КЛАССИКА COMPUTER SCIENCE

КОМПЬЮТЕРНЫЕ СЕТИ

ПЯТОЕ ИЗДАНИЕ

Э. ТАНЕНБАУМ
Д. УЭЗЕРОЛЛ

PEARSON Prentice Hall

ПИТЕР®

Andrew Tanenbaum, David Wetherall

Computer Networks

5th Edition

Upper Saddle River, New Jersey 07458

КЛАССИКА COMPUTER SCIENCE

Э. ТАНЕНБАУМ, Д. УЭЗЕРОЛЛ

КОМПЬЮТЕРНЫЕ СЕТИ

ПЯТОЕ ИЗДАНИЕ

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2012

ББК 32.973.202+32.988.02
УДК 004.738.5
Т18

Таненбаум Э., Уэзеролл Д.
Т18 Компьютерные сети. 5-е изд. — СПб.: Питер, 2012. — 960 с.: ил.

ISBN 978-5-459-00342-0

Перед вами — очередное, пятое издание самой авторитетной книги по современным сетевым технологиям, написанной признанным экспертом в этой области Эндрю Таненбаумом в соавторстве с профессором Вашингтонского университета Дэвидом Уэзеролом. Первая версия этого классического труда появилась на свет в далеком 1980 году, и с тех пор каждое издание книги неизменно становилось бестселлером и использовалось в качестве базового учебника в ведущих технических вузах.

В книге последовательно изложены основные концепции, определяющие современное состояние и тенденции развития компьютерных сетей. Авторы подробнейшим образом объясняют устройство и принципы работы аппаратного и программного обеспечения, рассматривают все аспекты и уровни организации сетей — от физического до уровня прикладных программ. Изложение теоретических принципов дополняется яркими, показательными примерами функционирования Интернета и компьютерных сетей различного типа. Пятое издание полностью переработано с учетом изменений, происшедших в сфере сетевых технологий за последние годы и, в частности, освещает такие аспекты, как беспроводные сети стандарта 802.12 и 802.16, сети 3G, технология RFID, инфраструктура доставки контента CDN, пиринговые сети, потоковое вещание, интернет-телефония и многое другое.

ББК 32.973.202+32.988.02
УДК 004.738.5

ISBN 978-0132126953 англ.
ISBN 978-5-459-00342-0

Краткое оглавление

Оглавление

Сьюзан, Барбаре, Даниэлю, Арону, Марвину,
Матильде, а также памяти Брэма и Крошки π.
Э. Таненбаум

Кетрин, Люси и Пепперу.
Д. Уэзеролл

Предисловие

Вот и вышло в свет уже пятое издание этой книги. Каждое издание соответствует своему периоду развития компьютерных сетей. Так в 1980 году, когда появилось первое из них, сети были лишь академической диковинкой. Второе издание (1988 г.) пришлось на те времена, когда сетевые технологии стали применяться в университетах и большом бизнесе. В 1996 году появилось третье издание, и уже тогда сети, особенно Интернет, стали ежедневной реальностью для миллионов людей. В четвертом издании (2003 г.) появились беспроводные сети и мобильный Интернет. А в пятом издании необходимо было рассмотреть вопросы распределения контента (например, видео с использованием CDN и одноранговых сетей) и мобильные телефоны, как маленькие компьютеры на просторах Интернета.

Среди множества изменений в этой книге наиболее существенным является то, что у книги появился еще один автор — профессор Дэвид Уэзеролл. Дэвид занимается сетями уже более 20 лет. Он продолжает интересоваться Интернетом и беспроводными сетями и сейчас. Последние десять лет Дэвид Уэзеролл занимает должность профессора в университете Вашингтон, где занимается исследованиями и читает лекции по тематике компьютерных сетей.

Естественно, изменения, произошедшие в мире компьютерных сетей, не могли не отразиться в книге:

- беспроводные сети (802.12 и 802.16);
- 3G-сети, используемые смартфонами;
- RFID и сенсорные сети;
- распределение контента с использованием CDN;
- одноранговые сети;
- медиа, работающие в режиме реального времени;
- интернет-телефония (IP-телефония);
- сети, устойчивые к задержкам.

В изданиях по компьютерной тематике всегда много сокращений. Данная книга не стала исключением. После ее прочтения вы будете легко оперировать следующими аб-

бревиатурами: ADSL, AES, AMPS, AODV, ARP, ATM, BGP, CDMA, CDN, CGI, CIDR, DCF, DES, DHCP, DMCA, FDM, FHSS, GPRS, GSM, HDLC, HFC, HTML, HTTP, ICMP, IMAP, ISP, ITU, LAN, LMDS, MAC, MACA, MIME, MPEG, MPLS, MTU, NAP, NAT, NSA, NTSC, OFDM, OSPF, PCF, PCM, PGP, PHP, PKI, POTS, PPP, PSTN, QAM, QPSK, RED, RFC, RPC, RSA, RSVP, RTP, SSL, TCP, TDM, UDP, URL, UTP, VLAN, VPN, VSAT, WAN, WAP, WDMA, WEP, WWW и XML. Каждая из них будет расшифрована, так что волноваться не стоит, достаточно внимательно читать книгу.

От издательства

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на веб-сайте издательства http://www.piter.com.

Глава 1
Введение

Каждое из трех прошедших столетий ознаменовалось преобладанием своей господствующей технологии. XVIII столетие было веком индустриальной революции и механизации. В XIX веке наступила эпоха паровых двигателей. В течение XX века главной технологией стали сбор, обработка и распространение информации. Среди прочих разработок следует отметить создание глобальных телефонных сетей, изобретение радио и телевидения, рождение и небывалый рост компьютерной индустрии, запуск спутников связи и, конечно, Интернет.

Благодаря высокой скорости технологического прогресса эти области очень быстро проникают друг в друга. При этом в XXI веке различия между сбором, транспортировкой, хранением и обработкой информации продолжают быстро исчезать. Организации с сотнями офисов, разбросанных по всему миру, должны иметь возможность получать информацию о текущем состоянии своего самого удаленного офиса мгновенно, нажатием кнопки. По мере роста нашего умения собирать, обрабатывать и распространять информацию, потребности в средствах еще более сложной обработки информации растут все быстрее.

Хотя компьютерная индустрия еще довольно молода по сравнению с другими производствами (например, автомобильной или авиационной промышленностью), прогресс в сфере производства компьютеров был весьма впечатляющим. В первые два десятилетия своего существования компьютерные системы были сильно централизованными и располагались, как правило, в пределах одного помещения. Часто эта комната оборудовалась стеклянными стенами, сквозь которые посетители могли полюбоваться на великое электронное чудо. Компания среднего размера или университет могли позволить себе один-два компьютера, тогда как у очень крупных организаций их число могло достигать нескольких десятков. Сама мысль о том, что через какие-нибудь 40 лет гораздо более мощные компьютеры будут иметь размеры почтовой марки и производиться миллиардами, тогда казалась чистой фантастикой.

На слияние компьютеров и коммуникаций существенно влияет принцип организации компьютерных систем. Некогда доминирующее понятие «вычислительного центра» как комнаты с большим компьютером, к которому пользователи приносят свою работу для обработки, является теперь полностью устаревшим (хотя распространены информационные центры, содержащие тысячи интернет-серверов). Старая модель единственного компьютера, служащего всем вычислительным потребностям организации, была заменена на схему, при которой задание выполняет большое количество отдельных, но связанных компьютеров. Эти системы называют компьютерными сетями. Проектирование и организация этих сетей — тема нашей книги.

Мы будем использовать термин «компьютерная сеть», чтобы означать набор автономных компьютеров, связанных одной технологией. Два компьютера называют связанными, если они в состоянии обмениваться информацией. Соединение не обязательно должно представлять собой медный провод; может использоваться волоконная оптика, микроволны, инфракрасный диапазон и спутники связи. Сети бывают различных размеров, формы и конфигураций. Они обычно соединяются вместе, чтобы создать большие сети, самым известным примером сети сетей является Интернет.

В литературе существует путаница между понятиями компьютерная сеть и распределенная система. Основное их различие заключается в том, что в распределенной системе наличие многочисленных автономных компьютеров незаметно для пользователя. С его точки зрения, это единая связанная система. Обычно имеется набор программного обеспечения на определенном уровне (над операционной системой), которое называется связующим ПО и отвечает за реализацию этой идеи. Хорошо известный пример распределенной системы — это Всемирная паутина (World Wide Web), в которой, с точки зрения пользователя, все выглядит как документ (веб-страница).

В компьютерных сетях нет никакой единой модели, нет и программного обеспечения для ее реализации. Пользователи имеют дело с реальными машинами, и со стороны вычислительной системы не осуществляется никаких попыток связать их воедино. Скажем, если компьютеры имеют разное аппаратное и программное обеспечение, пользователь не сможет этого не заметить. Если он хочет запустить программу на удаленной машине, ему придется явно зарегистрироваться на ней и явно дать задание на запуск.

На самом деле, распределенная система является программной системой, построенной на базе сети. Эта программная система обеспечивает высокую степень связности и прозрачности элементов. Таким образом, различие между компьютерной сетью и распределенной системой заключается в программном обеспечении (особенно в операционной системе), а не в аппаратуре.

Тем не менее эти два понятия имеют очень много общего. Например, как компьютерная сеть, так и распределенная система занимаются перемещением файлов. Разница заключается в том, кто вызывает эти перемещения — система или пользователь.

Хотя основной темой этой книги являются сети, многие разделы будут касаться и распределенных систем. Дополнительную информацию о распределенных системах см. [Tannenbaum, van Steen, 2007].

1.1. Применение компьютерных сетей

Прежде чем приступить к изучению технических подробностей, стоит посвятить некоторое время обсуждению вопроса, почему люди интересуются компьютерными сетями и для чего они могут быть использованы. В конце концов, если бы никто не был заинтересован в развитии этих технологий, то не было бы построено такое огромное количество самых разных сетей. Мы начнем с обсуждения таких традиционных вещей, как сети в организациях, затем перейдем к домашним сетям и новым технологиям, связанным с мобильной связью и мобильными пользователями, и закончим социальными вопросами.

1.1.1. Сети в организациях

Большинство современных организаций используют большое количество компьютеров. Например, компания может иметь компьютер для каждого сотрудника и использовать их, чтобы разрабатывать продукты, писать брошюры и делать платежные ведомости. Первоначально некоторые из этих компьютеров, возможно, работали в изоляции от других, но в некоторый момент управление, возможно, решило соединить их, чтобы быть в состоянии передавать информацию по всей компании.

Если посмотреть на эту проблему с более общих позиций, то вопросом здесь является совместное использование ресурсов, а целью — предоставление доступа к программам, оборудованию и особенно данным для любого пользователя сети, независимо от физического расположения ресурса и пользователя. В качестве примера можно привести сетевой принтер, то есть устройство, доступ к которому может осуществляться с любой рабочей станции сети. Это выгодное решение, поскольку нет никакой необходимости в том, чтобы свое печатающее устройство было у каждого служащего, к тому же, содержание и обслуживание одного принтера, очевидно, обходится дешевле.

Но, наверное, даже более важной проблемой, нежели совместное использование физических ресурсов, таких как принтеры и устройства резервного копирования, является совместное использование информации . В наше время любая компания, независимо от ее размеров, просто немыслима без данных, представленных в электронном виде. Маленькие и большие компании жизненно зависят от компьютеризированной информации. У большинства компаний в сети доступны потребительские отчеты, информация о продукте, материальные запасы, финансовые отчеты, информация о налоге и многое дгугое. Если бы вдруг внезапно отказали все компьютеры какого-нибудь банка, даже самого крупного, он обанкротился бы минут за пять, не более. Современное автоматизированное производство с использованием вычислительной техники в этом случае не продержалось бы и пяти секунд. Да что там говорить, если даже маленькое туристическое агентство, весь штат которого состоит из трех человек, находится в очень сильной зависимости от компьютерных сетей, позволяющих получать доступ к необходимой информации и документам.

В маленьких компаниях все компьютеры обычно собраны в пределах одного офиса или, в крайнем случае, одного здания. Если же речь идет о больших фирмах, то и вычислительная техника, и служащие могут быть разбросаны по десяткам представительств в разных странах. Несмотря на это, продавец, находящийся в Нью-Йорке, может запросить и сразу же получить информацию о товарах, имеющихся на складе в Сингапуре. Для соединения сетей, расположенных в разных местах, могут быть использованы сети, называемые VPN (Virtual Private Networks — виртуальные частные сети). Другими словами, тот факт, что пользователь удален от физического хранилища данных на 15 тысяч километров, никак не ограничивает его возможности доступа к этим данным. Можно сказать, что одной из целей сетей является борьба с «тиранией географии».

Проще всего информационную систему компании можно представить себе как совокупность одной или более баз данных с информацией компании и некоторого количества работников, которым удаленно предоставляется информация. В этом случае данные хранятся на мощном компьютере, называемом **сервером**. Довольно часто

сервер располагается в отдельном помещении и обслуживается системным администратором. С другой стороны, компьютеры служащих могут быть менее мощными, они идентифицируются в сети как **клиенты**, могут в большом количестве располагаться даже в пределах одного офиса и иметь удаленный доступ к информации и программам, хранящимся на сервере. (Иногда мы будем называть «клиентом» пользователя такой машины. Я думаю, вы сможете по контексту догадаться, когда речь идет о компьютере, а когда о человеке.) Клиентская и серверная машины объединены в сеть, как показано на рис. 1.1. Обратите внимание: пока что мы показываем сеть просто в виде овала, не вдаваясь в детали. Такое представление мы будем использовать при ведении наиболее абстрактного разговора о компьютерных сетях. При обсуждении того или иного аспекта их функционирования мы будем «раскрывать» этот овал, узнавая о нем все новые подробности.

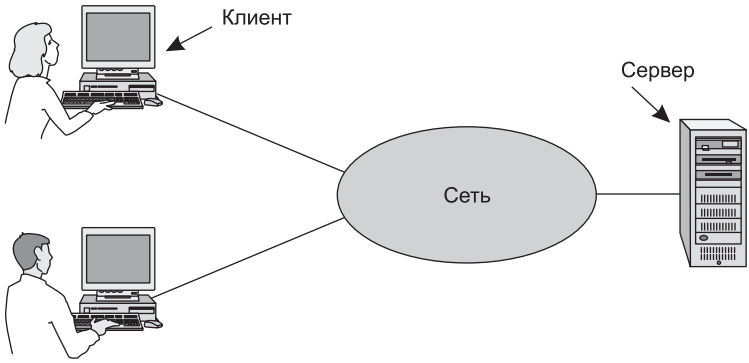


Рис. 1.1. Сеть, состоящая из двух клиентов и одного сервера

Такая система называется **клиент-серверной моделью**. Она используется очень широко и зачастую является основой построения всей сети. Самая популярная реализация — веб-приложение, в котором сервер производит веб-страницы, основанные на его базе данных в ответ на запросы клиента, которые могут обновить базу данных. Она применима, когда клиент и сервер находятся в одном здании и принадлежат одной компании, а также когда они расположены далеко друг от друга. Скажем, когда пользователь получает доступ к интернет-сайту, работает та же модель. При этом веб-сервер играет роль серверной машины, а пользовательский компьютер — клиентской. В большинстве случаев один сервер одновременно занимается обслуживанием большого (сотен или тысяч) числа клиентов.

Если мы посмотрим на модель «клиент-сервер» чуть пристальнее, то станет очевидно, что в работе сети можно всегда выделить два процесса (то есть работающие программы): серверный и клиентский. Обмен информацией чаще всего происходит так. Клиент посылает запрос серверу через сеть и начинает ожидать ответ. При принятии запроса сервер выполняет определенные действия или ищет запрашиваемые данные, затем отсылает ответ. Все это показано на рис. 1.2.

Вторая цель работы компьютерной сети связана в большей степени с людьми, чем с информацией или вычислительными машинами. Дело в том, что сеть — это

замечательная **коммуникационная среда** для работников предприятия. Почти в любой компании найдется хотя бы один компьютер, умеющий принимать и отправлять **электронную почту (e-mail)**, а ведь именно ее большинство людей предпочитает использовать для общения. На самом деле, обычное ворчание начальства на тему того, что люди проводят много времени за чтением и написанием электронной почты, совершенно беспочвенно: многие руководители давно уже поняли, что они и сами могут рассылать своим подчиненным электронные послания, это удобно и просто.

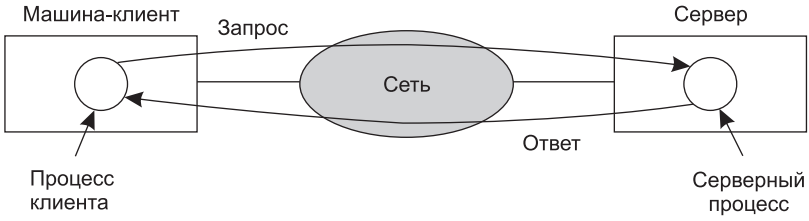


Рис. 1.2. В модели «клиент-сервер» различают запросы и ответы

Телефонные звонки между служащими могут передаваться по компьютерной сети, вместо телефонной. Эту технологию называют IP-телефонией или VoIP (Voice over IP). Микрофон и динамик в каждом конце могут принадлежать VoIP-включенному телефону или компьютеру сотрудника. Компании считают это замечательным способом экономить на телефонных счетах.

С помощью компьютерных сетей возможны и другие, более богатые формы общения. К аудио может быть добавлено видео, так чтобы сотрудники в отдаленных местоположениях могли видеть и слышать друг друга в течение встречи. Этот метод — мощный инструмент для того, чтобы устранить стоимость и время, которые раньше тратились на поездки. Совместный доступ к рабочему столу позволяет удаленным сотрудникам видеть и взаимодействовать с графическим монитором. Это облегчает работу для двух или нескольких человек, которые работают далеко друг от друга, при необходимости совместно проводить расчеты или писать отчет. Когда один сотрудник производит изменение в документе онлайн, другие могут немедленно видеть изменение, вместо ожидания письма в течение нескольких дней. Такое ускорение делает сотрудничество среди обширных групп людей легким там, где это ранее было невозможно. Начинают использоваться и более амбициозные формы удаленной координации, такие как телемедицина (например, осмотр удаленного пациента), но они могут стать намного более важными. Иногда говорят, что происходит гонка между коммуникациями и транспортом, и тот, кто не победит, устареет.

Третья цель для многих компаний — заниматься коммерцией с помощью электроники, особенно с клиентами и поставщиками. Эту новую модель называют электронной коммерцией (e-commerce), и она в последние годы быстро растет. Авиалинии, книжные магазины и другие продавцы обнаружили, что многим клиентам так удобно посещать магазин из дома. Следовательно, много компаний обеспечивают каталоги своих товаров и услуг онлайн и делают заказы онлайн.

Производители автомобилей, летательных аппаратов, компьютеров закупают комплектующие и детали у огромного числа поставщиков, а затем занимаются сборкой

конечной продукции. С помощью компьютерных сетей процесс составления и отправки заказов можно автоматизировать. Более того, заказы могут формироваться строго в соответствии с производственными нуждами, что позволяет резко повысить эффективность.

1.1.2. Использование сетей частными лицами

В 1977 году Кен Олсен (Ken Olsen) был президентом корпорации DEC (Digital Equipment Corporation), которая на тот момент была второй по величине (после IBM) компанией, производящей компьютерную технику. Когда у него спросили, почему DEC не поддерживает идею создания персональных компьютеров, он сказал: «Я не вижу никакого смысла в том, чтобы в каждом доме стоял компьютер». Возможно, он и был прав, но исторический факт заключается в том, что все оказалось как раз наоборот, а корпорация DEC вообще прекратила свое существование. Зачем люди устанавливают компьютеры у себя дома? Изначально основной целью было редактирование текстов и электронные игры. Недавно самой большой причиной купить домашний компьютер был, вероятно, доступ к Интернету. Теперь многие бытовые электронные устройства, такие как цифровые приемники, игровые приставки и радиоприемники с часами, производятся со встроенными компьютерами и компьютерными сетями, особенно беспроводными сетями, и домашние сети широко используются для развлечения, включая слушание, просмотр, и создание музыки, фотографий и видео.

Доступ к Интернету предоставляет домашним пользователям связь с удаленными компьютерами. Как и в случае с компаниями, домашние пользователи могут получить доступ к информации, общаться с другими людьми и купить продукты и услуги с помощью электронной коммерции. Основная выгода теперь возникает из соединения за пределами дома. Боб Меткэйлф, изобретатель Ethernet, выдвигал гипотезу, что значение сети пропорционально квадрату числа пользователей, потому что это — примерно число различных соединений, которые могут быть сделаны (Гилдер, 1993). Эта гипотеза известна как «закон Меткэйлфа». Она помогает объяснить, как огромная популярность Интернета возникает из его размера.

Доступ к удаленной информации может осуществляться в различной форме. Можно бродить по Сети в поисках нужной или просто интересной информации. При этом практически невозможно найти такую область знаний, которая не была бы представлена в Интернете. Там есть искусство, бизнес, кулинария, политика, здоровье, история, различные хобби, отдых, наука, спорт, путешествия и многое-многое другое. Развлекательных тем безумное количество, причем некоторые из них не стоит даже упоминать.

Многие газеты стали доступны в электронном виде, их можно персонализировать. Например, можно заказать себе все статьи, касающиеся коррупции среди политических деятелей, больших пожаров, скандалов, связанных со знаменитостями, эпидемий, но отказаться от статей о футболе. Можно сделать и так, чтобы ночью газета загружалась на ваш компьютер пока вы спите. Конечно, это все может привести к массовой безработице среди подростков, продающих на улицах газеты, но сами редакции довольны таким поворотом событий, поскольку распространение всегда было самым слабым звеном в их производственной цепочке. Разумеется, чтобы эта модель

заработала, надо сначала выяснить, как же делать деньги в этом новом мире, что не совсем очевидно, так как пользователи Интернета ожидают, что все будет бесплатно.

Следующий шаг после создания электронных версий газет и журналов — это онлайновые библиотеки. Многие профессиональные организации, такие как ACM (*www.acm.org*) и даже объединение IEEE (*www.computer.org*), уже занимаются этим. Да и другие фирмы и частные лица выкладывают свои коллекции самых разнообразных материалов в Интернете. Электронные книги и он-лайн библиотеки могут привести к тому, что печатные издания начнут морально устаревать. Скептики уже сейчас сравнивают это с эффектом от появления в средние века печатного станка, который заменил ручное письмо.

Доступ к большей части информации осуществляется по модели клиент-сервер, но есть и другой популярный тип сетевого общения, основанный на технологии **равноранговых сетей (peer-to-peer)**, (Parameswaran и др., 2001). Люди, входящие в некоторую группу пользователей, могут общаться друг с другом. В принципе, каждый может связаться с каждым, разделение на клиентские и серверные машины в этом случае отсутствует. Это показано на рис. 1.3.

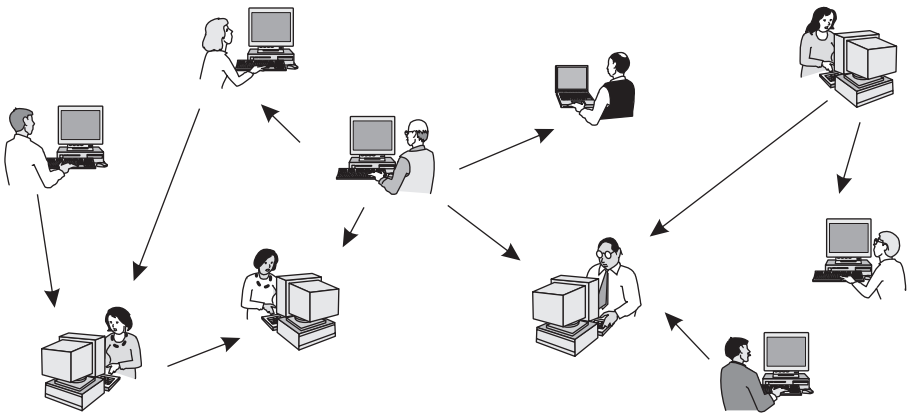


Рис. 1.3. В равноранговой сети нет четко определенных клиентов и серверов

Многие одноранговые сети, такие как BitTorrent (Cohen, 2003), не имеют никакой центральной базы данных контента. Вместо этого каждый пользователь поддерживает свою собственную базу данных в местном масштабе и обеспечивает список других людей по соседству, которые являются членами системы. Новый пользователь может пойти к любому существующему участнику, увидеть то, что он имеет, и получить имена других участников, таким образом получая доступ к большему количеству контента и большему количеству имен. Этот процесс поиска может повторяться сколь угодно долго, чтобы в результате создать большую местную базу данных того, что имеется в наличии. Такая деятельность была бы утомительной для людей, но компьютер прекрасно с этим справляется.

Коммуникация соединения равноправных узлов ЛВС часто задействуется, чтобы совместно использовать музыку и видео. Этого рода коммуникации стали очень популярны примерно в 2000 году, они были реализованы с помощью службы Napster,

которая была закрыта после того, как произошел самый большой скандал, связанный с нарушением закона об авторских правах за всю историю звукозаписи (Lam and Tan, 2001; и Macedonia, 2000). Между тем, существуют и легальные равноранговые сети. Они включают поклонников, совместно использующих общественную музыку домена, семьи, обменивающиеся фотографиями и фильмами и пользовательскими пакетами программ открытого доступа. Между прочим, не стоит забывать и о том, что самая популярная интернет-технология — e-mail — выросла именно из идеи равноранговых сетей. Такой вид коммуникаций вообще является перспективным, и в будущем он еще будет довольно активно развиваться.

Все вышеупомянутые приложения вовлекают взаимодействия между человеком и удаленной базой данных, полной информации. Вторая широкая категория использования сетей — коммуникация от человека к человеку, ответ 21-го столетия телефону XIX века. Электронная почта уже используется ежедневно миллионами людей во всем мире и ее использование быстро растет. Она уже привычно содержит не только текст и изображения, но и аудио и видео. Передача запаха, возможно, вопрос времени.

Все уважающие себя подростки увлекаются **обменом мгновенными сообщениями**. Эта возможность, полученная из UNIX-программы *talk*, использовавшейся приблизительно с 1970 года, позволяет обмениваться сообщениями в режиме реального времени. Есть и службы обмена сообщениями для нескольких человек, такие как **Twitter**, которая позволяет посылать короткие текстовые сообщения, названные tweets, «чириканьем», своему кругу друзей или тем, кто на это согласился.

Интернет может использоваться приложениями для передачи аудио (например, интернет-радиостанции) и видео (например, YouTube). Кроме того, что это дешевый способ позвонить тому, кто находится далеко, эти приложения могут обеспечить, например, дистанционное обучение, когда можно быть на уроке в 8 утра, не вставая для этого из кровати. В конечном счете, использование сетей, чтобы улучшить общение между людьми, может оказаться важнее, чем любое другое. Это может быть очень важно для людей, ущемленных географически, которые получат такие же возможности доступа, как и люди, живущие в центре большого города.

Промежуточное положение между коммуникациями от человека к человеку и доступом к информации занимают **социальные сети**. Здесь поток информации управляется отношениями, которые люди объявляют друг между другом. Один из самых популярных сайтов социальных сетей — **Facebook**. Он позволяет обновлять свои личные профили и получать общий доступ к этим обновлениям с теми, кого они объявили своими друзьями. Другие приложения социальных сетей могут быть представлены через друзей друзей, посылать сообщения новостей друзьям, такие как упомянутый выше Twitter, и многое другое.

Еще более свободно группы людей могут сотрудничать, создавая контент. Технология **wiki**, например, это веб-сайт, который совместно редактируют члены сообщества. Самая известная wiki — **Википедия**, энциклопедия, которую любой может редактировать, но есть тысячи других сайтов wiki.

Третьей выделенной нами категорией является электронная коммерция в самом широком смысле слова. Закупка продуктов и вещей для дома через Интернет уже давно стала привычным делом. У покупателя появился богатый выбор, поскольку компаний, предлагающих приобрести что-либо по веб-каталогу, с каждым днем становится

все больше, и счет уже идет на тысячи. Некоторые из этих каталогов интерактивны, показывают товары с различных точек зрения и имеют возможности персональной настройки. Если клиент купил некий продукт, но не знает, что с ним делать, ему поможет сетевая служба поддержки.

Электронная коммерция широко используется и в другом серьезном направлении: организуется доступ к финансовым учреждениям. Многие уже давно оплачивают свои векселя, управляют банковскими счетами и работают с вкладами с помощью соответствующих сетевых служб. По мере прогресса в области защиты передаваемой информации эта тенденция, безусловно, продолжится.

Еще одна область, которую вряд ли кто-то мог предвидеть, это электронные «блошиные» рынки. Онлайновые аукционы, на которых распродают подержанные вещи, стали довольно мощной сетевой индустрией. В отличие от традиционных форм электронной коммерции, использующих модель «клиент-сервер», такие разновидности бизнеса ближе к равноранговым сетям, в том смысле, что пользователи могут действовать и как продавцы, и как покупатели. Некоторые формы сетевой коммерции со временем обрели сокращенные обозначения, наиболее популярные приведены в табл. 1.1. (Английский предлог *to*, отвечающий на вопросы *«кому»*, *«чему»*, произносится так же, как *2* (*two*). Исходя из этого и выбирались обозначения.)

Таблица 1.1. Некоторые формы электронной коммерции

Обозначение	Полное название	Пример
B2C	Коммерсант клиенту (Business-to-Consumer)	Заказ книг в режиме on-line
B2B	Коммерсант коммерсанту (Business-to-Business)	Производитель автомобилей заказывает покрышки у поставщика
G2C	Государство клиенту (Government-to-Client)	Распространение бланков квитанций через Интернет
C2C	Клиент клиенту (Client-to-Client)	Продажа подержанных вещей
P2P	Равноранговые сети (Peer-to-Peer)	Совместный доступ к музыкальным файлам

Наша четвертая категория — развлечения. Она делает огромные успехи в последние годы с распространением музыки, радио- и телевизионных программ и фильмов в Интернете, начинающем конкурировать с традиционными механизмами. Пользователи могут найти, купить и загрузить песни в MP3 и фильмы качества DVD и добавить их в свою личную коллекцию. Телевизионные программы многие смотрят теперь по системе **IPTV** (IPTeleVision), которая основана на IP-технологии вместо кабельного телевидения или радиопередач. Приложения, передающие потоковое мультимедиа, позволяют пользователям слушать интернет-радиостанции или смотреть новые эпизоды их любимых сериалов. Естественно, весь этот контент может быть перемещен у вас дома от устройства к устройству, между дисплеями и динамиками, обычно с помощью беспроводной сети.

Скоро, может быть, будет возможно выбрать любой фильм или телевизионную передачу из когда-либо и где-либо снятых и увидеть ее мгновенно на своем экране.

Новые фильмы могут стать интерактивными, где пользователю периодически будет предлагаться выбор сюжетной линии (должен ли Макбет убить Дункана сейчас или немного подождать?) с альтернативными сценариями, заготовленными для всех случаев. Телевидение тоже может стать интерактивным — с участием аудитории в викторинах, с возможностью выбора победителя и т. п.

Другая форма развлечения — игры. Уже сейчас существуют игры-симуляторы в реальном времени с большим количеством участников, например прятки в виртуальном средневековом замке или симуляторы полетов, в которых одна команда пытается сбить игроков команды противника. Виртуальные миры обеспечивают постоянное место действия, где тысячи пользователей могут иметь совместный доступ к виртуальной реальности с трехмерной графикой.

Наша последняя категория — **вездесущие вычисления**, когда вычисления встроены в повседневную жизнь, как в видении Марка Веизера (1991). Во многих домах уже установлены системы обеспечения безопасности, которые включают датчики на дверях и окнах и еще много вариантов датчиков, которые могут быть подключены к умному домашнему монитору, например, контролирующие потребление энергии. Ваши счетчики электричества, газа и воды тоже могут сообщать показания по сети. Это экономило бы деньги, поскольку не будет никакой потребности посылать контролеров учета. А детекторы дыма могли звонить в отдел пожарной охраны, вместо того чтобы издавать громкий звук (в котором мало смысла, если никого нет дома). В результате снижения стоимости датчиков и коммуникаций количество измерений, производимых с задействованием сетей, будет расти.

Все большее количество потребительских электронных устройств объединяется в сеть. Например, некоторые высококачественные камеры уже имеют способность присоединяться к беспроводной сети и используют ее, чтобы послать фотографии на дисплей для просмотра. Фотографы профессионального спорта могут послать свои фотографии редакторам в реальном времени, соединившись с помощью беспроводных технологий с точкой доступа и далее по Интернету. Устройства, такие как телевизоры, могут использовать линии сети электропередачи, чтобы послать информацию всюду по дому по электрическим проводам. Возможно, не очень удивительно иметь эти объекты в сети, но обмениваться информацией могут и объекты, о которых мы совсем не думаем, как о компьютерах. Например, ваш душ может сделать запись использования воды, дать вам визуальную обратную связь, в то время как вы намыливаетесь, и сообщить домашнему экологическому контрольному приложению, когда вы закончили мыться, чтобы помочь экономить на расходах за воду.

Технология под названием **RFID** (Radio Frequency Identification, радиочастотная идентификация) продвинет эту идею еще далее в будущем. Метки RFID пассивны (то есть не имеют никакой батареи), имеют размер почтовой марки и уже могут быть прикреплены к книгам, паспортам, домашним животным, кредитным картам и другим элементам дома и на улице. Это позволяет считывателям RFID определять местонахождение и «общаться» на расстоянии до нескольких метров, в зависимости от вида RFID. Первоначально коммерческой целью RFID было заменить штриховые коды. Это еще не произошло, потому что штриховые коды бесплатны, а теги RFID стоят несколько центов. Конечно, теги RFID предлагают намного больше, и их цена быстро уменьшается. Они могут превратить реальный мир в Интернет (ITU, 2005).

1.1.3. Использование беспроводных сетей

Мобильные компьютеры — ноутбуки и наладонные компьютеры — это еще одна область, в которой сейчас наблюдается бурное развитие. Их продавцы уже обогнали производителей настольных систем. Их продажи уже достигли уровня продаж настольных компьютеров. Почему люди хотят их иметь? Потому что они хотят использовать мобильные устройства при перемещении, чтобы читать и посылать электронные письма, записи твиттера, смотреть кино, скачивать музыку, играть в игры или просто искать информацию в сети. То есть делать все то, что они делают дома и в офисе. Естественно, они хотят делать это всюду на земле, море или в воздухе.

Соединение с Интернетом делает доступным полезные мобильные возможности. Так как обычные сети, в которых информация передается по проводам, невозможно использовать в машине, лодке или самолете, люди проявляют большой интерес к беспроводным сетям. Сотовые сети, которыми управляют телефонные компании, — это хорошо известный вид беспроводной сети, которая осуществляет покрытие для использования мобильных телефонов. Другой вид беспроводной сети для мобильных компьютеров — беспроводные точки доступа, основанные на стандарте 802.11. Они возникли всюду, куда идут люди, образуя лоскутное покрытие в кафе, отелях, аэропортах, школах, поездах и самолетах.

Имея ноутбук и беспроводный модем, можно включить его и подключиться к Интернету через точку доступа, как если бы компьютер был подключен к проводной сети.

Такого рода сети уже давно и с успехом применяются в больших компаниях, занимающихся грузоперевозками, в таксопарках, службах доставки почты и ремонта. Там это жизненно необходимо — как для отслеживания пути следования транспорта и груза, так и для поддержки постоянной связи с диспетчерами. Например, во многих местах водители такси являются частными предпринимателями, не относящимися к какому-либо таксопарку. У них в машинах имеется электронное табло. Когда на централизованный пульт поступает заявка, диспетчер вводит данные о местонахождении пассажира и требуемом месте назначения. Информация появляется одновременно на экранах у всех водителей и сопровождается звуковым сигналом. Тот водитель, который первым нажмет кнопку, считается принявшим заявку.

Беспроводные сети находят широкое применение и в военном деле. При ведении боевых действий в совершенно произвольном месте планеты не приходится рассчитывать на инфраструктуру местных сетей связи, и нужно организовывать свою сеть.

Хотя мобильные компьютеры и беспроводные сети тесно связаны между собой, все же это не одно и то же. Это отражено в табл. 1.2. Можно заметить, что есть разница между **стационарными** и **мобильными беспроводными сетями**. Даже ноутбуки иногда подключают к обычной компьютерной сети. Если пользователь, скажем, в номере гостиницы подключит его к телефонной розетке, он получит мобильность при отсутствии беспроводной сети.

С другой стороны, и наличие беспроводной сети еще не говорит о наличии мобильности.

Дома, в офисах или отелях, которые испытывают недостаток в проложенных кабелях, может быть более удобно соединить настольные компьютеры или медиапроигрыватели с помощью беспроводных технологий, чем провести провода.

Таблица 1.2. Сочетания беспроводных сетей и мобильных компьютеров

Беспроводная сеть	Мобильность	Применения
Нет	Нет	Настольные компьютеры в офисах
Нет	Есть	Ноутбук в номере гостиницы
Есть	Нет	Сети в старых зданиях, в которых не проложена проводка
Есть	Есть	Наладонный компьютер, хранящий информацию о товарах

Наконец, есть и полноценные мобильные применения беспроводных сетей, такие как складской учет с помощью карманных компьютеров. Во многих крупных и загруженных аэропортах служащие, принимающие на стоянках арендованные машины, зачастую используют мобильные компьютеры. Они сканируют штрих-код или RFID-метку машины, и их мобильное устройство, имеющее встроенный принтер, связывается с главным компьютером, получают от него информацию и сразу же распечатывает счет.

Возможно, основной двигатель развития приложений беспроводной связи — мобильный телефон. Обмен текстовыми сообщениями чрезвычайно популярен. Это возможность для пользователя мобильного телефона вводить короткие сообщения, которые будут доставлены сотовой связью другому мобильному абоненту. Немного людей предсказали бы десять лет назад, что подростки, бесконечно набирающие короткие текстовые сообщения на мобильных телефонах, станут огромным источником дохода для телефонных компаний. Но **texting** (или **Short Message Service**, как это называют вне США) очень выгоден, так как передача текстового сообщения стоит малую долю цента, а берут за это гораздо больше.

Долгожданная конвергенция телефонов и Интернета наконец наступила, и это ускорит рост мобильных приложений. **Умные телефоны** (**Smart Phones**), такие как популярный iPhone, комбинируют аспекты мобильных телефонов и мобильных компьютеров. Сотовая связь третьего и четвертого поколений, с помощью которой они соединяются, может обеспечить быстрые информационные службы для использования Интернета, так же как для обработки телефонных звонков. Много усовершенствованных телефонов соединяются и с беспроводными точками доступа, и автоматически переключаются между сетями, чтобы выбрать наилучший вариант для пользователя.

Другие устройства бытовой электроники могут использовать сотовые сети и сети с точками доступа для соединения с удаленными компьютерами. Электронные устройства чтения книг могут загрузить недавно купленную книгу или очередной выпуск журнала или сегодняшней газеты везде, где бы они ни находились. Электронные фоторамки могут обновить изображения на своих дисплеях.

Так как мобильные телефоны знают свое местоположение, часто потому что они оборудованы **GPS** (**Система глобального позиционирования**), некоторые службы зависят от местоположения. Мобильные карты и указатели — очевидный кандидат, поскольку и ваш телефон, и автомобиль с GPS, вероятно, лучше вас знают, где вы находитесь. Такая же ситуация с поиском ближайшего книжного магазина, или китайского ресторана, или местного прогноза погоды. Другие службы могут записывать текущее

местоположение, например указывать для фотографий и видео место, в котором они были сделаны. Такие указания называют «геотегирование».

Областью, в которой начинают использоваться мобильные телефоны, является m-commerce (мобильная торговля) (Senn, 2000). Короткие текстовые сообщения с мобильного телефона используются вместо наличных денег и кредитных карт для оплаты продуктов в торговых автоматах, билетов в кино и других мелочах. Оплата будет включена в счет мобильного телефона. Мобильный телефон, снабженный технологией NFC (Near Field Communication), может действовать как карта RFID и взаимодействовать для оплаты с ближайшим считывателем. Движущие силы этого явления — производители мобильных устройств и сетевые операторы, которые очень стараются узнать, как получить часть пирога электронной коммерции. С точки зрения магазина, эта схема может сохранить им большую часть дохода компаний кредитных карт, который может составить несколько процентов. Впрочем, есть и оборотная сторона вопроса, вовсе не выгодная для магазина: клиент, прежде чем совершить покупку, с помощью своего PDA может узнать, где выбранные им товары можно купить дешевле. Более того, PDA могут быть снабжены небольшим встроенным сканером для чтения штрих-кода продукции и получения детальной информации о том, где еще и по какой цене она продается.

У операторов мобильных сетей, а значит и у м-коммерции есть одно замечательное преимущество. В отличие от пользователей Интернета, абоненты мобильных телефонов привыкли за все платить. Если на каком-нибудь сайте промелькнет упоминание о том, что за оплату с помощью кредитной карты будут взиматься какие-то сборы, то посетители поднимут большой шум. Если же оператор мобильной связи за небольшую плату любезно предоставит возможность оплатить покупки в магазине с помощью телефона, наверное, это будет воспринято нормально. Впрочем, время покажет.

В будущем, вероятно, будут развиваться технологии, основанные на всеобщей тенденции миниатюризации вычислительной техники. Давайте бросим беглый взгляд на некоторые возможности. Сенсорные сети составляются из узлов, которые собирают информацию о состоянии материального мира и с помощью беспроводных технологий передают ее. Узлы могут быть частью знакомых элементов, таких как автомобили или телефоны, или отдельными маленькими устройствами. Например, ваш автомобиль мог бы собрать данные о своем местоположении, скорости, вибрации и топливной экономичности от своей бортовой диагностической системы и загрузить эту информацию в базу данных (Hull и др., 2006). Эти данные могут помочь объехать выбоины, спланировать объезд переполненных дорог и сообщат вам, тратите ли вы на данном участке дороги слишком много бензина по сравнению с другими водителями.

Сенсорные сети видоизменяют науку, обеспечивая сбор ранее недоступного количества данных. Например, перемещение отдельных зебр отслеживается путем размещения маленького датчика на каждом животном (Juang и др., 2002). Исследователи упаковали беспроводный компьютер в куб со стороной 1 мм (Warneke и др., 2001). С помощью мобильных компьютеров можно проследить перемещения даже маленьких животных, птиц, грызунов и насекомых.

Даже приземленное использование, например в парковочных счетчиках, может быть существенным, так как используют данные, которые не были ранее доступны. Беспроводные счетчики времени стоянки могут принять кредит или платежи

дебетовой карты с мгновенной проверкой по беспроводной ссылке. Они могут также сообщить по беспроводной сети о своей занятости. Это позволило бы водителям загружать актуальную карту парковки и таким образом легче найти свободное место. По окончании оплаченного времени счетчик мог бы проверить присутствие автомобиля (отразив от него сигнал) и сообщить об истечении оплаченного времени. Было подсчитано, что только в США таким образом могло бы быть собрано дополнительно $10 миллиардов (Harte и др., 2000).

Еще одно многообещающее приложение — компьютеры, которые можно носить на себе. К умным часам со встроенным радиоприемником мы начали привыкать тогда, когда о них впервые рассказал в юмористическом монологе Дик Трэйси (Dick Tracy) в 1946 году, а сейчас вы можете их купить. Другие такие устройства могут быть внедрены, такие как насосы инсулина и электронные стимуляторы сердца. Некоторыми из них можно управлять по беспроводной сети. Это позволяет врачам легче проверять их и менять конфигурацию. Но может привести и к проблемам, если устройства столь же незащищены, как средний компьютер, и легко могут быть взломаны (Halperin и др., 2008).

1.1.4. Социальный аспект

Итак, подобно печатному станку 500 лет назад, компьютерные сети предоставляют новые способы распространения гражданами их взглядов среди самой различной аудитории. Новая свобода распространения информации несет с собой и новые нерешенные политические, социальные и этические проблемы. Упомянем кратко только некоторые из них; полное исследование потребовало бы, по крайней мере, книги.

Социальные сети, доски объявлений, сайты для хранения контента и узлы с другими приложениями позволяют людям делиться своими взглядами с аналогично мыслящими людьми.

До тех пор пока обсуждаемый предмет не выходил за рамки техники или увлечений вроде возделывания огородов, проблем, которые могут возникнуть, было не так уж много.

Проблемы начались с возникновением конференций, посвященных темам, по-настоящему волнующим людей, таким как политика, религия или секс. Взгляды, излагаемые одними людьми, могут оказаться оскорбительными для других. Они и в самом деле зачастую далеки от норм политкорректности. Кроме того, сетевые технологии, как известно, не ограничены только лишь передачей текста. Без особых проблем по Сети ходят фотографии с высоким разрешением и даже видеофрагменты. Некоторые люди придерживаются позиции «живи и дай жить другим», однако другие считают, что помещение в сети некоторых материалов (словесные угрозы в адрес отдельных стран или религий, порнографиии т. д.) просто недопустимо и такой контент должен подвергаться цензуре. Законодательства разных стран имеют разные взгляды на эту проблему; таким образом, страсти накаляются.

Раньше люди подавали в суд на сетевых операторов, считая их ответственными за содержимое сайтов, подобно тому, как газеты и журналы несут ответственность за содержимое своих страниц. В ответ же операторы сетей утверждают, что сеть подобна телефонной компании или почтовому отделению, и они не могут отвечать за то, что говорят их клиенты, а тем более управлять содержанием этих разговоров.

Теперь немного удивительно узнавать, что некоторые сетевые операторы блокируют контент по своим собственным причинам. Некоторых пользователей приложений соединения одноранговых узлов отключали от сети, потому что сетевые операторы не считали выгодным передачу большого количества трафика, посылаемого этими приложениями. Те же самые операторы, вероятно, хотели бы по-разному обслуживать разные компании. Если вы — крупная компания и платите хорошо, тогда вы получаете хорошую услугу, но если вы — мелкий игрок, вы получаете услугу худшего качества. Противники этой практики утверждают, что соединение одноранговых узлов и другой контент должны быть обработаны одинаково, потому что все они — биты в сети. Такая позиция, выступающая за коммуникации, которые не дифференцированы их контентом, источником или тем, кто создал контент, известна как сетевой нейтралитет (Wu, 2003). Вероятно, эти дебаты продолжатся еще некоторое время.

В споре относительно контента есть и другие стороны. Например, пиратская музыка и фильмы питали массивный рост одноранговых сетей, что не нравилось правообладателям, которые угрожали судебными исками (и иногда так и делали). Есть теперь автоматизированные системы, которые ищут одноранговые сети и посылают операторам предупреждения о пользователях, которые подозреваются в том, что посягают на авторское право. В Соединенных Штатах эти предупреждения известны как уведомления DMCA, появившиеся после принятия Закона о защите авторских прав в цифровую эпоху (Digital Millennium Copyright Act). Такой поиск — гонка вооружений, потому что достоверно поймать нарушение авторского права трудно. Даже ваш принтер мог бы быть принят за преступника (Piatek и др., 2008).

Компьютерные сети делают общение очень легким. Они также упрощают возможности отслеживать трафик. Так областью конфликтов оказались права наемных работников, вступившие в противоречие с правами нанимателей. Некоторые наниматели считают себя вправе читать и, возможно, подвергать цензуре сообщения своих работников, включая сообщения, посланные с домашних терминалов после работы. Не все с этим согласны.

Другой конфликт разворачивается вокруг проблемы взаимоотношений государства и граждан. Известно, что в поисках крупиц информации ФБР установило на серверах многих поставщиков услуг специальные системы, позволяющие просматривать входящую и исходящую почту. Система изначально называлась **Carnivore** (хищник. — *Примеч. перев.*), однако такое зловещее название обращало на себя слишком много внимания общественности. Было решено переименовать систему и назвать ее невинным именем — DCS1000 (Blaze и Bellovin, 2000; Sobel, 2001; Zacks, 2001). Цель таких систем состоит в том, чтобы шпионить за миллионами людей в надежде на обнаружение информации о незаконной деятельности. К сожалению для шпионов, Четвертая поправка к американской Конституции запрещает правительственные поиски без ордера на обыск, но правительство часто игнорирует это.

Конечно, не только правительство угрожает частной жизни. Частный сектор вносит свою лепту, также профилируя пользователей. Например, маленькие cookie-файлы, содержащие информацию о том, что пользователь делал в Сети, и позволяющие нечистым на руку компаниям узнавать конфиденциальную информацию и передавать через Интернет номера кредитных карт и другие важные идентификаторы (Berghel, 2001). Компании, которые оказывают услуги в сети, могут хранить большое количе-

ство персональных данных об их пользователях, что позволяют им изучать пользовательские действия. Например, если вы используете электронную почту Gmail, то Google может читать вашу электронную почту и показывать вам рекламные объявления, основанные на ваших интересах.

С распространением мобильных устройств возник вопрос конфиденциальности местоположения (Beresford и Stajano, 2003). В процессе оказывания услуги вашему мобильному устройству сетевые операторы узнают, где вы находитесь в течение дня. Это позволяет им отслеживать ваши передвижения. Они могут знать, какой ночной клуб или медицинский центр вы посещаете.

Компьютерные сети также предоставляют возможность и увеличить защиту частной жизни, например, путем отправки анонимных сообщений. В некоторых ситуациях такая необходимость есть. Помимо защиты от изучения компаниями ваших личных привычек, студенты, солдаты, служащие и граждане могут, таким образом, пожаловаться на незаконные действия профессоров, офицеров, начальства и политиков, не опасаясь репрессий. С другой стороны, в США и многих других демократических странах законом особо предусмотрено право обвиняемой стороны на очную ставку со своим обвинителем в суде, а также на встречный иск. Поэтому анонимные обвинения не могут рассматриваться в качестве свидетельств в суде.

Интернет позволяет с огромной скоростью находить нужную информацию, однако кто проверит ее качество и достоверность? Совет врача, которого вы ждете по поводу боли в груди, может на самом деле исходить как от лауреата Нобелевской премии в области медицины, так и от разгильдяя, которого выгнали из школы и которому нечем заняться.

Другая информация часто нежелательна. Электронная макулатура, «спам», стала, увы, частью нашей жизни, потому что есть способы собрать миллионы адресов e-mail и дешево рассылать по ним все, что угодно. Получающаяся волна спама конкурирует с потоком сообщений от настоящих людей. К счастью, фильтрующее программное обеспечение с меньшим или большим успехом в состоянии вычислить и отсечь спам, произведенный другими компьютерами.

Другие виды контента созданы для совершения преступлений. Веб-страницы и электронные письма, содержащие активный контент (в основном программы или макросы, которые выполняются на машине получателя), могут содержать вирусы, которые могут захватить ваш компьютер. Они могут использоваться, чтобы украсть ваши пароли банковского счета или вовлечь ваш компьютер в рассылку спама как часть **ботента** или группы зараженных машин.

Фишинговые сообщения (phishing messages) притворяются происходящими из заслуживающего доверия источника, например вашего банка, и пытаются узнать от вас частую информацию, например номер кредитной карточки. Хищение личных данных становится серьезной проблемой, поскольку воры собирают достаточно информации о жертве, чтобы получить кредитные карты и другие документы на имя жертвы.

Бывает трудно препятствовать тому, чтобы компьютеры в Интернете исполнили роль людей. Эта проблема привела к развитию CAPTCHA-теста для распознавания людей и машин, в котором компьютер просит решить короткую задачу распознавания, например, вводя буквы, показанные в искаженном изображении, чтобы показать, что клиент является человеком (von Ahn, 2001). Этот процесс — вариация на тему зна-

менитого теста Тьюринга, в котором человек задает вопросы по сети, чтобы судить, является ли отвечающий человеком.

Многие из этих проблем могут быть решены, если компьютерная индустрия всерьез займется вопросами защиты информации. Если бы все сообщения передавались в зашифрованном виде, это позволило бы избежать огромных убытков, понесенных как частными лицами, так и крупными компаниями. Более того, системы кодирования уже давно разработаны, и мы подробно изучим их в главе 8. Проблема в том, что производители аппаратного и программного обеспечения прекрасно знают, каких денег стоит внедрение защитных систем, и понимают, что попытки продать такую дорогостоящую продукцию обречены. Немало хлопот доставляют и «баги» (ошибки в программах), «дыры» в защите и т. д. Они возникают потому, что производители добавляют все новые и новые функции, а это приводит к увеличению числа неполадок. Возможным выходом из такой ситуации является взимание платы за расширенные версии программ, однако поди заставь конторы, занимающиеся разработкой ПО, отказаться от такого хорошего рекламного хода, как бесплатные обновления. Можно, конечно, обязать фирмы возмещать убытки, нанесенные выпущенными ими дефектными программами, однако это приведет к банкротству практически всей программной индустрии в первый же год.

Компьютерные сети поднимают новые правовые проблемы, когда они взаимодействуют со старыми законами. Например, электронные азартные игры. Компьютеры в течение многих десятилетий моделировали различные процессы, так почему бы не моделировать игровые автоматы, рулетку, дилеров блэк джека и другое игорное оборудование? Ну, потому что это во многих местах незаконно. Но в большом количестве других мест (Англия, например) азартная игра является законной, и владельцы казино там оценили потенциал для азартной игры через Интернет. Что происходит, если игрок, казино и сервер — все в разных странах, с противоречивыми законами? Хороший вопрос.

1.2. Сетевое оборудование

Теперь пора от вопросов применения сетей и социальных аспектов (десерта) перейти к рассмотрению технической стороны разработки сетей (шпинату). Единой общепринятой системы, которой удовлетворяют все сети, не существует, однако есть два важнейших параметра: технология передачи и размеры. Рассмотрим оба параметра по очереди.

Если смотреть в общих чертах, существует два типа технологии передачи:

- Широковещательные сети.
- Сети с передачей от узла к узлу.

Сети с передачей от узла к узлу состоят из соединенных пар машин. Чтобы пойти от источника до места назначения в сети, составленной из двухточечных линий, коротким сообщениям, называемым в определенных контекстах **пакетами**, вероятно, придется сначала посетить одну или несколько промежуточных машин. Часто возможны несколько маршрутов различных длин, поэтому в двухточечных сетях важно найти лучшие из них. Двухточечную передачу с ровно одним отправителем и ровно одним получателем иногда называют **однонаправленной передачей (unicasting)**.

Широковещательные сети существенно отличаются тем, что обладают единым каналом связи, совместно используемым всеми машинами сети. Пакеты посылаются одной машиной, а получаются всеми машинами. Поле адреса в каждом пакете указывает, кому направляется сообщение. При получении пакета машина проверяет его адресное поле. Если пакет адресован этой машине, она его обрабатывает. Пакеты, адресованные другим машинам, игнорируются.

Беспроводная сеть — типичный пример широковещательного канала, с коммуникацией, совместно использованной в зоне обслуживания, которая зависит от беспроводного канала и передающей машины. В качестве иллюстрации представьте себе человека, стоящего в комнате и кричащего: «Ватсон, идите сюда. Вы мне нужны». И хотя это сообщение может быть получено (услышано) многими людьми, ответит только Ватсон. Остальные просто не обратят на него внимания.

Широковещательные сети также позволяют адресовать пакет одновременно всем машинам с помощью специального кода в поле адреса. Когда передается пакет с таким кодом, он получается и обрабатывается всеми машинами сети. Такая операция называется **широковещательной передачей**. Некоторые широковещательные системы также предоставляют возможность посылать сообщения подмножеству машин, и это называется **многоадресной передачей**.

Другим признаком классификации сетей является их размер. Размеры сетей являются весьма важным классификационным фактором, поскольку в сетях различного размера применяются различные технологии.

На рис. 1.4 приведена классификация мультипроцессорных систем в зависимости от их размеров. В верхней строке таблицы помещаются **персональные сети**, то есть сети, предназначенные для одного человека. Далее в таблице следуют более протяженные сети. Их можно разделить на локальные, муниципальные и глобальные сети. И замыкают таблицу объединения двух и более сетей.

Расстояние между процессорами	Процессоры расположены	Пример
1 м	На одном квадратном метре	Персональная сеть
10 м	Комната	Локальная сеть
100 м	Здание	
1 км	Кампус	
10 км	Город	Муниципальная сеть
100 км	Страна	Глобальная сеть
1000 км	Континент	
10 000 км	Планета	Интернет

Рис. 1.4. Классификация многопроцессорных систем по размеру

Хорошо известным (но не единственным) примером такого объединения является Интернет.

В данной книге мы рассмотрим сети всех размеров, а также их объединения. Ниже мы дадим краткое описание сетевого оборудования.

1.2.1. Персональные сети

Персональные сети (PAN) позволяют общаться устройствам в близи человека. Типичный пример — беспроводная сеть, которая соединяет компьютер с его периферийными устройствами. Почти у каждого компьютера есть присоединенный монитор, клавиатура, мышь и принтер. При отсутствии беспроводной сети они должны быть присоединены кабелями. Очень многие новые пользователи испытывают трудности с поиском нужных кабелей и включением их в нужные отверстия (даже при том, что они обычно обозначены соответствующим цветом), поэтому большинство продавцов компьютеров предлагают опцию визита специалиста. Чтобы помочь этим пользователям, несколько компаний собрались для разработки беспроводной сети малой дальности, названной Bluetooth, чтобы соединять компоненты без проводов. Идея состоит в том, что если у ваших устройств есть Bluetooth, то вы не нуждаетесь ни в каких кабелях. Вы только ставите их, включаете, и они взаимодействуют. Для многих людей эта непринужденность работы — большой плюс.

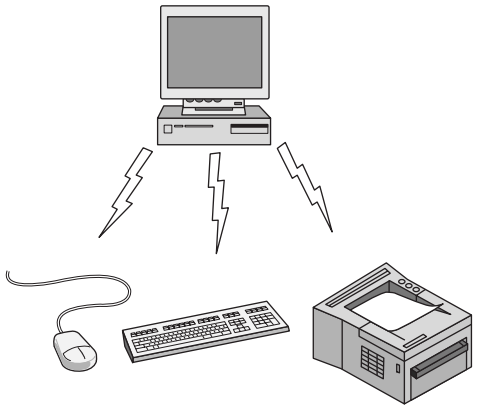


Рис. 1.5. Конфигурация персональной сети Bluetooth

В самой простой форме сети Bluetooth используют парадигму ведущий—ведомые (Master—Slave), см. рис. 1.5. Системный модуль (PC) обычно является ведущим устройством и общается с мышью, клавиатурой и т. д., как с ведомыми устройствами. Ведущее устройство говорит ведомым устройствам, какие адреса использовать, когда они могут осуществлять широковещательную передачу, сколько времени они могут передавать, какие частоты могут использовать и т. д.

Bluetooth может использоваться также и в других устройствах. Он часто используется, чтобы соединить наушники с мобильным телефоном без шнуров, и может позволить вашему цифровому музыкальному проигрывателю соединяться с вашим

автомобилем, как только он находится в пределах диапазона. Абсолютно другой вид PAN возникает, когда встроенное медицинское устройство, такое как кардиостимулятор, насос инсулина или слуховой аппарат, говорит с управляемым пользователем дистанционным управлением. Мы обсудим Bluetooth более подробно в главе 4.

PAN могут также быть созданы с другими технологиями, которые общаются на малые расстояния, такие как RFID на смарт-картах и библиотечных книгах. Мы изучим RFID в главе 4.

1.2.2. Локальные сети

Локальными сетями называют частные сети, размещающиеся, как правило, в одном здании или на территории какой-либо организации. Их часто используют для объединения компьютеров и рабочих станций в офисах компании или предприятия бытовой электроники для предоставления совместного доступа к ресурсам (например, принтерам) и обмена информацией. Когда локальные сети используются предприятиями, их называют **сеть предприятия (enterprise networks)**.

Беспроводные ЛВС сейчас очень популярны, особенно в домах, более старых офисных зданиях, кафетериях и других местах, где слишком сложно провести кабели. В этих системах у каждого компьютера есть радиомодем и антенна, которую он использует, чтобы общаться с другими компьютерами. В большинстве случаев каждый компьютер говорит с устройством в потолке, как показано на рис. 1.6, *а*. Это устройство, названное AP (Точка доступа), беспроводный маршрутизатор, или базовая станция, передает пакеты между беспроводными компьютерами, а также между ними и Интернетом. Точка доступа похожа на популярного ребенка в школе, потому что все хотят говорить с ним. Однако, если другие компьютеры достаточно близки, они могут общаться непосредственно друг с другом в конфигурации соединения равноправных узлов ЛВС.

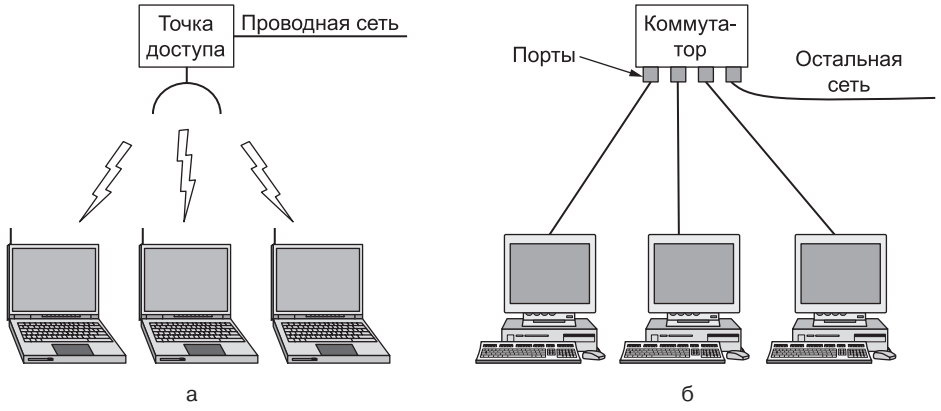


Рис. 1.6. Беспроводные и проводные сети: *а* — 802.11; *б* — коммутируемый Ethernet

Стандарт для беспроводных ЛВС, названный IEEE 802.11, более известный как WiFi, стал очень широко распространенным. Он работает на скоростях от 11 до сот-

ни мегабит в секунду. (В этой книге мы будем придерживаться традиции и измерять скорости линии в мегабитах в секунду, где 1 Мбит/с составляет 1 000 000 битов в секунду, и гигабитах в секунду, где 1 Гбит/с составляет 1 000 000 000 битов в секунду.) Мы обсудим 802.11 в главе 4.

В проводных ЛВС используются различные технологии передачи. Большинство из них использует медные провода, а некоторые — оптоволокно. ЛВС ограничены в размере, это означает, что максимальное время передачи ограничено и известно заранее. Знание этих границ помогает с задачей разработки сетевых протоколов. Как правило, проводные ЛВС работают на скоростях от 100 Мбит/с до 1 Гбит/с, имеют низкую задержку (микросекунды или наносекунды) и делают очень немного ошибок. Более новые ЛВС могут работать со скоростью 10 Гбит/с. По сравнению с беспроводными сетями проводные ЛВС превышают их по всем параметрам работы. Послать сигналы по проводу или через волокно проще, чем по воздуху.

Топология многих проводных ЛВС создана из магистральных линий. Стандарт **IEEE 802.3**, обычно называемый **Ethernet**, является, безусловно, наиболее распространенным типом проводной ЛВС. Рисунок 1.6, *б* показывает типовую топологию коммутируемого Ethernet. Каждый компьютер говорит на протоколе Ethernet и соединяется с устройством, названным коммутатором с магистральной линией. Отсюда и название. У коммутатора есть несколько портов, каждый из которых может соединиться с одним компьютером. Работа коммутатора — передать пакеты между компьютерами, которые к нему присоединены; для определения нужного компьютера используется адрес в каждом пакете.

Чтобы создать большие ЛВС, коммутаторы могут быть подключены друг в друга с помощью портов. Что происходит, если вы включаете их вместе в цикл? Будет ли сеть работать? К счастью, проектировщики подумали об этом варианте. Протокол определяет пути для пакетов так, чтобы они достигли нужного компьютера. Мы увидим как это работает в главе 4.

Также возможно разделить одну большую физическую ЛВС на две меньших логических ЛВС. Вы могли бы задаться вопросом, чем это будет полезно. Иногда расположение сетевого оборудования не соответствует структуре организации. Например, у инженерного и финансового отделов компании могли бы быть компьютеры в одной физической ЛВС, потому что они находятся в одном крыле здания, но было бы легче управлять системой, если бы у каждого отдела была своя виртуальная ЛВС (VLAN). В этой конструкции каждый порт отмечен «цветом», скажем, зеленый для инженерного отдела и красный для финансового. Коммутатор направляет пакеты так, чтобы компьютеры, присоединенные к зеленым портам, были отделены от компьютеров, присоединенных к красным портам. Широковещательные пакеты, посланные красным портом, например, не будут получены зеленым портом, как если бы это были две различных ЛВС. Мы обсудим виртуальные ЛВС в конце главы 4.

Существуют и другие топологии проводной ЛВС. Фактически, коммутируемый Ethernet — современная версия оригинального проекта Ethernet, который передавал все пакеты по единственному линейному кабелю. В один момент времени передачу могла вести только одна машина, а конфликты решал распределенный арбитражный механизм. Использовался простой алгоритм: компьютеры могли передать всякий раз, когда кабель был неактивен. Если два или несколько пакетов столкнулись, каждый

компьютер ждал в течение случайного промежутка времени и делал еще одну попытку. Мы будем называть этот вариант классический Ethernet, и вы узнаете о нем в главе 4.

И проводные, и беспроводные широковещательные сети в зависимости от способа назначения канала подразделяются на статические и динамические. При статическом назначении используется циклический алгоритм, и все время делится между всеми машинами на равные интервалы, так что машина может передавать данные только в течение выделенного ей интервала времени. При этом емкость канала расходуется неэкономно, так как временной интервал предоставляется машинам независимо от того, есть им что сказать или нет. Поэтому чаще используется динамическое (то есть по требованию) предоставление доступа к каналу.

Методы динамического предоставления доступа к каналу также могут быть централизованными либо децентрализованными. При централизованном методе предоставления доступа к каналу должно существовать одно устройство, такое как базовая станция в сотовой сети, определяющее машину, получающую право на передачу. Оно должно получать множество пакетов и принимать решение о приоритетах на основании некоего внутреннего алгоритма. При децентрализованном методе каждая машина должна сама решать, передавать ей что-нибудь или нет. Можно подумать, что подобный метод обязательно приводит к беспорядку, однако это не так. Ниже мы рассмотрим различные алгоритмы, специально созданные для внесения порядка в возможный хаос.

Стоит дополнительно обсудить домашние ЛВС. В будущем, вероятно, каждое устройство в доме будет способно к общению с любым другим устройством, и все они будут доступны по Интернету. Это развитие, вероятно, будет одним из тех новшеств, которых никто не ожидает (как телевизионные дистанционные пульты управления или мобильные телефоны), но когда они появились никто уже не может представить жизнь без них.

Многие устройства уже способны стать сетевыми. Это компьютеры, устройства развлечения, такие как телевизоры и DVD, телефоны и другая бытовая электроника, такая как камеры, радиочасы и оборудование, такое как счетчики и термостаты. Эта тенденция только продолжится. Например, в обычном доме, вероятно, есть дюжина часов (например, в приборах), если бы они были подключены к Интернету, они все могли бы переходить на летнее время автоматически. Дистанционный мониторинг дома — вероятный лидер в этом направлении, так многие выросшие дети были бы готовы потратить некоторые деньги, чтобы помочь своим стареющим родителям жить безопасно в их собственных домах.

Хотя мы могли бы думать о домашней сети просто как о еще одной ЛВС, она, вероятно, будет отличаться от других сетей. Во-первых, сетевые устройства должно быть очень легко установить. Беспроводные маршрутизаторы — наиболее возвращаемые покупателями электронные устройства. Люди покупают их, потому что они хотят иметь дома беспроводную сеть, обнаруживают, что он не работает «из коробки», и затем возвращает его, вместо того чтобы слушать музыку в ожидания ответа на линии технической помощи.

Во-вторых, сеть и устройства должны быть надежными в работе. Кондиционеры обычно имеют один переключатель — кнопку с четырьмя вариантами: выкл, низкий, средний и высокий. К ним прилагаются руководства из 30 страниц. Когда они объ-

единены в сеть, только одна глава по безопасности составит 30 страниц. Это — проблема, потому что только пользователи компьютеров приучены к установке продуктов, которые не работают; покупатели автомобилей и телевизоров намного менее терпеливы. Они ожидают, что продукты будут работать 100 % без потребности привлекать компьютерного специалиста.

В-третьих, для успеха важна низкая цена. Люди не будут платить дополнительные $50 за интернет-термостат, потому что мало кто считает важным возможность контролировать температуру дома, находясь на работе. А за дополнительные $5 они могли бы это купить.

В-четвертых, нужна возможность начать с одного или двух устройств и постепенно расширять сеть. Это означает — отсутствие войн форматов. Если предлагать потребителям купить периферийные устройства с интерфейсом IEEE 1394 (FireWire), а несколько лет спустя отрекаться от этого и предлагать USB 2.0 как «интерфейс месяца», а затем переключиться на 802.11g, или нет, 802.11n, а лучше 802.16 (все это различные беспроводные сети), пользователи станут весьма сдержанны. Сетевой интерфейс должен оставаться постоянным в течение многих десятилетий, как стандарты телевещания.

В-пятых, будут очень важны безопасность и надежность. Потеря нескольких файлов из-за вируса в электронной почте — это одно; преступник, который разоружает вашу систему обеспечения безопасности со своего мобильного компьютера и затем грабит ваш дом — совсем другое. Интересный вопрос состоит в том, будут ли домашние сети проводными или беспроводными. Удобство и стоимость говорят в пользу беспроводной сети, потому что не надо заботиться о соответствии или, еще хуже, модификации проводов. Проводные сети безопаснее, потому что радиоволны, которые используют беспроводные сети, довольно хорошо проходят сквозь стены. Вы не будете рады соседям, которые пользуются вашим интернет-соединением и читают вашу электронную почту. В главе 8 мы изучим, как может использоваться шифрование, чтобы обеспечить безопасность, но для неопытных пользователей это легче сказать, чем сделать.

Третий вариант сети — использовать то, что уже есть у вас дома. Очевидный кандидат — электрические провода. Сети линии электропередачи позволяют устройствам, которые в них включены, посылать информацию всюду по дому. Вы должны в любом случае включить телевизор и тем самым можете сразу получить связь с Интернетом. Трудность состоит в том, как одновременно перенести и электроэнергию, и сигналы данных. Часть ответа — они используют различные диапазоны частот.

Короче говоря, домашние ЛВС предлагают много возможностей и проблем. Большинство проблем относится к необходимости в простоте, надежности и безопасности, особенно в руках не разбирающихся в технике пользователей, а так же в низкой цене.

1.2.3. Муниципальные сети

Муниципальные сети (metropolitan area network, MAN) объединяют компьютеры в пределах города. Самым распространенным примером муниципальной сети является система кабельного телевидения. Она стала правопреемником обычных антенных

телесетей в тех местах, где по тем или иным причинам качество эфира было слишком низким. Общая антенна в этих системах устанавливалась на вершине какого-нибудь холма, и сигнал передавался в дома абонентов.

Вначале стали появляться специализированные, разработанные прямо на объектах сетевые структуры. Затем компании-разработчики занялись продвижением своих систем на рынке, начали заключать договоры с местным правительством и в итоге охватили целые города. Следующим шагом стало создание телевизионных программ и даже целых каналов, предназначенных только для кабельного телевидения. Зачастую они представляли какую-то область интересов. Можно было подписаться на новостной канал, спортивный, посвященный кулинарии, саду-огороду и т. д. До конца девяностых годов эти системы были предназначены исключительно для телевизионного приема.

С тех пор как Интернет привлек к себе массовую аудиторию, операторы кабельного телевидения поняли, что, внеся небольшие изменения в систему, можно сделать так, чтобы по тем же каналам, в неиспользуемой части спектра, передавались (причем в обе стороны) цифровые данные. С этого момента кабельное телевидение стало постепенно превращаться в муниципальную компьютерную сеть. В первом приближении систему MAN можно представить себе такой, как она изображена на рис. 1.7. На этом рисунке видно, что по одним и тем же линиям передается и телевизионный, и цифровой сигнал. Во **входном устройстве** они смешиваются и передаются абонентам. Мы еще вернемся к этому вопросу позднее в главе 2.

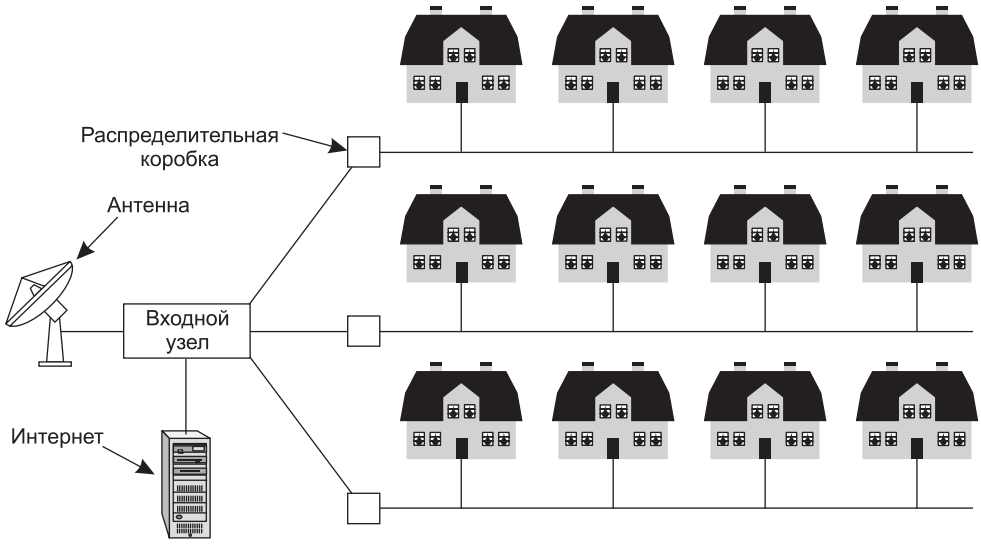


Рис. 1.7. Муниципальная сеть на базе кабельного ТВ

Впрочем, муниципальные сети — это не только кабельное телевидение. Недавние разработки, связанные с высокоскоростным беспроводным доступом в Интернет, привели к созданию других MAN, которые описаны в стандарте IEEE 802.16, известном как WiMax. Мы рассмотрим их в главе 4.

1.2.4. Глобальные сети

Глобальная сеть (wide area network, WAN) охватывает значительную географическую область, часто целую страну или даже континент. Мы начнем разговор о них с проводных глобальных сетей, используя в качестве примера компанию, имеющую подразделения в разных городах.

Сеть, показанная на рис. 1.8, соединяет офисы, находящиеся в Перте, Мельбурне и Брисбене. Каждый из них содержит компьютеры, предназначенные для выполнения программ пользователя (то есть приложений). Мы будем следовать традиционной терминологии и называть эти машины хостами. Хосты соединяются коммуникационными подсетями, называемыми для краткости просто подсетями. Задачей подсети является передача сообщений от хоста хосту, подобно тому, как телефонная система переносит слова (то есть просто звуки) от говорящего слушающему.

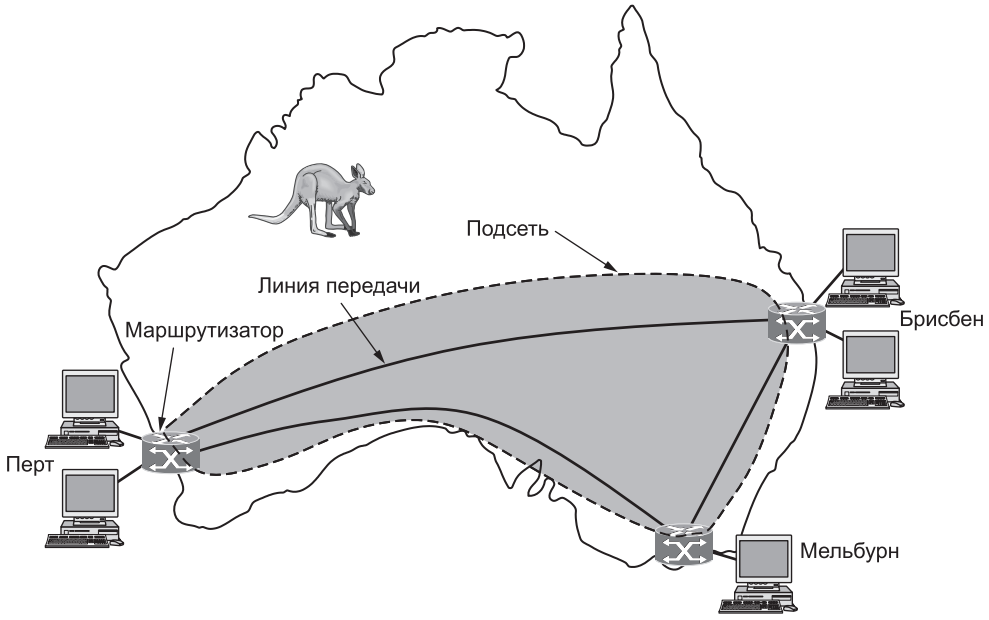


Рис. 1.8. WAN, соединяющая три офиса на территории Австралии

В большинстве глобальных сетей подсеть состоит из двух раздельных компонентов: линий связи и переключающих элементов. Линии связи переносят данные от машины к машине. Они могут представлять собой медные провода, оптоволокно или даже радиосвязь. Большинство компаний не имеют собственных линий связи, поэтому они арендуют их у телекоммуникационной компании. Переключающие элементы являются специализированными компьютерами, используемыми для соединения двух или более линий связи. Когда данные появляются на входной линии, переключающий элемент должен выбрать выходную линию для дальнейшего маршрута этих данных. В прошлом для названия этих компьютеров не было стандартной терминологии. Сейчас их называют **маршрутизаторами (router)**, однако читателю следует знать, что по поводу терминологии в данном случае единого мнения не существует. К сожалению,

многие остряки-самоучки любят рифмовать «router» с «doubter», что в переводе означает «скептик», а некоторые вместо «router» пишут «rooter» («корчеватель»). Определение правильного произношения оставим как упражнение читателю. (Это может зависеть от того, где вы живете.)

Следует также сделать замечание по поводу термина «подсеть» (subnet). Изначально его *единственным* значением являлся набор маршрутизаторов и линий связи, используемый для передачи пакета от одного хоста к другому. Однако читателям следует обратить внимание, что этот термин приобрел второй смысл, связанный с адресацией в сети (что будет обсуждаться в главе 5). Однако до тех пор мы будем использовать исходное значение (набор линий и маршрутизаторов).

Глобальные сети, как мы их описали, выглядят похоже на большую проводную ЛВС, но есть некоторые важные различия, которые не только в длинных проводах. Обычно в глобальной сети узлы и подсеть принадлежат и управляются различными людьми. В нашем примере сотрудники могли бы быть ответственными за свои собственные компьютеры, в то время как IT-отдел компании отвечает за остальную часть сети. Мы увидим более ясные границы в ближайших примерах, в которых подсетью управляют сетевой провайдер или телефонная компания. Разделение чистых коммуникационных аспектов сети (подсеть) от аспектов приложения (узлы) очень упрощает полное проектирование сети.

Второе различие — то, что маршрутизаторы будут обычно соединять различные виды сетевых технологий. Сети в офисах могут быть, например, коммутируемыми Ethernet, в то время как дальние линии передачи могут быть линиями SONET (которые мы обсудим в главе 2). Какое-то устройство должно присоединиться к ним. Проницательный читатель заметит, что это идет вне нашего определения сети. Это означает, что многие глобальные сети фактически будут объединенными сетями, или сложными сетями, которые составлены больше чем из одной сети. В следующем разделе мы скажем больше об объединенных сетях.

Заключительное различие состоит в том, что связано с подсетью. Это могут быть отдельные компьютеры, которые соединяются с ЛВС, или целые ЛВС. Вот каким образом большие сети создаются из небольших. В случае подсетей все происходит так же.

Теперь мы можем посмотреть на два других варианта глобальных сетей. Во-первых, вместо того чтобы арендовать линии передачи, компания могла бы соединить свои офисы с Интернетом. Это позволяет соединениям между офисами быть виртуальными и использовать основные возможности Интернета. Это расположение, показанное в рис. 1.9, называют **VPN (Virtual Private Network**, **Виртуальная частная сеть**). По сравнению со специализированным расположением у VPN есть преимущество виртуализации — гибкое повторное использование ресурса (интернет-связи). Посмотрите, как легко добавить четвертый офис. У VPN есть и обычный недостаток виртуализации — недостаточное управление основными ресурсами. Пропускная способность выделенной линии ясна. С VPN ваши затраты на единицу расстояния могут меняться в зависимости от интернет-сервиса.

Второе изменение состоит в том, что подсеть может обслуживаться другой фирмой. Оператор подсети — провайдер сетевой службы, а офисы — его клиенты. Эта структура показана на рис. 1.10. Оператор подсети соединяется с клиентами и предоставляет им услугу, пока они за это платят. Так как было бы неудобно, если бы и клиенты могли посылать пакеты только друг другу, оператор подсети соединяется и с другими сетями,

которые являются частью Интернета. Такого оператора подсети называют провайдером (ISP (Internet Service Provider)), а такую подсеть — сетью провайдера. Клиенты, которые соединяются с провайдером, получают интернет-сервис.

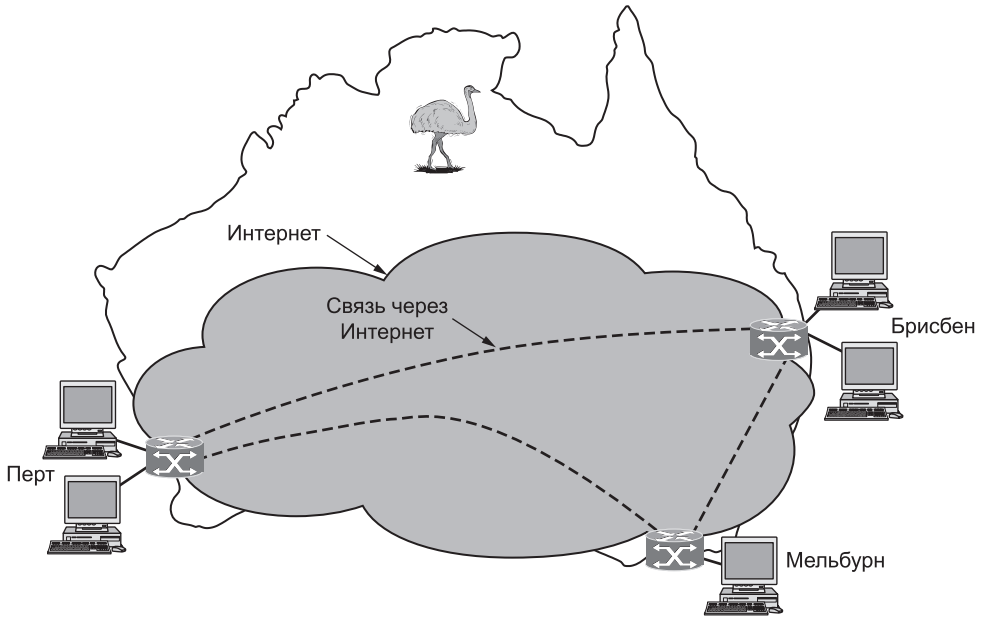


Рис. 1.9. WAN, использующая виртуальную частную сеть

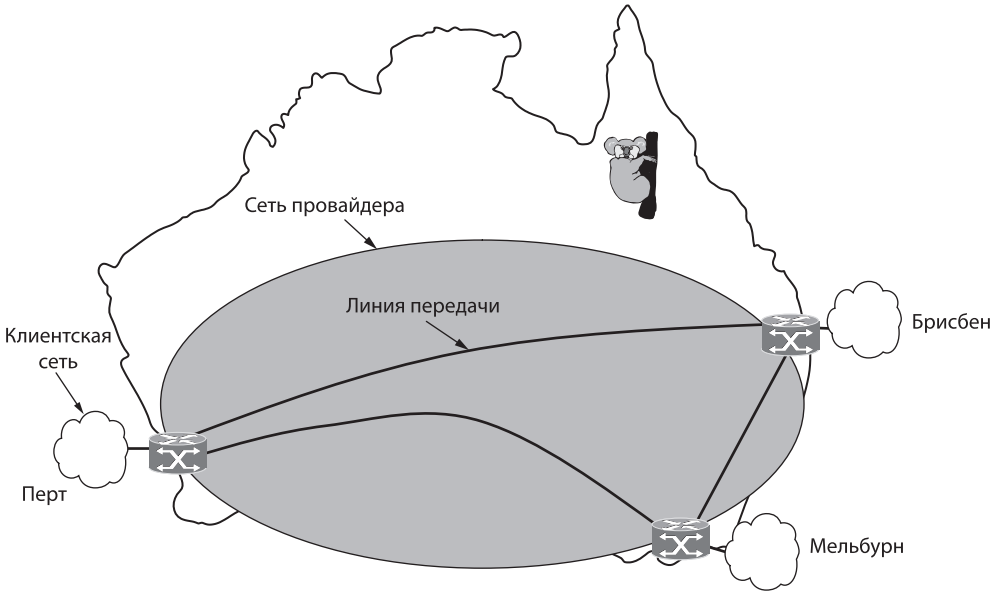


Рис. 1.10. WAN, использующая сеть провайдера

Мы можем использовать сеть провайдера, чтобы анонсировать некоторые ключевые вопросы, которые мы изучим в более поздних главах. Большинство глобальных сетей содержит большое количество кабелей или телефонных линий, соединяющих пару маршрутизаторов. Если какие-либо два маршрутизатора не связаны линией связи напрямую, то они должны общаться при помощи других маршрутизаторов. В сети может быть много путей, которые соединяют эти два маршрутизатора. Метод принятия решения называется **алгоритмом маршрутизации**. Существует много таких алгоритмов. То, как каждый маршрутизатор принимает решение, куда послать пакет, называется **алгоритмом пересылки**. Их существует огромное множество, и некоторые алгоритмы обоих типов мы изучим в главе 5.

Другие глобальные сети используют беспроводные технологии. В спутниковых системах каждый компьютер на земле снабжается антенной, при помощи которой он может принимать и посылать сигнал спутнику на орбите. Все компьютеры могут принимать сигналы со спутника, а в некоторых случаях они могут также слышать передачи соседних компьютеров, передающих данные на спутник. Спутниковые сети являются широковещательными и наиболее полезны там, где требуется широковещание.

Сеть мобильной телефонной связи — другой пример глобальной сети, использующей беспроводную технологию. Эта система уже прошла три поколения, а четвертое на горизонте. Первое поколение осуществляло только аналоговую передачу звука. Второе поколение было цифровым, но тоже только для передачи голоса. Третье поколение является цифровым и предназначено для передачи и речи, и данных. Каждая сотовая базовая станция покрывает намного большую площадь, чем беспроводная ЛВС, расстояния измеряются в километрах, а не в десятках метров. Базовые станции соединены друг с другом базовой сетью, которая обычно проводная. Скорости передачи данных в сотовых сетях достигают порядка 1 Мбит/с, что намного меньше, чем у беспроводной ЛВС, где скорость может достигать порядка 100 Мбит/с. Мы многое расскажем об этих сетях в главе 2.

1.2.5. Объединения сетей

Существующие ныне сети часто используют различное оборудование и программное обеспечение. Люди, связанные с одной сетью, хотят общаться с людьми, подключенными к другой. Для выполнения этого желания необходимо объединить вместе различные и часто несовместимые сети. Набор соединенных сетей называется **интерсетью** (internetwork, internet). Слово «интерсеть» (internet, написанный со строчной буквы) всегда будет использоваться в этой книге в его исконном смысле, в отличие от слова «Интернет» (с прописной буквы) — всемирной сети, являющейся одной из интерсетей. Интернет использует сети провайдеров, чтобы соединить сети предприятий, домашние сети и многие другие. Позже в этой книге мы рассмотрим Интернет очень подробно.

Часто путают подсети, сети и интерсети. Термин «подсети» обычно употребляется в контексте глобальных сетей, где он означает набор маршрутизаторов и линий связи, принадлежащих одному сетевому оператору. Аналогично этому, телефонная система состоит из телефонных станций, соединенных друг с другом высокоскоростными каналами, а с домами и офисами — низкоскоростными каналами. Эти каналы и оборудование принадлежат телефонным компаниям, являющимся аналогами подсетей.

Сами телефонные аппараты (аналоги хостов) не являются частью подсетей. Вместе с хостами подсеть образует сеть. В случае локальной сети сеть состоит из кабеля и хостов. Подсетей там нет.

Сеть формируется комбинацией подсети и ее узлов. Однако слово «сеть» часто используется и в свободном смысле. Подсеть могла бы быть описана как сеть, как в случае «сети провайдера» на рис. 1.10. Интерсеть могла бы также быть описана как сеть, как в случае глобальной сети на рис. 1.8. Мы будем поступать также, а если когда отличаем сеть от других расположений, то будем придерживаться нашего оригинального определения набора компьютеров, связанных одной технологией.

Немного больше о том, что составляет объединенную сеть (internetwork). Мы знаем, что интерсеть образуется путем объединения нескольких сетей. С нашей точки зрения, объединение локальной и глобальной сети или объединение двух локальных сетей — обычный способ образования интерсети, однако в индустрии нет единого мнения по поводу терминологии в данной области. Существует два мнемонических правила. Первое — если создание и поддержку сети оплачивают разные организации, то мы имеем дело с интерсетью, а не единой сетью. Второе — если работа основана на применении нескольких технологий (например, широковещательная в одной ее части и двухузловая в другой), то, вероятно, это интерсеть.

Чтобы пойти глубже, мы должны поговорить о том, как могут быть соединены две различных сети. Общее название машины, которая обеспечивает соединение между двумя или более сетями и обеспечивает необходимый перевод, с точки зрения как аппаратного, так и программного обеспечения, это шлюз. Шлюзы различаются по уровням, в которых они работают в иерархии протокола. Начиная со следующего раздела, мы расскажем об уровнях и иерархиях протокола намного больше, но пока предположим, что более высокие уровни более привязаны к приложениям, таким как Web, а нижние уровни более привязаны к каналам передачи, таким как Ethernet.

Так как польза от формирования Интернета — в соединении компьютеров через сети, мы не хотим использовать слишком низкоуровневый шлюз, иначе будем неспособны делать соединения между различными видами сетей. Мы не хотим использовать и слишком высокоуровневый шлюз, иначе соединение будет работать лишь с некоторыми приложениями. Уровень в середине, который нам «в самый раз», часто называют сетевым уровнем, и маршрутизатор является шлюзом, который обрабатывает пакеты на сетевом уровне. Теперь мы можем определить интерсеть, как сеть, у которой есть маршрутизаторы.

1.3. Сетевое программное обеспечение

Когда собирались первые сети, то основное внимание уделялось аппаратуре, а вопросы программного обеспечения откладывались на будущее. Подобная стратегия больше не работает. Сегодняшнее сетевое программное обеспечение в высокой степени структурировано. В следующих разделах мы узнаем, как осуществляется эта структуризация. Описанный подход является краеугольным камнем всей книги и будет часто встречаться и далее.

1.3.1. Иерархия протоколов

Для упрощения структуры большинство сетей организуются в наборы **уровней** или **слоев**, каждый последующий возводится над предыдущим. Количество уровней, их названия, содержание и назначение разнятся от сети к сети. Однако во всех сетях целью каждого уровня является предоставление неких сервисов для вышестоящих уровней. При этом от них скрываются детали реализации предоставляемого сервиса.

Такая концепция не нова и используется в вычислительной технике уже давно. Ее вариации известны как сокрытие информации, абстрактные типы данных, свойство инкапсуляции и объектно-ориентированное программирование. Фундаментальной идеей является предоставление неким программным или аппаратным уровнем сервисов своим пользователям без раскрытия деталей своего внутреннего состояния и подробностей алгоритмов.

Уровень *n* одной машины поддерживает связь с уровнем *n* другой машины. Правила и соглашения, используемые в данном общении, называются **протоколом** уровня *n*. По сути, протокол является договоренностью общающихся сторон о том, как должно происходить общение. По аналогии, когда женщину представляют мужчине, она может протянуть ему свою руку. Он, в свою очередь, может принять решение либо пожать, либо поцеловать эту руку, в зависимости от того, является ли эта женщина американским адвокатом на деловой встрече или же европейской принцессой на официальном балу. Нарушение протокола создаст затруднения в общении, а может и вовсе сделает общение невозможным.

На рис. 1.11 показана пятиуровневая сеть. Объекты, включающие в себя соответствующие уровни на разных машинах, называются равноранговыми или равноправными узлами сети. Именно они общаются при помощи протокола.

В действительности, данные не пересылаются с уровня *n* одной машины на уровень *n* другой машины. Вместо этого каждый уровень передает данные и управление уровню, лежащему ниже, пока не достигается самый нижний уровень. Ниже первого уровня располагается **физическая среда**, по которой и производится обмен информацией. На рис. 1.11 виртуальное общение показано пунктиром, тогда как физическое — сплошными линиями.

Между каждой парой смежных уровней находится **интерфейс**, определяющий набор примитивных операций, предоставляемых нижним уровнем верхнему. Когда разработчики сетей решают, сколько уровней включить в сеть и что должен делать каждый уровень, одной из важнейших задач является определение ясных интерфейсов между уровнями. Подобная задача требует, в свою очередь, чтобы каждый уровень выполнял особый набор хорошо понятных функций. В дополнение к минимизации количества информации, передаваемой между уровнями, ясно разграниченные интерфейсы также значительно упрощают изменение реализации уровня на совершенно другой протокол или реализацию (например, замену всех телефонных линий спутниковыми каналами), так как при этом всего лишь требуется, чтобы новый протокол или реализация предоставляла такой же набор услуг вышестоящему уровню, что и предыдущая. Вполне нормальное явление — использование хостов, принадлежащих к разным реализациям, одного и того же протокола (часто написанного различными

компаниями). Фактически может изменяться сам протокол уровня, так что уровней выше и ниже это не затронет.

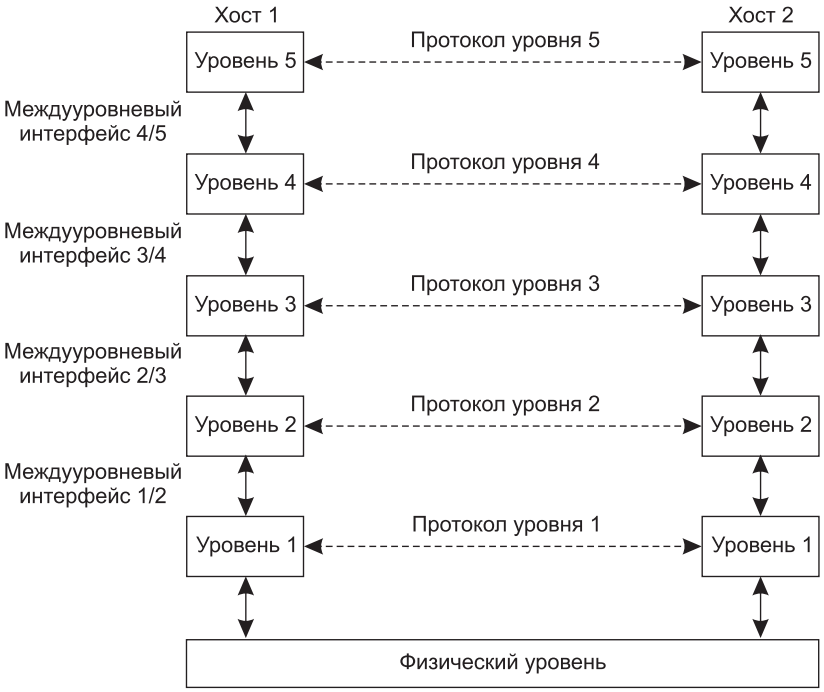


Рис. 1.11. Уровни, протоколы и интерфейсы

Набор уровней и протоколов называется **архитектурой сети**. Спецификация архитектуры должна содержать достаточно информации для написания программного обеспечения или создания аппаратуры для каждого уровня, чтобы они корректно выполняли требования протокола. Ни детали реализации, ни спецификации интерфейсов не являются частями архитектуры, так как они спрятаны внутри машины и не видны снаружи. При этом даже не требуется, чтобы интерфейсы на всех машинах сети были одинаковыми, лишь бы каждая машина правильно применяла все протоколы. Список протоколов, используемых системой, по одному протоколу на уровень, называется **стеком протоколов**. Сетевые архитектуры, стеки протоколов и сами протоколы и являются основной темой данной книги.

Чтобы было проще понять идею многоуровневого общения, можно воспользоваться следующей аналогией. Представьте себе двух философов (одноранговый процесс уровня 3), один из которых говорит на урду и английском, а другой — на китайском и французском. Поскольку нет общего языка, на котором они смогли бы общаться, каждый из них использует переводчика (одноранговый процесс уровня 2), каждый из которых, в свою очередь, нанимает секретаршу (одноранговый процесс уровня 1). Философ 1 желает выразить своему собеседнику свою привязанность к виду oryctolagus cuniculus. Для этого он передает сообщение (на английском) по интерфейсу 2/3 своему переводчику, говоря «я люблю кроликов», как изображено на

рис. 1.12. Переводчики договорились общаться на нейтральном языке, голландском, таким образом, сообщение преобразуется к виду «Ik hou van konijnen». Выбор языка является протоколом второго уровня и осуществляется одноранговыми процессами уровня 2.

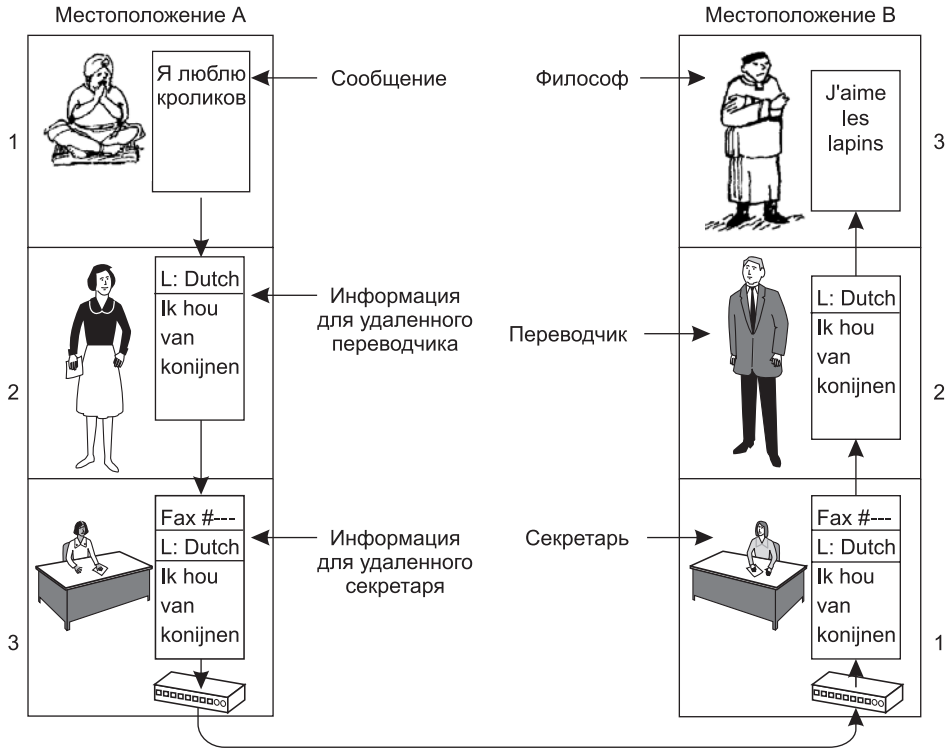


Рис. 1.12. Архитектура философ-переводчик-секретарь

Затем переводчик отдает сообщение секретарю для передачи, например, по электронной почте (протокол первого уровня). Когда сообщение получено другим секретарем, оно переводится на французский и через интерфейс 2/3 передается философу 2. Заметим, что каждый протокол полностью независим от других, поскольку интерфейсы одинаковы с каждой стороны. Переводчики могут переключиться с голландского, скажем, на финский, при условии, что оба будут согласны, при этом в интерфейсах второго уровня с первым или с третьим уровнем ничего не изменится. Подобным же образом секретари могут сменить факс на электронную почту или телефон, не затрагивая (и даже не информируя) другие уровни. Каждое изменение добавит лишь обмен информацией на своем уровне. Эта информация не будет передаваться на более высокий уровень.

Теперь рассмотрим более технический пример: как обеспечить общение для верхнего уровня пятиуровневой сети на рис. 1.13. Сообщение M производится приложением, работающим на уровне 5, и передается уровню 4 для передачи. Уровень 4 добавляет к сообщению **заголовок** для идентификации сообщения и передает результат уров-

ню 3. Заголовок включает управляющую информацию, например адреса, позволяющие уровню 4 принимающей машины доставить сообщения. Другими примерами управляющей информации, используемой в некоторых уровнях, являются порядковые номера (в случае если нижний уровень не сохраняет порядок сообщения), размеры и время.

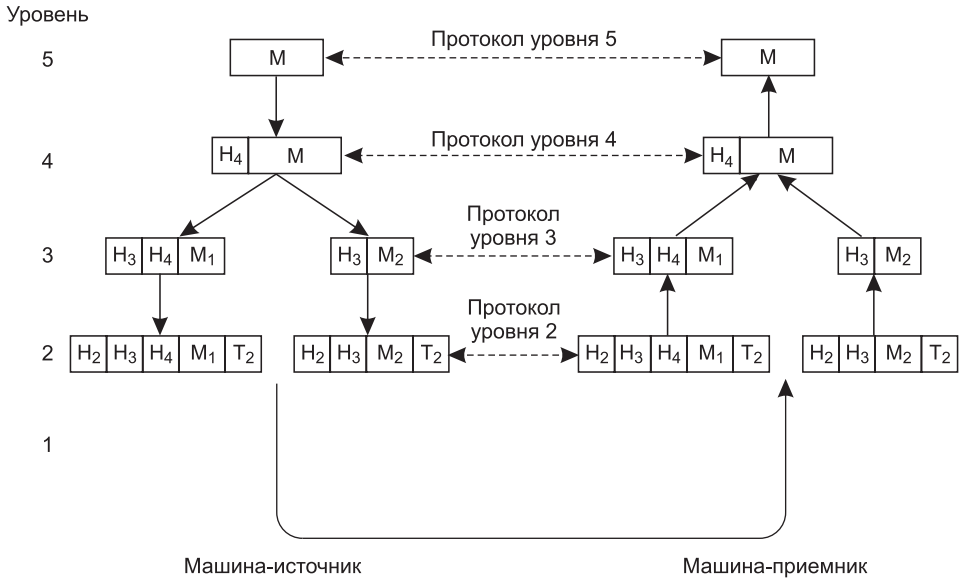


Рис. 1.13. Пример потока информации на уровне 5

Во многих сетях сообщения, передаваемые на уровне 4, не ограничиваются по размеру, однако подобные ограничения почти всегда накладываются на протокол третьего уровня. Соответственно, уровень 3 должен разбить входящие сообщения на более мелкие единицы — пакеты, предваряя каждый пакет заголовком уровня 3. В данном примере сообщение M разбивается на две части, M_1 и M_2.

Уровень 3 решает, какую из выходных линий использовать, и передает пакеты уровню 2. Уровень 2 добавляет не только заголовки к каждому пакету, но также и завершающую последовательность с контрольной суммой **(trailer)**, после чего передает результат уровню 1 для физической передачи. На получающей машине сообщение двигается по уровням вверх, при этом заголовки убираются на каждом уровне по мере продвижения сообщения. Заголовки нижних уровней более высоким уровням не передаются.

Необходимо понять соотношение между виртуальным и реальным общением и разницу между протоколом и интерфейсом. Одноранговые процессы уровня 4, например, считают свое общение горизонтальным, использующим протокол 4-го уровня. У каждого из них имеется процедура с названием вроде *SendToOtherSide* (Отправить другой стороне) и *GetFromOtherSide* (Получить от другой стороны), даже если на самом деле эти процедуры общаются не друг с другом, а с нижними уровнями при помощи интерфейсов 3/4.

Абстракция одноранговых процессов является ключевой для проектирования сетей. С ее помощью невыполнимая задача разработки целой сети может быть разбита на несколько меньших по размеру и вполне разрешимых проблем разработки, а именно разработки индивидуальных уровней.

Хотя этот раздел называется «Сетевое программное обеспечение», следует отметить, что нижние уровни в иерархии протоколов часто реализуются аппаратно или программно-аппаратно. Тем не менее при этом используются сложные алгоритмы протоколов, хотя они и внедряются в аппаратуру частично или целиком.

1.3.2. Разработка уровней

Некоторые из ключевых аспектов разработки, возникающие при создании компьютерных сетей, будут возникать от уровня к уровню. Ниже мы кратко опишем наиболее важные из них.

Надежность — проблема проектирования создания сети, которая работает правильно, даже если она составлена из набора компонентов, которые сами по себе ненадежны. Думайте о битах пакета, перемещающегося по сети. Есть шанс, что некоторые из этих битов будут получены поврежденными (инвертированными) из-за случайного электрического шума, случайных беспроводных сигналов, недостатков аппаратных средств, ошибок программного обеспечения и т. д. Возможно ли находить и исправлять эти ошибки?

Один из механизмов, для того чтобы найти ошибки в полученной информации использует коды для обнаружения ошибок. Информация, которая неправильно получена, может быть передана повторно, пока она не будет получена правильно. Более сильные коды учитывают устранение ошибки, где правильное сообщение восстанавливается из первоначально неправильно полученных битов. Оба эти механизма работают, добавляя избыточную информацию. Они используются в нижних уровнях, чтобы защитить пакеты, посланные по отдельным каналам, и в верхних уровнях, чтобы проверить, что было получено правильное содержание.

Другая проблема надежности — найти рабочий путь через сеть. Часто есть разнообразные пути между источником и местом назначения, а в большой сети некоторые каналы или маршрутизаторы могут выйти из строя. Предположите, что в Германии сеть вышла из строя. Пакеты, посланные из Лондона в Рим через Германию, не будут проходить, но мы могли вместо этого послать пакеты из Лондона в Рим через Париж. Сеть должна автоматически принять это решение. Эта тема носит название маршрутизации.

Вторая проблема проектирования касается развития сети. В течение долгого времени сети становятся больше, и появляются новые проекты, которые должны быть соединены с существующей сетью. Мы видели, что основной механизм структурирования, поддерживающий изменения — делить полную проблему и скрывать детали выполнения: иерархическое представление протокола. Существует и много других стратегий.

Когда в сети много компьютеров, каждый уровень нуждается в механизме для того, чтобы идентифицировать отправителей и получателей для каждого сообщения. Этот механизм называют адресацией или наименованием соответственно в нижних и верхних уровнях.

Аспект роста заключается в том, что у различных сетевых технологий часто есть различные ограничения. Например, не все каналы связи сохраняют порядок сообщений, посланных по ним, приводя к решению нумеровать сообщения. Другой пример — различия в максимальном размере сообщения, которое могут передать сети. Это приводит к механизмам для того, чтобы разделять, передавать и затем повторно собирать сообщения. Эту тему называют работа с объединенной сетью.

Когда сети становятся большими, возникают новые проблемы. В городах могут быть пробки, нехватка номеров телефона и возможность легко потеряться. Не у многих людей есть эти проблемы вблизи дома, но, охватив весь город, они могут быть большой проблемой. Проекты, которые продолжают работать хорошо, когда сеть становится большой, как говорят, масштабируемы.

Третья проблема проектирования — распределение ресурсов. Сети оказывают услугу узлам из их основных ресурсов, таких как способность линий передачи. Чтобы сделать это хорошо, они нуждаются в механизмах, которые делят их ресурсы так, чтобы один узел не слишком мешал работе другого.

Многие проекты совместно используют сетевую пропускную способность динамически, согласно краткосрочным потребностям узлов, а не выделяя каждому узлу фиксированной части пропускной способности, что может использоваться или не использоваться. Этот способ называют статистическим мультиплексированием, означая совместное использование основанного на статистике требования. Это может быть применено в нижних уровнях для одного канала связи или в верхних уровнях для сети или даже приложений, которые используют сеть.

Проблема распределения, которая происходит на каждом уровне, состоит в том, как препятствовать тому, что быстрый отправитель затопит данными медленного получателя. Часто используется обратная связь от получателя к отправителю. Ее называют управлением потоками. Иногда проблема состоит в том, что сеть перегружена, потому что слишком много компьютеров хотят послать слишком большие объемы информации, и сеть не может передать все. Эту перегрузку сети называют скоплением. Одна стратегия решения — требовать в таком случае от каждого компьютера уменьшения его запросов. Это также может использоваться во всех уровнях.

Интересно заметить, что сеть может предложить больше ресурсов, чем просто пропускная способность. Для использования передачи видео в реальном времени, своевременность доставки имеет большое значение. Большинство сетей должно предоставить сервис приложениям, которые хотят получить эту доставку в реальном времени в то же самое время, когда они работают с приложениями, которые хотят получить высокую пропускную способность. Качество службы — это механизмы, которые регулируют эти конкурирующие требования.

Последняя главная проблема устройства сети — обеспечить сеть защитой от различных видов угроз. Одна из угроз, которые мы упомянули ранее, — подслушивание коммуникаций. Механизмы, которые обеспечивают конфиденциальность, защищают от этой угрозы, и они используются во многих уровнях. Механизмы для аутентификации препятствуют тому, чтобы кто-то исполнил роль кого-то другого. Они могли бы использоваться, чтобы отличать поддельные банковские веб-сайты от реальных или позволить сотовой связи проверять, что вызов действительно происходит из вашего телефона, чтобы вы оплатили счет. Другие механизмы для целостности предотвраща-

ют тайные изменения сообщений, таких как изменение «снимите с моего счета $10» на «снимите с моего счета $1000». Все эти проекты основаны на криптографии, которую мы изучим в главе 8.

1.3.3. Службы на основе соединений и службы без установления соединений

Уровни могут предлагать вышестоящим уровням услуги двух типов: с наличием или отсутствием установления соединения. В этом разделе мы рассмотрим, что означает каждый из этих типов и в чем состоит разница между ними.

Типичным примером **сервиса с установлением соединения** является телефонная связь. Чтобы поговорить с кем-нибудь, необходимо поднять трубку, набрать номер, а после окончания разговора положить трубку. Нечто подобное происходит и в компьютерных сетях: при использовании сервиса с установлением соединения абонент сначала устанавливает соединение, а после окончания сеанса разрывает его. Это напоминает трубу: биты сообщения влетают в один ее конец, а вылетают с другого. В большинстве случаев не возникает путаницы с последовательностью передачи этих битов.

В некоторых случаях перед началом передачи отправляющая и получающая машины обмениваются приветствиями, отсылая друг другу приемлемые параметры соединения: максимальный размер сообщения, необходимое качество сервиса и др. В большинстве случаев одна из сторон посылает запрос, а другая его принимает, отвергает или же выставляет встречные условия. Линия — другое название соединения со связанными ресурсами, такими как фиксированная пропускная способность. Это название происходит из истории телефонной сети, в которой линия была путем по медному проводу, который переносил телефонный разговор.

Противоположный пример — **сервисы без установления соединения**, типичный пример такой технологии — почтовые системы. Каждое письмо содержит полный адрес назначения и проходит по некому маршруту, который совершенно не зависит от других писем. Есть различные названия для сообщений в различных контекстах; пакет — сообщение на сетевом уровне. Когда промежуточные узлы получают сообщение полностью перед пересылкой его к следующему узлу, это называют коммутацией с промежуточной буферизацией. Другой вариант, когда передача сообщения начинается прежде, чем оно будет полностью получено узлом, называют сквозной передачей. Обычно то письмо, которое отправлено раньше, в место назначения приходит раньше. Тем не менее возможна ситуация, что первое письмо задерживается и раньше приходит то, которое было послано вторым.

Каждая служба характеризуется **качеством обслуживания**. Некоторые службы являются надежными, в том смысле, что они никогда не теряют данные. Обычно надежная служба реализуется при помощи подтверждений, посылаемых получателем в ответ на каждое принятое сообщение, так что отправитель знает, дошло очередное сообщение или нет. Процесс пересылки подтверждений требует некоторых накладных расходов и снижает пропускную способность канала. Впрочем, подобные затраты обычно не очень велики и окупаются, хотя иногда могут быть нежелательными.

Типичным примером необходимости надежной службы на основе соединений является пересылка файлов. Владелец файла хочет быть уверен, что все биты файла прибыли без искажений и в том же порядке, в котором были отправлены. Вряд ли кто-нибудь отдаст предпочтение службе, которая случайным образом искажает информацию, даже если передача происходит значительно быстрее.

Надежные службы на основе соединений бывают двух типов: последовательности сообщений и байтовые потоки. В первом варианте сохраняются границы между сообщениями. Когда посылаются два сообщения размером по 1 Кбайт, то они прибывают в виде двух сообщений размером по 1 Кбайт и никогда как одно двухкилобайтное сообщение. При втором варианте связь представляет собой просто поток байтов, без разделения на отдельные сообщения. Когда 2048 байт прибывают к получателю, то нет никакой возможности определить, было это одно сообщение длиной 2 Кбайт, два сообщения длиной 1 Кбайт или же 2048 однобайтных сообщений. Если страницы книги посылаются по сети фотонаборной машине в виде отдельных сообщений, то, возможно, необходимо сохранить границы между сообщениями. С другой стороны, чтобы загрузить DVD-фильм, вполне достаточно потока байтов с сервера на компьютер пользователя. Границы сообщений внутри фильма не важны.

Существуют системы, для которых задержки, связанные с пересылкой подтверждений, неприемлемы. В качестве примера такой системы можно назвать цифровую голосовую связь и IP-телефонию. В данном случае предпочтительнее допустить шумы на линии или искаженные слова, нежели большие паузы, вызванные отсылкой подтверждений и повторной передачей блоков данных. Аналогично, при проведении видеоконференции отдельные неправильные пикселы окажутся меньшей проблемой, нежели движение изображения резкими толчками; из-за того, что поток останавливается и начинает исправлять ошибки, мы видим дергающиеся и останавливающиеся кадры.

Не все приложения требуют установки соединения. Например, спаммеры рассылают рекламу по электронной почте большому количеству получателей. Спаммер, вероятно, не хочет устанавливать связь для пересылки каждого отдельного сообщения, а хочет отправить один объект. Также не требуется в этом случае и 100-процентная надежность, особенно если это существенно увеличит стоимость. Все, что нужно, — это способ переслать сообщение с высокой вероятностью его получения, но без гарантии. Ненадежная (то есть без подтверждений) служба без установления соединения часто называется **службой дейтаграмм** или дейтаграммной службой, по аналогии с телеграфной службой, также не предоставляющей подтверждений отправителю. Несмотря низкую надежность, это — доминирующая форма в большинстве сетей по причинам, которые станут ясными позже.

В других ситуациях бывает желательно не устанавливать соединение для пересылки сообщений, но надежность, тем не менее, существенна. Такая служба называется **службой дейтаграмм с подтверждениями**. Она подобна отправке заказного письма с подтверждением получения. Получив подтверждение, отправитель уверен, что письмо доставлено адресату, а не потеряно по дороге. Примером являются текстовые сообщения на мобильных телефонах.

Кроме того, существует **служба запросов и ответов**, в которой отправитель посылает дейтаграммы, содержащие запросы, и получает ответы от получателя. Обычно

модель запросов и ответов применяется для реализации общения в модели клиент-сервер: клиент посылает запрос, а сервер отвечает на него. Например, пользователь мобильного телефона мог бы послать запрос в сервер карт, чтобы получить данные о карте для текущего местоположения. Обсуждавшиеся выше типы служб сведены в таблицу на рис. 1.14.

	Служба	Пример
Ориентированная на соединение	Надежный поток сообщений	Последовательность страниц
	Надежный поток байт	Удаленная регистрация
	Ненадежное соединение	Цифровая голосовая связь
Без установления соединения	Ненадежная дейтаграмма	Рассылка рекламы электронной почтой
	Дейтаграмма с подтверждениями	Заказные письма
	Запрос — ответ	Запрос к базе данных

Рис. 1.14. Шесть типов служб

Концепция использования ненадежной связи поначалу может показаться несколько странной. В самом деле, почему это может возникать такая ситуация, когда выгоднее предпочесть ненадежную связь надежной? Во-первых, надежное соединение (в том смысле, который был оговорен выше, то есть с подтверждением) не всегда можно установить на данном уровне. Скажем, Ethernet не является «надежным» средством коммуникации. Пакеты при передаче могут искажаться, но решать эту проблему должны протоколы более высоких уровней. В частности, много надежных служб создано над ненадежной службой дейтаграмм. Во-вторых, задержки, связанные с отсылкой подтверждения, в некоторых случаях неприемлемы, особенно при передаче мультимедиа в реальном времени. Именно благодаря этим факторам продолжают сосуществовать надежные и ненадежные соединения.

1.3.4. Примитивы служб

Служба (сервис) формально описывается набором **примитивов** или операций, доступных пользователю или другому объекту для получения сервиса. Эти примитивы заставляют службу выполнять некоторые действия или служат ответами на действия объекта того же уровня. Если набор протоколов входит в состав операционной системы (как часто и бывает), то примитивы являются системными вызовами. Они приводят к возникновению системных прерываний в привилегированном режиме, в результате чего управление машиной передается операционной системе, которая и отсылает нужные пакеты.

Набор доступных примитивов зависит от природы сервиса. Скажем, примитивы сервисов с установлением соединения и без него различаются. В табл. 1.3 приведен минимальный набор примитивов, обеспечивающий надежную передачу битового потока

в среде типа «клиент-сервер». Они будут знакомы поклонникам сокет-интерфейса Беркли, поскольку примитивы — упрощенная версия этого интерфейса.

Таблица 1.3. Шесть сервисных примитивов, обеспечивающих простую передачу с установлением соединения

Примитив	Значение
LISTEN (ожидание)	Блокировка, ожидание входящего соединения
CONNECT (соединение)	Установка соединения с ожидающим объектом того же ранга
ACCEPT (прием)	Прием входящего соединения от объекта того же ранга
RECEIVE (прием)	Блокировка, ожидание входящего сообщения
SEND (отправка)	Отправка сообщения ожидающему объекту того же ранга
DISCONNECT (разрыв)	Разрыв соединения

Эти примитивы могут использоваться для взаимодействия запрос-ответ в среде клиент-сервер. Чтобы проиллюстрировать, как это происходит, мы покажем набросок простого протокола, который осуществляет службу, используя общепризнанные дейтаграммы.

Вначале сервер исполняет LISTEN, показывая тем самым, что он готов устанавливать входящие соединения. Этот примитив обычно реализуется в виде блокирующего системного вызова. После его исполнения процесс сервера приостанавливается до тех пор, пока не будет установлено соединение.

Затем процесс клиента выполняет примитив CONNECT, устанавливая соединение с сервером. В системном вызове должно быть указано, с кем именно необходимо установить связь. Для этого может вводиться специальный параметр, содержащий адрес сервера. Далее операционная система клиента посылает равноранговой сущности пакет с запросом на соединение, как показано на рис. 1.15 стрелочкой с пометкой (1). Процесс клиента приостанавливается в ожидании ответа.

Когда пакет приходит на сервер, операционная система обнаруживает запрос на соединение. Она проверяет, есть ли слушатель, и в этом случае разблокирует его. Затем процесс сервера может установить соединение с помощью ACCEPT. Он посылает ответ (2) назад к процессу клиента, чтобы принять соединение. Прибытие этого ответа освобождает клиента. Начиная с этого момента считается, что сервер и клиент установили соединение.

Самым очевидным жизненным примером такого взаимодействия может служить звонок покупателя (клиента) в сервисный центр компании. Менеджер сервисного центра должен находиться у телефона, чтобы иметь возможность ответить на звонок. Клиент совершает звонок. Когда менеджер поднимает трубку, считается, что соединение установлено.

Следующим шагом будет выполнение сервером примитива RECEIVE, подготавливающего систему к принятию первого запроса. В нормальной ситуации это происходит сразу же после прекращения ожидания (LISTEN), даже до того, как клиент получает подтверждение соединения. Системный вызов RECEIVE вновь блокирует сервер.

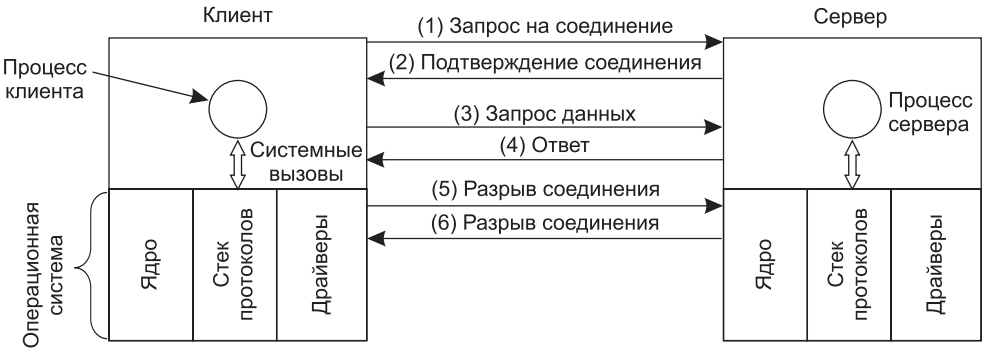


Рис. 1.15. Простейшее взаимодействие клиента и сервера с использованием общепризнанных дейтаграмм

Клиент выполняет SEND, передает запрос (3) и сразу же выполняет RECEIVE, ожидая ответ. Прием пакета с запросом разблокирует сервер, благодаря чему он может обработать запрос. По окончании обработки сервер выполняет примитив SEND, и ответ отсылается клиенту (4). Прием пакета разблокирует клиента, теперь наступает его очередь обрабатывать пакет. Если у клиента есть еще запросы к серверу, он может отослать их.

Когда запросы клиента окончены, он осуществляет разрыв соединения с помощью DISCONNECT. Обычно первый примитив DISCONNECT отсылает пакет, уведомляющий сервер об окончании сеанса, и блокирует клиента (5). В ответ сервер генерирует свой примитив DISCONNECT, являющийся подтверждением для клиента и командой, разрывающей связь. Клиент, получив его, разблокируется, и соединение считается окончательно разорванным. Именно так в двух словах можно описать схему коммуникации с установлением соединения.

Конечно, жизнь не настолько проста. Описанный выше алгоритм работы весьма схематичен, а кое-что просто неправильно (например, CONNECT на самом деле выполняется до LISTEN). При этом пакеты, бывает, теряются, возникают и другие проблемы. Позднее мы рассмотрим все это гораздо более подробно, но на данный момент можно по рис. 1.15 получить общее представление о работе клиент-серверной системы с установлением соединения, использованием общепризнанных дейтаграмм и игнорированием потерянных пакетов.

Увидев эти шесть пакетов, необходимых для работы протокола, можно удивиться, почему же не используется протокол без установления соединения? Ответ таков: в идеальном мире, где нужны всего два пакета — один для запроса и один для ответа, — это, возможно, имело бы смысл. Но стоит представить себе передачу большого сообщения (скажем, мегабайтного файла), причем в обе стороны, причем с ошибками при передаче, потерянными пакетами и т. д., как ситуация меняется. Если ответ сервера состоит из нескольких сотен пакетов, парочка из которых затерялась по пути, то как клиент узнает, что он получил сообщение не в полном объеме? Как он узнает о том, что последний принятый пакет является действительно последним? Допустим, клиент запросил второй файл. Как он отличит пакет 1 из второго файла от потерянного пакета 1 из первого файла, который вдруг нашелся? Короче говоря, в реальном мире простой протокол запросов-ответов без подтверждений часто не подходит. В главе 3

мы обсудим протоколы, позволяющие решать самые разные проблемы, возникающие при передаче данных. А сейчас поверьте на слово: наличие надежной связи с упорядоченным байтовым потоком между процессами — это удобно.

1.3.5. Службы и протоколы

Службы и протоколы являются различными понятиями. Различие между ними столь важно, что мы хотели бы еще раз обратить на него ваше внимание. *Служба* (или *сервис*) — это набор примитивов (операций), которые более низкий уровень предоставляет более высокому. Служба определяет, какие именно операции уровень будет выполнять от лица своих пользователей, но никак не оговаривает, как должны реализовываться эти операции. Служба описывает интерфейс между двумя уровнями, в котором нижний уровень является поставщиком сервиса, а верхний — его потребителем.

Напротив, *протокол* — это набор правил, описывающих формат и назначение кадров, пакетов или сообщений, которыми обмениваются объекты одного ранга внутри уровня. Объекты используют протокол для реализации определений своих служб. Они могут менять протокол по желанию, при условии, что при этом остаются неизменными службы, предоставляемые ими своим пользователям. Таким образом, служба и протокол оказываются практически независимыми. Это — ключевое понятие, которое должен хорошо понять любой проектировщик сетей.

Повторим этот важный момент, службы — это нечто, связанное с межуровневыми интерфейсами, тогда как протоколы связаны с пакетами, передающимися объектами одного уровня, расположенными на разных машинах. Это показано на рис. 1.16. Очень важно не путать эти два понятия.

Стоит провести аналогию с языками программирования. Службу можно уподобить абстрактному типу данных или объекту в объектно-ориентированных языках программирования. Он определяет операции, которые могут выполняться с объектом, но не описывает, как реализованы эти операции. В этом случае протокол, напротив, относится к *реализации* службы и, таким образом, невидим для пользователей службы.

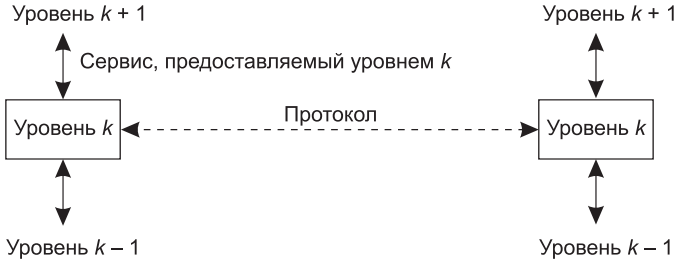


Рис. 1.16. Связь между службой и протоколом

Во многих старых системах служба не отделялась от протокола. В результате типичный уровень мог содержать примитив службы SEND PACKET, в котором пользователь должен был указать ссылку на полностью собранный пакет. Это означало, что любые изменения протокола тут же становились видимыми для пользователей. Большинство разработчиков сетей сегодня считают подобный подход серьезнейшей ошибкой.

1.4. Эталонные модели

Обсудив многоуровневые сети в общих чертах, пора рассмотреть несколько примеров. Мы опишем два важных архитектурных типа — эталонные модели OSI и TCP/IP. Несмотря на то что *протоколы*, связанные с эталонной моделью OSI, сейчас не используются, сама *модель* до сих пор весьма актуальна, а свойства ее уровней, которые будут обсуждаться в этом разделе, очень важны. В эталонной модели TCP/IP все наоборот: сама модель сейчас почти не используется, а ее протоколы являются едва ли не самыми распространенными. Исходя из этого, мы обсудим подробности, касающиеся обеих моделей. К тому же иногда приходится больше узнавать из поражений, чем из побед.

1.4.1. Эталонная модель OSI

Эталонная модель OSI (за исключением физической среды) показана на рис. 1.17. Эта модель основана на разработке Международной организации по стандартизации (International Organization for Standardization, ISO) и является первым шагом к международной стандартизации протоколов, используемых на различных уровнях (Day и Zimmerman, 1983). Затем она была пересмотрена в 1995 году (Day, 1995). Называется эта структура эталонной моделью взаимодействия открытых систем ISO (**ISO OSI** (**Open System Interconnection**) **Reference Model**), поскольку она связывает открытые системы, то есть системы, открытые для связи с другими системами. Для краткости мы будем называть эту модель просто «модель OSI».

Модель OSI имеет семь уровней. Появление именно такой структуры было обусловлено следующими соображениями.

1. Уровень должен создаваться по мере необходимости отдельного уровня абстракции.
2. Каждый уровень должен выполнять строго определенную функцию.
3. Выбор функций для каждого уровня должен осуществляться с учетом создания стандартизированных международных протоколов.
4. Границы между уровнями должны выбираться так, чтобы поток данных между интерфейсами был минимальным.
5. Количество уровней должно быть достаточно большим, чтобы различные функции не объединялись в одном уровне без необходимости, но не слишком высоким, чтобы архитектура не становилась громоздкой.

Ниже мы обсудим каждый уровень модели, начиная с самого нижнего. Обратите внимание: модель OSI не является сетевой архитектурой, поскольку она не описывает службы и протоколы, используемые на каждом уровне. Она просто определяет, что должен делать каждый уровень. Тем не менее ISO также разработала стандарты для каждого уровня, хотя эти стандарты не входят в саму эталонную модель. Каждый из них был опубликован как отдельный международный стандарт. Эта модель (частично) широко используется, хотя связанные с ней протоколы долго были забыты.

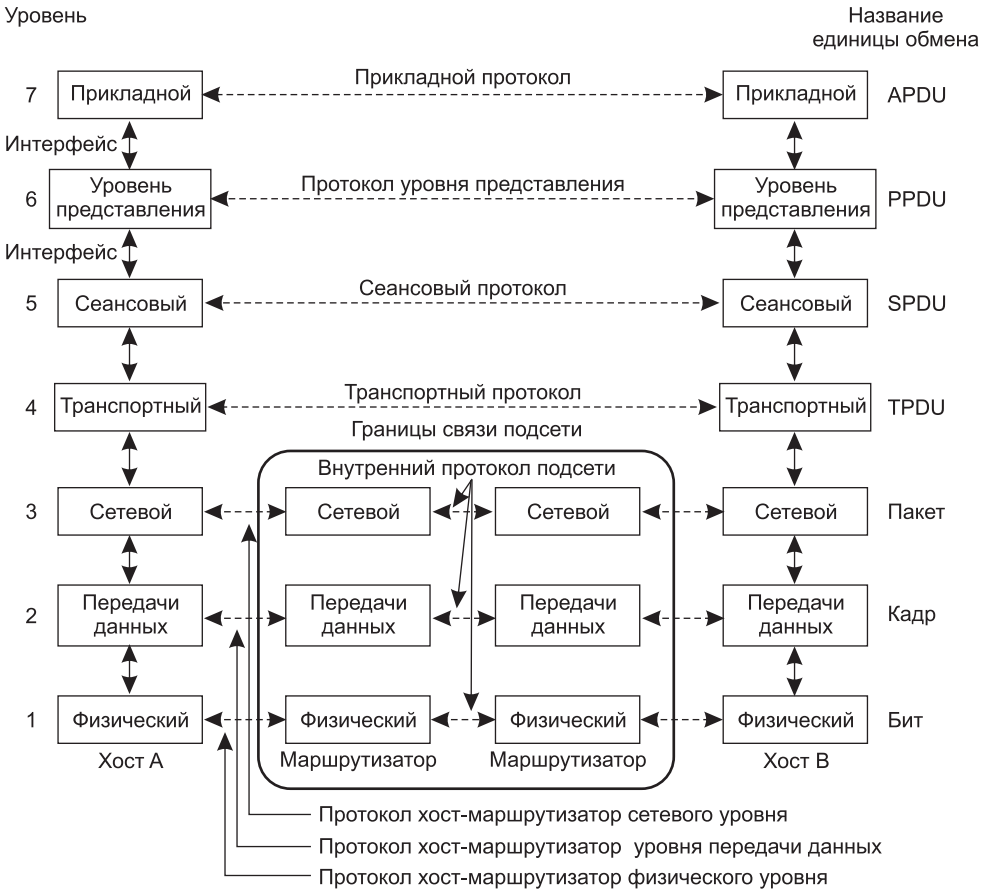


Рис. 1.17. Эталонная модель OSI

Физический уровень

Физический уровень занимается реальной передачей необработанных битов по каналу связи. При разработке сети необходимо убедиться, что когда одна сторона передает единицу, то принимающая сторона получает также единицу, а не ноль. Принципиальными вопросами здесь являются следующие: какое напряжение должно использоваться для отображения единицы, а какое для нуля; сколько микросекунд длится бит; может ли передача производиться одновременно в двух направлениях; как устанавливается начальная связь и как она прекращается, когда обе стороны закончили свои задачи; из какого количества проводов должен состоять кабель и какова функция каждого провода. Вопросы разработки в основном связаны с механическими, электрическими и процедурными интерфейсами, а также с физическим носителем, лежащим ниже физического уровня.

Уровень передачи данных

Основная задача **уровня передачи данных** — быть способным передавать «сырые» данные физического уровня по надежной линии связи, свободной от необнаруженных ошибок, и маскировать реальные ошибки, так что сетевой уровень их не видит. Эта задача выполняется при помощи разбиения входных данных на **кадры**, обычный размер которых колеблется от нескольких сот до нескольких тысяч байт. Кадры данных передаются последовательно с обработкой **кадров подтверждения**, отсылаемых обратно получателем.

Еще одна проблема, возникающая на уровне передачи данных (а также и на большей части более высоких уровней), — как не допустить ситуации, когда быстрый передатчик заваливает приемник данными. Может быть предусмотрен некий механизм регуляции, который информировал бы передатчик о наличии свободного места в буфере приемника на текущий момент.

В широковещательных сетях существует еще одна проблема уровня передачи данных: как управлять доступом к совместно используемому каналу. Эта проблема разрешается введением специального дополнительного подуровня уровня передачи данных — подуровня доступа к носителю.

Сетевой уровень

Сетевой уровень занимается управлением операциями подсети. Важнейшим моментом здесь является определение маршрутов пересылки пакетов от источника к пункту назначения. Маршруты могут быть жестко заданы в виде таблиц и редко меняться либо, что бывает чаще, автоматически изменяться, чтобы избегать отказавших компонентов. Кроме того, они могут задаваться в начале каждого соединения, например, терминальной сессии, такого как подключения к удаленной машине. Наконец, они могут быть в высокой степени динамическими, то есть вычисляемыми заново для каждого пакета с учетом текущей загруженности сети.

Если в подсети одновременно присутствует слишком большое количество пакетов, то они могут закрыть дорогу друг другу, образуя заторы в узких местах. Недопущение подобной закупорки также является задачей сетевого уровня в соединении с более высокими уровнями, которые адаптируют загрузку. В более общем смысле, сетевой уровень занимается предоставлением определенного уровня сервиса (это касается задержек, времени передачи, вопросов синхронизации).

При путешествии пакета из одной сети в другую также может возникнуть ряд проблем. Так, способ адресации, применяемый в одной сети, может отличаться от принятого в другой. Сеть может вообще отказаться принимать пакеты из-за того, что они слишком большого размера. Также могут различаться протоколы и т. д. Именно сетевой уровень должен разрешать все эти проблемы, позволяя объединять разнородные сети.

В широковещательных сетях проблема маршрутизации очень проста, поэтому в них сетевой уровень очень примитивный или вообще отсутствует.

Транспортный уровень

Основная функция **транспортного уровня** — принять данные от сеансового уровня, разбить их при необходимости на небольшие части, передать их сетевому уровню

и гарантировать, что эти части в правильном виде прибудут по назначению. Кроме того, все это должно быть сделано эффективно и таким образом, чтобы изолировать более высокие уровни от каких-либо изменений в аппаратной технологии с течением времени.

Транспортный уровень также определяет тип сервиса, предоставляемого сеансовому уровню и, в конечном счете, пользователям сети. Наиболее популярной разновидностью транспортного соединения является защищенный от ошибок канал между двумя узлами, поставляющий сообщения или байты в том порядке, в каком они были отправлены. Однако транспортный уровень может предоставлять и другие типы сервисов, например пересылку отдельных сообщений без гарантии соблюдения порядка их доставки или одновременную отправку сообщения различным адресатам по принципу широковещания. Тип сервиса определяется при установке соединения. (Строго говоря, полностью защищенный от ошибок канал создать совершенно невозможно. Говорят лишь о таком канале, уровень ошибок в котором достаточно мал, чтобы им можно было пренебречь на практике.)

Транспортный уровень является настоящим сквозным уровнем, то есть доставляющим сообщения от источника адресату. Другими словами, программа на машине-источнике поддерживает связь с подобной программой на другой машине при помощи заголовков сообщений и управляющих сообщений. На более низких уровнях для поддержки этого соединения устанавливаются соединения между всеми соседними машинами, через которые проходит маршрут сообщений. Различие между уровнями с 1-го по 3-й, действующих по принципу звеньев цепи, и уровнями с 4-го по 7-й, являющимися сквозными, проиллюстрировано на рис. 1.17.

Сеансовый уровень

Сеансовый уровень позволяет пользователям различных компьютеров устанавливать **сеансы связи** друг с другом. При этом предоставляются различные типы сервисов, среди которых **управление диалогом** (отслеживание очередности передачи данных), **управление маркерами** (предотвращение одновременного выполнения критичной операции несколькими системами) и **синхронизация** (установка служебных меток внутри длинных сообщений, позволяющих продолжить передачу с того места, на котором она оборвалась, даже после сбоя и восстановления).

Уровень представления

В отличие от более низких уровней, задача которых — достоверная передача битов и байтов, **уровень представления** занимается по большей части синтаксисом и семантикой передаваемой информации. Чтобы было возможно общение компьютеров с различными внутренними представлениями данных, необходимо преобразовывать форматы данных друг в друга, передавая их по сети в неком стандартизированном виде. Уровень представления занимается этими преобразованиями, предоставляя возможность определения и изменения структур данных более высокого уровня (например, записей баз данных).

Прикладной уровень

Прикладной уровень содержит набор популярных протоколов, необходимых пользователям. Одним из наиболее распространенных является протокол передачи гипертекста **HTTP** (HyperText Transfer Protocol), который составляет основу технологии Всемирной паутины. Когда браузер запрашивает веб-страницу, он передает ее имя (адрес) и рассчитывает на то, что сервер, на котором расположена страница, будет использовать HTTP. Сервер в ответ отсылает страницу. Другие прикладные протоколы используются для передачи файлов, электронной почты, сетевых рассылок.

1.4.2. Эталонная модель TCP/IP

Рассмотрим теперь эталонную модель, использовавшуюся в компьютерной сети ARPANET, которая является бабушкой нынешних сетей, а также в ее наследнице, всемирной сети Интернет. Хотя краткую историю сети ARPANET мы рассмотрим чуть позднее, некоторые ключевые моменты ее следует отметить прямо сейчас. ARPANET была исследовательской сетью, финансируемой Министерством обороны США. В конце концов, она объединила сотни университетов и правительственных зданий при помощи выделенных телефонных линий. Когда впоследствии появились спутниковые сети и радиосети, возникли большие проблемы при объединении с ними других сетей с помощью имеющихся протоколов. Понадобилась новая эталонная архитектура. Таким образом, возможность объединять различные сети в единое целое являлась одной из главных целей с самого начала. Позднее эта архитектура получила название **эталонной модели TCP/IP**, в соответствии со своими двумя основными протоколами. Первое ее описание встречается в книге Cerf и Kahn (1974), позднее превращается в стандарт (Braden, 1989). Конструктивные особенности модели обсуждаются в издании Clark, 1988.

Поскольку Министерство обороны США беспокоилось, что ценные хосты, маршрутизаторы и межсетевые шлюзы могут быть мгновенно уничтожены, другая важная задача состояла в том, чтобы добиться способности сети сохранять работоспособность при возможных потерях подсетевого оборудования, так чтобы при этом связь не прерывалась. Другими словами, Министерство обороны США требовало, чтобы соединение не прерывалось, пока функционируют приемная и передающая машины, даже если некоторые промежуточные машины или линии связи внезапно вышли из строя. Кроме того, от архитектуры нужна была определенная гибкость, поскольку предполагалось использовать приложения с различными требованиями, от переноса файлов до передачи речи в реальном времени.

Канальный уровень

Все эти требования привели к выбору сети с пакетной коммутацией, основанной на уровне без установления соединения, который работает в различных сетях. Самый низкий уровень в модели, уровень канала, описывает то, как и что каналы, такие как последовательные линии и классический Ethernet, должны сделать, чтобы удовлет-

ворить потребности этого межсетевого уровня без установления соединения. Это на самом деле не уровень вообще, в нормальном смысле слова, а скорее интерфейс между каналами передачи и узлами. В ранних материалах о модели TCP/IP мало что об этом говорится.

Межсетевой уровень

Все эти требования обусловили выбор модели сети с коммутацией пакетов, в основе которой лежал не имеющий соединений межсетевой уровень. Он показан на рис. 1.18 и примерно соответствует сетевому уровню в OSI. Этот уровень, называемый **интернет-уровнем** или **межсетевым уровнем**, является основой всей архитектуры. Его задача заключается в обеспечении возможности каждого хоста посылать пакеты в любую сеть и независимо двигаться к пункту назначения (например, в другой сети). Они могут прибывать совершенно в другом порядке, чем были отправлены. Если требуется соблюдение порядка отправления, эту задачу выполняют более верхние уровни. Обратите внимание, что слово «интернет» здесь используется в своем первоначальном смысле, несмотря на то что этот уровень присутствует в сети Интернет.

Здесь можно увидеть аналогию с почтовой системой. Человек может бросить несколько международных писем в почтовый ящик в одной стране, и, если повезет, большая часть из них будет доставлена по правильным адресам в других странах. Вероятно, письма по дороге пройдут через несколько международных почтовых шлюзов, однако это останется тайной для корреспондентов. В каждой стране (то есть в каждой сети) могут быть свои марки, свои предпочитаемые размеры конвертов и правила доставки, незаметные для пользователей почтовой службы.

Межсетевой уровень определяет официальный формат пакета и протокол **IP**, с дополнительным протоколом ICMP (Internet Control Message Protocol, межсетевой протокол управления сообщениями). Задачей межсетевого протокола является доставка IP-пакетов к пунктам назначения. Основными аспектами здесь являются выбор маршрута пакета и недопущение закупорки транспортных артерий (хотя IP не оказался эффективным для избегания скоплений).

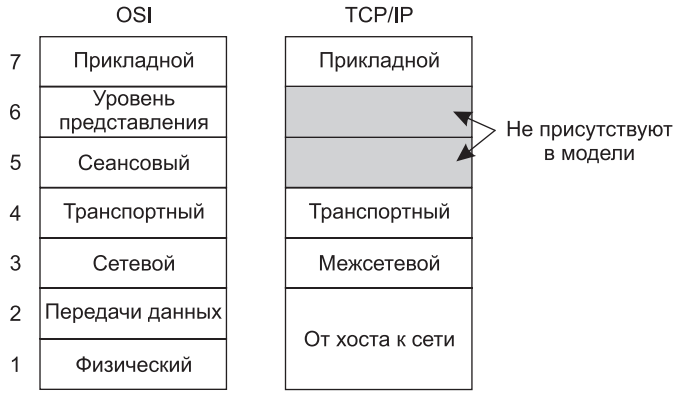


Рис. 1.18. Эталонная модель TCP/IP

Транспортный уровень

Уровень, расположенный над межсетевым уровнем модели TCP/IP, как правило, называют **транспортным**. Он создан для того, чтобы объекты одного ранга на приемных и передающих хостах могли поддерживать связь, подобно транспортному уровню модели OSI. На этом уровне должны быть описаны два сквозных протокола. Первый, **TCP (Transmission Control Protocol — протокол управления передачей)**, является надежным протоколом с установлением соединений, позволяющим без ошибок доставлять байтовый поток с одной машины на любую другую машину объединенной сети. Он разбивает входной поток байтов на отдельные сообщения и передает их межсетевому уровню. На пункте назначения получающий TCP-процесс собирает из полученных сообщений выходной поток. Кроме того, TCP осуществляет управление потоком, чтобы быстрый отправитель не завалил информацией медленного получателя.

Второй протокол этого уровня, **UDP (User Dataram Protocol — протокол пользовательских дейтограмм2)**, является ненадежным протоколом без установления соединения, не использующим последовательное управление потоком протокола TCP, а предоставляющим свое собственное. Он также широко используется в одноразовых клиент-серверных запросах и приложениях, в которых оперативность важнее аккуратности, например при передаче речи и видео. Взаимоотношения протоколов IP, TCP и UDP показаны на рис. 1.19. Со времени создания протокола IP этот протокол был реализован во многих других сетях.

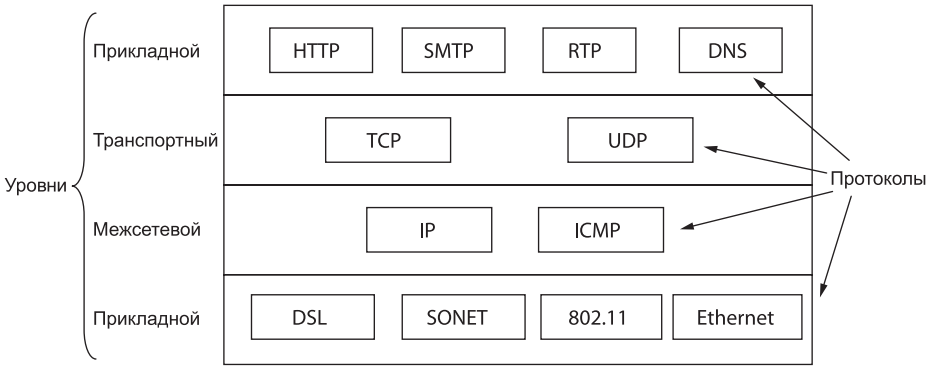


Рис. 1.19. Протоколы и сети в модели TCP/IP

Прикладной уровень

В модели TCP/IP нет сеансового уровня и уровня представления. В этих уровнях просто не было необходимости, поэтому они не были включены в модель. Вместо этого приложения просто включают все функции сеансов и представления, которые им нужны. Опыт работы с моделью OSI доказал правоту этой точки зрения: большинство приложений мало нуждаются в этих уровнях.

Над транспортным уровнем располагается прикладной уровень. Он содержит все протоколы высокого уровня. К старым протоколам относятся протокол виртуального

терминала (TELNET), протокол переноса файлов (FTP) и протокол электронной почты (SMTP). С годами было добавлено много других протоколов. Некоторые наиболее важные, которые мы рассмотрим, показаны на рис. 1.19. Это **DNS (Domain Name Service — служба имен доменов)**, позволяющая преобразовывать имена хостов в сетевые, HTTP, протокол, используемый для создания страниц на World Wide Web, а также RTP, протокол для представления мультимедиа в реальном времени, таких как звук или фильмы.

1.4.3. Модель, используемая в книге

Как упомянуто ранее, сила эталонной модели OSI — сама модель (минус представление и уровни сеансов), которая оказалась исключительно полезной для обсуждения компьютерных сетей. Напротив, сила эталонной модели TCP/IP — протоколы, которые широко использовались много лет. Чтобы совместить эти качества, мы будем использовать в книге гибридную модель, показанную на рис. 1.20.

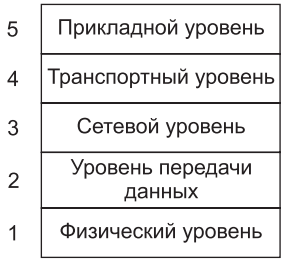


Рис. 1.20. Эталонная модель, используемая в этой книге

Задача сетевого уровня объединить многочисленные каналы в сети и сети сетей, а также в объединенные сети, чтобы мы могли посылать пакеты между удаленными компьютерами. Это включает задачу обнаружения пути, по которому можно послать пакеты. IP — основной протокол, который мы изучим в качестве примера для этого уровня. Транспортный уровень усиливает гарантии доставки сетевого уровня, обычно с увеличенной надежностью, и обеспечивает параметры доставки, такие как надежный поток байтов, соответствующий потребности различных приложений. TCP — важный пример протокола транспортного уровня.

Наконец, прикладной уровень содержит программы, которые используют сеть. У многих, но не всех сетевых приложений есть пользовательские интерфейсы, такие как веб-браузер. Нас интересует, однако, та часть программы, которая использует сеть. Это — протокол HTTP в случае веб-браузера. Есть также важные программы поддержки в прикладном уровне, такие как DNS, которые используются многими приложениями.

Последовательность глав в книге основана на этой модели. Таким образом, мы сохраняем значение модели OSI для понимания сетевой архитектуры, но концентрируемся, прежде всего, на протоколах, которые важны практически, от TCP/IP и связанных с ним протоколов до более новых, таких как 802.11, SONET и Bluetooth.

1.4.4. Сравнение эталонных моделей OSI и TCP

У моделей OSI и TCP имеется много общих черт. Обе модели основаны на концепции стека независимых протоколов. Функциональность уровней также во многом схожа. Например, в обеих моделях уровни, начиная с транспортного и выше, предоставляют сквозную, не зависящую от сети транспортную службу для процессов, желающих обмениваться информацией. Эти уровни образуют поставщика транспорта. Также в каждой модели уровни выше транспортного являются прикладными потребителями транспортных сервисов.

Несмотря на это фундаментальное сходство, у этих моделей имеется и ряд отличий. В данном разделе мы рассмотрим ключевые различия. Обратите внимание на то, что мы сравниваем именно *эталонные модели*, а не соответствующие им *стеки протоколов*. Сами протоколы будут обсуждаться несколько позднее. Книга (Piscitello и Chapin, 1993) целиком посвящена сравнению моделей TCP/IP и OSI.

Для модели OSI центральными являются три концепции.

1. Службы.
2. Интерфейсы.
3. Протоколы.

Вероятно, наибольшим вкладом модели OSI стало явное разделение этих трех концепций. Каждый уровень предоставляет некоторые сервисы для расположенного выше уровня. *Сервис* определяет, что именно делает уровень, но не то, как он это делает и каким образом объекты, расположенные выше, получают доступ к данному уровню.

Интерфейс уровня определяет способ доступа к уровню для расположенных выше процессов. Он описывает параметры и ожидаемый результат. Он также ничего не сообщает о внутреннем устройстве уровня.

Наконец, равноранговые *протоколы*, применяемые в уровне, являются внутренним делом самого уровня. Для выполнения поставленной ему задачи (то есть предоставления сервиса) он может использовать любые протоколы. Кроме того, уровень может менять протоколы, не затрагивая работу приложений более высоких уровней.

Эти идеи очень хорошо соответствуют современным идеям объектно-ориентированного программирования. Уровень может быть представлен в виде объекта, обладающего набором методов (операций), к которым может обращаться внешний процесс. Семантика этих методов определяет набор служб, предоставляемых объектом. Параметры и результаты методов образуют интерфейс объекта. Внутреннее устройство объекта можно сравнить с протоколом уровня. За пределами объекта оно никого не интересует и никому не видно.

Изначально в модели TCP/IP не было четкого разделения между службами, интерфейсом и протоколами, хотя и производились попытки изменить это, чтобы сделать ее более похожей на модель OSI. Так, например, единственными настоящими сервисами, предоставляемыми межсетевым уровнем, являются SEND IP PACKET (послать IP-пакет) и RECEIVE IP PACKET (получить IP-пакет).

В результате в модели OSI протоколы скрыты лучше, чем в модели TCP/IP, и при изменении технологии они могут быть относительно легко заменены. Возможность

проводить подобные изменения, не затрагивая другие уровни, является одной из главных целей многоуровневых протоколов.

Эталонная модель OSI была разработана *прежде*, чем были изобретены протоколы для нее. Такая последовательность событий означала, что эта модель не была настроена на какой-то конкретный набор протоколов, что делало ее универсальной. Обратной стороной такого порядка действий было то, что у разработчиков было мало опыта в данной области и не было четкого представления о том, какие функции должен выполнять каждый уровень.

Например, уровень передачи данных изначально работал только в сетях с передачей от узла к узлу. С появлением широковещательных сетей в модель потребовалось ввести новый подуровень. В дальнейшем, когда на базе модели OSI начали строить реальные сети с использованием существующих протоколов, обнаружилось, что они не соответствуют требуемым спецификациям служб. Поэтому в модель пришлось добавить подуровни для устранения несоответствия. Наконец, изначально ожидалось, что в каждой стране будет одна сеть, управляемая правительством и использующая протоколы OSI, поэтому никто и не думал об объединении различных сетей. В действительности все оказалось не так.

С моделью TCP/IP было все наоборот: сначала появились протоколы, а уже затем была создана модель, описывающая существующие протоколы. Таким образом, не было проблемы с соответствием протоколов модели. Они ей соответствовали прекрасно. Единственной проблемой было то, что *модель* не соответствовала никаким другим стекам протоколов. В результате она не использовалась для описания каких-нибудь других сетей, отличных от TCP/IP.

Если взглянуть на эти две модели поближе, то, прежде всего, обратит на себя внимание различие в количестве уровней: в модели OSI семь уровней, в модели TCP/IP — четыре. В обеих моделях имеются межсетевой, транспортный и прикладной уровни, а остальные уровни различные.

Еще одно различие между моделями лежит в сфере возможности использования связи на основе соединений и связи без установления соединения. Модель OSI на сетевом уровне поддерживает оба типа связи, а на транспортном уровне — только связь на основе соединений (поскольку транспортные службы являются видимыми для пользователя). В модели TCP/IP на сетевом уровне есть только один режим связи (без установления соединения), но на транспортном уровне она поддерживает оба режима, предоставляя пользователям выбор. Этот выбор особенно важен для простых протоколов запрос-ответ.

1.4.5. Критика модели и протоколов OSI

Ни описанные выше модели (OSI и TCP/IP), ни их протоколы не являются совершенными. Довольно много критики было высказано по поводу обеих моделей. Некоторые критические замечания мы рассмотрим в данном и в следующем разделах. Сначала проанализируем модель OSI, а затем TCP/IP.

В то время, когда вышло второе (английское. — *Примеч. ред.*) издание этой книги (1989), многим экспертам в данной области казалось, что модель OSI и ее протоколы завоюют весь мир и вытеснят все остальное. Этого не случилось. Почему? Может быть,

полезно оглянуться и учесть некоторые из причин этого. Основных причин неудачи модели OSI было четыре:

- несвоевременность;
- неудачная технология;
- неудачная реализация;
- неудачная политика.

Несвоевременность

Прежде всего рассмотрим причину номер один: несвоевременность. Для успеха стандарта чрезвычайно важно, в какое время он устанавливается. У Дэвида Кларка (David Clark) из M.I.T. есть теория стандартов, которую он называет *апокалипсисом двух слонов* (рис. 1.21).

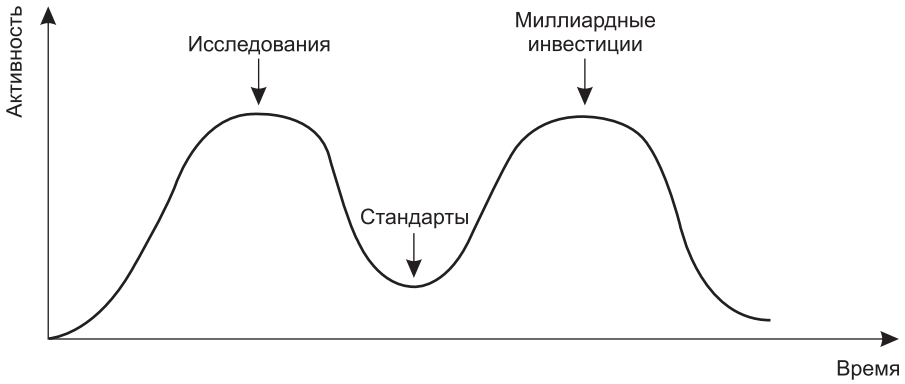


Рис. 1.21. Апокалипсис двух слонов

На этом рисунке изображена активность, сопровождающая любую новую разработку. Открытие новой темы вначале вызывает всплеск исследовательской активности в виде дискуссий, статей и собраний. Через некоторое время наступает спад активности, эту тему открывают для себя корпорации, и в результате в нее инвестируются миллиарды долларов.

Существенным является то, что стандарты пишутся именно в период между двумя «слонами». Если их создавать слишком рано, прежде чем закончатся исследования, предмет может оказаться еще слишком мало изучен и понят, что повлечет принятие плохих стандартов. Если создавать их слишком поздно, компании могут успеть вложить деньги в несколько отличные от стандартов технологии, так что принятые стандарты могут оказаться проигнорированными. Если интервал между двумя пиками активности будет слишком коротким (а все стремятся делать деньги как можно быстрее), разработчики стандартов могут просто не успеть их выработать.

Теперь становится ясно, почему стандартные протоколы OSI потерпели неудачу. К моменту их появления среди исследовательских университетов уже получили широкое распространение конкурирующие с ними протоколы TCP/IP. И хотя волна

инвестиций еще не обрушилась на данную область, рынок университетов был достаточно широк для того, чтобы многие разработчики стали осторожно предлагать продукты, поддерживающие протоколы TCP/IP. Когда же появился OSI, разработчики не захотели поддерживать второй стек протоколов; таким образом, начальных предложений не было. Каждая компания выжидала, пока первым начнет кто-нибудь другой, поэтому OSI так никто и не стал поддерживать.

Плохая технология

Второй причиной, по которой модель OSI не была реализована, оказалось несовершенство как самой модели, так и ее протоколов. Выбор семиуровневой структуры стал больше политическим решением, чем техническим. В результате два уровня (сеансовый и уровень представления) почти пусты, тогда как два других (сетевой и передачи данных) перегружены.

Эталонная модель OSI вместе с соответствующими определениями служб и протоколами оказалась невероятно сложной. Если сложить в стопку распечатку официального описания стандартов, получится кипа бумаги высотой в один метр. Модель тяжело реализуема и неэффективна в работе. В этом контексте вспоминается шутка Пола Мокапетриса (Paul Mockapetris), процитированная в издании Rose, 1993.

Вопрос. Что получится, если скрестить гангстера с международным стандартом?

Ответ. Человек, делающий вам предложения, которые вы не способны понять.

Еще одна проблема, помимо невозможности понять стандарты OSI, заключалась в том, что некоторые функции, такие как адресация, управление потоком и обработка ошибок, повторялись снова и снова в каждом уровне. Так, например, в книге Saltzer и др. (1984) указывается, что для того чтобы контроль за ошибками был эффективным, он должен осуществляться на самом верхнем уровне, поэтому повторение его снова и снова на каждом уровне часто оказывается излишним и неэффективным.

Неудачная реализация

Учитывая огромную сложность модели и протоколов, громоздкость и медлительность первых реализаций не стали неожиданностью. Неудачу потерпели все, кто попытался реализовать эту модель. Поэтому вскоре понятие «OSI» стало ассоциироваться с плохим качеством. И хотя со временем продукты улучшились, ассоциации остались.

Первые реализации TCP/IP, основанные на Berkley UNIX, напротив, были достаточно хороши (не говоря уже о том, что они были открытыми). Они довольно быстро вошли в употребление, что привело к появлению большого сообщества пользователей. Это вызвало исправления и улучшения реализации, в результате чего сообщество пользователей еще выросло. В данном случае обратная связь явно была положительной.

Неудачная политика

Из-за особенностей первоначальной реализации многие, особенно в университетских кругах, считали TCP/IP частью системы UNIX. А к системе UNIX в университетских кругах в 80-е годы испытывали чувства средние между родительскими (в те времена

некорректно, ущемляя права мужского населения, называемые материнскими) и чувствами к яблочному пирогу.

С другой стороны, OSI считался детищем европейских телекоммуникационных министерств, европейского сообщества и (позднее) правительства США. Все это было лишь отчасти верным, однако сама мысль о группе правительственных чиновников, пытающихся протолкнуть неудачный в техническом отношении стандарт в глотки бедных исследователей и программистов, прокладывавших компьютерные сети в траншеях, не способствовала продвижению этой модели. Кое-кто рассматривал это развитие в том же свете, что и заявления корпорации IBM в 1960-м году о том, что PL/I будет языком будущего, или Министерства обороны США, поправлявшим позднее это утверждение своим заявлением, что в действительности таким языком будет Ada.

1.4.6. Критика эталонной модели TCP/IP

У модели TCP/IP и ее протоколов также имеется ряд недостатков. Во-первых, в этой модели нет четкого разграничения концепций служб, интерфейсов и протоколов. При разработке программного обеспечения желательно провести четкое разделение между спецификацией и реализацией, что весьма тщательно делает OSI и чего не делает TCP/IP. В результате модель TCP/IP довольно бесполезна при разработке сетей, использующих новые технологии.

Во-вторых, модель TCP/IP отнюдь не является общей и довольно плохо описывает любой стек протоколов, кроме TCP/IP. Так, например, описать технологию Bluetooth с помощью модели TCP/IP совершенно невозможно.

В-третьих, канальный уровень в действительности не является уровнем в том смысле, который обычно используется в контексте уровневых протоколов. Это скорее интерфейс между сетью и уровнями передачи данных. Различие между интерфейсом и уровнем является чрезвычайно важным, и здесь не следует быть небрежным.

В-четвертых, в модели TCP/IP не различаются физический уровень и уровень передачи данных. Об этом различии даже нет упоминания. Между тем, они абсолютно разные. Физический уровень должен иметь дело с характеристиками передачи информации по медному кабелю, оптическому волокну и по радио, тогда как задачей уровня передачи данных является определение начала и конца кадров и передача их с одной стороны на другую с требуемой степенью надежности. Правильная модель должна содержать их как два различных уровня. В модели TCP/IP этого нет.

И, наконец, хотя протоколы IP и TCP были тщательно продуманы и неплохо реализованы, многие другие протоколы были созданы несколькими студентами, работавшими над ними, пока это занятие им не наскучило. Реализации этих протоколов свободно распространялись, в результате чего они получили широкое признание, глубоко укоренились, и теперь их трудно заменить на что-либо другое. Некоторые из них в настоящее время оказались серьезным препятствием на пути прогресса. Например, протокол виртуального терминала TELNET, созданный еще для механического терминала типа Teletype, работавшего с огромной скоростью 10 символов в секунду. Ему ничего не известно о графических интерфейсах пользователя и о мышках. Тем не менее сейчас, 30 лет спустя, он все еще используется.

1.5. Примеры сетей

Компьютерные сети бывают очень разными: большими и маленькими, всемирно известными и почти никому не известными. Они преследуют в своей работе разные цели, имеют разные масштабы, используют разные технологии. В этом разделе мы рассмотрим несколько примеров, помогающих осознать, насколько многообразен мир сетей. Первым примером будет самая известная сеть сетей, Интернет. Вы узнаете, как она появилась, как эволюционировала и какие технологии при этом использовались. Затем мы обратимся к технологии мобильных сетей. Технически она довольно сильно отличается от Интернета. Затем мы представим IEEE 802.11, основной стандарт беспроводных локальных сетей. Наконец, последним примером в этом разделе будет RFID и сенсорные сети; это технологии, которые позволяют включить в сети физический мир и предметы быта.

1.5.1. Интернет

Для начала следует еще раз напомнить о том, что Интернет на самом деле не является сетью, это собирательное название разных сетей, использующих определенные общие протоколы и предоставляющие определенные сервисы. Это система необычна тем, что ее никто специально не планировал и не контролировал. Чтобы лучше понять, почему так получилось, мы начнем с самых истоков существования Интернета. В качестве прекрасного пособия по истории Интернета можно порекомендовать книгу, которую написал Джон Нотон (John Naughton) в 2000 году. Это редкое издание, потому что оно не только легко читается, но и содержит двадцатистраничный библиографический список параллельных мест и цитат, которые будут полезны людям, всерьез занимающимся историей. Часть материала, представленного в этом разделе, основывается именно на этой книге. Конечно, об Интернете, а также о его протоколах написаны бесчисленные технические книги. Для получения дополнительной информации см., например, Maufer (1999).

ARPANET

История глобальных сетей началась в конце пятидесятых годов. В самый разгар холодной войны Министерство обороны США пожелало иметь сеть, которая могла бы пережить даже ядерную войну. В то время все военные телекоммуникации базировались на общественной телефонной сети, которая была сочтена слишком уязвимой. Графически эта уязвимость демонстрируется на рис. 1.22, *а*. Здесь черными точками обозначены коммутационные станции, с каждой из которых были связаны тысячи абонентов. Эти коммутаторы, в свою очередь, являлись абонентами для станций более высокого уровня — междугородных. Междугородные станции формировали национальные сети. При этом степень резервной избыточности была минимальной. Уязвимость заключалась в том, что потеря всего одного ключевого коммутатора или междугородной станции разделила бы сеть на изолированные участки.

Для решения этой проблемы в 1960-х годах Министерство обороны США обратилось к корпорации RAND. Один из ее работников, Пол Бэрен (Paul Baran), разработал

проект высоконадежной распределенной сети (рис. 1.22, *б*). Поскольку по линиям такой большой длины тяжело было бы передать аналоговый сигнал с допустимым уровнем искажений, Бэрэн предложил передавать цифровые данные и использовать технологию коммутации пакетов. Им было написано несколько отчетов для Министерства обороны, в которых описывались подробности реализации его идей. Пентагону понравилась предложенная концепция, и компании AT&T (тогдашнему единственному в США монополисту в области телефонных сетей) было поручено разработать прототип. AT&T сразу же отклонила идеи Бэрена. Конечно, богатейшая и крупнейшая компания не могла позволить какому-то мальчишке указывать ей, как следует строить телефонные сети. Было заявлено, что бэреновскую сеть построить невозможно, и проект был на этом закрыт.

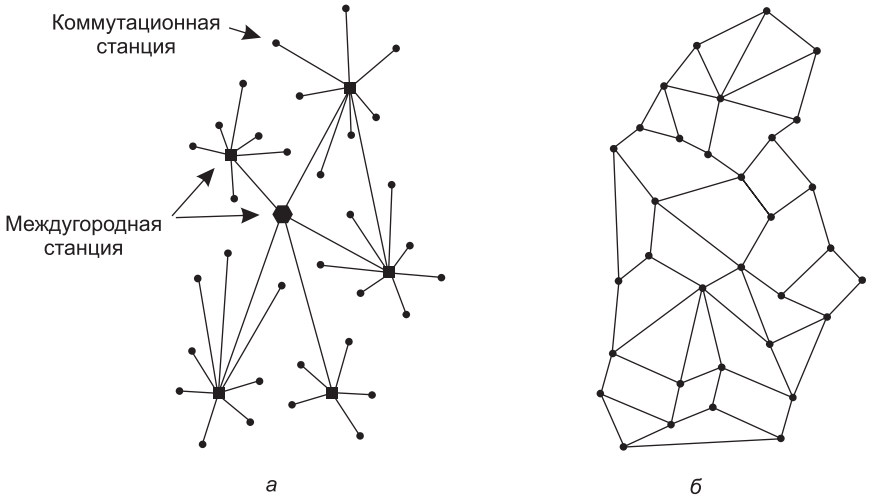


Рис. 1.22. Структура телефонной сети (*а*); предложенная Бэреном архитектура распределенной сети (*б*)

Прошло еще несколько лет, но Министерству обороны США так и не было предложено никакой замены существующей оперативной системе управления. Чтобы понять, как развивались события дальше, мы вспомним октябрь 1957 года, когда в СССР был запущен первый в мире искусственный спутник Земли, и тем самым основной соперник США получил преимущество в космосе. Тогда президент Эйзенхауэр задумался о том, кто же допустил такой прокол. И выяснилось, что армия, флот и ВВС США только зря проедают деньги, отпущенные Пентагоном на научные исследования. Было немедленно решено создать единую научную организацию под покровительством Министерства обороны, **ARPA (Advanced Research Projects Agency**, **Управление перспективного планирования научно-исследовательских работ)**. У ARPA не было ни ученых, ни лабораторий. У нее вообще практически ничего не было, за исключением небольшого офиса и скромного (по меркам Пентагона) бюджета. ARPA занималась тем, что выделяла из множества предлагаемых университетами и компаниями проектов наиболее перспективные и организовывала получение грантов под эти проекты и заключение контрактов с этими организациями.

В первые годы своего существования ARPA пыталась определиться с направлением своей деятельности. В 1967 году внимание Ларри Робертса, диспетчера программ в ARPA, который пытался выяснить, как обеспечить удаленный доступ к компьютерам, привлекли компьютерные сети. Он наладил контакты с различными экспертами, пытаясь понять, какие разработки могут представлять наибольший интерес для Министерства обороны. Один из экспертов, Весли Кларк (Wesley Clark), предложил построить подсеть с коммутацией пакетов, где каждый хост имел бы собственный маршрутизатор.

После преодоления собственного скептицизма Робертс все же решился приобрести эту идею и представил некий смутный отчет, касающийся этого, на симпозиуме ACM SIGOPS, посвященном принципам работы операционных систем. Симпозиум состоялся в Гетлинбурге, штат Теннесси, в конце 1967 года (Roberts, 1967). К большому удивлению Робертса он услышал доклад, в котором описывалась очень похожая система. Причем эта система была не только спроектирована, но и полностью реализована под руководством Дональда Дэвиса (Donald Davis) в Национальной физической лаборатории (NPL) Англии. Разработанная NPL сеть, конечно, не охватывала всю страну — она вообще лишь соединяла несколько компьютеров на территории организации, но ее реализация доказала, что пакетная коммутация может с успехом применяться на практике. Более того, то, что услышал Робертс, практически цитировало отвергнутую когда-то разработку Бэрена! Директор ARPA уехал из Гетлингбурга с твердым намерением создать в Америке то, что позднее будет названо **ARPANET**.

Подсеть должна была состоять из специализированных мини-компьютеров, называемых **IMP (Interface Message Processor)**, соединенных линиями связи, передающими информацию со скоростью 56 Кбит/с. Для повышения надежности каждый IMP должен был соединяться минимум с двумя другими IMP. Подсеть должна была быть дейтаграммной, чтобы, в случае если какие-либо линии и IMP будут разрушены, сообщения могли автоматически выбрать альтернативный путь.

Каждый узел сети должен был состоять из IMP и хоста, находящихся в одной комнате и соединенных коротким проводом. Хост мог пересылать своему IMP сообщения длиной до 8063 бит, которые IMP разбивал на пакеты, как правило, по 1008 бит, и пересылал их далее, независимо друг от друга, к пункту назначения. Пакет пересылался дальше только после того, как он был получен целиком, таким образом, это была первая электронная коммутирующая пакеты сеть с промежуточным хранением.

Затем агентство ARPA предложило тендер на строительство подсети. В тендере участвовало двенадцать компаний. Оценив предложения, агентство ARPA выбрало BBN, консалтинговую фирму в Кембридже, штат Массачусетс, и в декабре 1968 года подписало с ней контракт на постройку подсети и создание для нее программного обеспечения. BBN решило использовать специально модифицированные мини-компьютеры Honeywell DDP-316 с 12K 16-разрядных слов оперативной памяти в качестве IMP. У IMP не было дисков, поскольку движущиеся детали были сочтены ненадежными. Их соединили линиями с пропускной способностью по 56 Кбит/с, арендованными у телефонных компаний. Хотя в наше время 56 Кбит/с — это выбор подростков, которые еще не могут позволить себе DSL или прокладку кабеля, в 1968 году ничего более высокоскоростного просто не существовало.

Программное обеспечение было разбито на две части: для подсети и хостов. Подсетевое программное обеспечение состояло из части соединения хост-IMP со стороны IMP, протокола IMP-IMP и протокола между IMP-источником и IMP-приемником, разработанного для улучшения надежности. Оригинальная структура сети ARPANET показана на рис. 1.23.

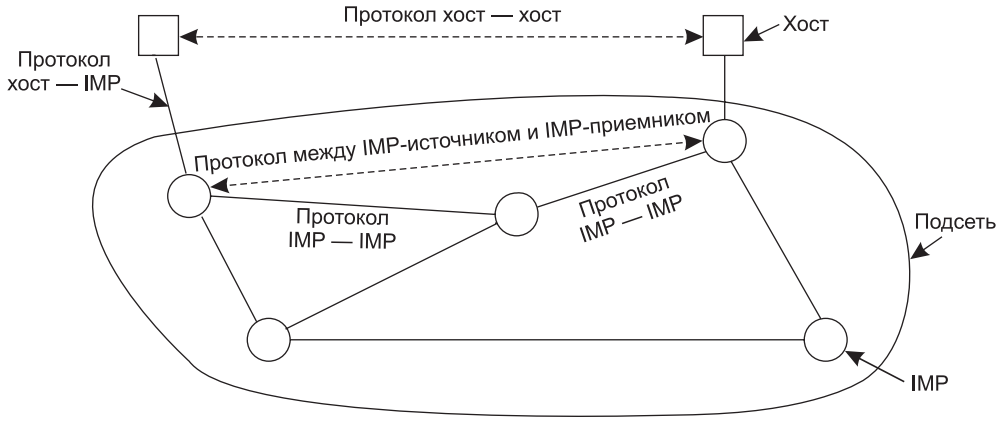


Рис. 1.23. Оригинальная структура сети ARPANET

Вне подсети также требовалось программное обеспечение, а именно соединение хост—IMP со стороны хоста, протокол хост—хост и прикладные программы. Как вскоре выяснилось, фирма BBN полагала, что ее задача ограничивается приемом сообщения на линии хост—IMP и передачей его на линию IMP—хост приемника.

Чтобы решить проблему программного обеспечения для хостов, Ларри Робертс летом 1969 года созвал совещание сетевых исследователей, большей частью аспирантов в городе Сноуберд (Snowbird), штат Юта. Аспиранты ожидали, что какой-нибудь эксперт в области сетей объяснит им устройство сети и его программное обеспечение, после чего распределит между ними работу. С изумлением они обнаружили, что не было ни специалиста по сетям, ни плана. Они должны были сами решать, что нужно сделать.

Тем не менее в декабре 1969 года удалось запустить экспериментальную сеть, состоящую из четырех узлов, расположенных в Калифорнийском университете в Лос-Анджелесе (UCLA), Калифорнийском университете в Санта-Барбаре (UCSB), Исследовательском институте Стэнфорда (SRI, Stanford Research Institute) и университете штата Юта. Были выбраны эти четыре университета, поскольку у них был большой опыт общения с агентством ARPA, кроме того, у всех имелись различные и совершенно несовместимые компьютеры-хосты (чтобы было веселее). Первое сообщение от узла к узлу было послано двумя месяцами ранее из узла UCLA командой во главе с Леном Клеинроком (пионер теории пакетной коммутации) к узлу SRI. Сеть быстро росла по мере создания и установки новых IMP. Вскоре она охватила все Соединенные Штаты. На рис. 1.24 показано, как быстро росла сеть ARPANET в первые три года.

Помимо помощи развивающейся сети ARPANET, агентство ARPA также финансировало исследовательские работы по спутниковым сетям и разработку мобильных пакетных радиосетей. На одной знаменитой демонстрации грузовик, который ездил

по Калифорнии, посылал сообщения по пакетной радиосети в SRI, которые затем передавались по ARPANET на Атлантическое побережье США и по спутниковой сети транслировались в University College в Лондоне. Таким образом, исследователь в грузовике мог работать с компьютером, находящимся в Лондоне.

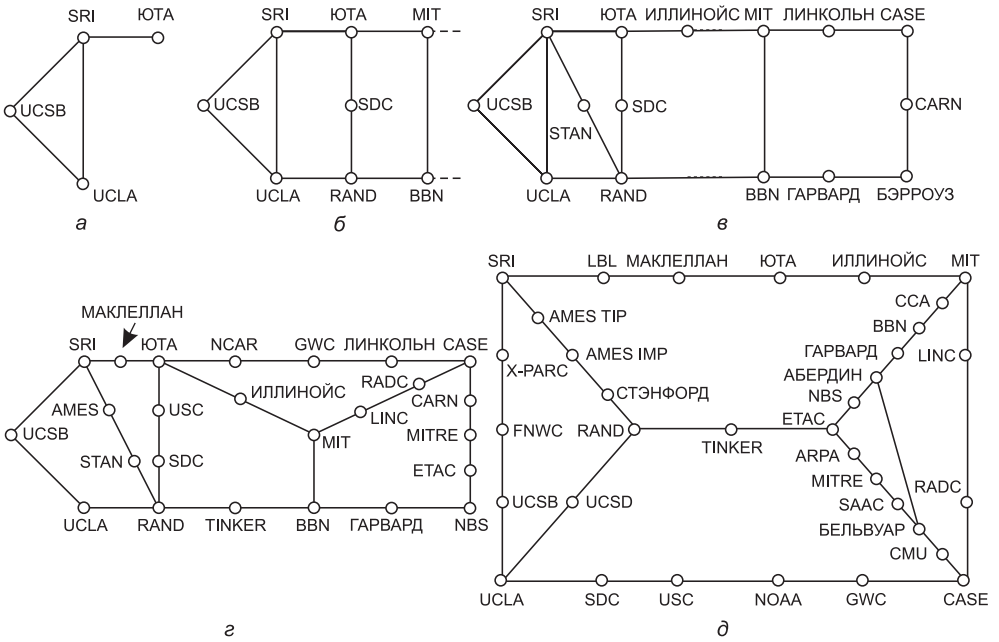


Рис. 1.24. Рост сети ARPANET: декабрь 1969 (*а*); июль 1970 (*б*); март 1971 (*в*); апрель 1972 (*г*); сентябрь 1972 (*д*)

При этой демонстрации также выяснилось, что имеющиеся протоколы сети ARPANET непригодны для работы с различными сетями. В результате были произведены дополнительные исследования в области протоколов, завершившиеся изобретением модели и протоколов TCP/IP (Cerf и Kahn, 1974). TCP/IP был специально разработан для управления обменом данными по интерсетям, что становилось все более и более важным по мере подключения все новых сетей к ARPANET.

Чтобы поощрить принятие новых протоколов, ARPA заключила несколько контрактов для внедрения TCP/IP на различных компьютерных платформах, в том числе на системах IBM, DEC и HP, а также UNIX Беркли. Исследователи в Калифорнийском университете в Беркли переписали TCP/IP с новым программным интерфейсом, названным **сокетом**, для следующего 4.2BSD выпуска UNIX Беркли. Они также написали много приложений, утилит и программ управления, чтобы показать, как удобно использовать сеть с сокетами.

Время было выбрано прекрасно. Многие университеты только что приобрели второй или третий компьютер VAX и LBC, чтобы их соединить, но у них не было сетевого программного обеспечения. С появлением системы UNIX 4.2 BSD, в которую вошли TCP/IP, сокеты и большое количество сетевых утилит, полный пакет был

принят немедленно. Кроме того, TCP/IP позволял легко соединить локальную сеть с ARPANET, что многие и делали.

В течение 80-х годов к ARPANET был подсоединен еще ряд сетей, в основном ЛВС. По мере роста размеров глобальной сети задача поиска хостов становилась все сложнее. В результате была создана система **DNS (Domain Name System — служба имен доменов)**, позволившая организовывать компьютеры в домены и преобразовывать имена хостов в IP-адреса. С тех пор DNS стала обобщенной распределенной системой баз данных, хранящей имена хостов и доменов. Мы рассмотрим ее более подробно в главе 7.

NSFNET

В конце 70-х годов Национальный научный фонд США (NSF, National Science Foundation) пришел к выводу, что сеть ARPANET оказывает огромное влияние на исследовательские работы университетов, позволяя ученым всей страны обмениваться информацией и совместно работать над проектами. Однако для получения доступа к ARPANET университет должен был заключить контракт с Министерством обороны, которого у многих университетов не было. Ответом NSF было основание в 1981 году сети Computer Science Network (CSNET). Она соединила кафедры информатики и индустриальные научно-исследовательские лаборатории с ARPANET через коммутируемый доступ и арендованные линии. В конце 1980-х NSF пошел далее и решил разработать преемника ARPANET, который будет открыт для всех университетских исследовательских групп.

Чтобы начать с чего-нибудь конкретного, Национальный научный фонд решил построить сетевую магистраль, соединив ею шесть суперкомпьютерных центров в Сан-Диего, Боулдере, Шампейне, Питтсбурге, Итаке и Принстоне. К каждому суперкомпьютеру был присоединен небольшой микрокомпьютер LSI-11, называемый **фаззбол** (fuzzball). Эти мини-компьютеры соединили выделенными линиями по 56 Кбит/с и сформировали подсеть по той же аппаратной технологии, которая использовалась в ARPANET. Однако программная технология была другой — мини-компьютеры с самого начала использовали протокол TCP/IP, образуя, таким образом, первую в мире глобальную сеть на основе протокола TCP/IP.

Национальный научный фонд также профинансировал создание нескольких (всего около 20) региональных локальных сетей, соединенных с магистралью, что позволило пользователям в тысячах университетов, исследовательских лабораторий, библиотек и музеев получить доступ к суперкомпьютерам. Вся сеть, состоящая из магистрали и региональных сетей, получила имя **NSFNET**. Она соединялась с ARPANET через линию между IMP и микрокомпьютером в компьютерном зале университета Карнеги-Меллона (Carnegie-Mellon University). Первоначальная магистраль сети NSFNET изображена на рис. 1.25.

Сеть NSFNET имела мгновенный успех, ей предсказывали большое будущее. Национальный научный фонд сразу же после завершения работы над NSFNET начал планировать следующую сеть и с этой целью подписал контракт с базирующимся в штате Мичиган консорциумом MERIT. Для создания второй версии магистрали сети у оператора междугородной связи MCI (Microwave Communications, Inc. — компания,

объединившаяся с тех пор с WorldCom) были арендованы волоконно-оптические каналы с пропускной способностью в 448 Кбит/с. В качестве маршрутизаторов использовались IBM PC-RT (RT-PC — RISC Technology Personal Computer — персональный компьютер на основе процессора с сокращенным набором команд). Вскоре и этого стало недостаточно, и вторая магистраль была ускорена до 1,5 Мбит/с.

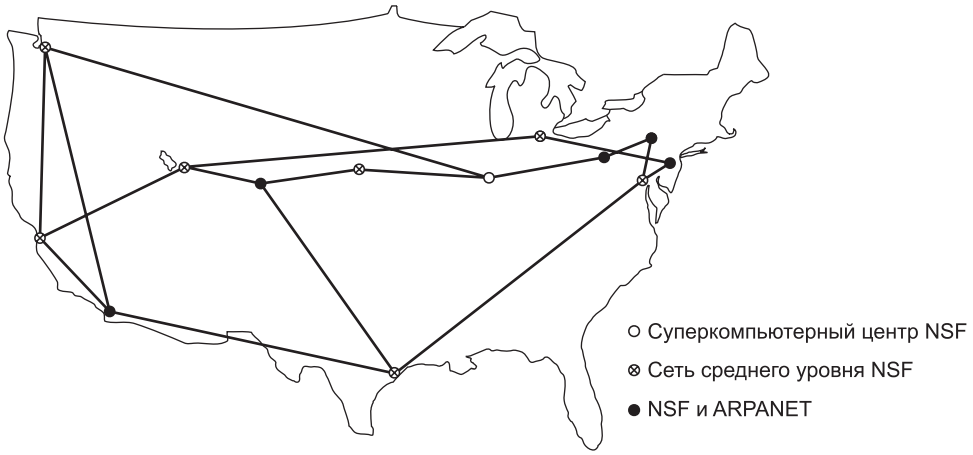


Рис. 1.25. Магистраль сети NSFNET в 1988 году

Рост отрасли продолжался, но Национальный научный фонд понимал, что правительство не сможет финансировать развитие сетей постоянно. Кроме того, коммерческие организации выражали желание поучаствовать в общем деле, но уставом фонда им было запрещено использовать сети, за которые заплатил Национальный научный фонд. Впоследствии Национальный научный фонд поддержал создание компаниями MERIT, MCI и IBM некоммерческой корпорации ANS (Advanced Networks and Services, Inc.) в качестве первого шага на пути коммерциализации. В 1990 году ANS вступила во владение сетью NSFNET и усовершенствовала линии со 1,5 Мбит/с до 45 Мбит/с, сформировав **ANSNET**. Эта сеть проработала пять лет, после чего была продана компании America Online. Но к тому времени уже появилось множество коммерческих фирм, предлагающих свои услуги в области IP-коммуникаций. Стало понятно, что государству не удастся выдержать конкуренцию с ними и оно должно уйти из этого бизнеса.

Для того чтобы облегчить переход с одних сетей на другие и гарантировать, что все региональные сети могут связаться друг с другом, Национальный научный фонд заключил контракт с четырьмя различными сетевыми операторами об организации пункта доступа к сети (NAP, Network Access Point). Этими операторами были компании PacBell (Сан-Франциско), Ameritech (Чикаго), MFS (Вашингтон) и Sprint (Нью-Йорк, с которым для удобства NAP были объединены Пеннсаукен и Нью-Джерси). Каждый сетевой оператор, который хотел предоставлять услуги по соединению региональных сетей NSF, должен был подключиться ко всем пунктам NAP.

Таким образом, пакет, пересылаемый с одной сети в другую, мог выбирать, по какому каналу перемещаться от одного пункта NAP до другого. Из-за этого операторы были вынуждены соперничать друг с другом по ценам и предоставляемым услугам,

как, собственно, и было задумано. Концепция единой магистрали была заменена коммерчески управляемой конкурентной инфраструктурой. Многие любят критиковать государственные структуры США за их консерватизм, а между тем, не кто иной, как Министерство обороны и государственный Национальный научный фонд создали все необходимые условия для развития Интернета, а затем передали свои закрытые разработки массовому пользователю.

В 90-х годах в других странах и регионах также были построены сети, сравнимые с NSFNET. Так, в Европе EuropaNET является IP-магистралью для исследовательских организаций, а EBONE представляет собой коммерчески-ориентированную сеть. Обе сети соединяют большое число европейских городов. Скорость каналов изначально составляла 2 Мбит/с, но впоследствии была увеличена до 34 Мбит/с. В конечном счете, сетевая инфраструктура в Европе, как и в США, превратилась в промышленную отрасль.

Интернет сильно изменился со времени своего возникновения. Он стремительно вырос в начале 1990-х годов с развитием World Wide Web (WWW). По данным Internet Systems Consortium количество видимых узлов превышает 600 миллионов. Это количество — только неточная оценка, но она сильно превышает те несколько миллионов узлов, которые были на момент первой конференции о WWW в 1994 году в CERN.

Радикально изменилось и то, как мы используем Интернет. Первоначально доминировали такие приложения, как электронная почта для академиков, группы новостей, удаленный вход в систему и передача файлов. Затем появилась электронная почта для всех, затем Web и одноранговые сети распространения контента, такие как закрытый теперь Napster. Теперь растет распространение СМИ в реальном времени, социальные сети (например, Facebook) и микроблогинг (например, Twitter). Эти изменения принесли более богатые виды медиа в Интернет и, следовательно, привели к росту трафика. Фактически, доминирующий трафик в Интернете, кажется, изменяется с некоторой регулярностью, так, например, новые и лучшие способы обработки музыки или видео могут стать популярными очень быстро.

Архитектура Интернета

Архитектура Интернета также сильно изменилась с его стремительным ростом. В этом разделе мы попытаемся дать краткий обзор того, на что она похожа сегодня. Картина осложнена непрерывными изменениями в фирмах телефонных компаний (telcos), кабельных компаний и интернет-провайдеров, из-за которых часто трудно сказать, кто что делает. Один из двигателей этих изменений — телекоммуникационная конвергенция, когда сеть начинает использоваться в новом качестве. Например, в «тройной игре» одна компания продает вам телефонию, телевидение и интернет-сервис по тому же самому сетевому соединению, поскольку это сэкономит вам деньги. Следовательно, описание, данное здесь, будет по необходимости несколько более простым, чем действительность. И то, что является истиной сегодня, возможно, не будет истиной завтра.

Общая картина показана на рис. 1.26. Давайте исследуем ее по частям, начиная с компьютера дома (на краях рисунка). Чтобы присоединиться к Интернету, компьютер соединяется с интернет-провайдером (Internet Service Provider, ISP), у которого пользователь покупает доступ к Интернету или связь. Это позволяет компьютеру

обмениваться пакетами со всеми другими доступными узлами в Интернете. Пользователь может посылать пакеты, чтобы перемещаться по сети или для любой из тысяч других возможностей ее использования, это не имеет значения. Есть много видов доступа к Интернету и их обычно отличают тем, сколько пропускной способности они обеспечивают и сколько они стоят, но самый важный признак — связь.

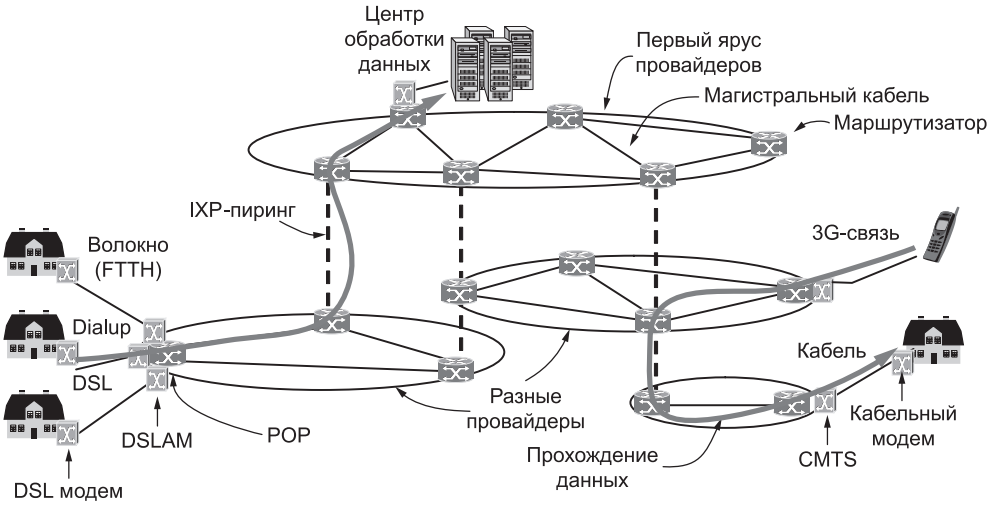


Рис. 1.26. Краткий обзор архитектуры

Распространенный способ соединиться с провайдером состоит в том, чтобы использовать телефонную линию, ведущую к вашему дому в этом случае вашим провайдером является ваша телефонная компания. DSL (сокращение Digital Subscriber Line) использует телефонную линию, которая соединяется с вашим домом для цифровой передачи данных. Компьютер соединен с устройством, названным модемом DSL, которое осуществляет преобразование между цифровыми пакетами и аналоговыми сигналами, которые могут идти по телефонной линии. С другой стороны, устройство, называемое DSLAM (Цифровой мультиплексор доступа линии подписчика, Digital Subscriber Line Access Multiplexer), осуществляет преобразование между сигналами и пакетами.

Несколько других популярных способов соединиться с провайдером показаны на рис. 1.26. DSL имеет более высокую пропускную способность при использовании местной телефонной линии, чем пересылка битов по традиционному телефонному звонку вместо голосового разговора. Это называется коммутируемым доступом и осуществляется с различными видами модемов на обоих концах. Модем — сокращение для слов «модулятор демодулятор», так называют любое устройство, которое осуществляет преобразование между цифровыми битами и аналоговыми сигналами.

Другой метод — передавать сигналы по системе кабельного телевидения. Как и DSL, это способ использовать существующую инфраструктуру, в этом случае неиспользованные каналы кабельного телевидения. Устройство в домашнем конце называют кабельным модемом, а устройство в головном узле кабеля называют **CMTS (Cable Modem Termination System — система завершения кабельного модема)**.

DSL и кабель обеспечивают доступ к Интернету на скоростях от небольшой доли мегабит в секунду до многих мегабит в секунду, в зависимости от системы. Эти скорости намного больше, чем коммутируемые скорости, которые ограничены 56 Кбит/с из-за узкой пропускной способности, используемой для голосовых вызовов. Доступ к Интернету с намного большими, чем коммутируемые, скоростями называют широкополосным. Название означает более широкую пропускную способность, которая используется для более быстрых сетей, а не для какой-то конкретной скорости.

Методы доступа, упоминаемые до сих пор, ограничиваются пропускной способностью «последней мили» или последнего этапа передачи. При проведении оптоволокна к местам жительства может быть обеспечен более быстрый доступ к Интернету, на скоростях порядка 10–100 Мбит/с. Этот проект называют FTTH (Волокно в дом). Для фирм в коммерческих областях может иметь смысл арендовать высокоскоростную линию передачи от офисов до самого близкого провайдера. Например, в Северной Америке линии T3 работают на скоростях около 45 Мбит/с.

Для доступа к Интернету также используется беспроводная связь. Примером, который мы исследуем коротко, является пример сетей мобильной связи третьего поколения. Они могут обеспечить доставку данных на скорости 1 Мбит/с или выше к мобильным телефонам и неподвижным абонентам в зоне охвата.

Теперь мы можем перемещать пакеты между домом и провайдером. Мы называем местоположение, в котором потребительские пакеты входят в сеть провайдера, POP (Point of Presence, Точка присутствия). Далее мы объясним, как пакеты перемещаются между точками присутствия различных провайдеров. С этого момента система является полностью цифровой и использует коммутацию пакетов.

Сети могут быть региональными, национальными или международного масштаба. Мы уже видели, что их архитектура составлена из дальних линий передачи, которые связывают маршрутизаторы в точках присутствия в различных городах, где действуют провайдеры. Это оборудование называют магистралью ISP. Если пакет предназначен для узла, обслуживаемого непосредственно ISP, этот пакет будет направлен по магистрали и поставлен узлу. Иначе он должен быть передан другому ISP.

Провайдеры соединяют свои сети, чтобы обмениваться трафиком в IXP (Internet eXchange Points, Точки обмена интернет-трафиком). Соединенные провайдеры, как говорят, видят друг друга. Множество провайдеров в городах во всем мире нарисованы вертикально на рис. 1.26, потому что сети накладываются географически. Обычно IXP — это комната, полная маршрутизаторов, по крайней мере по одному на каждого провайдера. ЛВС в комнате соединяет все маршрутизаторы, таким образом, пакеты могут быть отправлены от любой магистрали провайдера до любой другой. Точки обмена интернет-трафиком могут быть большими и находящимися в независимой собственности. Одна из самых больших — Amsterdam Internet Exchange, с которым соединяются сотни провайдеров, где они обмениваются сотнями гигабит в секунду трафика.

Равноправный информационный обмен, который происходит в точках обмена интернет-трафиком, зависит от деловых отношений между провайдерами. Есть много возможных отношений. Например, маленький провайдер мог бы заплатить большому провайдеру за интернет-связь, чтобы достигнуть отдаленных узлов; очень похоже на то, как клиент покупает услугу у интернет-провайдера. В этом случае маленький провайдер, как говорят, платит за транзит. Альтернативно, два больших провайдера

могли бы обмениваться трафиком так, чтобы каждый из них мог поставить некоторый трафик другому, не имея необходимости платить за транзит.

Один из парадоксов интернет-технологий заключается в том, что зачастую провайдеры, которые открыто конкурируют друг с другом в борьбе за клиентов, в то же самое время организуют частную равноранговую связь между собой (Metz, 2001).

Путь, по которому пакет перемещается по Интернету, зависит от выбора связи между провайдерами. Если провайдер, отправляющий пакет, связан с местом назначения, он может доставить пакет непосредственно. Иначе он может направить пакет к самому близкому месту, в котором есть соединение с платным провайдером транзита так, чтобы провайдер мог отправить пакет. На рисунке 1.26 показаны два примера пути через провайдеров. Часто путь пакета через Интернет не будет кратчайшим путем.

Наверху «пищевой цепи» находится маленькая горстка компаний, таких как AT&T и Sprint, которые управляют большими международными базовыми сетями с тысячами маршрутизаторов, соединенных линиями оптоволокна высокой пропускной способности. Эти провайдеры не платят за транзит. Их обычно называют первым ярусом провайдеров, и они, как говорят, формируют магистраль Интернета, так как все остальные должны соединиться с ними, чтобы быть в состоянии достигнуть всего Интернета.

Компании, которые обеспечивают много контента, такие как Google и Yahoo!, располагают свои компьютеры в информационных центрах, которые хорошо соединены с остальной частью Интернета. Эти информационные центры разработаны для компьютеров, а не для людей, и могут быть наполнены стойками машин, называемых серверной фермой. Расположение клиентов информационных центров хостинга, которым позволяют поместить оборудование, такое как серверы в точках присутствия провайдеров, таково, чтобы короткие, быстрые соединения могли быть сделаны между серверами и магистралями провайдера. Индустрия интернет-хостинга становится все более и более виртуальной, так что теперь, вместо того чтобы установить физический компьютер, распространена возможность арендовать виртуальную машину, которая работает на серверной ферме. Эти информационные центры являются настолько крупными (десятки или сотни тысяч машин), что основными их затратами является электричество, поэтому информационные центры иногда создаются в областях, где электричество дешево.

На этом мы закончим наш краткий обзор архитектуры Интернета. Впереди еще много разделов, посвященных изучению отдельных компонентов этого вопроса: проектирования, алгоритмов, протоколов. Еще один момент, стоящий упоминания, — изменение понятия «быть в Интернете». Мы привыкли, что машина находится в Интернете, если: (1) выполняется стек протокола TCP/IP; (2) у нее имеется IP-адрес; и (3) она может пересылать IP-пакеты другим машинам в Интернете. Однако провайдеры часто повторно используют IP-адреса, в зависимости от того, какие компьютеры работают в настоящее время, и домашние сети часто совместно задействуют один IP-адрес для нескольких компьютеров. Эта практика подрывает второе условие. Меры по безопасности, такие как брандмауэры, могут также частично заблокировать компьютеры от получения пакетов, подрывая третье условие. Несмотря на эти трудности, имеет смысл расценивать такие машины, как находящиеся в Интернете, в то время как они соединены со своими провайдерами.

Хочется напоследок отметить еще один нюанс: некоторые компании, объединяющие свои внутренние сети, часто используют те же технологии, которые используются в глобальной сети Интернет. Доступ к данным этих **интрасетей** обычно ограничивается пределами компании или ноутбуками, принадлежащими компании, но во всем остальном это тот же самый Интернет, только в миниатюре.

1.5.2. Мобильная телефонная сеть третьего поколения

Люди любят говорить по телефону даже больше, чем путешествовать на просторах Интернета, и это сделало мобильную телефонную сеть самой успешной сетью в мире. Количество абонентов уже превысило четыре миллиарда — это примерно 60 % населения Земли и больше чем количество интернет-узлов и стационарных телефонных линий (ITU, 2009). За последние 40 лет сеть мобильной связи очень разрослась, а ее архитектура сильно изменилась. Системы первого поколения передавали голосовые вызовы в виде непрерывных (аналоговых) сигналов, а не как последовательность битов. Система AMPS (Advanced Mobile Phone System), развернутая в Соединенных Штатах в 1982 году, была самой популярной системой первого поколения. Системы второго поколения перешли на передачу голосовых вызовов в цифровом виде, что увеличило пропускную способность, повысило безопасность и позволило осуществлять обмен текстовыми сообщениями. GSM (Глобальная система мобильной связи) была развернута с 1991 года и стала наиболее широко используемой системой мобильной телефонной связи в мире, относится к системам 2-го поколения.

Третье поколение, или 3G-системы, были развернуты в 2001 году. Они предлагают как цифровую передачу голоса, так и широкополосную цифровую передачу данных. В 3G-системах используется множество различных стандартов. ITU (который мы обсудим в следующем разделе) утверждает, что 3G должен обеспечивать скорость передачи данных не менее 2 Мбит/с для неподвижных или идущих пользователей и 384 Кбит/с при перемещении в транспортном средстве. UMTS (Universal Mobile Telecommunications System), также названный WCDMA (Wideband Code Division Multiple Access), является основной 3G-системой, которая быстро развертывается во всем мире. Она может обеспечить до 14 Мбит/с для входящей и почти 6 Мбит/с для исходящей информации. Следующие версии 3G-системы будут использовать комплексы антенн и передатчиков, чтобы предоставить пользователям еще большие скорости.

Наиболее критичный ресурс в системах 3G (как и в более ранних 2G- и 1G-системах) — ограниченность полосы радиочастот. Правительства предоставляют операторам мобильных сетей права на использование частотных диапазонов, для этого могут проводиться аукционы, на которых заинтересованные лица делают ставки. Лицензирование частотного диапазона облегчает разработку и управление системами, так как никто больше не может получить разрешение передавать на этих частотах, но обходится достаточно дорого. В Великобритании в 2000 году, например, пять лицензий 3G ушли с аукциона за сумму около $40 млрд.

Дефицит частотного диапазона привел к появлению схемы сотовой связи, показанной на рис. 1.27, которая используется сейчас в мобильных сетях. Чтобы снизить

вероятность возникновения радиопомех между пользователями, зона охвата разделена на ячейки сот. В пределах ячейки пользователям назначаются каналы, не затрагивающие друг друга и не вызывающие проблем для смежных ячеек. Таким образом обеспечивается эффективное использование спектра и повторное использование частотных диапазонов в соседних ячейках, что увеличивает пропускную способность сети. В системах первого поколения, которые передавали каждое голосовое сообщение в определенном диапазоне частот, приходилось тщательно выбирать частоты, чтобы не возникало конфликтов передачи данных в соседних ячейках. Одна частота могла использоваться в группе соседних ячеек только однократно. Современные 3G-системы позволяют каждой ячейке использовать полный диапазон доступных частот, обеспечивая при этом удовлетворительную работу соседних ячеек. Существует множество способов построения отдельных ячеек, например с использованием направленных и секторных антенн, позволяющих уменьшить взаимное влияние ячеек, но основная идея — одна и та же.

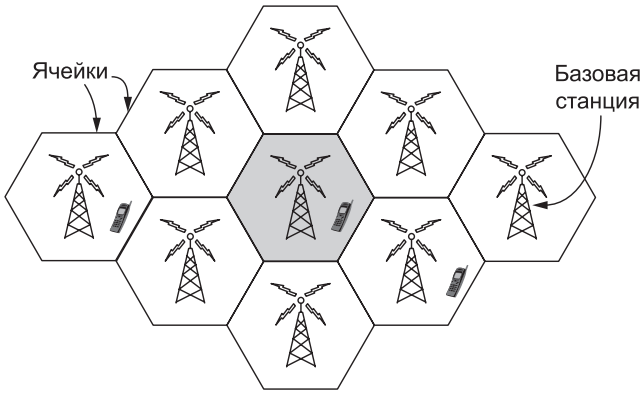


Рис. 1.27. Схема сотовой сети мобильных телефонов

Архитектура мобильной сети сильно отличается от Интернета. Мобильную сеть можно разделить на несколько блоков, как показано в упрощенной схеме архитектуры UMTS (рис. 1.28). Первый блок — радиоинтерфейс — название протокола радиосвязи, который используется при беспроводной передаче данных между мобильным устройством (например, сотовый телефон) и сотовой базовой станцией. Совершенствование радиоинтерфейса за прошлые десятилетия очень увеличило скорость беспроводной передачи данных. Радиоинтерфейс UMTS основан на множественном доступе с кодовым разделением каналов (CDMA, Code Division Multiple Access), метод, который мы изучим в главе 2.

Сотовая базовая станция вместе с контроллером формирует сеть с радиодоступом. Этот фрагмент схемы — беспроводная сторона сети мобильных телефонов. Узел контроллера или RNC (Radio Network Controller, контроллер радиосети) управляет использованием спектра. Базовая станция обеспечивает радиоинтерфейс (так называемый «узел B»).

Остальная часть сети передает трафик в сеть с радиодоступом. Ее называют базовой сетью (core network). Базовая сеть UMTS происходит от базовой сети GSM,

используемой в 2G-системах предыдущего поколения. Однако в базовой сети UMTS есть много интересных нюансов.

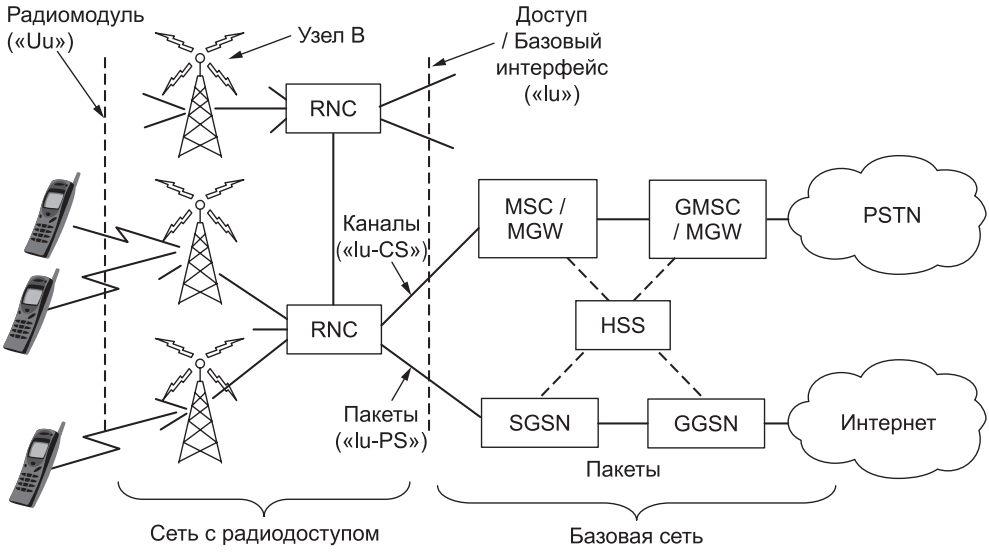


Рис. 1.28. Архитектура сети мобильной связи 3G UMTS

С начала существования сетей продолжается война между поклонниками пакетных сетей (то есть подсетей без установления соединения) и теми, кто поддерживает сети с коммутацией каналов (то есть подсетей, ориентированных на соединения). Основные сторонники пакетов происходят из интернет-сообщества. В схеме без установления соединений каждый пакет двигается независимо от остальных. Как следствие, если во время сеанса некоторые маршрутизаторы выйдут из строя, то не возникнет никаких проблем, пока система может динамически реконфигурировать себя, позволяя последующим пакетам найти маршрут к месту назначения, даже если он отличается от того, который использовали предыдущие пакеты.

Сторонники сетей с коммутацией каналов принадлежат миру телефонных компаний. В телефонной сети звонящий должен набрать номер абонента и ждать, пока установится соединение, чтобы получить возможность говорить или передавать данные. Данное соединение устанавливает маршрут движения по телефонной сети, который остается стабильным, пока звонок не закончен. Все слова или пакеты следуют одним и тем же маршрутом. Если линия или коммутатор отключаются, вызов обрывается. Такой вариант менее устойчив к ошибкам, чем сети без установления соединения.

Преимущество сетей с коммутацией каналов состоит в том, что в этом случае легче поддерживать качество обслуживания. Настраивая соединение заранее, подсеть может зарезервировать ресурсы, например частотную полосу канала, буфер коммутатора, и загрузку центрального процессора. Если предпринята попытка настроить вызов, а ресурсов недостаточно, вызов отклоняется и вызывающий получает своего рода сигнал «занято». Таким образом, как только соединение оказывается настроено, можно гарантировать высокое качество обслуживания.

Если в сети без установления соединения на один из маршрутизаторов поступает слишком много пакетов одновременно, произойдет сбой и маршрутизатор, вероятно, потеряет часть информации. Отправитель, в конечном счете, заметит это и снова их отправит. Такого уровня обслуживания будет недостаточно для работы с аудио или видео, особенно если сеть окажется загруженной. Очевидно, что обеспечение соответствующего качества звука волнует телефонные компании более, чем проблема соединения.

Особенность схемы, представленной на рис. 1.28, состоит в том, что в базовой сети используются и пакетная и канальная маршрутизация. Это демонстрирует промежуточное положение сетей мобильной связи, когда оба подхода реализуются как по отдельности, так и одновременно. Более старые сети использовали ядро с канальной маршрутизацией в стиле традиционной телефонной сети, чтобы передавать голосовые сообщения. Это же «наследие» можно заметить в сетях UMTS с элементами MSC (Mobile Switching Center), GMSC (Gateway Mobile Switching Center) и MGW (Media Gateway), которые настраивают сети с коммутацией каналов, такие как PSTN (Public Switched Telephone Network).

Цифровая обработка стала намного более важной частью мобильных сетей, начиная с возникновения обмена текстовыми сообщениями и появления ранних пакетных информационных протоколов, таких как GPRS (General Packet Radio Service) в системе GSM. Эти более старые информационные протоколы работали на скоростях в десятки килобит в секунду, но пользователи хотели больше. Скорость передачи пакетных данных в более новых сетях меряется уже в мегабитах в секунду. Голосовой звонок передается со скоростью 64 Кбит/с, а со сжатием — в 3–4 раза быстрее.

Чтобы передать все эти данные, основные сетевые узлы UMTS соединяются непосредственно с пакетно-коммутируемой сетью. SGSN (Serving GPRS Support Node) и GGSN (Gateway GPRS Support Node) доставляют и получают пакеты данных от мобильных телефонов и связывают их с внешними сетями с пакетной коммутацией, такими как Интернет.

Этот переход продолжится в сетях мобильных телефонов, которые разрабатываются и запускаются в настоящее время. Используются даже специальные интернет-протоколы для мобильной телефонии, чтобы настроить соединения для голосовых сообщений по сетям с пакетной коммутацией, например IP-телефония. IP и пакеты используются на всех уровнях — от радиодоступа до базовой сети. Конечно, методика, по которой создаются IP-сети, также изменяется, чтобы обеспечивать лучшее качество обслуживания. Если бы так не происходило, то проблемы с прерванным аудио- или видеопотоком заставили бы платежеспособных клиентов усомниться в данной технологии. Мы еще вернемся к этой теме в главе 5.

Другое различие между мобильными сетями и традиционным Интернетом — движение. Когда пользователь перемещается из района действия одной базовой станции в другой район, поток данных должен перенаправляться между базовыми станциями. Этот метод известен как переадресация вызова и проиллюстрирован на рис. 1.29.

Мобильное устройство или базовая станция могут запросить смену базовой станции, когда качество сигнала понижается. В некоторых сетях, обычно основанных на CDMA-технологии, можно соединиться с новой базовой станцией прежде, чем будет разорвано соединение с предыдущей базовой станцией. Это повышает качество со-

единения, поскольку не происходит никакого перерыва в обслуживании. В течение непродолжительного времени мобильный телефон соединяется с двумя базовыми станциями одновременно. Этот способ передачи данных называют мягкой передачей, в отличие от жесткой передачи, при которой мобильный телефон разъединяется со старой базовой станцией прежде, чем успевает соединиться с новой.

Рис. 1.29. Передача мобильного телефона: *а* — до, *б* — после

В связи с этим возникает дополнительная проблема — необходимость найти мобильный телефон в первый момент при входящем вызове. У каждой мобильной сети есть HSS (Home Subscriber Server) в базовой сети, которая знает местоположение каждого абонента, а также другую информацию о профиле, использующуюся для аутентификации и авторизации. Таким образом, каждый мобильный телефон может быть обнаружен с помощью HSS.

Последнее, что осталось обсудить, это вопросы безопасности. Исторически телефонные компании относились к вопросам безопасности более серьезно, чем интернет-компании. Это было связано с необходимостью своевременного выставления счетов за выполненные услуги и с задачей избежать мошенничества при оплате. К сожалению, это не все. Однако в развитии технологий от 1G до 3G компании мобильной связи смогли выработать некоторые основные механизмы безопасности.

Начиная с 2G систем, мобильный телефон состоит из двух частей — собственно телефонного аппарата и сменного чипа, который содержит идентификатор подписчика и информацию об учетной записи. Чип неофициально называют SIM-картой (сокращение от Subscriber Identity Module). SIM-карты могут вставляться в различные телефонные трубки, и они обеспечивают базис безопасности. Когда клиенты GSM путешествуют, отправляются в другие страны или уезжают в командировку, они часто берут с собой свой телефон, но по прибытии на место покупают за несколько долларов новую SIM-карту, чтобы делать местные звонки без платы за роуминг.

Чтобы сократить мошенничество, информация о SIM-картах также используется сетью, позволяет подтвердить подлинность абонентов и проверить, что они имеют право использовать сеть. В UMTS мобильный телефон также использует информацию о SIM-карте, чтобы проверить, что он законно использует сеть.

Другой аспект безопасности — частная жизнь. Беспроводные сигналы передаются всем получателям одновременно, поэтому, чтобы помешать подслушивать переговоры, SIM-карты используют ключи шифрования, позволяющие зашифровать информацию. Этот подход обеспечивает намного лучшую защиту частной жизни, чем это было в 1G-системах, которые легко прослушивались, но не является панацеей из-за дыр в схемах шифрования.

Мобильные сети разработаны так, чтобы играть центральную роль в будущих сетях. Теперь они в большей степени занимаются мобильными широкополосными приложениями, чем голосовыми сообщениями. Наиболее важными следствиями этого стало появление радиоинтерфейсов, изменение архитектуры базовых сетей и обеспечение безопасности будущих сетей. 4G-технологии, которые быстрее и лучше, пока находятся в разработке под названием LTE (Long Term Evolution), а развитие 3G продолжается. Существуют и другие беспроводные технологии, которые также предлагают широкополосный доступ к Интернету стационарным и мобильным клиентам, например 802.16 сети под общим названием WiMAX. Технологии LTE и WiMAX можно назвать конкурирующими и трудно предсказать, что произойдет с ними в будущем.

1.5.3. Беспроводные ЛВС: 802.11

Почти одновременно с появлением ноутбуков у людей появились мысли о том, что неплохо было бы иметь возможность, например, по пути на работу каким-нибудь волшебным образом выйти в Интернет и почитать последние новости. Разумеется, многие компании стали заниматься разработкой соответствующих аппаратных решений. Вскоре был найден очень практичный подход. Он состоял в том, чтобы оборудовать ноутбук и настольный компьютер, находящийся в офисе, радиопередатчиком небольшого радиуса действия, что позволило бы им связываться между собой.

Вскоре на рынке появились первые беспроводные локальные сети, созданные разными производителями. Проблема была в том, что сети разных фирм оказались совершенно несовместимы между собой. Например, компьютер, оборудованный передатчиком фирмы А, был не способен работать в помещении, в котором находилась базовая станция фирмы Б. В середине 1990-х годов было решено привести все беспроводные ЛВС к единому стандарту. Разобраться во всем многообразии существующих технологий и выработать единую концепцию было поручено институту IEEE, который уже имел опыт стандартизации обычных ЛВС.

Первое решение было самым легким: название. У всех других стандартов ЛВС были номера 802.1, 802.2, 802.3 и так до 802.10, таким образом, беспроводный стандарт ЛВС был назван 802.11.

На профессиональном жаргоне стандарт получил название **WiFi**. Но этот стандарт важен и заслуживает уважения, поэтому мы будем называть его как положено — 802.11.

Дальнейшее было труднее. Первая проблема — найти подходящий диапазон частот, который был бы доступен, желательно во всем мире. Решение сильно отличалось от варианта, используемого в мобильных сетях. Отказавшись от необходимости покупать дорогую лицензию на конкретный диапазон частот, 802.11 работает на частотах ISM-организаций (для некоммерческого использования в промышленности, научных и медицинских организациях), например, 902–928 МГц, 2.4–2.5 ГГц, 5.725–5.825 ГГц. Всем устройствам разрешается использовать эти частоты при условии, что они ограничивают свою мощность передачи, чтобы не создавать помех в работе других устройств. Конечно, это означает, что 802.11-передатчики могут помешать работе домашних беспроводных телефонов, устройств, открывающих двери гаражей, и микроволновых печей.

802.11-сети образуются ноутбуками, мобильными телефонами и AP-инфраструктурами (AP, access point — точка доступа), которые располагаются в зданиях. Точки доступа иногда называют базовыми станциями. Точки доступа соединяются с проводной сетью, и вся связь между клиентами сети проходит через точку доступа. Кроме того, клиенты сети могут общаться друг с другом напрямую, например пара офисных компьютеров может обмениваться информацией без точки доступа в здании. Такая конфигурация называется беспроводной локальной сетью (ad hoc network). Она используется намного менее часто, чем режим с точкой доступа. Оба режима показаны на рис. 1.30.

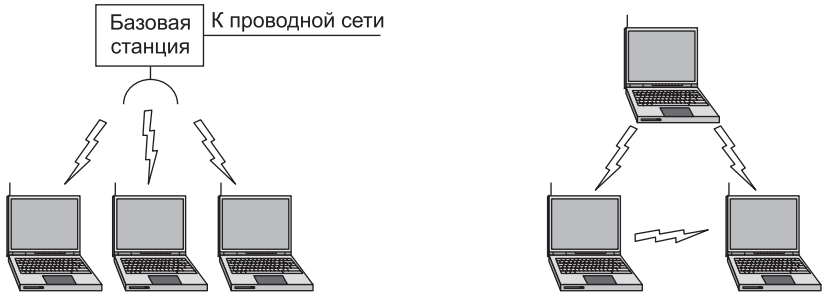


Рис. 1.30. Беспроводная сеть с точкой доступа (*а*); специальная сеть (*б*)

Передача сигналов в 802.11 осуществляется через воздушное пространство, а условия сильно зависят от окружающей среды. На частотах, используемых 802.11, радиосигналы отражаются от твердых объектов, таким образом, получатель может регистрировать не только основной сигнал, но и многократное эхо с нескольких направлений одновременно. Переотраженные сигналы могут заглушать или усиливать друг друга, приводя к колебанию уровня полученного сигнала. Это так называемые **замирания вследствие многолучевого распространения**. Данный эффект продемонстрирован на рис. 1.31.

Основной способ, позволяющий преодолеть меняющиеся условия беспроводной передачи, — передача информации несколькими независимыми путями. Таким образом, данные, вероятно, будут получены, даже если один из путей окажется заглушен из-за эффекта замирания. Эти независимые пути, как правило, встраиваются в цифровую схему модуляции на физическом уровне. Вариантов много — использование нескольких частот в пределах разрешенной полосы, варьирование путей передачи между разными парами антенн или повторение битов через некоторые промежутки времени.

Различные версии 802.11 использовали все перечисленные методы. Согласно начальному (1997) стандарту беспроводная ЛВС работала на скорости 1 или 2 Мбит/с, скачкообразно переключая частоты или размазывая сигнал по разрешенному частотному диапазону. Почти сразу люди стали жаловаться, что это слишком медленно, и началась разработка более быстрого стандарта. Вариант с широкополосными сигналами стал стандартом в 1999 году, 802.11b работал со скоростью до 11 Мбит/с. Стандарты 802.11a (1999) и 802.11g (2003) стали использовать другую схему модуля-

ции — **OFDM (Orthogonal Frequency Division Multiplexing — мультиплексирование с ортогональным частотным разделением сигналов)**. Данная схема делит широкую полосу спектра на множество узких фрагментов, по которым параллельно передаются различные биты. Улучшенная схема, которую мы изучим в главе 2, повысила скорость передачи 802.11a/g до 54 Мбит/с. Это — существенное увеличение, но люди все еще хотели, чтобы поддерживалась бóльшая пропускная способность. Последняя версия — 802.11n (2009) — использует более широкие частотные диапазоны и до четырех антенн на компьютер, что позволяет достигнуть скоростей около 450 Мбит/с.

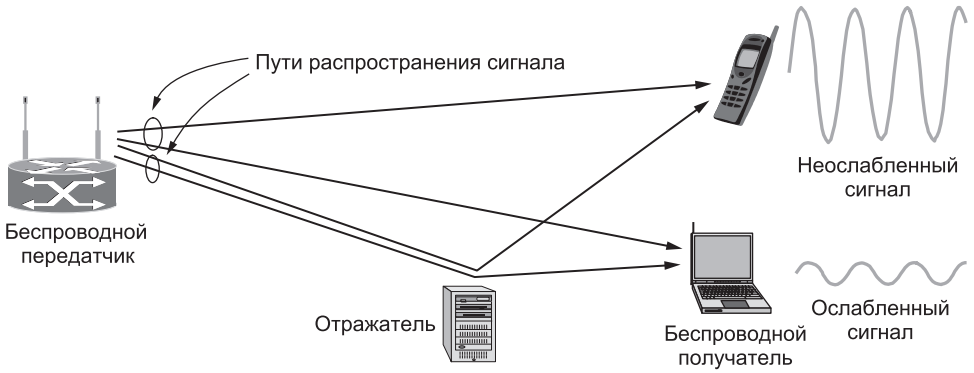


Рис. 1.31. Замирания вследствие многолучевого распространения

Так как беспроводная передача принципиально является широковещательной, 802.11-передатчики сталкиваются с проблемой одновременной передачи множества сигналов, которые интерферируют друг с другом, что может оказывать влияние на прием. Чтобы решить эту проблему, в 802.11 задействована схема CSMA (Carrier Sense Multiple Access), использующая идеи классического проводного Ethernet, которые были взяты из еще более ранней беспроводной сети — ALOHA, созданной на Гавайях. Компьютеры находятся в режиме ожидания передачи в течение короткого случайного интервала времени и задерживают сигнал, если какое-либо устройство передает информацию. Эта схема снижает вероятность ситуации, когда два компьютера отправят сигналы в одно и то же время. Но сама схема работы не такая, как в проводных сетях. Чтобы увидеть различия, изучите рис. 1.32. Предположим, что компьютер A передает сигнал компьютеру B, но уровня передатчика A недостаточно, чтобы сигнал достиг компьютера C. Допустим C собирается передать информацию компьютеру B, и тот факт, что он не «слышит» информации, передаваемой A, не означает, что его сигнал будет принят. Такая «глухота» C становится причиной некоторых проблем. После любого конфликта отправитель делает паузу на более длительный случайный интервал времени и повторно передает пакет. Несмотря на такие трудности и некоторые другие проблемы, на практике схема работает достаточно хорошо.

Еще одна проблема — движение. Если перемещающийся клиент отходит от используемой точки доступа и попадает в зону действия другой точки, возникает проблема, которую необходимо преодолеть. 802.11-сеть может объединять множество ячеек,

каждая из которых имеет собственную точку доступа, а управляющая система объединяет ячейки. Такая система часто подключается к Ethernet, но может использовать любые другие технологии. Когда клиенты перемещаются, они могут попадать в зону действия точек доступа с лучшим уровнем сигнала и переключаться на них. Извне такая система похожа на обычную проводную локальную сеть.

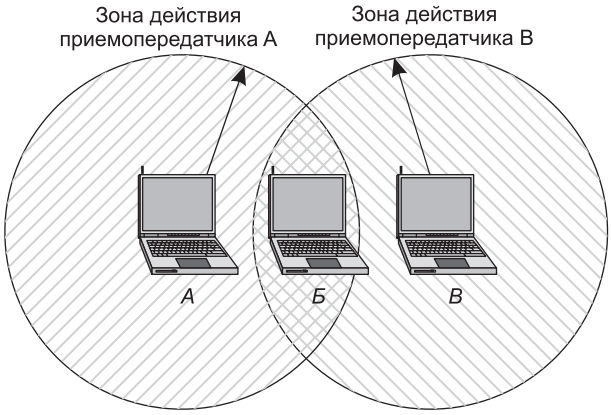


Рис. 1.32. Радиус действия одного радиопередатчика может не покрывать всю систему

Подвижность клиентов в 802.11 существенно более ограничена, чем в мобильных телефонных сетях. Как правило, 802.11 используется клиентами, которые двигаются от одного стационарного местоположения до другого, а не тогда, когда они непрерывно перемещаются. Для такого использования полноценная мобильность не требуется. Даже когда 802.11 используется в движении, перемещения ограничены одной сетью, зона действия которой не превышает одного большого здания. Будущие схемы должны будут обеспечить подвижность клиентов при работе в разных сетях и с использованием различных технологий (например, 802.21).

Наконец, существует проблема безопасности. Так как беспроводная передача является широковещательной, соседние компьютеры легко могут получить пакеты информации, которые не были для них предназначены. Чтобы избежать этого, в стандарте 802.11 используется схема шифрования WEP (Wired Equivalent Privacy). Основная идея — создать в беспроводной сети защиту аналогичную проводной. Идея была хорошая, но, к сожалению, в схеме оказалось много недостатков (Borisov и др., 2001). Позднее появились новые схемы шифрования, зафиксированные в стандарте 802.11i, который получил название WiFi Protected Access (WPA). В настоящее время используется версия WPA2.

Стандарт 802.11 вызвал революцию в беспроводных сетях, которая продолжается и сейчас. Помимо зданий такие сети используются в поездах, самолетах, судах и автомобилях, так чтобы люди могли выходить в Интернет везде, где бы они ни оказались. Мобильные телефоны и вся бытовая электроника, от игровых приставок до цифровых фотоаппаратов, могут получить доступ в сеть. Мы возвратимся к этому подробно в главе 4.

1.5.4. RFID и сенсорные сети

Сети, которые мы изучили до сих пор, образуются вычислительными устройствами — от компьютеров до мобильных телефонов. Радиочастотная идентификация (RFID) позволяет включать в сеть предметы повседневного пользования.

Электронная метка похожа на почтовую марку, которая может быть прикреплена к предмету (или встроена в него), чтобы его можно было отслеживать. Таким предметом может быть что угодно — животное, паспорт, книга или любая коробка. Метка состоит из маленького чипа с уникальным идентификатором и антенны, которая принимает радиосигнал. Считыватели RFID, установленные в точках отслеживании, обнаруживают метки, когда они оказываются вблизи, и посылают запрос, получая информацию, как показано на рис. 1.33. Это позволяет осуществлять идентификацию, управление системой поставок, измерение скорости и отказаться от использования штрихкодов.

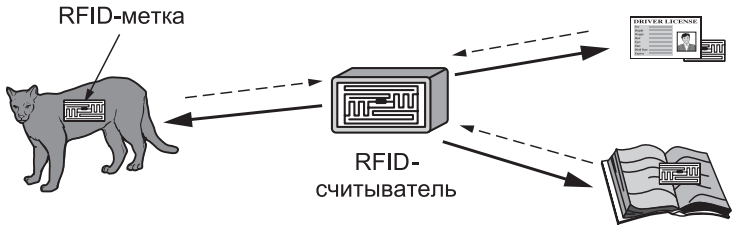


Рис. 1.33. RFID используется для объединения в сеть предметов повседневного пользования

Свойства RFID могут различаться, но, возможно, самый захватывающий аспект технологии RFID — то, что у большинства меток RFID нет ни электрического штепселя, ни батареи. Вместо этого вся необходимая энергия поставляется считывателями RFID в виде радиоволн. Эту технологию называют пассивным RFID, чтобы отличить это от (реже встречающегося) активного RFID, в котором метка обладает собственным источником энергии.

Стандартная форма RFID — УВЧ RFID (ультравысокочастотный RFID) — используется для контейнеров и некоторых водительских лицензий. В США считыватели излучают сигналы в полосе 902–928 МГц. Общение с метками происходит на расстоянии нескольких метров, метка отражает и изменяет сигнал, который считыватель принимает и обрабатывает. Этот способ называют обратным излучением.

Другой популярный вид RFID — ВЧ RFID (высокочастотный RFID) — работает на частоте 13,56 МГц и, вероятно, может встраиваться в паспорта, кредитные карты, книги и бесконтактные платежные системы. У ВЧ RFID малая дальность — не больше метра, потому что физический механизм излучения сигнала основан на индукции, а не обратном излучении. Существуют и другие формы RFID, использующие другие частоты, такие как LF RFID (низкочастотный RFID), который использовался до ВЧ RFID для отслеживания животных. Этот вид RFID, вероятно, можно найти у вашей кошки. Считыватели RFID должны, так или иначе, решать проблему контакта с несколькими метками в пределах диапазона чтения. Это означает, что метка не может просто отвечать, обнаружив считыватель, иначе сигналы от нескольких меток могут помешать друг другу. Решение проблемы похоже на подход, использованный в 802.11:

перед ответом метки выдерживают паузу в течение случайного короткого интервала, которая позволяет считывателю разделять сигналы.

Безопасность — еще одна проблема. Способность считывателей RFID легко отследить объект и, следовательно, человека, который его использует, может быть вторжением в личную жизнь. К сожалению, трудно обеспечить информационную безопасность для RFID-меток, потому что их вычислительные возможности ограничены. Поэтому используются более слабые меры, такие как пароли (которые легко взломать). Если паспортный контроль на границе может считать информацию с вашего паспорта удаленно, помешает ли что-нибудь другим людям отслеживать ту же самую метку без вашего ведома? Навряд ли.

Метки RFID появились как чипы идентификации, но быстро превращаются в достаточно серьезные компьютеры. Например, у многих меток есть память, которая может сохранять информацию о том, что произошло с объектом, и позже передавать ее. Rieback и др. (2006) продемонстрировал, что это приводит к тому, что в этой области также должны появиться все проблемы, свойственные программному обеспечению компьютера. Например, ваша кошка или ваш паспорт могут использоваться для распространения RFID-вируса.

Следующий шаг — сенсорная сеть. Сенсорные сети призваны контролировать все аспекты материального мира. До сих пор они главным образом использовались для научных экспериментов, например для отслеживания поведения птиц, вулканической деятельности и миграции зебр. Теперь они внедряются в бизнес-приложения, здравоохранение, оборудование для контроля колебаний и контроля замороженных, охлажденных и других скоропортящихся товаров.

Сенсорные узлы — маленькие компьютеры размером с брелок для ключей, в которые встроены датчики температуры, вибрации и т. п. Множество узлов находится в зоне наблюдения. Как правило, у них есть собственные источники питания, они могут пользоваться солнечной энергией или получать энергию от колебаний. Для этих устройств важно обладать достаточным количеством энергии, чтобы доставить данные датчиков на внешнюю точку сбора информации. Общая схема самоорганизации узлов для передачи друг другу сообщений показана на рис. 1.34. Такую схему называют сетью с ретрансляторами.

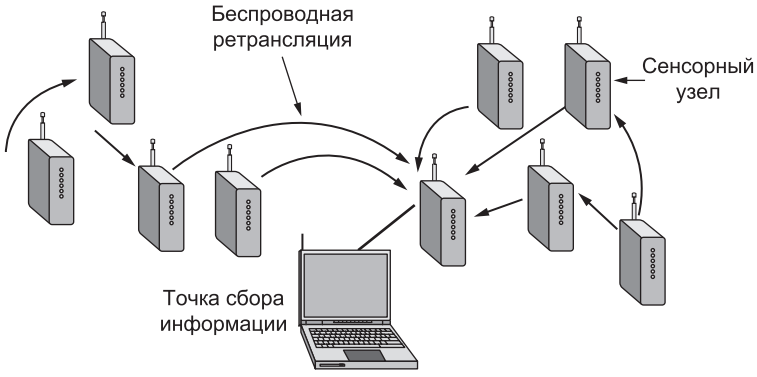


Рис. 1.34. Топология сенсорной сети с ретрансляторами

В будущем RFID и сенсорные сети, вероятно, получат больше возможностей и станут более распространенными. Исследователи уже объединили лучшее из обеих технологий, создав прототипы программируемых RFID-меток с датчиками света, движения и др. (Sample и др., 2008).

1.6. Стандартизация сетей

В мире существует большое количество разработчиков сетей, каждый из которых имеет свои представления о способах реализации различных функций. Без координации их действий наступила бы полная неразбериха, и пользователи не смогли бы работать. Единственным способом борьбы с хаосом является достижение согласия по определенным вопросам на основе сетевых стандартов. Стандарты не только обеспечивают возможность общения компьютеров, но также расширяют рынок для продукции, придерживающейся стандарта, что приводит к массовому выпуску совместимой друг с другом аппаратуры, очень широкомасштабной интеграции, удешевлению производства и, следовательно, еще большему притоку потребителей на этот рынок.

В этом разделе мы рассмотрим чрезвычайно важный, но плохо изученный мир международных стандартов. Но сначала давайте обсудим, что должно относиться к стандартам. Разумный человек предположит, что стандарт разъясняет, как работает протокол, чтобы вы могли сделать его хорошую реализацию. И этот человек оказывается неправ.

Стандарты определяют то, что необходимо для реализации взаимодействия: ни больше, ни меньше. Это позволяет рынку развиваться и также позволяет компаниям конкурировать, совершенствуя собственные продукты. Например, стандарт 802.11 задает скорости передачи, но не говорит, при каких условиях отправитель должен использовать конкретную скорость, — данная информация является ключевой характеристикой хорошей работы. Этот вопрос должен решать тот, кто создает продукт. Часто такой подход затрудняет взаимодействие, поскольку стандарты обычно предлагают множество возможностей, что приводит к появлению огромного количества вариантов реализаций. С 802.11 было связано столько проблем, что на вооружение была взята стандартная стратегия — для отлаживания взаимодействия в рамках 802.11 была создана специальная группа — WiFi Alliance.

Точно так же стандарт протокола определяет протокол проводной связи, но не дает описания реального интерфейса, позволяющего объяснить протокол. Существующие интерфейсы часто являются объектами собственности. Например, способ, которым TCP взаимодействует с IP в компьютере, не имеет значения для обеспечения связи с удаленным узлом. Значение имеет только то, что удаленный узел говорит через TCP/IP. Фактически TCP и IP обычно работаю вместе без дополнительного интерфейса. Это говорит о том, что хорошие служебные интерфейсы, как и хорошие интерфейсы приложений, важны для работы протоколов и лучшие (такие как сокеты Беркли) могут стать очень популярны.

Стандарты делятся на две категории: de facto и de jure. Стандарты **de facto** установились сами собой, без какого-либо предварительного плана. HTTP — протокол, на котором работает Сеть, появился как стандарт де-факто. Он происходит от пер-

вых WWW-браузеров, созданных Тимом Бернерсом-Ли (Tim Berners-Lee) в CERN. Bluetooth — еще один пример — был придуман фирмой Ericsson, но теперь им пользуются все.

Стандарты **de jure**, напротив, приняты по неким формальным законам стандартизации. Международные организации по стандартизации обычно делятся на две категории: созданные на основе межправительственных договоренностей и добровольные организации. В области сетевых компьютерных стандартов существуют несколько организаций каждого типа, наиболее известны ITU, ISO, IETF и IEEE, которые мы обсудим ниже.

Взаимоотношения между стандартами и компаниями достаточно сложны на практике. Фактические стандарты часто трансформируются в юридические, особенно если они успешны. Так произошло с HTTP, который был быстро принят IETF. Комитеты по стандартизации могут ратифицировать стандарты друг друга, чтобы стимулировать рост рынка технологий. В настоящее время существует множество специальных деловых организаций, которые образуются вокруг некоторых технологий и также играют существенную роль в развитии и усовершенствовании сетевых стандартов. Примером может послужить технология 3GPP (Third Generation Partnership Project), которая объединила телекоммуникационные ассоциации, занимающиеся стандартами мобильной связи 3G UMTS.

1.6.1. Кто есть кто в мире телекоммуникаций

Официальный статус телефонных компаний мира в разных странах различен. На одном полюсе находятся Соединенные Штаты, в которых имеется 2000 отдельных (в основном очень маленьких) частных телефонных компаний. До разделения в 1984 году AT&T, самая большая на тот момент корпорация в мире, полностью доминировала в этом рыночном секторе. Ее услугами пользовались около 80 % абонентов Америки, и закон о Телекоммуникациях 1996 года перестроил регулирование, чтобы способствовать конкуренции.

На другом полюсе располагаются страны, в которых правительство обладает полной монополией на все средства связи, включая почту, телеграф, телефон, а часто также и радио, и телевидение. В эту категорию попадает большая часть мира. В некоторых случаях владельцем средств связи выступает национализированная компания, в других им является просто особая ветвь правительства, обычно называемая **Министерством связи** или **PTT** (Postal Telegraph and Telephone administration — Управление почтово-телеграфной и телефонной связи). Сейчас во всем мире наблюдается тенденция перехода от государственной монополии к либерализации и конкуренции. Большинство европейских стран уже передало целиком или частично эту отрасль в руки независимых собственников, но кое-где процесс пока еще идет слишком медленно.

При таком разнообразии поставщиков услуг, очевидно, имеется необходимость в обеспечении совместимости во всемирном масштабе. Совместимости, гарантирующей пользователям (и компьютерам) разных стран возможность связаться друг с другом. На самом деле, эта потребность существовала уже очень давно. В 1865 году представители многих европейских государств собрались, чтобы сформировать союз, который явился предшественником сегодняшнего Международного союза теле-

коммуникаций (ITU, International Telecommunications Union). Задачей этого союза стала стандартизация международных средств связи. В то время еще нечего было стандартизировать, кроме телеграфа, но уже тогда было ясно, что если половина стран будет использовать азбуку Морзе, а другая половина — какой-либо другой код, то возникнут проблемы. С появлением международной телефонной связи ITU занялся также разработкой стандартов в области телефонии. В 1947 году международный союз телекоммуникаций вошел в состав учреждений Организации Объединенных Наций.

В ITU входят примерно 200 представителей различных ведомств, включая практически все организации ООН. В США отсутствует PTT, но кто-то же должен представлять государственные структуры. Эта роль выпала Государственному департаменту. Возможно, он участвует в решении вопросов взаимодействия ITU с другими государствами, что коррелирует с собственным направлением деятельности Госдепартамента. В состав ITU входит примерно 700 представителей соответствующего промышленного сектора и ассоциированные члены, среди которых есть и телефонные компании (например, AT&T, Vodafone, Sprint), и производители телекоммуникационной аппаратуры (например, Cisco, Nokia, Nortel), и производители компьютеров (например, Microsoft, Agilent, Toshiba), производители микросхем (например, Intel, Motorola, TI) и другие заинтересованные компании (например, Boeing, CBS, VerySign).

У ITU есть три основных сектора. Мы сосредоточимся прежде всего на **ITU-T**, **Телекоммуникационном секторе стандартизации**, который касается систем передачи данных и телефонии. До 1993 года этот сектор называли CCITT, что является акронимом для его французского названия, Comité Consultatif International Télégraphique et Téléphonique. **ITU-R**, **Сектор радиосвязи**, занимается координированием использования конкурирующими заинтересованными группами радиочастот во всем мире. Другой сектор — **ITU-D**, **Сектор развития**. Он способствует развитию информационно-коммуникационных технологий, чтобы сузить «цифровую пропасть» между странами с эффективным доступом к информационным технологиям и странами с ограниченным доступом.

Задачей ITU-T является разработка технических рекомендаций в области телефонии, телеграфа и интерфейсов передачи данных. Подобные рекомендации часто становятся международными стандартами, хотя технически рекомендации ITU-T являются всего лишь предложениями, которые правительство любой страны может принять или проигнорировать. (Потому что правительства подобны тринадцатилетним мальчишкам, которые терпеть не могут, когда им приказывают что-то делать. Лучше скромно порекомендовать.) На практике никто не может помешать стране принять свой собственный телефонный стандарт, отличающийся от всех других, однако тем самым эта страна сама себя отрежет от всего мира. Возможно, такой подход будет работать в Северной Корее, но во всем остальном мире это будет настоящей проблемой.

Собственно, работа в ITU-T осуществляется исследовательскими группами (Study Groups), состав которых часто достигает 400 человек. В настоящее время работают 10 таких групп, занимающихся самыми разными вопросами, начиная от концепции оплаты телефонных услуг и заканчивая службами мультимедиа и вопросами безопасности. Группа SG 15, например, стандартизирует технологии DSL, часто используемые для соединения с Интернетом. Чтобы сделать возможной какую-либо работу,

исследовательские группы разделяются на рабочие группы (Working Parties), подразделяющиеся, в свою очередь, на экспертные команды (Expert Teams), которые также делятся на группы. Бюрократия и здесь остается бюрократией.

Несмотря на все это, ITU-T справляется с работой. Текущий выход этой организации составляет более 3000 рекомендаций, многие из которых широко применяются на практике. Например, рекомендация H.264 (также стандарт Международной организации по стандартизации, известный как MPEG-4 AVC) широко используется для сжатия видео, а сертификаты открытого ключа X.509 используются для безопасного просмотра веб-страниц и цифровой подписи электронной почты.

По мере того как телекоммуникации завершают начатый ими в 80-е годы переход от национальных систем к глобальным, стандарты становятся все более важным аспектом их развития, и все большее число организаций желает принять участие в их формировании. Дополнительную информацию об ITU см. Irner, 1994.

1.6.2. Кто есть кто в мире международных стандартов

Международные стандарты разрабатываются **Международной организацией по стандартизации (International Organization for Standardization, ISO)**, добровольной организацией, созданной в 1946 году. В нее входят национальные организации по стандартизации из 89 стран. Среди ее членов ANSI (США), BSI (Великобритания), AFNOR (Франция), DIN (Германия) и еще 153 другие организации.

ISO выпускает стандарты, касающиеся широкого спектра вопросов, начиная от болтов и гаек (буквально) до покраски телефонных столбов (не говоря уж о стандартах на какао-бобы (ISO 2451), рыболовные сети (ISO 1530), женское нижнее белье (ISO 4416)... Сложно придумать такую вещь, для которой не существовало бы стандарта ISO). По проблемам телекоммуникационных стандартов Международная организация по стандартизации ISO и ITU-T часто сотрудничают (ISO — член ITU-T), чтобы избежать парадокса двух официальных и взаимно несовместимых международных стандартов.

К настоящему времени выпущено более 17 000 стандартов, включая стандарты OSI. В ISO входят более 200 технических комитетов (Technical Committee, TC), нумеруемых последовательно, по мере их создания, каждый из которых занимается своим отдельным вопросом. Так, например, TC1 занимается болтами и гайками (стандартизацией диаметра и шага резьбы). JTC1 имеет дело с информационной технологией, включая сети, компьютеры и программное обеспечение. Первый (и до сих пор единственный) Объединенный технический комитет, созданный в 1987 году слиянием TC97 с действиями в IEC, — еще один субъект стандартизации.

Каждый технический комитет делится на подкомитеты (subcommittee, SC), которые, в свою очередь, состоят из рабочих групп (working group, WG).

Основная работа проводится в рабочих группах, в которые входит более 100 000 добровольцев по всему миру. Многие из этих «добровольцев» работают по найму в компаниях, чьи продукты должны быть стандартизованы. Другие являются государственными служащими, пытающимися сделать свой национальный стандарт

международным. Ученые эксперты также принимают активное участие во многих рабочих группах.

Для принятия стандартов Международной организацией по стандартизации ISO разработана процедура, позволяющая добиться принятия мнения, одобренного максимально возможным количеством сторон. Процесс начинается, когда одна из национальных организаций по стандартизации чувствует потребность в появлении международного стандарта в определенной области. Тогда формируется рабочая группа, которая вырабатывает **предварительный план** (Committee Draft, CD). Этот набросок передается всем остальным членам технического комитета. На критику проекта отводится срок в шесть месяцев. Если проект получает одобрение значительным большинством, то откорректированный документ получает название **чернового международного стандарта** (DIS, Draft International Standard); после этого он опять циркулирует в различных группах, где за него голосуют и снабжают его комментариями. На основании этого подготавливается, утверждается и публикуется окончательный документ, называемый **международным стандартом** (IS, International Standard). В некоторых случаях черновым вариантам, CD или DIS, приходится проходить через многократные изменения и голосования, пока они не наберут достаточного количества голосов. Подобный процесс может длиться несколько лет.

Национальный институт стандартов и технологий США (NIST, National Institute of Standards and Technology) является подразделением Министерства торговли США (U.S. Dept. of Commerce). Ранее он назывался Национальным бюро стандартов (National Bureau of Standards). Он выпускает стандарты, обязательные для закупок, проводимых правительством США, кроме закупок Министерства обороны, которое определяет свои стандарты.

Одним из основных игроков на поле стандартизации является **Институт инженеров по электротехнике и электронике (IEEE, Institute of Electrical and Electronics Engineers)** — крупнейшая профессиональная организация в мире. Помимо выпуска ряда журналов и организации разнообразных конференций, IEEE также разрабатывает стандарты в области электротехники и электроники. Например, комитет IEEE 802 выпустил ряд ключевых стандартов в области локальных компьютерных сетей, некоторые из них мы рассмотрим в этой книге. Реальная работа проводится, как правило, внутри рабочих групп, которые перечислены в табл. 1.4. Рабочие группы комитета 802 никогда не считались преуспевающими. Номер 802.x вовсе не был залогом успеха. Однако стремительный взлет популярности стандартов (особенно 802.3 и 802.11) в промышленности и в мире изменил ситуацию.

Таблица 1.4. Рабочие группы комитета 802. Наиболее важные отмечены звездочкой (*). Те, что помечены стрелочкой (↓), бездействуют. Отмеченные крестиком (†) самоликвидировались за ненадобностью

Номер	Темы разработок
802.1	Общее представление и архитектура ЛВС
802.2 ↓	Управление логическим каналом
802.3 *	Ethernet
802.4 ↓	Маркерная шина (одно время использовалась в промышленных сетях)

Номер	Темы разработок
802.5	Маркерное кольцо (вклад фирмы IBM в технологии ЛВС)
802.6 ↓	Двойная двунаправленная шина (ранние региональные сети)
802.7 ↓	Техническая консультативная группа по широкополосным технологиям
802.8 †	Техническая консультативная группа по оптоволоконным технологиям
802.9 ↓	Изохронные ЛВС (для приложений реального времени)
802.10 ↓	Виртуальные ЛВС и защита информации
802.11 *	Беспроводные ЛВС
802.12 ↓	Приоритеты запросов (для AnyLAN фирмы Hewlett-Packard)
802.13	Счастливый номер. Почему-то его никто не выбрал
802.14 ↓	Кабельные модемы (рабочая группа распалась: в области кабельных модемов ее опередил промышленный консорциум)
802.15 *	Персональные сети (Bluetooth)
802.16 *	Широкополосные беспроводные ЛВС
802.17	Гибкая технология пакетного кольца
802.18	Радиорегулирование
802.19	Сосуществование сетей
802.20	Мобильный широковещательный беспроводной доступ (аналог 802.16e)
802.21	Переключение, не зависимое от среды передачи данных (для переключения между технологиями)
802.22	Местные беспроводные сети

1.6.3. Кто есть кто в мире стандартов Интернета

Всемирная сеть Интернет имеет свой механизм стандартизации, значительно отличающийся от ITU-T и ISO. В двух словах, основное отличие заключается в том, что сотрудники ITU и ISO носят деловые костюмы, тогда как стандарты Интернета разрабатывают в основном люди в джинсах (ну, кроме тех, кто работает в Сан-Диего — на них надеты шорты и футболки с коротким рукавом).

На совещания ITU-T и ISO собираются администраторы корпораций и государственные гражданские служащие, для которых стандартизация является их работой. Они считают, что стандартизация — Очень Нужная Вещь, и посвящают ей свою жизнь. Для людей Интернета, напротив, анархия является делом принципа, однако когда сотни миллионов делают какое-то общее дело, иногда им все же приходится о чем-то договариваться, чтобы хоть что-то работало. Волей-неволей стандарты оказываются необходимыми. Дэвид Кларк (David Clark) как-то высказал замечание, ставшее ныне популярным, о стандартизации Интернета, состоящей из «грубого консенсуса и работающей программы».

Когда была запущена сеть ARPANET, Министерство обороны США создало неофициальный комитет для наблюдения за сетью. В 1983 году этот комитет был пере-

именован в **Совет по деятельности Интернета** (Internet Activities Board, **IAB**). Перед советом были поставлены несколько расширенные задачи, а именно удерживать исследователей, включенных в проекты ARPANET и Интернет, в более-менее одном направлении, что напоминало попытку выпаса стада кошек. Значение сокращения IAB было затем изменено на **Совет по архитектуре Интернета** (Internet Architecture Board).

Каждый из приблизительно десяти членов IAB возглавлял специальную комиссию по отдельному важному вопросу. Совет по архитектуре Интернета собирался несколько раз в год для обсуждения результатов работы и предоставления отчета Министерству обороны и NSF, которые в то время осуществляли основное финансирование в этой области. Когда требовался какой-либо стандарт (например, новый алгоритм маршрутизации), члены совета прорабатывали этот вопрос, после чего объявляли об изменениях аспирантам, занимавшимся реализацией программного обеспечения сетей. Стандарты оформлялись в виде набора технических отчетов, называемых **RFC** (Requests for Comments). RFC доступны в Интернете для всех желающих (*www.ietf.org/frc*). Они пронумерованы в хронологическом порядке их создания. На сегодняшний день существует около 5000 этих документов. На многие из них мы будем ссылаться в этой книге.

К 1989 году Интернет вырос настолько, что подобный неформальный подход к его стандартам перестал работать. К тому моменту многие производители предлагали продукцию на основе протокола TCP/IP и не хотели их менять просто потому, что десятку исследователей пришла в головы одна хорошая идея. Летом 1989 года IAB был снова реорганизован. Исследователи были переведены в **группу исследования Интернета** (Internet Research Task Force, IRTF), подконтрольную IAB, и в **группу проектирования Интернета** (Internet Engineering Task Force, IETF). В совете IAB появились люди, представляющие более широкий спектр организаций, чем исследовательское сообщество. Вначале это была группа, в которой члены работали в течение двух лет, после которых сами назначали своих преемников. Затем было создано **Общество Интернета** (Internet Society), в которое вошли люди, заинтересованные в Интернете. Таким образом, интернет-сообщество в каком-то смысле сравнимо с Ассоциацией по вычислительной технике (ACM, Association for Computing Machinery) или IEEE. Оно управляется избираемыми доверенными лицами, которые утверждают состав IAB.

Идея этого разделения заключалась в том, чтобы сосредоточить IRTF на долгосрочных исследованиях, а IETF — на краткосрочных инженерных вопросах. Проблемная группа IETF была разделена на рабочие группы, каждая из которых решала свою задачу. Первое время председатели рабочих групп встречались друг с другом в составе руководящего комитета для координации совместных исследовательских усилий. Рабочие группы занимались такими вопросами, как новые приложения, информация для пользователей, OSI-интеграция, маршрутизация и адресация, безопасность, управление сетью и стандарты. В конце концов, было сформировано так много рабочих групп (более 70), что их сгруппировали по областям, после чего в руководящем комитете стали собираться председатели областей.

Кроме того, был принят более формальный процесс стандартизации по аналогии с процедурой, принятой в ISO. Чтобы стать **предлагаемым стандартом**, основная

идея должна быть полностью изложена в RFC и представлять достаточный интерес, гарантирующий ее рассмотрение. Затем, чтобы стать **проектом стандарта**, должна быть создана работающая реализация, которую нужно тщательно протестировать минимум двумя независимыми сайтами в течение 4-х месяцев. Если IAB уверен, что идея здравая и программное обеспечение работает, он может объявить RFC стандартом Интернета. Некоторые стандарты Интернета стали стандартами Министерства обороны США (MIL-STD), что сделало их обязательными к применению поставщиками министерства.

Консорциум World Wide Web (W3C) развивает протоколы и направляющие линии для веб-стандартов, чтобы облегчить долгосрочный рост Сети. Это — промышленный консорциум во главе с Тимом Бернерсом-Ли, организованный в 1994 году, когда Web действительно начал стремительно расти. Теперь W3C имеет более 300 участников со всего мира и произвел больше чем 100 Рекомендаций W3C, как называют его стандарты, затрагивая такие темы, как HTML и веб-безопасность.

1.7. Единицы измерения

Во избежание путаницы необходимо предварить дальнейшие рассуждения замечанием по поводу единиц измерения. В вычислительной технике традиционной английской системе обычно предпочитают десятичную систему мер. Основные префиксы, используемые при этом, приведены в табл. 1.5. Обычно они сокращаются по первым буквам их названий, причем если префикс имеет вес, больший 1, то он пишется с заглавной буквы (Кб, Мб и т. д.). Единственное исключение исторически составляет сокращение Кбит/с. Таким образом, линия, работающая на скорости 1 Мбит/с, передает 10^6 бит в секунду, а таймер на 100 пс изменяет свое состояние каждую 10^{-10}-ю долю секунды. Поскольку «милли» и «микро» начинаются с одной и той же буквы, то принято обозначать «милли» буквой «м», а «микро» — буквами «мк» или греческой буквой «μ».

Таблица 1.5. Основные префиксы метрической системы

Степень	В явном виде	Префикс	Степень	В явном виде	Префикс
10^{-3}	0,001	милли	10^{3}	1000	Кило
10^{-6}	0,000001	микро	10^{6}	1 000 000	Мега
10^{-9}	0,000000001	нано	10^{9}	1 000 000 000	Гига
10^{-12}	0,000000000001	пико	10^{12}	1 000 000 000 000	Тера
10^{-15}	0,000000000000001	фемто	10^{15}	1 000 000 000 000 000	Пета
10^{-18}	0,000000000000000001	атто	10^{18}	1 000 000 000 000 000 000	Экза
10^{-21}	0,000000000000000000001	цепто	10^{21}	1 000 000 000 000 000 000 000	Цетта
10^{-24}	0,000000000000000000000001	йокто	10^{24}	1 000 000 000 000 000 000 000 000	Йотта

Также необходимо отметить, что единицы измерения, использующиеся для обозначения объемов памяти, емкости дисков, размеров файлов и баз данных, несколько

отличаются от принятых в других областях. Например, «кило» означает не 1000 (10^3), а 1024 (2^{10}), что соответствует общей двоичной концепции вычислительной техники. Размеры памяти всегда представляют собой степени двойки. Так, в 1 Кб содержится 1024 байт, а не 1000 байт. Заметим также, что заглавная Б используется для обозначения байта (участка из 8 битов), а строчная б означает «бит». Аналогично, в 1 Мб содержится 2^{20}, то есть 1 048 576 байт, а в 1 Гб соответственно 2^{30} (1 073 741 824) байт. База данных на 1 Тб содержит 2^{40} (1 099 511 627 776) байт. Тем не менее линия при скорости 1 Кбит/с передает 1000 бит/с, а ЛВС, работающая со скоростью 10 Мбит/с, может передавать 10 000 000 бит/с — скорости не измеряются степенями двойки. К сожалению, многие путают эти две системы счисления, особенно когда дело касается емкости дисков. Чтобы избежать двусмысленности, еще раз повторюсь: по крайней мере, в нашей книге символы Кб, Мб, Гб и Тб будут означать 2^{10}, 2^{20}, 2^{30} и 2^{40} байт соответственно. При этом буквой «б» мы будем обозначать байты, а «биты» так и будут «битами». Поэтому символы Кбит/с, Мбит/с, Гбит/с и Тбит/с будут означать соответственно 10^3, 10^6, 10^9 и 10^{12} бит в секунду.

1.8. Краткое содержание следующих глав

В этой книге обсуждаются как теоретические, так и практические вопросы построения компьютерных сетей. Большая часть глав начинается с обсуждения основных принципов, за которыми следует ряд примеров, иллюстрирующих эти принципы. На протяжении всей книги в качестве примеров используются действующие сети — Интернет и беспроводные сети, такие как сеть мобильной телефонной связи. Эти сети очень важны и актуальны, а кроме того, они очень разные. Там, где это важно, будут даны и другие примеры.

Структура книги соответствует гибридной модели, изображенной на рис. 1.20. Начиная со второй главы, мы будем рассматривать иерархию протоколов с самого нижнего из них. Мы изучим основные положения в области обмена данными, для проводной и для беспроводной связи. Этот материал относится к передаче информации на физическом уровне, хотя мы будем обсуждать не столько аппаратные, сколько архитектурные аспекты. Также будут обсуждаться несколько примеров физического уровня, такие как коммутируемая телефонная сеть общего пользования, мобильная телефонная сеть, а также сеть кабельного телевидения.

Главы 3 и 4 обсуждают уровень канала связи в двух частях. В главе 3 рассматривается проблема того, как послать пакеты через канал, включая обнаружение ошибок и исправление. Как реальный пример протокола канала связи мы рассмотрим DSL (используемый для широкополосного доступа к Интернету по телефонным линиям).

В главе 4 мы исследуем средний подуровень доступа. Это — часть уровня канала связи, который имеет дело с тем, как совместно использовать канал несколькими компьютерами. Примеры, которые мы рассмотрим, включают беспроводные, такие как 802.11 и RFID, и проводные ЛВС, такие как классический Ethernet. Также мы обсудим здесь коммутаторы канального уровня, которые соединяют ЛВС, такие как коммутируемый Ethernet.

В главе 5 описывается сетевой уровень, в частности маршрутизация. Обсуждаются как статические, так и динамические алгоритмы маршрутизации. Даже очень хороший алгоритм рассчитан на сеть с определенным уровнем загрузки. При превышении этого уровня могут возникать заторы, пакеты могут задерживаться или пропадать. Мы обсудим предотвращение заторов и обеспечение необходимого уровня обслуживания. Кроме того, затрагивается ряд проблем, возникающих при связи разнородных сетей. Особое внимание уделяется сетевому уровню применительно к Интернету.

Глава 6 посвящена транспортным уровням. Основное внимание уделяется протоколам, ориентированным на соединение, и надежности, поскольку в них нуждаются многие приложения. Подробно описываются транспортные протоколы Интернета — TCP и UDP. Обсуждаются также вопросы их эффективности.

В главе 7 описывается прикладной уровень, его протоколы и приложения. Первое, к чему мы обратимся, будет служба DNS — интернет-аналог телефонной книги. Затем речь пойдет об электронной почте и о протоколах, которые в ней используются. Затем мы переходим к веб-технологиям. В разделах, посвященных им, подробно обсуждается статическое и динамическое содержимое страниц, рассказывается о том, что происходит на стороне клиента и сервера. После этого рассматриваются вопросы мультимедиа в Сети, включая потоковый звук, интернет-радио и видео. Наконец, мы обсудим сети доставки контента, включая технологию одноранговых (пиринговых) сетей.

Глава 8 посвящена вопросам защиты информации. В этой теме есть аспекты, касающиеся всех уровней, именно поэтому данная глава завершает книгу. Она начинается с объяснения основ криптографии. После этого приводятся примеры применения криптографии для налаживания безопасных соединений, защиты электронной почты и веб-страниц. Заканчивается глава обсуждением некоторых вопросов, в которых защита информации сталкивается с частной жизнью. Речь идет о свободе слова, цензуре и других насущных социальных проблемах.

Глава 9 содержит аннотированный список предлагаемой литературы, организованный по главам. Его цель — помочь читателям, желающим продолжить изучение сетей. В главе также содержится алфавитный список литературы, упоминаемой в тексте книги.

Сайт автора в Pearson: *http://www.pearsonhighered.com/tanenbaum* содержит страничку со ссылками на различные самоучители, ответы на часто задаваемые вопросы, а также ссылки на сайты компаний и промышленных консорциумов, профессиональных организаций, комитетов по стандартизациии, технологиям, документации и т. д.

Резюме

Существует множество вариантов использования компьютерных сетей, и для компаний, и для частных лиц, и стационарно, и во время движения. Компании используют сети компьютеров, чтобы делиться корпоративной информацией, как правило, используя клиент-серверную модель с компьютерами сотрудников, действующими как клиенты, получающие доступ к мощным серверам в машинной комнате. Для частных лиц сети предлагают доступ к множеству информации и ресурсов развлечения, так же как способы купить и продать товары и услуги. Люди часто получают доступ к Интернету

дома через свой телефон или кабельных провайдеров, хотя беспроводной доступ для ноутбуков и телефонов используется все шире. Технологические усовершенствования включают новые виды мобильных приложений и сетей с компьютерами, встроенными в приборы и другие потребительские устройства. Те же самые усовершенствования поднимают социальные вопросы, такие как проблемы частной жизни.

Грубо говоря, все сети могут быть разделены на локальные, региональные, глобальные и объединенные. Локальные сети обычно охватывают здание и работают с очень высокой скоростью. Региональные обычно охватывают город — примером тому могут служить сети кабельного телевидения, используемые в последнее время для доступа в Интернет. Глобальные сети охватывают страну или континент. Некоторые из технологий, используемых, чтобы создать эти сети, являются двухточечными (например, кабель), другие — широковещательными (например, беспроводные). Сети могут быть связаны с маршрутизаторами, чтобы сформировать объединенные сети, из которых Интернет — самый большой и самый известный пример. Беспроводные сети, например 802.11 ЛВС и 3G мобильная телефония, также становятся чрезвычайно популярными.

Сетевое программное обеспечение строится вокруг протоколов, или правил, по которым процессы обмениваются информацией. Большая часть сетей поддерживает иерархию протоколов, в которой каждый уровень предоставляет услуги вышестоящему уровню, не раскрывая ему подробностей своей работы. Стек протоколов обычно базируется на модели OSI или модели TCP/IP. В обеих моделях имеется канальный, сетевой, транспортный и прикладной уровень, но они различаются в остальных уровнях. Вопросы разработки включают в себя уплотнение каналов, управление передачей, обнаружение ошибок и т. д. Вопросы проектирования включают надежность, распределение ресурсов, рост, безопасность и др. В книге уделяется много внимания протоколам и их проектированию.

Итак, сети предоставляют пользователям различные услуги. Эти услуги могут варьироваться от низкоприритетной сетевой доставки без установления соединения до гарантируемой доставки с установлением соединения. В некоторых сетях на одних уровнях используют первый из этих принципов, на других (более низких) — второй.

К хорошо известным сетям относятся Интернет, мобильные телефонные сети третьего поколения и сети стандарта 802.11. Сеть Интернет выросла из ARPANET, к которой был добавлен ряд других сетей. Таким образом, Интернет является объединенной сетью, Сетью сетей, число которых сегодня измеряется тысячами, использующей стек протоколов TCP/IP. Сеть мобильной связи 3G обеспечивает беспроводный и мобильный доступ к Интернету на скорости нескольких мегабит в секунду, и, конечно, переносит и голосовые вызовы. Беспроводные ЛВС, основанные на стандарте IEEE 802.11, развернуты во многих домах и кафе и могут обеспечить связь на уровнях свыше 100 Мбит/с. Появляются и новые виды сетей, такие как встроенные сенсорные сети и сети, основанные на технологии RFID.

Для того чтобы миллионы компьютеров могли общаться друг с другом, нужны как аппаратные, так и программные стандарты. Разрабатываются они такими организациями, как ITU-T, ISO, IEEE и IAB. Каждая из них работает в своей области, и все вместе они реализуют процесс стандартизации компьютерных сетей.

Вопросы

1. Представьте, что вы научили свою собаку, сенбернара Берни, приносить вам коробку с тремя 8-мм магнитными лентами вместо бутылки бренди. (Потому что с некоторых пор вы стали рассматривать заканчивающееся место на жестком диске как трагедию.) На каждой ленте помещается 7 Гб информации. Собака обучена бежать к вам, где бы вы ни находились, со скоростью 18 км/ч. В каком диапазоне расстояний скорость передачи данных собакой будет выше, чем у линии, чья фактическая скорость работы составляет 150 Мбит/с? Как изменится ваш ответ, если (а) скорость Берни увеличится в два раза; (б) емкость каждой ленты увеличится в два раза (в) скорость передачи данных по линии увеличится в два раза.
2. Альтернативой локальной сети является большая система разделения времени с терминалом для каждого пользователя. Приведите два преимущества клиент-серверной системы, использующей локальную сеть.
3. На производительность системы «клиент-сервер» сильнее всего влияют два главных параметра сети: пропускная способность (сколько бит в секунду она может передавать) и время ожидания (сколько секунд требуется на доставку первого бита от клиента до сервера). Приведите пример а) сети с высокой пропускной способностью и большим временем ожидания; б) сети с низкой пропускной способностью и малым временем ожидания.
4. Какие еще характеристики, кроме пропускной способности и времени ожидания, нужно оптимизировать для получения высокого качества обслуживания в а) сетях цифровой передачи речи; б) передаче видео; в) передачи финансовых транзакций?
5. Одним из факторов, влияющих на время ожидания в сетях с коммутацией пакетов и промежуточным хранением, является задержка при сохранении и переадресации пакета коммутатором. Если время коммутации составляет 10 мкс, будет ли это основной задержкой в работе клиент-серверной системы, в которой клиент находится в Нью-Йорке, а сервер — в Калифорнии? Скорость распространения сигнала в медной линии принять равной 2/3 скорости света в вакууме.
6. Система «клиент-сервер» использует спутниковую сеть. Орбита вращения спутника удалена от поверхности Земли на 40 000 км. Какова будет минимально возможная задержка при ожидании ответа на запрос в такой системе?
7. В будущем, когда у всех дома будет терминал, подключенный к компьютерной сети, станут возможными мгновенные референдумы по важным законодательным вопросам. В конце концов, существующие законодательные органы могут быть распущены, позволив народу выражать свою волю напрямую. Позитивные аспекты такой прямой демократии очевидны. Обсудите некоторые негативные аспекты.
8. Пять маршрутизаторов необходимо соединить в подсеть с двухточечным соединением. Каждые два маршрутизатора разработчики могут соединить высокоскоростной, среднескоростной, низкоскоростной линией или никак не соединять. Предположим, компьютеру требуется 100 мс для моделирования и анализа каждой топологии. Сколько компьютерного времени понадобится для выбора варианта, лучше всего соответствующего ожидаемой нагрузке?
9. Отрицательной чертой широковещательной подсети является потеря мощности вследствие попытки одновременного доступа к каналу нескольких хостов. В качестве простейшего примера предположим, что время делится на равные интервалы, в которые каждый из n хостов пытается использовать канал с вероятностью p. Какой процент интервалов будет потерян из-за конфликтов?
10. Назовите две причины использования многоуровневых протоколов. Каковы возможные неудобства их использования?

11. Президент корпорации Specialty Paint Corp. решает объединить усилия с местной пивоваренной фабрикой для производства невидимой пивной банки (в качестве средства борьбы с мусором). Президент просит свой юридический отдел рассмотреть эту идею, а те в свою очередь обращаются за помощью в технический отдел. В результате начальник технического отдела звонит начальнику технического отдела пивоваренного завода, чтобы обсудить технические аспекты проекта. После этого оба инженера докладывают своим юридическим отделам, которые затем обсуждают друг с другом по телефону юридические вопросы. Наконец, два президента корпораций обсуждают финансовую сторону дела. Какой принцип многоуровневого протокола (в соотвтетсвии с моделью OSI) мешает этому процессу?
12. Представим себе две сети, предоставляющие надежные, ориентированные на соединение службы. Одна из сетей обеспечивает надежный поток байтов, а другая — надежный поток сообщений. Идентичны ли эти сети? Если да, то почему проводится различие? Если нет, объясните на примере, чем они отличаются.
13. Что означает термин «согласование» (negotiation) в контексте обсуждения сетевых протоколов? Приведите пример.
14. На рис. 1.16 изображен некоторый сервис. Подразумевается ли наличие других сервисов на этом рисунке? Если да, то где? Если нет, то почему?
15. В некоторых сетях уровень передачи данных обрабатывает ошибки передачи, требуя повторной передачи поврежденных кадров. Если вероятность повреждения кадра равна p, каким будет среднее число попыток, необходимых для передачи кадра, при условии, что подтверждения никогда не теряются?
16. Система обладает *n*-уровневой иерархией протоколов. Приложения обмениваются сообщениями длиной *M* байт. На каждом из уровней добавляется заголовок из *h* байт. Какой процент пропускной способности занят заголовками?
17. В чем основное различие между протоколами TCP и UDP?
18. Подсеть на рис. 1.22, *б* проектировалась таким образом, чтобы выстоять во время ядерной войны. Сколько бомб потребуется, чтобы разбить сеть на две изолированные части, если одна бомба разрушает узел со всеми линиями, подходящими к нему.
19. Интернет удваивается в размерах приблизительно каждые 18 месяцев. Точное число хостов неизвестно, но один аналитик в 2009 году назвал цифру в 600 млн хостов. Сколько будет хостов в Интернете в 2018-м году? Вы сами верите в это? Поясните свою точку зрения.
20. При передаче файла между двумя компьютерами возможны (как минимум) две стратегии подтверждений. В первом случае файл может быть разбит на отдельные пакеты, получение которых подтверждается получателем индивидуально, но получение всего файла не подтверждается. Во втором случае получение каждого пакета не подтверждается, а подтверждается получение всего файла. Обсудите оба варианта.
21. Операторам мобильной телефонной сети необходимо знать местонахождения мобильных телефонов (и, следовательно, тех, кто ими пользуется). Объясните, чем это плохо для пользователей. Назовите причины, по которым это хорошо для пользователей.
22. Какой длины (в метрах) был один бит, в соответствии со стандартом 802.3? Для вычислений примите скорость работы равной 10 Мбит/с, а скорость распространения сигнала равной 2/3 скорости света в вакууме.
23. Имеется несжатое изображение размером 1600 × 1200 пикселов, 3 байта/пиксел. Сколько времени потребуется на его передачу с помощью модема, работающего со скоростью 56 Кбит/с? С помощью кабельного модема, работающего на 1 Мбит/с? По Ethernet со скоростью передачи 10 Мбит/с? По Ethernet со скоростью 100 Мбит/с? По гигабитной Ethernet?

24. Ethernet и беспроводные сети имеют много общего, но есть и различия. Одним из свойств Ethernet является возможность передачи только одного кадра в каждый момент времени. Унаследовал ли стандарт 802.11 это свойство?
25. Назовите два преимущества и два недостатка наличия международных стандартов для сетевых протоколов.
26. Когда у системы имеются постоянная и съемная части, как у проигрывателя компакт-дисков и компакт-диска, важно, чтобы система была стандартизирована, чтобы различные компании могли выпускать как постоянные, так и съемные части, стыкующиеся друг с другом. Приведите три примера международных стандартов вне компьютерной области. Теперь приведите три примера отсутствия международных стандартов вне компьютерной области.
27. Предположите, что алгоритмы, используемые для операций в уровне k, изменились. Как это воздействует на операции в уровнях $k - 1$ и $k + 1$?
28. Предположите, что произошло изменение в службе (набор операций) реализуемой уровнем k. Как это воздействует на службы в уровнях $k - 1$ и $k + 1$?
29. Приведите список причин того, почему время отклика клиента может быть больше, чем задержка лучшего случая.
30. Каковы недостатки использования маленьких клеток фиксированной длины в ATM?
31. Перечислите, какие действия вы производите ежедневно при помощи компьютерных сетей. Как изменится ваша жизнь, если сети вдруг перестанут существовать?
32. Узнайте, какие сети установлены в вашем учебном заведении или на работе. Опишите их типы, топологии, методы коммутации.
33. Программа `ping` позволяет отправлять тестовый пакет по указанному адресу и исследовать время его прохождения в одну и в другую сторону. Попробуйте воспользоваться этой программой, чтобы выяснить, сколько времени следуют данные по различным известным адресам. Используя полученные данные, постройте график зависимости времени прохождения пакета от расстояния. Лучше всего использовать для экспериментов серверы университетов, поскольку их расположение очень хорошо известно. Например, berkley.edu находится в Беркли, штат Калифорния, mit.edu — в Кембридже, Массачусетс, vu.nl — в Амстердаме, Голландия, www.usyd.edu.au — в Сиднее, Австралия и www.uct.ac.za — в Кейптауне, Южная Африка.
34. Посетите сайт IETF (www.ietf.org) и изучите, чем занимается эта организация. Возьмите в качестве примера любой понравившийся вам проект и напишите краткий отчет о том, что он собой представляет.
35. Интернет состоит из огромного числа сетей. Их взаимное расположение определяет топологию Интернета. Очень много информации на тему топологии Интернета можно найти на различных веб-сайтах. С помощью поисковых программ найдите соответствующую информацию и напишите краткий отчет по итогам исследования.
36. Поищите в Интернете и узнайте некоторые из важных пиринговых точек, используемых для направления пакетов в Интернете в настоящее время.
37. Напишите программу, которая осуществляет поток сообщений от верхнего слоя до нижнего слоя модели протокола с 7 уровнями. Ваша программа должна включать отдельную функцию протокола для каждого уровня. Заголовки протокола — последовательность до 64 символов. У каждой функции есть два параметра: сообщение, пришедшее из протокола более высокого уровня (буфер случайной работы), и размер сообщения. Эта функция присоединяет свой заголовок перед сообщением, печатает новое сообщение на стандартном выводе и затем вызывает функцию протокола нижнего уровня. Ввод программы — сообщение приложения (последовательность 80 символов или меньше).

Глава 2
Физический уровень

В этой главе мы рассмотрим нижний уровень сетевой модели — физический уровень. Он определяет электрические, временны́е и прочие характеристики сетей, по которым биты информации пересылаются в форме электрических сигналов. Физический уровень — это фундамент сети. Производительность каналов передачи данных (их полоса пропускания, время запаздывания и частота ошибок) определяется различными свойствами физических носителей. Базовые характеристики каналов передачи и станут отправной точкой нашего увлекательного путешествия в мир компьютерных сетей.

Начнем с теоретического анализа передачи данных, чтобы с удивлением обнаружить, что природа накладывает определенные ограничения на то, что и как можно передавать с помощью физического носителя. Затем мы обсудим три типа сред передачи — проводниковые (медный провод и оптоволокно), радиоэфир (наземная радиосвязь) и радиоэфир, связанный со спутниковыми системами. Уникальные отличительные свойства каждой из этих технологий в значительной степени влияют на принципы построения и производительность сетей. Мы изучим основы ключевых технологий передачи данных, применяемых в современных сетях.

Нашей следующей темой станет цифровая модуляция — мы поговорим о том, как аналоговые сигналы превращаются в цифровые и обратно. После этого мы рассмотрим схемы уплотнения и узнаем, как данные нескольких сеансов связи могут передаваться по одному носителю, не мешая друг другу.

Оставшаяся часть главы посвящена трем примерам систем связи, которые используются на практике в глобальных сетях. Мы начнем с телефонной системы (стационарной), второй пример — мобильная телефонная система, и третий — кабельное телевидение. Все эти технологии крайне важны и повсеместно применяются на практике, поэтому мы уделим достаточно внимания каждому примеру.

2.1. Теоретические основы передачи данных

Информация может передаваться по проводам за счет изменения какой-либо физической величины, например напряжения или силы тока. Представив значение напряжения или силы тока в виде однозначной функции времени, $f(t)$, мы сможем смоделировать поведение сигнала и подвергнуть его математическому анализу. Этому анализу и посвящены следующие разделы.

2.1.1. Ряды Фурье

В начале XIX столетия французский математик Жан-Батист Фурье (Jean-Baptiste Fourier) доказал, что любая периодическая функция $g(t)$ с периодом T может быть разложена в ряд (возможно, бесконечный), состоящий из сумм синусов и косинусов:

$$g(t) = \frac{1}{2}c + \sum_{n=1}^{\infty} a_n \sin(2\pi nft) + \sum_{n=1}^{\infty} b_n \cos(2\pi nft), \qquad (2.1)$$

где $f = 1/T$ — основная частота (гармоника), a_n и b_n — амплитуды синусов и косинусов n-й **гармоники**, а c — константа. Подобное разложение называется **рядом Фурье**. Разложенная в ряд Фурье функция может быть восстановлена по элементам этого ряда, то есть если период T и амплитуды гармоник известны, то исходная функция может быть восстановлена с помощью суммы ряда (2.1).

Информационный сигнал, имеющий конечную длительность (все информационные сигналы имеют конечную длительность), может быть разложен в ряд Фурье, если представить, что весь сигнал бесконечно повторяется снова и снова (то есть интервал от T до $2T$ полностью повторяет интервал от 0 до T и т. д.).

Амплитуды a_n могут быть вычислены для любой заданной функции $g(t)$. Для этого нужно умножить левую и правую стороны выражения (2.1) на $\sin(2\pi kft)$, а затем проинтегрировать от 0 до T. Поскольку:

$$\int_0^T \sin(2\pi kft)\sin(2\pi nft)\,dt = \begin{cases} 0 & \text{для } k \neq n; \\ T/2 & \text{для } k = n. \end{cases}$$

остается только один член ряда: a_n. Ряд b_n исчезает полностью. Аналогично, умножая выражение (2.1) на $\cos(2\pi kft)$ и интегрируя по времени от 0 до T, мы можем вычислить значения b_n. Если проинтегрировать обе части выражения, не изменяя его, то можно получить значение константы c. Результаты этих действий будут следующими:

$$a_n = \frac{2}{T}\int_0^T g(t)\sin(2\pi nft)\,dt; \quad b_n = \frac{2}{T}\int_0^T g(t)\cos(2\pi nft)\,dt; \quad c = \frac{2}{T}\int_0^T g(t)\,dt.$$

2.1.2. Сигналы с ограниченным спектром

Вы спросите, какое отношение все это имеет к передаче данных? В зависимости от физических характеристик каналов сигналы с разными частотами ведут себя в них по-разному. Рассмотрим конкретный пример — передачу двоичного кода ASCII символа «b». Для этого потребуется 8 бит (то есть 1 байт). Задача — передать следующую последовательность бит: 01100010. На рис. 2.1, *а* слева изображена зависимость выходного напряжения от времени на передающем компьютере. В результате анализа Фурье для данного сигнала получаем следующие значения коэффициентов:

$$a_n = \frac{1}{\pi n}\left[\cos(\pi n/4) - \cos(3\pi n/4) + \cos(5\pi n/4) - \cos(7\pi n/4)\right];$$

$$b_n = \frac{1}{\pi n}\left[\sin(3\pi n/4) - \sin(\pi n/4) + \sin(7\pi n/4) - \sin(6\pi n/4\right];$$

$$c = 3/4.$$

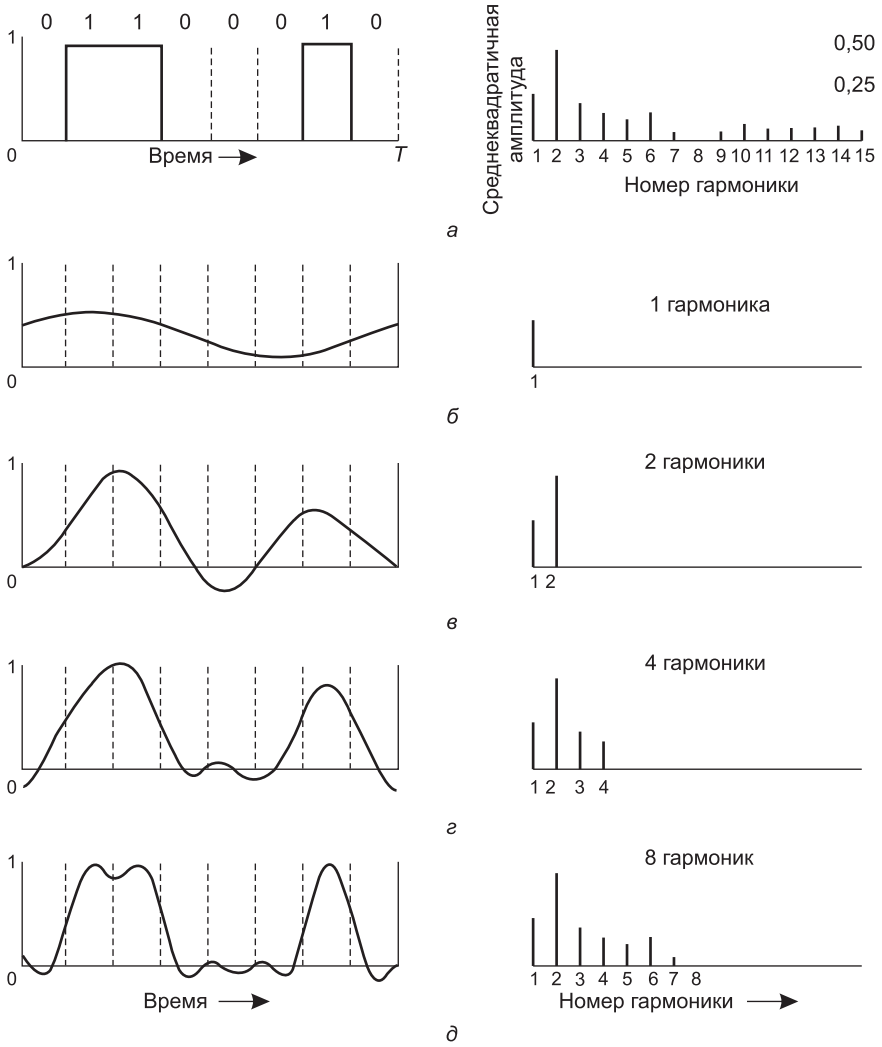


Рис. 2.1. Двоичный сигнал и его среднеквадратичные гармоники Фурье (*а*); последовательные приближения к оригинальному сигналу (*б–д*)

Среднеквадратичные амплитуды, $\sqrt{a_n^2 + b_n^2}$, для нескольких первых гармоник показаны на рис. 2.1, *а* справа. Эти значения представляют интерес, поскольку их квадраты пропорциональны энергии, передаваемой на соответствующей частоте.

Ни один канал связи не может передавать сигналы без потери мощности. Если бы все гармоники ряда Фурье уменьшались при передаче в равной степени, то сигнал уменьшался бы по амплитуде, но не искажался (то есть у него была бы та же самая замечательная прямоугольная форма, как на рис. 2.1, *а*). К сожалению, все каналы связи уменьшают гармоники ряда Фурье в разной степени, тем самым искажая передаваемый сигнал. Как правило, по кабельным сетям амплитуды передаются почти без уменьшения в частотном диапазоне от 0 до некоей частоты f_c (измеряемой в периодах

в секунду или герцах (Гц)), при этом высокочастотная составляющая сигнала (выше частоты f_c, называемой *частотой среза*) заметно ослабляется. Этот диапазон частот называется **полосой пропускания**. На практике срез вовсе не является таким резким, поэтому обычно в упомянутую выше полосу пропускания включают те частоты, которые передаются с потерей мощности, не превышающей 50 %.

Полоса пропускания является физической характеристикой среды передачи данных и зависит, например, от конструкции, толщины и длины носителя — провода или оптоволокна. Иногда для намеренного уменьшения полосы пропускания, доступной абонентам, в линию включается специальное устройство — *фильтр*. Например, беспроводным каналам стандартов 802.11 выделяется полоса пропускания шириной примерно 20 МГц, поэтому радиопередатчики соответствующим образом урезают сигнал. Еще один пример: у традиционных (аналоговых) телевизионных каналов полоса пропускания равна 6 МГц, независимо от того, передаются данные по проводам или беспроводным способом. Благодаря такой фильтрации в определенном диапазоне спектра можно передать большее количество сигналов, за счет чего повышается общая эффективность системы. Диапазон частот отдельных сигналов будет начинаться со значения, отличного от нуля, но это не играет никакой роли. Полоса пропускания — это все также некий разрешенный диапазон частот, и возможность передачи информации зависит только от его ширины, но не от начального и конечного значения частот. Сигналы, передающиеся в диапазоне частот от 0 и до верхней границы полосы, называются модулирующими сигналами. Сигналы, которые сдвигаются в верхний диапазон частот, как, например, для всех видов беспроводной передачи данных, называются сигналами в полосе.

Теперь посмотрим, как будет выглядеть сигнал, изображенный на рис. 2.1, *а*, если полоса пропускания канала будет такой, что через него будут проходить только самые низкие частоты (то есть функция $g(t)$ будет аппроксимирована лишь несколькими первыми членами рядов выражения (2.1)). На рис. 2.1, *б* показан сигнал на выходе канала, пропускающего лишь первую (основную, f) гармонику сигнала. Аналогично, рис. 2.1, *в*–*д* показывают спектры и восстановленные сигналы для каналов с более широкой полосой пропускания. Что касается цифровых данных, то главная задача — передавать их с минимальным качеством, позволяющим восстановить первоначальную последовательность битов. Для этого достаточно сигнала, показанного на рис. 2.1, *д*; следовательно, совершенно не нужно тратить ресурсы и использовать большее количество гармоник для получения более точной копии.

При заданной скорости передачи в битах, равной b бит/с, время, требуемое для передачи, скажем, 8 бит, будет равно $8/b$ секунд. Таким образом, частота первой гармоники равна $b/8$ Гц. Обычная телефонная линия, часто называемая **речевым каналом**, имеет искусственно созданную частоту среза около 3000 Гц. Это ограничение означает, что номер самой высокой гармоники, прошедшей сквозь телефонный канал, примерно (срез не очень крутой) равен $3000/(b/8)$ или $24\,000/b$.

Для некоторых скоростей передачи данных эти значения показаны в табл. 2.1. Из приведенных данных ясно, что попытка передать по речевому каналу данные на скорости 9600 бит/с превратит сигнал, показанный на рис. 2.1, *а*, в нечто подобное рис. 2.1, *в*, что сделает прием исходного потока битов с приемлемым качеством практически невозможным. Очевидно, что у сигналов, передаваемых со скоростью 38 400 бит/с и выше, нет никаких шансов пройти через речевой канал, даже при полном отсутствии

помех на линии. Другими словами, ограничение полосы пропускания частот канала ограничивает его пропускную способность для передачи *двоичных* данных даже для идеальных каналов. Однако схемы, использующие несколько уровней напряжений, существуют и позволяют достичь более высоких скоростей передачи данных. Мы обсудим это ниже в этой главе.

Таблица 2.1. Соотношение между скоростью передачи данных и числом гармоник для нашего примера

Бит/с	T, мс	1-я гармоника, Гц	Количество пропускаемых гармоник
300	26,67	37,5	80
600	13,33	75	40
1200	6,67	150	20
2400	3,33	300	10
4800	1,67	600	5
9600	0,83	1200	2
19 200	0,42	2400	1
38 400	0,21	4800	0

С термином «полоса пропускания» связано множество недоразумений, так как для инженеров-электриков и компьютерных специалистов он означает разные вещи. Для инженера-электрика (аналоговая) полоса пропускания, как уже говорилось выше, это значение в герцах, указывающее ширину диапазона частот. Для компьютерного специалиста (цифровая) полоса пропускания — это максимальная скорость данных в канале, то есть значение, измеряемое в битах в секунду. Фактически скорость данных определяется аналоговой полосой пропускания физического канала, применяемого для передачи цифровой информации, и эти два показателя связаны, как мы увидим далее. В этой книге будет понятно из контекста, какой термин имеется в виду в каждом конкретном случае — аналоговая (Гц) или цифровая (бит/с) полоса пропускания.

2.1.3. Максимальная скорость передачи данных через канал

В 1924 году американский ученый Х. Найквист (H. Nyquist) из компании AT&T пришел к выводу, что существует некая предельная скорость передачи даже для идеальных каналов. Он вывел уравнение, позволяющее найти максимальную скорость передачи данных в бесшумном канале с ограниченной полосой пропускания частот. В 1948 году Клод Шеннон (Claude Shannon) продолжил работу Найквиста и расширил ее для случая канала со случайным (то есть термодинамическим) шумом. Это важнейшая работа во всей теории передачи информации. Мы кратко рассмотрим результаты работы Найквиста и Шеннона, ставшие сегодня классическими.

Найквист доказал, что если произвольный сигнал прошел через низкочастотный фильтр с полосой пропускания B, то такой отфильтрованный сигнал может быть полностью восстановлен по дискретным значениям этого сигнала, измеренным с частотой

$2B$ в секунду. Производить измерения сигнала чаще, чем $2B$ в секунду, нет смысла, так как более высокочастотные компоненты сигнала были отфильтрованы. Если сигнал состоит из V дискретных уровней, то уравнение Найквиста будет выглядеть так:

$$\text{максимальная скорость передачи данных} = 2B \log_2 V, \text{ бит/с}.$$

Так, например, бесшумный канал с частотой пропускания в 3 кГц не может передавать двоичные (то есть двухуровневые) сигналы на скорости, превосходящей 6000 бит/с.

Итак, мы рассмотрели случай бесшумных каналов. При наличии в канале случайного шума ситуация резко ухудшается. Уровень термодинамического шума в канале измеряется отношением мощности сигнала к мощности шума и называется **отношением сигнал/шум**. Если обозначить мощность сигнала S, а мощность шума — N, то отношение сигнал/шум будет равно S/N. Обычно величина отношения выражается через ее десятичный логарифм, умноженный на 10: $10 \lg S/N$, так как ее значение может меняться в очень большом диапазоне. Единица такой логарифмической шкалы называется **децибелом** (decibel, dB, дБ); здесь приставка «деци» означает «десять», а «бел» — это единица измерения, названная в честь изобретателя телефона Александра Грэма Белла. Таким образом, отношение сигнал/шум, равное 10, соответствует 10 дБ, отношение, равное 100, равно 20 дБ, отношение, равное 1000, равно 30 дБ и т. д. Производители стереоусилителей часто указывают полосу частот (частотный диапазон), в которой их аппаратура имеет линейную амплитудно-частотную характеристику в пределах 3 дБ. Отклонение в 3 дБ соответствует ослаблению сигнала примерно в два раза (потому что $10 \log_{10} 0{,}5 \approx -3$).

Главным результатом, который получил Шеннон, было утверждение о том, что максимальная скорость передачи данных или емкость канала с полосой частот B Гц и отношением сигнал/шум, равным S/N, вычисляется по формуле:

$$\text{максимальная скорость передачи данных} = B \log_2(1 + S/N), \text{ бит/с}.$$

Это наилучшее значение емкости, которое можно наблюдать для реального канала. Например, полоса пропускания канала ADSL (Asymmetric Digital Subscriber Line, ассиметричная цифровая абонентская линия), по которому осуществляется доступ в Интернет через телефонные сети, равна приблизительно 1 МГц. Отношение сигнал/шум в значительной степени зависит от расстояния между компьютером пользователя и телефонной станцией. Для коротких линий длиной от 1 до 2 км очень хорошим считается значение около 40 дБ. С такими характеристиками канал никогда не сможет передавать более 13 Мбит/с, независимо от способа модуляции сигнала, то есть количества используемых уровней сигнала, частоты дискретизации и т. д. Поставщики услуг заявляют скорость передачи данных до 12 Мбит/с, однако пользователям редко удается наблюдать такое качество передачи данных. Тем не менее это великолепный результат для шестидесяти лет развития технологий передачи информации, в течение которых произошел огромный скачок от емкости каналов, характерной для времен Шеннона, и до существующей в современных реальных сетях.

Результат, полученный Шенноном и подкрепленный постулатами теории информации, применим к любому каналу с Гауссовским (термальным) шумом. Попытки доказать обратное заранее обречены на провал. Для того чтобы добиться в канале ADSL скорости, превышающей 13 Мбит/с, необходимо либо улучшить отношение

сигнал/шум (например, добавив цифровые повторители в линии передачи данных, подходящие к компьютерам пользователей), либо расширить полосу пропускания, как это сделано в новой версии технологии, ADSL2+.

2.2. Проводниковые среды передачи информации

Назначением физического уровня сети является передача битов от одной машины к другой. Для передачи могут использоваться различные физические носители информации, называемые также средой распространения сигнала. Каждый из них имеет характерный набор полос пропускания, задержек, цен и простоты установки и использования. Среды передачи информации можно разделить на две группы: проводниковые среды, такие как медный провод и оптоволоконный кабель, и беспроводные, например предназначенные для наземной беспроводной и спутниковой связи, а также передача по лазерному лучу без кабеля. Мы рассмотрим проводниковые среды в этом разделе, а беспроводные — в последующих.

2.2.1. Магнитные носители

Один из самых простых способов перенести данные с одного компьютера на другой — записать их на магнитную ленту или другой съемный носитель (например, перезаписываемый DVD), физически перенести эти ленты и диски к пункту назначения и там прочитать их. Поскольку такой метод значительно проще применения, скажем, геостационарного спутника связи, он часто оказывается гораздо более эффективным в экономическом отношении, особенно для приложений, в которых высокая пропускная способность или цена за бит являются ключевыми факторами.

Разобраться в данном вопросе нам помогут несложные вычисления. Стандартная кассета с лентой Ultrium вмещает 800 Гбайт. В коробку размером 60 × 60 × 60 см помещается около 1000 таких кассет, что дает общую емкость 800 терабайт или 6400 терабит (6,4 петабит). Коробка с кассетами может быть доставлена в пределах США в течение 24 часов службой Federal Express или другой компанией. Эффективная полоса пропускания при такой передаче составляет 6400 терабит/86 400 с или немногим больше 70 Гбит/с. Если же пункт назначения находится всего в часе езды, то пропускная способность составит свыше 1700 Гбит/с. Ни одна компьютерная сеть пока не в состоянии даже приблизиться к таким показателям.

Если мы теперь взглянем на этот вопрос с экономической точки зрения, то получим сходную картину. Оптовая цена кассеты составляет около $40. Коробка с лентами обойдется в $4000, при этом одну и ту же ленту можно использовать десятки раз. Прибавим $1000 на перевозку (а на самом деле, гораздо меньше), и мы получим около $5000 за передачу 800 Тбайт или чуть более половины цента за гигабайт. Ни одна сеть на земле не может соперничать с этим. Мораль этой истории такова.

Не думай свысока о скорости передачи данных автомобилем, полным кассет, с грохотом передвигающимся по дороге.

2.2.2. Витая пара

Хотя скорость передачи данных с помощью магнитных лент отличная, однако величина задержки при такой передаче очень велика. Время передачи измеряется минутами или часами, а не миллисекундами. Для многих приложений требуется мгновенная реакция удаленной системы (в подключенном режиме). Одним из первых и до сих пор часто применяемых средств передачи является витая пара. Этот носитель состоит из двух изолированных медных проводов, обычный диаметр которых составляет 1 мм. Провода свиваются один вокруг другого в виде спирали, чем-то напоминая молекулу ДНК. Это позволяет уменьшить электромагнитное взаимодействие нескольких расположенных рядом витых пар. (Два параллельных провода образуют простейшую антенну, витая пара — нет.) Сигнал обычно передается в виде разницы потенциалов в двух проводах, составляющих пару. Это обеспечивает лучшую устойчивость к внешнему шуму, так как шум одинаково влияет на оба провода, и, таким образом, разница потенциалов остается неизменной.

Самым распространенным применением витой пары является телефонная линия. Почти все телефоны соединяются с телефонными компаниями при помощи этого носителя. По витой паре передаются не только телефонные звонки; доступ в Интернет по технологии ADSL также осуществляется через витую пару. Витая пара может передавать сигнал без ослабления мощности на расстояние, составляющее несколько километров. На более дальних расстояниях из-за ослабевания сигнала требуются повторители. Большое количество витых пар, тянущихся на большое расстояние в одном направлении, объединяются в кабель, на который надевается защитное покрытие. Если бы пары проводов, находящиеся внутри таких кабелей, не были свиты, то сигналы, проходящие по ним, накладывались бы друг на друга. Телефонные кабели диаметром несколько сантиметров можно видеть протянутыми на столбах.

Витые пары могут использоваться для передачи как аналоговых, так и цифровых данных. Полоса пропускания зависит от диаметра и длины провода, но в большинстве случаев на расстоянии до нескольких километров может быть достигнута скорость несколько мегабит в секунду. Благодаря довольно высокой пропускной способности и небольшой цене витые пары широко распространены и, скорее всего, будут популярны и в будущем.

Витые пары применяются в нескольких вариантах. В офисных зданиях наиболее распространена витая пара категории 5 или «Cat 5». Витая пара категории 5 состоит из двух изолированных проводов, свитых друг с другом. Четыре такие пары обычно помещаются вместе в пластиковую оболочку. Подобная схема показана на рис. 2.2.

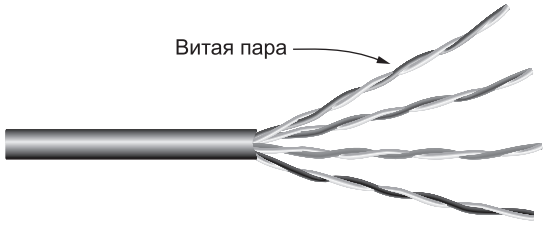


Рис. 2.2. Кабель UTP категории 5 с четырьмя витыми парами

В сетях разных стандартов витая пара используется по-разному. Например, в 100-мегабитной сети Ethernet данные передаются по двум (из четырех) парам, по одной паре в каждом направлении.

Для того чтобы достичь более высокой скорости, в 1-гигабитных сетях Ethernet данные передаются по всем четырем парам одновременно в обоих направлениях. Во избежание ошибок принимающее устройство должно уметь отделять сигнал, передаваемый локально.

Немного сухой терминологии. Линии, по которым данные могут одновременно передаваться в обе стороны, как на улице с двухсторонним движением, называются дуплексными. А те линии, по которым данные в каждый момент времени могут пересылаться лишь в одном направлении, как на дороге с реверсивным движением, называются полудуплексными. Третья категория — это дороги с односторонним движением — линии, по которым передача сигнала возможна только в одну сторону. Они называются симплексными.

Возвращаясь к витой паре, на смену витой паре третьей категории пришла витая пара категории 5. У витой пары пятой категории такой же разъем, но число витков на метр длины проводов больше, чем у кабеля третьей категории. Большее число витков обеспечивает лучшее качество передачи сигнала на большие расстояние и уменьшает наводки. Таким образом, витая пара категории 5 лучше подходит для высокоскоростной компьютерной связи, особенно для 100-мегабитных и 1-гигабитных сетей Ethernet.

Новым стандартом станут, вероятно, кабели категорий 6 или даже 7. Жесткие спецификации этих категорий обеспечивают обработку сигналов с большой полосой пропускания. Некоторые кабели категории 6 и выше предназначены для сигналов с частотой 500 МГц; их можно использовать в 10-гигабитных сетях, развертывание которых планируется в ближайшем будущем.

Вплоть до категории 6, все эти типы соединений часто называются UTP (unshielded twisted pair — неэкранированная витая пара), так как они состоят всего лишь из проводов и изоляции. В противоположность им, в кабелях категории 7 экранированы не только отдельные витые пары, но и весь кабель (помещенный в защитную пластиковую оболочку). Экранирование снижает чувствительность к внешним помехам и наводки между соседними кабелями. Благодаря этому витая пара седьмой категории отвечает высоким требованиям к производительности. Кабель категории 7 напоминает высококачественные, но громоздкие экранированные кабели из витых пар корпорации IBM, которые она представила на рынке в 1980 году. Они так и не стали популярными за пределами фирмы IBM. Очевидно, настало время попробовать еще раз.

2.2.3. Коаксиальный кабель

Другим распространенным средством передачи данных является коаксиальный кабель. Он лучше экранирован, чем витая пара, поэтому может обеспечить передачу данных на более дальние расстояния с более высокими скоростями. Широко применяются два типа кабелей. Один из них, 50-Омный, обычно используется для передачи исключительно цифровых данных. Другой тип кабеля, 75-Омный, часто применяется для передачи аналоговой информации, а также в кабельном телевидении. В основе такого разделения лежат скорее исторические, нежели технические факторы

(например, первые дипольные антенны имели импеданс 300 Ом, и проще всего было использовать уже существующие преобразователи с отношением импеданса 4:1). Начиная с середины 1990-х годов операторы кабельного телевидения стали также предоставлять услуги доступа к Интернету, что сделало 75-Омный кабель еще более значимой средой передачи информации.

Коаксиальный кабель состоит из твердого медного провода, расположенного в центре кабеля, покрытого изоляцией. Поверх изоляции натянут цилиндрический проводник, обычно выполненный в виде мелкой медной сетки. Он покрыт наружным защитным слоем изоляции (пластиковой оболочкой). Вид кабеля в разрезе показан на рис. 2.3.

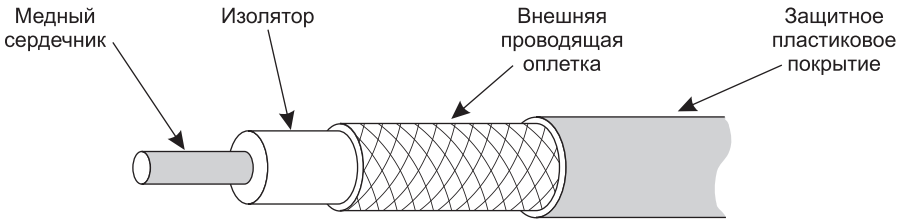


Рис. 2.3. Коаксиальный кабель

Конструкция и специальный тип экранирования коаксиального кабеля обеспечивает высокую пропускную способность и отличную помехозащищенность. Максимальная пропускная способность зависит от качества и длины. Современные кабели имеют полосу пропускания до нескольких гигагерц. Коаксиальные кабели широко применялись в телефонных системах, но теперь на линиях большой протяженности они все чаще заменяются оптоволоконными кабелями. Однако коаксиальные кабели все еще широко используются для кабельного телевидения, а также в некоторых региональных сетях.

2.2.4. Линии электропитания

Телефонные сети и кабельное телевидение — не единственные линии передачи данных, которые можно приспособить для пересылки информации. Есть и еще одна распространенная проводная сеть: электрическая. По линиям электропитания электричество поставляется в дома. Внутри домов электричество по проводам подводится к розеткам.

Идея использовать линии электропитания для передачи данных возникла довольно давно. Электрические компании таким способом много лет удаленно снимают показания. Кроме того, подобная низкоскоростная передача данных позволяет управлять различными домашними устройствами (например, по стандарту X10). В последние годы снова возродился интерес к высокоскоростной передаче информации по линиям электропитания, как внутри дома, так и за его пределами — для доступа к Интернету. Мы рассмотрим наиболее распространенный вариант: использование электропроводки внутри дома.

Удобство такого варианта очевидно. Просто подключите телевизор и ресивер к стенной розетке (это необходимо сделать в любом случае, так как устройствам нужно электропитание), и они смогут отправлять и получать фильмы по электрическим про-

водам. Подобная конфигурация показана на рис. 2.4. Никаких других разъемов или точек передачи радиосигналов нет. Сигнал данных накладывается на низкочастотный сигнал электропитания (на активном проводе — том, что находится под напряжением), и оба сигнала передаются по проводам одновременно.

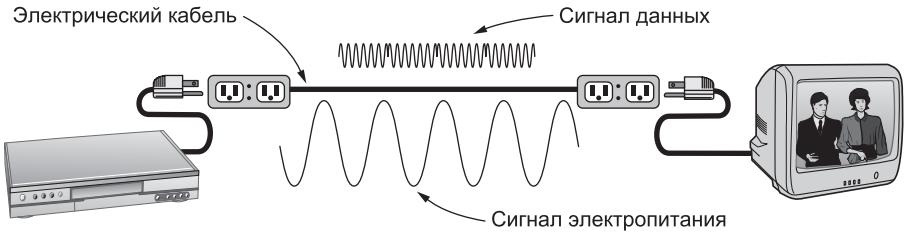


Рис. 2.4. Сеть на основе домашней электропроводки

Сложность применения домашней электропроводки для построения сети передачи данных заключается в том, что основное ее предназначение — это распределение электроэнергии. Очевидно, что эти две задачи кардинально различаются. Электрические сигналы пересылаются с частотой 50–60 Гц, а более высокочастотные сигналы (их частота измеряется с единиц мегагерц) затухают. Свойства проводки в разных домах могут сильно отличаться, к тому же они меняются при включении и выключении бытовых приборов, что приводит к нестабильности сигналов данных. Неустановившийся ток при включении или выключении устройства создает электрический шум в большом диапазоне частот. А без тщательного скручивания (как в витой паре) электрическая проводка действует как тонкая антенна, собирая внешние сигналы и излучая собственные. Такое поведение означает, что для того чтобы отвечать нормативам стандартов, сигналам данных нужно исключать лицензированные частоты, такие как частоты, выделенные для любительских радиостанций.

Несмотря на перечисленные трудности, по обычной домашней электросети можно отправлять данные со скоростью как минимум 100 Мбит/с, используя схемы с предотвращением ослабления сигнала и подавлением ошибок. Для многих продуктов применяются запатентованные стандарты передачи данных по линиям электропитания, но международные стандарты пока что находятся в активной разработке.

2.2.5. Волоконная оптика

Быстрое развитие компьютерных технологий в соответствии с законом Мура (который предсказал, что число транзисторов на кристалле будет удваиваться приблизительно каждые два годы) вызывает чувство гордости у многих представителей этой индустрии. Первый персональный компьютер фирмы IBM, созданный в 1981 году, работал с тактовой частотой 4,77 МГц. Спустя 28 лет этот показатель вырос до 3 ГГц на четырехъядерных процессорах. Прирост множителя составил около 2500 или 16 за декаду. Не так уж плохо.

За тот же период скорость передачи данных выросла с 45 Мбит/с (линия T3 по телефонным проводам) до 100 Гбит/с (современная длинная линия), это означает не менее впечатляющий рост в 2000 раз или 16 раз за 10 лет. При этом вероятность

ошибки при передаче уменьшилась с 105 на бит почти до нуля. Помимо этого, процессоры начинают приближаться к своим физическим пределам, поэтому теперь на одном кристалле их используется сразу несколько. Существующая ныне оптоволоконная технология, напротив, может развивать скорость передачи данных вплоть до 50 000 Гбит/с (50 Тбит/с), и до достижения ее физического предела нам еще далеко. Сегодняшний практический предел в 100 Гбит/с обусловлен нашей неспособностью быстрее преобразовывать электрические сигналы в оптические и обратно. Для того чтобы достичь более высокой скорости, по одному волокну просто одновременно передаются данные нескольких каналов.

В этом разделе мы познакомимся с оптическим волокном и узнаем, как данные передаются по оптоволокну. В гонке компьютеров и средств связи у последних еще есть шанс на победу — благодаря волоконной оптике. Если это произойдет, то в мире появится не только совершенно новое понятие о почти бесконечной полосе пропускания, но и неслыханная доселе идея о том, что все компьютеры безнадежно медленны и сетям следует любой ценой избегать вычислений, независимо от того, какая часть полосы пропускания при этом будет потеряна. Необходимо время, чтобы изменения впитались в умы поколений ученых-компьютерщиков и инженеров, приученных думать в терминах низкоскоростных медных линий и ограничений, сформулированных Шенноном.

Конечно, в этом представлении не хватает одной немаловажной детали: стоимости. Затраты на прокладку оптоволокна до компьютера каждого пользователя, чтобы обойти характерные для проводов ограничения — низкую полосу пропускания и небольшой диапазон частот, — попросту огромны. Помимо этого, на пересылку битов тратится больше энергии, чем на вычисления. Всегда будут существовать островки неравенства, в которых стоимость либо вычислений, либо пересылки данных будет приближаться к нулю. Например, перед тем как выйти в Интернет, мы применяем все имеющиеся вычислительные возможности и расходуем место на диске, чтобы решить проблему сжатия и кэширования содержимого — все для того, чтобы наиболее эффективно воспользоваться доступом к Всемирной сети. В Интернете же может происходить обратное. Такие компании, как Google, перемещают по сети огромные объемы данных, сбрасывая их туда, где хранение и обработка будут стоить дешевле.

Оптоволокно используется для пересылки информации на очень большие расстояния по сетевым магистральным соединениям, внутри высокоскоростных локальных сетей (хотя пока что ему не удается достаточно далеко уйти вперед от медных проводов) и для высокоэффективного доступа в Интернет, например, по технологии FTTH (Fiber to the Home — волокно прямо к дому). Оптоволоконная система передачи данных состоит из трех основных компонентов: источника света, носителя, по которому распространяется световой сигнал, и приемника сигнала, или детектора. Световой импульс принимают за единицу, а отсутствие импульса — за ноль. Свет распространяется в сверхтонком стеклянном волокне. При попадании на него света детектор генерирует электрический импульс. Присоединив к одному концу оптического волокна источник света, а к другому — детектор, мы получим однонаправленную систему передачи данных. Система принимает электрические сигналы и преобразует их в световые импульсы, передающиеся по волокну. На другой стороне происходит обратное преобразование в электрические сигналы.

Такая передающая система была бы бесполезна, если бы свет по дороге рассеивался и терял свою мощность. Однако в данном случае используется один интересный физический закон. Когда луч света переходит из одной среды в другую, например из стекла (расплавленного и застывшего кварца) в воздух, луч отклоняется (эффект рефракции или преломления) на границе стекло—воздух, как показано на рис. 2.5, *а*. Здесь мы видим, что луч света падает под углом $\alpha 1$, выходя под углом β_1. Соотношение углов падения и отражения зависит от свойств смежных сред (в частности, от их коэффициентов преломления). Если угол падения превосходит некоторую критическую величину, луч света целиком отражается обратно в стекло, а в воздух ничего не проходит. Таким образом, луч света, падающий на границу сред под углом, превышающим критический, оказывается запертым внутри волокна, как показано на рис. 2.5, б, и может быть передан на большое расстояние почти без потерь.

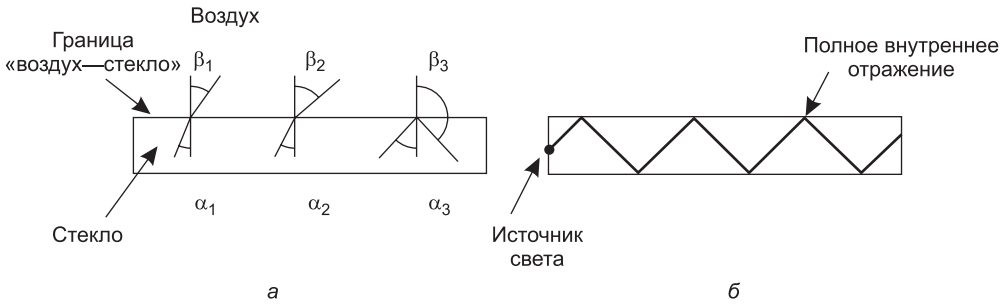


Рис. 2.5. Три примера преломления луча света, падающего под разными углами, на границе кварцевого волокна и воздуха (*а*); луч света, пойманный полным внутренним отражением (*б*)

На рис. 2.5, *б* показан только один пойманный луч света, однако поскольку любой луч света с углом падения, превышающим критический, будет отражаться от стенок волокна, то и множество лучей будет одновременно отражаться под различными углами. Про каждый луч говорят, что он обладает некоторой модой, а оптическое волокно, обладающее свойством передавать сразу несколько лучей, называется многомодовым.

Однако если уменьшить диаметр волокна до нескольких длин волн света, то волокно начинает действовать подобно волноводу, и свет может двигаться только по прямой линии, без отражений от стенок волокна. Такое волокно называется одномодовым. Оно стоит дороже, но может использоваться при передаче данных на большие расстояния. Сегодняшние одномодовые волоконные линии могут работать со скоростью 100 Гбит/с на расстоянии до 100 км. В лабораториях были достигнуты и более высокие скорости, правда, на меньших дистанциях.

Прохождение света по волокну

Оптическое волокно изготавливается из стекла, которое, в свою очередь, производится из песка — недорогого необработанного материала, доступного в неограниченных количествах. Изготовление стекла было известно уже в древнем Египте, однако, чтобы свет мог проникнуть сквозь стекло, его толщина не должна превышать 1 мм, чего в то

время было невозможно достичь. Стекло, достаточно прозрачное, чтобы его можно было использовать в окнах зданий, было изобретено в эпоху Возрождения. Для современных оптических кабелей применяется настолько прозрачное стекло, что если бы океаны вместо воды состояли из него, то дно океана было бы так же ясно видно, как поверхность суши с борта самолета в ясный день.

Ослабление силы света при прохождении через стекло зависит от длины волны (а также от некоторых физических свойств стекла). Оно определяется в виде отношения мощности входного сигнала к мощности выходного сигнала. Для стекла, используемого в оптическом волокне, зависимость ослабления от длины волны показана на рис. 2.6 в децибелах на километр длины волокна. Например, ослаблению мощности в два раза соответствует на графике $10 \lg 2 = 3$ дБ. На графике изображена ближняя инфракрасная часть спектра, используемая на практике. Видимый свет имеет несколько более короткие длины волн — от 0,4 до 0,7 мкм (1 мкм или 1 микрон равен 10^{-6} метра). Приверженцы точных наименований сказали бы, что длина волны измеряется в нанометрах — в данном случае речь о диапазоне от 400 до 700 нм, — однако мы будем использовать более привычные термины.

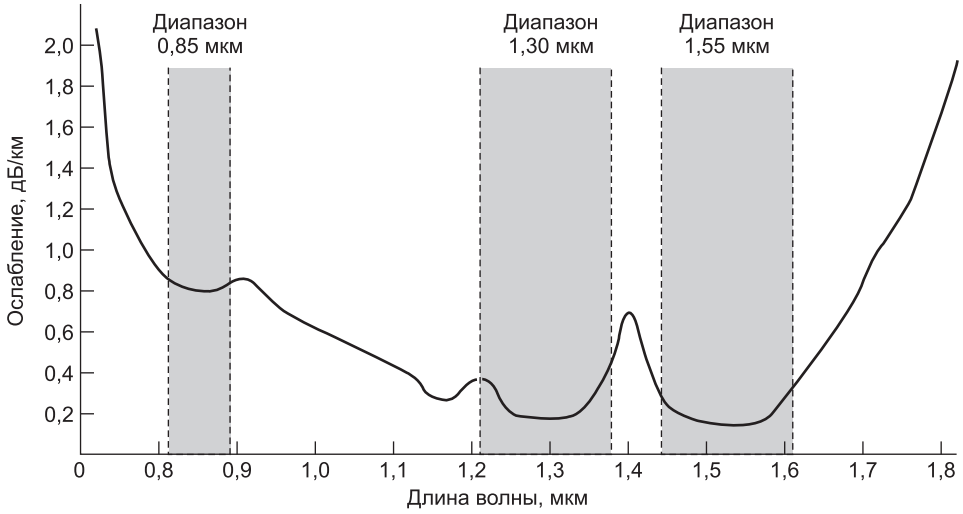


Рис. 2.6. Ослабление света в инфракрасной области спектра при прохождении через оптическое волокно

В системах связи используются три диапазона длин волн: 0,85, 1,30 и 1,55 мкм. Все три диапазона обладают полосой пропускания от 25 000 до 30 000 ГГц. Первым стал применяться диапазон с центром 0,85 мкм. Он обладает более высоким ослаблением, поэтому используется для передачи на короткие расстояния. Однако его преимуществом является то, что для этой длины волны лазеры и электроника могут быть сделаны из одного и того же материала (арсенида галлия). У двух остальных диапазонов показатели по ослаблению лучше (менее 5 % потерь на километр). В настоящее время широко используется диапазон 1,55 мкм и волоконные усилители с добавкой эрбия, которые работают прямо в оптическом домене.

Световые импульсы удлиняются по мере их продвижения по волокну. Это удлинение называется световой дисперсией. Величина удлинения зависит от длины волны. Чтобы не допустить перекрывания соседних расширяющихся импульсов, можно увеличить расстояние между ними, однако при этом придется уменьшить скорость передачи. К счастью, было обнаружено, что эффект дисперсии можно предотвратить, если придавать импульсам специальную форму, а именно обратной величины от гиперболического косинуса. В этом случае будет возможно посылать импульсы на тысячи километров без искажения формы. Такие импульсы называются уединенными волнами или солитонами. Значительная часть исследователей намерена перейти от лабораторных исследований уединенных волн к их промышленному использованию.

Оптоволоконные кабели

Структура оптоволоконного кабеля схожа с описанной выше структурой коаксиального провода. Разница состоит лишь в том, что в первом нет экранирующей сетки. На рис. 2.7, *а* показана отдельная оптоволоконная жила. В центре ее располагается стеклянная сердцевина, по которой распространяется свет. В многомодовом оптоволокне диаметр сердечника составляет 50 мкм, что примерно равно толщине человеческого волоса. Сердечник в одномодовом волокне имеет диаметр от 8 до 10 мкм.

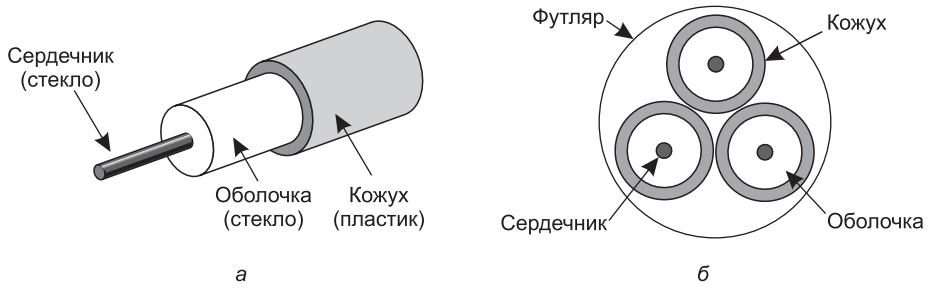


Рис. 2.7. Вид одиночного волокна сбоку (*а*); поперечное сечение трехжильного кабеля (*б*)

Сердечник покрыт слоем стекла с более низким, чем у сердечника, коэффициентом преломления. Он предназначен для более надежного предотвращения выхода света за пределы сердечника. Внешним слоем служит пластиковая оболочка, защищающая остекление. Оптоволоконные жилы обычно группируются в пучки, защищенные внешней оболочкой. На рис. 2.7, *б* показан трехжильный кабель.

Обычно кабели кладутся в грунт на глубину около метра, где их могут случайно повредить грызуны или экскаватор. У побережья трансокеанические кабели укладываются в траншеи специальным механизмом. На большой глубине их обычно просто кладут на дно, где их могут зацепить рыболовные траулеры или перегрызть акулы.

Соединение отрезков кабеля может осуществляться тремя способами. Во-первых, на конец кабеля может прикрепляться специальный разъем, с помощью которого кабель вставляется в оптическую розетку. Подобное соединение приводит к потере 10–20 % силы света, зато оно позволяет легко изменить конфигурацию системы.

Во-вторых, они могут механически сращиваться — два аккуратно отрезанных конца кабеля укладываются рядом друг с другом и зажимаются специальной муфтой. Улучшение прохождения света достигается выравниванием концов кабеля. При этом через соединение пропускается свет, и задачей является добиться максимального соответствия мощности выходного сигнала мощности входного. Одно механическое сращивание кабелей занимает у опытного монтажника сетей около 5 минут и дает в результате потерю 10 % мощности света.

В-третьих, два куска кабеля могут быть сплавлены вместе. Сплавное соединение почти так же хорошо, как и сплошной кабель, но даже при таком методе происходит небольшое уменьшение мощности света.

Во всех трех типах соединений в точке соединения могут возникнуть отражения, и отраженный свет может интерферировать с сигналом.

Для передачи сигнала по оптоволоконному кабелю могут использоваться два типа источника света: светоизлучающие диоды (LED, Light Emitting Diode) и полупроводниковые лазеры. Они обладают различными свойствами, как показано в табл. 2.2. Их длина волны может быть настроена при помощи интерферометров Фабри—Перо (Fabry—Perot) или Маха—Цандера (Mach—Zehnder), устанавливаемых между источником и кабелем. Интерферометры Фабри—Перо представляют собой простые резонансные углубления, состоящие из двух параллельных зеркал. Свет падает перпендикулярно зеркалам, углубление отбирает те длины волн, которые укладываются в его размер целое число раз. Интерферометры Маха—Цандера разделяют свет на два луча, которые проходят различное расстояние и снова соединяются на выходе. Синфазными на выходе интерферометра окажутся лучи строго определенной длины.

Таблица 2.2. Сравнительные характеристики светодиодов и полупроводниковых лазеров

Характеристика	Светодиод	Полупроводниковые лазеры
Скорость передачи данных	Низкая	Высокая
Тип волокна	Многомодовые	Многомодовые или одномодовые
Расстояние	Короткое	Дальнее
Срок службы	Долгий	Короткий
Чувствительность к температуре	Невысокая	Значительная
Цена	Низкая	Высокая

Приемный конец оптического кабеля представляет собой фотодиод, генерирующий электрический импульс, когда на него падает свет. Обычное время срабатывания фотодиода, который преобразует оптический сигнал в электрический, ограничивает скорость передачи данных 100 Гбит/с. Термальный шум также имеет место, поэтому импульс света должен быть довольно мощным, чтобы его можно было обнаружить на фоне шума. При достаточной мощности импульса можно добиться пренебрежимо малой частоты ошибок.

Сравнение характеристик оптического волокна и медного провода

Сравнение характеристик оптического волокна и медного провода весьма поучительно. Оптическое волокно обладает рядом преимуществ. Во-первых, оно обеспечивает значительно более высокие скорости передачи, чем медный провод. Уже благодаря этому именно оптическое волокно должно применяться в высококачественных профессиональных сетях. Благодаря низкому коэффициенту ослабления повторители для оптоволоконной связи требуются лишь через каждые 50 км, по сравнению с 5 км для медных проводов, что существенно снижает затраты для линий дальней связи. Преимуществом оптического волокна также является его толерантность по отношению к внешним электромагнитным возмущениям. Оно не подвержено коррозии, поскольку стекло является химически нейтральным. Это важно для применения на химических предприятиях.

Это может показаться странным, но телефонные компании любят оптическое волокно еще по одной причине: оно тонкое и легкое. Многие каналы для кабелей заполнены до отказа, так что новый кабель некуда положить. Если вынуть из такого канала все медные кабели и заменить их оптическими, то останется еще много свободного места, а медь можно очень выгодно продать скупщикам цветного металла. Кроме того, оптический кабель значительно легче медного. Тысяча медных витых пар длиной в 1 км весят около 8000 кг. Пара оптоволоконных кабелей весит всего 100 кг при гораздо большей пропускной способности, что снижает затраты на дорогие механические системы. При прокладке новых маршрутов оптоволоконные кабели выигрывают у медных благодаря гораздо более низким затратам на их прокладку. Наконец, оптоволоконные кабели не теряют свет и к ним сложно подключиться, что способствует их надежности и сохранности.

Отрицательной стороной оптоволоконной технологии является то, что для работы с ней требуются определенные навыки, которые имеются далеко не у всех инженеров. Кабель довольно хрупкий и ломается в местах сильных изгибов. Кроме того, поскольку оптическая передача данных является строго однонаправленной, для двухсторонней связи требуется либо два кабеля, либо две частотные полосы в одном кабеле. Наконец, оптический интерфейс стоит дороже электрического. Тем не менее, очевидно, что будущее цифровой связи на расстояниях более нескольких метров — за волоконной оптикой. Подробнее обо всех аспектах оптоволоконных сетей см. в книге (Hecht, 2005).

2.3. Беспроводная связь

В наше время появляется все большее количество информационных «наркоманов» — людей с потребностью постоянно находиться в подключенном режиме (on-line). Таким пользователям никакие кабельные соединения, будь то витая пара, коаксиальный кабель или оптическое волокно, не подходят. Им требуется получать данные непосредственно на переносные компьютеры, лэптопы, ноутбуки, электронные записные книжки, карманные компьютеры, палмтопы и компьютеры, встроенные в наручные часы. Короче говоря, они предпочитают пользоваться устройствами, не привязанными

к наземным инфраструктурам. Для таких пользователей беспроводная связь является необходимостью.

В следующих разделах мы познакомимся с основами беспроводной связи. У нее есть ряд других важных применений, кроме предоставления доступа в Интернет желающим побродить по нему, лежа на пляже. При некоторых обстоятельствах беспроводная связь может иметь свои преимущества и для стационарных устройств. Например, если прокладка оптоволоконного кабеля осложнена природными условиями (горы, джунгли, болота и т. д.), то беспроводная связь может оказаться предпочтительнее. Следует отметить, что современная беспроводная связь зародилась на Гавайских островах, где людей от компьютерных центров отделяли большие пространства Тихого океана, а качество обычной телефонной системы было далеко не на самом высоком уровне.

2.3.1. Электромагнитный спектр

Движение электронов порождает электромагнитные волны, которые могут распространяться в пространстве (даже в вакууме). Это явление было предсказано британским физиком Джеймсом Клерком Максвеллом (James Clerk Maxwell) в 1865 году Первый эксперимент, при котором их можно было наблюдать, поставил немецкий физик Генрих Герц (Heinrich Hertz) в 1887 году. Число электромагнитных колебаний в секунду называется **частотой**, f, и измеряется в герцах (в честь Генриха Герца). Расстояние между двумя последовательными максимумами (или минимумами) называется **длиной волны**. Эта величина традиционно обозначается греческой буквой λ (лямбда).

Если в электрическую цепь включить антенну подходящего размера, то электромагнитные волны можно с успехом принимать приемником на некотором расстоянии. На этом принципе основаны все беспроводные системы связи.

В вакууме все электромагнитные волны распространяются с одной и той же скоростью, независимо от их частоты. Эта скорость называется **скоростью света**, c. Ее величина приблизительно равна 3×108 м/с, или около одного фута (30 см) за наносекунду. (Можно было бы переопределить, воспользовавшись таким совпадением, фут, постановив, что он равен расстоянию, которое проходит электромагнитная волна в вакууме за 1нс. Это было бы логичнее, чем измерять длины размером сапога какого-то давно умершего короля.) В меди или стекле скорость света составляет примерно 2/3 от этой величины, кроме того, слегка зависит от частоты. Скорость света современная наука считает верхним пределом скоростей. Быстрее не может двигаться никакой объект или сигнал.

Величины f, λ и c (в вакууме) связаны фундаментальным соотношением:

$$\lambda f = c. \tag{2.2}$$

Поскольку c является константой, то, зная f, мы можем определить λ, и наоборот. Существует мнемоническое правило, которое гласит, что $\lambda f \approx 300$, если λ измеряется в метрах, а f — в мегагерцах. Например, волны с частотой 100 МГц имеют длину волны около трех метров, 1000 МГц соответствует 0,3 м, а длине волны 0,1 м соответствует частота 3000 МГц.

На рис. 2.8 изображен электромагнитный спектр. Радио, микроволновый, инфракрасный диапазоны, а также видимый свет могут быть использованы для передачи информации с помощью амплитудной, частотной или фазовой модуляции волн. Ультрафиолетовое, рентгеновское и гамма-излучения были бы даже лучше благодаря их высоким частотам, однако их сложно генерировать и модулировать, они плохо проходят сквозь здания и, кроме того, они опасны для всего живого. Диапазоны, перечисленные в нижней части рис. 2.8, представляют собой официальные названия ITU (International Telecommunication Union, международное телекоммуникационное сообщество), основанные на длинах волн. Так, например, низкочастотный диапазон (LF, Low Frequency) охватывает длины волн от 1 до 10 км (что приблизительно соответствует диапазону частот от 30 до 300 кГц). Сокращения LF, MF и HF обозначают Low Frequency (низкая частота), Medium Frequency (средняя частота) и High Frequency (высокая частота) соответственно. Очевидно, при назначении диапазонам названий никто не предполагал, что будут использоваться частоты выше 10 МГц, поэтому более высокие диапазоны получили названия VHF (very high frequency — очень высокая частота), UHF (ultrahigh frequency — ультравысокая частота, УВЧ), SHF (superhigh frequency — сверхвысокая частота, СВЧ), EHF (Extremely High Frequency — чрезвычайно высокая частота) и THF (Tremendously High Frequency — ужасно высокая частота). Выше последнего диапазона имена пока не придуманы, но если следовать традиции, появятся диапазоны Невероятно (Incredibly), Поразительно (Astonishingly) и Чудовищно (Prodigiously) высоких частот (ITF, ATF и PTF).

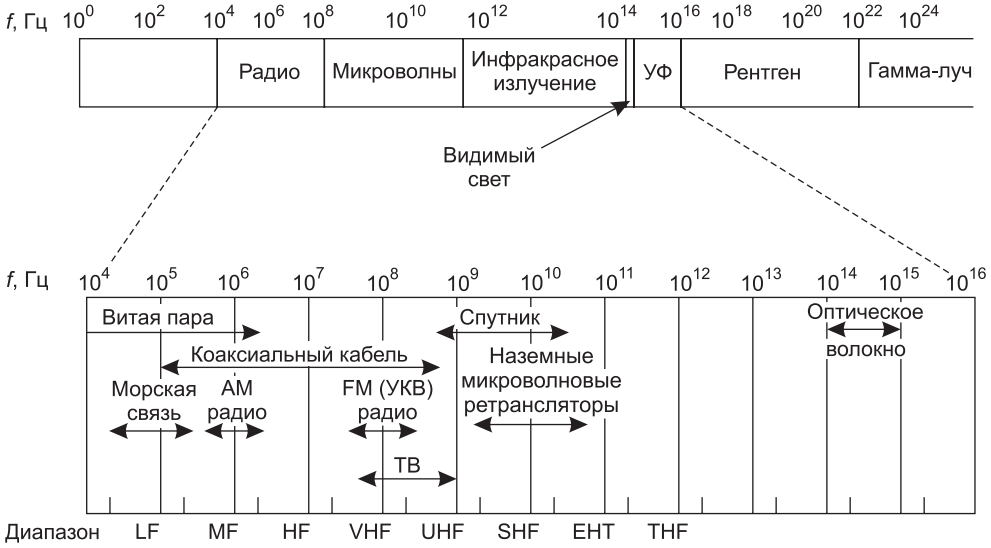


Рис. 2.8. Электромагнитный спектр и его применение в связи

Из выкладок Шеннона известно, что количество информации, которое может переносить сигнал, такой как электромагнитная волна, зависит от мощности приемника и пропорционально полосе пропускания. Из рис. 2.8 должно быть понятно, почему

разработчики сетей так любят оптоволоконную связь. В высокочастотном диапазоне для передачи данных доступна широкая полоса пропускания — шириной в несколько гигагерц, — особенно если речь идет об оптоволокне, которое находится в правой части нашей логарифмической шкалы. Например, рассмотрим 1,30-мк диапазон, изображенный на рис. 2.6; его ширина 0,17 мк. С помощью выражения (2.2) можно найти начальное и конечное значения частот диапазона, зная длину волн. Таким образом, диапазон составляет примерно 30 000 ГГц. При допустимом отношении «сигнал/шум» в 10 дБ это 300 Тбит/с.

Большинство систем связи используют относительно узкие полосы частот (то есть $\Delta f/f << 1$). Сигналы концентрируются в узкой полосе для эффективного использования спектра и достижения хорошей скорости передачи данных при достаточно мощной передаче. Однако иногда используются и широкие полосы. При этом возможны три варианта. Когда применяется расширенный спектр с перестройкой частоты, то передатчик изменяет частоту работы сотни раз в секунду. Этот метод очень популярен в военных системах связи, потому что такой сигнал тяжело перехватить и почти невозможно заглушить. Он также обладает хорошей защищенностью от многолучевого затухания и сосредоточенных помех, поскольку приемник не задерживается на искаженной частоте надолго, и разговор не прерывается. Устойчивость и надежность особенно важны в наиболее заполненных частях спектра, например в полосах ISM (industrial, scientific and medical band, промышленный, научный и медицинский диапазоны), которые мы вкратце рассмотрим. В коммерческих системах данная техника также применяется, например, в Bluetooth и старых версиях 802.11.

С историей изобретения метода перестройки частоты связан один курьез. Одним из его изобретателей была австрийская секс-богиня Хэди Ламмар (Hedy Lammar) — первая женщина, снявшаяся в кино в обнаженном виде (это был чешский фильм 1933 года под названием *Extase*). Ее первый муж занимался производством оружия и как-то раз рассказал Хэди, как легко блокируются радиосигналы управления торпедами. Когда вдруг обнаружилось, что он продает вооружение гитлеровской армии, Хэди была вне себя. Она переоделась горничной и сбежала из дома. Поехала в Голливуд, где продолжила свою актерскую карьеру. А в свободное от работы время взяла и изобрела метод перестройки частоты. Хэди мечтала хоть чем-нибудь помочь союзным войскам. В ее схеме использовалось 88 частот, по числу клавиш (и частот) на пианино. Вместе со своим другом, композитором Джорджем Антейлом (George Antheil), они запатентовали свое изобретение (патент № 2 292 387). К сожалению, Хэди не удалось убедить военно-морской флот США в том, что метод перестройки частот может иметь какое-то практическое значение, поэтому никаких гонораров за изобретение получено не было. Только через много лет после окончания срока действия патента метод передачи данных, придуманный киноактрисой и композитором, стал популярен.

Еще один метод, использующий широкую полосу частот, называется **расширенным спектром с прямой последовательностью**. Кодовая последовательность применяется для распределения сигнала данных по более широкой полосе частот. Этот метод широко используется в коммерческих системах, так как позволяет эффективно передавать несколько сигналов внутри одной полосы частот. Сигналам можно присваивать разные коды; этот метод называется CDMA (Code Division Multiple Access,

кодовое разделение каналов с множественным доступом). О нем мы поговорим чуть позже в этой главе. На рис. 2.9 показано, как данный метод отличается от метода с перестройкой частоты. Кстати, он лежит в основе мобильных телефонных сетей 3G, а также используется в системах GPS (Global Positioning System, глобальная система определения координат). Даже без назначения кодов расширенный спектр с прямой последовательностью, так же как и расширенный спектр с перестройкой частоты, устойчив к сосредоточенным помехам и многолучевому замиранию, так как теряется при этом только часть сигнала. Именно поэтому он применяется в старых беспроводных сетях 802.11b. Занимательную и подробную историю средств связи с расширенным спектром см. в книге (Scholtz, 1982).

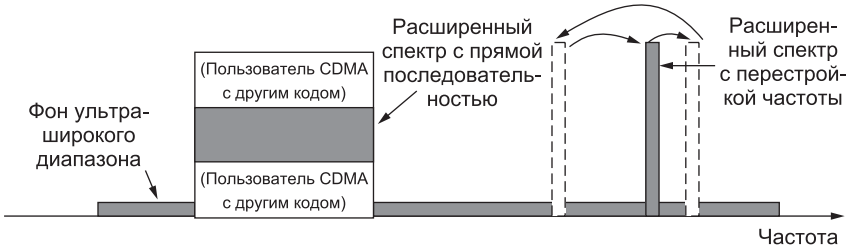


Рис. 2.9. Расширенный спектр и передача данных по сверхширокой полосе пропускания

Третий метод передачи данных в широкой полосе называется **UWB-коммуникацией** или **коммуникацией в ультраширокомдиапазоне**. Для пересылки информации отправляется последовательность коротких импульсов, изменяющих свое положение. Большое количество коротких импульсов формирует сигнал, распределенный по очень широкой полосе частот. Полоса пропускания UWB-коммуникации составляет минимум 500 МГц или минимум 20 % от значения центральной частоты соответствующей полосы частот. Рисунок 2.9 также иллюстрирует UWB-коммуникацию. С такой полосой пропускания возможна передача данных на очень высоких скоростях. А распределение по широкому диапазону частот позволяет сигналу выдерживать значительное количество относительно сильных помех со стороны других узкополосных сигналов. Также важно, что так как при передаче данных на короткое расстояние UWB-передатчик излучает на каждой конкретной частоте сигнал малой мощности, он не создает серьезных помех для этих узкополосных радиосигналов. Можно сказать, что по отношению к другим сигналам UWB-передача остается фоновой. Благодаря такому мирному существованию на свет появился новый вид сетей — PAN, Personal Area Network. Скорость передачи данных в персональной сети — до 1 Гбит/с. Нельзя сказать, однако, что это стало несомненным коммерческим успехом. UWB-коммуникацию можно применять для получения изображений объектов, находящихся за твердой преградой (земля, стены, тела людей или животных), а также в системах точного определения местоположения.

Теперь рассмотрим использование различных частей электромагнитного спектра, показанного на рис. 2.8, начиная с радиосвязи. Если не указано иное, будем предполагать, что передача данных осуществляется в узкой полосе частот.

2.3.2. Радиосвязь

Радиоволны просто сгенерировать, они могут преодолевать большие расстояния, проходить сквозь стены и огибать здания, поэтому их область применения довольно широка. Радиосвязь устанавливают как в помещениях, так и вне зданий. Кроме того, радиоволны могут распространяться одновременно во всех направлениях, поэтому для низких частот не требуется тщательного наведения антенн передатчика и приемника.

В некоторых случаях такое свойство радиоволн является удобным, но иногда оно нежелательно. В 1970-х годах компания General Motors решила оснастить новые автомобили «Кадиллак» управляемой с помощью компьютера системой антиблокировки тормозов. Когда водитель нажимал на педаль тормоза, компьютер, чтобы тормоза не заблокировались, выдавал серию импульсов команд включения и выключения тормозов. В один прекрасный день полицейский, патрулирующий шоссе в штате Огайо, решил связаться со своим участком с помощью новой портативной радиостанции. При этом едущий рядом с ним «Кадиллак» внезапно стал скакать, как дикая лошадь. Когда офицер остановил машину, водитель клялся, что не предпринимал никаких действий и что машина вдруг просто взбесилась.

Подобные случаи стали повторяться: «Кадиллаки» иногда сходили с ума, но только на главных автострадах штата Огайо и только под присмотром дорожного патруля. В течение очень долгого времени компания General Motors никак не могла понять, почему во всех других штатах и на небольших дорогах в Огайо «Кадиллаки» вели себя прекрасно. Только после долгих упорных исследований было обнаружено, что проводка «Кадиллака» представляла собой прекрасную антенну для частот, используемых новой радиосистемой дорожного патруля штата Огайо.

Свойства радиоволн зависят от частоты. При работе на низких частотах радиоволны хорошо проходят сквозь препятствия, однако мощность сигнала в воздухе резко падает по мере удаления от передатчика. Соотношение мощности и удаленности от источника выражается примерно так: $1/r2$. Энергия сигнала распределяется по большой поверхности более тонким слоем. Такое ослабление называется потерей на траектории. На высоких частотах радиоволны вообще имеют тенденцию распространяться исключительно по прямой линии и отражаться от препятствий. Потеря на траектории снижает мощность, однако полученный сигнал также сильно может зависеть от отражений. Высокочастотные радиоволны намного сильнее низкочастотных поглощаются дождем и другими препятствиями. Радиосигналы любых частот подвержены помехам со стороны двигателей с искрящими щетками и другого электрического оборудования.

Интересно сравнить ослабление радиоволн с ослаблением сигналов в проводниковых средах. В оптоволокне, коаксиальном кабеле и витой паре сигнал ослабевает пропорционально расстоянию, например, для витой пары это 20 дБ на каждые 100 м. Радиосигнал же ослабевает пропорционально квадрату расстояния, например, 6 дБ при удвоении расстояния в свободном пространстве. Это означает, что радиоволны способны распространяться на большие расстояния, и взаимные помехи, вызываемые одновременно работающими пользователями, представляют серьезную проблему. Поэтому все государства ведут очень строгий учет владельцев радиопередатчиков, за несколькими важными исключениями (обсуждаемыми ниже).

В диапазонах VLF, LF и MF радиоволны огибают поверхность земли, как показано на рис. 2.10, *а*. Эти волны можно поймать радиоприемником на расстоянии около 1000 км, если используются низкие частоты, и на несколько меньших расстояниях, если частоты повыше. Радиовещание с амплитудной модуляцией (AM) использует диапазон средних волн (MF), по этой причине, например, передачи Бостонской средневолновой радиостанции не слышны в Нью-Йорке. Радиоволны этих диапазонов легко проникают сквозь здания, вследствие чего переносные радиоприемники работают и в помещениях. Основным препятствием для использования этих диапазонов для передачи данных является их относительно низкая пропускная способность (см. выражение (2.2)).

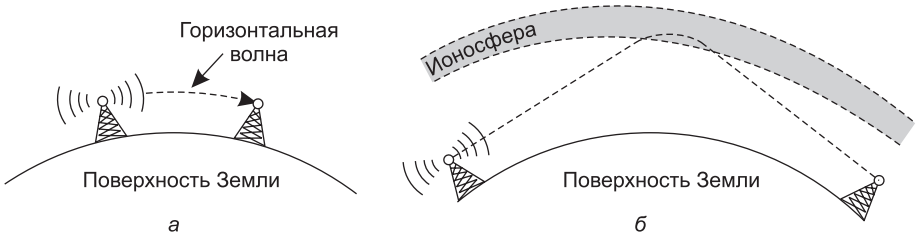


Рис. 2.10. Волны диапазонов VLF, LF и MF огибают неровности поверхности земли (*а*); волны диапазона HF отражаются от ионосферы (*б*)

Радиоволны диапазонов HF и VHF поглощаются землей. Однако те из них, которые доходят до ионосферы, представляющей собой слой заряженных частиц, расположенный на высоте от 100 до 500 км, отражаются ею и посылаются обратно к поверхности Земли, как показано на рис. 2.10, *б*. При определенных атмосферных условиях сигнал может отразиться несколько раз. Радиолюбители используют такие диапазоны частот для дальней связи. Военные также осуществляют связь в диапазонах HF и VHF.

2.3.3. Связь в микроволновом диапазоне

На частотах выше 100 МГц радиоволны распространяются почти по прямой, поэтому могут быть сфокусированы в узкие пучки. Концентрация энергии в виде узкого пучка при помощи параболической антенны (вроде всем известной спутниковой телевизионной тарелки) приводит к улучшению соотношения сигнал/шум, однако для подобной связи передающая и принимающая антенны должны быть довольно точно направлены друг на друга. Кроме того, подобная направленность позволяет использовать несколько передатчиков, установленных в ряд, сигналы от которых принимаются также установленными в ряд приемными антеннами без взаимных помех. До изобретения оптоволоконной связи подобные микроволновые антенны в течение десятков лет составляли основу междугородной телефонной связи. На самом деле, компания MCI, один из основных конкурентов AT&T, построила целую систему микроволновой связи с передачей сигнала от одной башни к другой. Расстояние между антеннами составляло десятки километров. Эта технология нашла отражение даже в названии компании: аббревиатура оператора междугородной связи MCI изначально расшифровывалась как Microwave Communications, Inc. С тех пор, впрочем, MCI уже успела перейти

на оптоволоконные сети и после множества корпоративных слияний и банкротств в сфере коммуникаций стала частью компании Verizon.

Микроволны распространяются строго по прямой, поэтому при слишком большом удалении антенн друг от друга на пути следования сигнала может оказаться земная поверхность (например, так случится, если поставить передатчик в Сиэттле, а приемник — в Амстердаме). Следовательно, на пути сигнала должны встречаться повторители. Чем выше ретрансляционные башни, тем больше может быть расстояние между ними. Максимальное расстояние между повторителями можно очень грубо оценить как корень квадратный из их высоты. Так, при высоте ретрансляторов 100 м расстояние между ними может быть около 80 км.

В отличие от радиоволн с более низкими частотами, микроволны плохо проходят сквозь здания. Кроме того, даже при точной фокусировке луча на приемной антенне при прохождении сквозь пространство луч довольно значительно расширяется в диаметре. Часть волн может отражаться атмосферными слоями, благодаря чему на своем пути к приемной антенне отраженные волны пройдут большее расстояние, чем прямые. Это означает, что первые будут отличаться от последних по фазе, что может привести к подавлению сигнала. Такой эффект называется **многолучевым затуханием** и довольно часто представляет собой серьезную проблему. Наличие этого эффекта зависит от погоды и частоты. Некоторые операторы связи держат около 10 % своих каналов свободными и временно переключаются на них в случае возникновения многолучевого затухания на какой-либо частоте.

Потребности во все большем диапазоне частот заставляют постоянно совершенствовать технологию, благодаря чему для связи используются все более высокие частоты. Диапазоны частот до 10 ГГц теперь применяются довольно широко, однако при частотах выше 4 ГГц появляется новая проблема: поглощение водой. Длина волн при такой частоте составляет всего несколько сантиметров, и такие волны сильно поглощаются дождем. Такой эффект может быть весьма полезен для тех, кто хочет соорудить огромную наружную микроволновую печь, чтобы жарить пролетающих мимо птичек, однако он представляет собой серьезную проблему в области радиосвязи. Пока что единственным решением является отключение линий связи, пересекаемых полосой дождя, и переключение на обходные пути.

Микроволновая радиосвязь стала настолько широко использоваться в междугородной телефонии, сотовых телефонах, телевещании и других областях, что начала сильно ощущаться нехватка ширины спектра. Данная связь имеет ряд преимуществ перед оптоволокном. Главное из них состоит в том, что не нужно прокладывать кабель, соответственно, не нужно платить за аренду земли на пути сигнала. Достаточно купить маленькие участки земли через каждые 50 км и установить на них ретрансляционные вышки, обойдя, таким образом, телефонные кабельные системы. Именно поэтому корпорации MCI удалось быстро внедриться в рынок междугородной связи. Компания Sprint, ставшая еще одним конкурентом AT&T после ослабления государственного регулирования, пошла другим путем: она была образована Южной Тихоокеанской железной дорогой (South Pacific Railroad), которая уже владела правами на большой участок пути и просто закапывала кабель рядом с железнодорожным полотном.

Кроме того, микроволновая связь является относительно недорогой. Установка двух примитивных вышек (это могут быть просто большие столбы на четырех растяж-

ках) с антеннами на каждой из них, скорее всего, обойдется дешевле, чем прокладка 50 км кабеля в перенаселенной городской местности или в горах. Это может быть также дешевле, чем аренда оптоволоконной линии у телефонной компании, особенно если телефонная компания еще не полностью расплатилась за медный кабель, который она уже сменила на оптоволоконный.

Политика распределения частот

Для предотвращения анархии при использовании частот существуют определенные национальные и международные соглашения, касающиеся политики их распределения. Понятно, что всем хочется сделать связь максимально быстрой, поэтому все хотели бы получить в свое распоряжение максимально широкий спектр. Национальные правительства распределяют частоты между AM- и FM-радиостанциями, телевидением, операторами сотовой связи, а также телефонными компаниями, полицией, морскими и аэронавигационными службами, военными, администрацией и еще многими другими потенциальными клиентами. Международное агентство ITU-R (WRC) пытается скоординировать действия различных структур, чтобы можно было производить устройства, способные работать в любой точке планеты. Тем не менее рекомендации ITU-R не являются обязательными для исполнения. Так, например, Федеральная комиссия по связи, FCC (Federal Communication Commision), занимающаяся раздачей частотных диапазонов в США, иногда пренебрегает этими рекомендациями — чаще всего из-за соответствующей убедительной просьбы какой-нибудь влиятельной политической группировки, которой требуется конкретная часть спектра.

Даже если определенный диапазон выделен под конкретные цели (например, под сотовую связь), то встает новый вопрос: как распределять те или иные частоты внутри диапазона между операторами связи? В прошлом были популярны три алгоритма. Первый из них, часто называемый **конкурсом красоты**, подразумевал подробные объяснения претендентов, доказывающие, что именно предлагаемый ими сервис лучше всего отвечает интересам общественности. После этого «жюри» решает, чья история выглядит самой красивой. Такая направленность на угождение администрации, зачастую подкрепленная, что называется, рублем, ведет лишь к развитию взяточничества, коррупции, непотизма и т. д. Более того, даже если какой-нибудь честный чиновник видит, что иностранная фирма может сделать для отечественного потребителя больше, чем иные национальные компании, то ему придется долго это доказывать, преодолевая сопротивление своих коллег.

Такие наблюдения в конце концов привели к созданию альтернативного алгоритма — обычной **лотереи** среди компаний, желающих получить свою долю спектрального пирога. Проблема здесь лишь в том, что в лотерее могут участвовать и фирмы, совершенно не заинтересованные ни в каких частотах. Например, если определенный частотный диапазон достается какому-нибудь крупному ресторану, он может очень выгодно продать его, ничем не рискуя.

Этот алгоритм очень бурно критиковался за то, что выиграть могут совершенно случайные лица. Результатом стало внедрение третьего алгоритма — **аукциона**. На аукционных торгах частоту выигрывал тот покупатель, который мог выложить наибольшую сумму. Когда в 2000 году. Британское правительство проводило аукцион

между операторами мобильной связи третьего поколения, ожидаемая сумма доходов составляла 4 млрд долларов. Она достигла 40 млрд, поскольку операторы просто впали в бешенство в борьбе за будущее своего бизнеса, предпочитая умереть, но не уйти с рынка мобильной связи. Эти события очень заинтересовали правительства других стран, которые тоже были не прочь получить такой доход, не прикладывая никак усилий. На аукционную систему перешли многие, и она работала, оставляя на своем пути обессиленные такой конкурентной борьбой компании, в один миг оказавшиеся на грани банкротства. В лучшем случае, компаниям, выигравшим частоту, требуется несколько лет, чтобы выплатить все свои долги.

Совершенно другим подходом является следующий: вообще не распределять частоты. Пусть каждый работает на той частоте, которая ему больше нравится, но следит за мощностью своих передатчиков: она не должна быть такой, чтобы сигналы накладывались друг на друга. В соответствии с этим принципом, было решено выделить несколько частотных диапазонов, называемых **ISM** (Industrial, Scientific, Medical, то есть промышленные, научные, медицинские). Для работы в этих диапазонах не требуется специальной лицензии. Устройства, открывающие ворота гаража, домашние радиотелефоны, радиоуправляемые игрушки, беспроводные мыши и многие-многие другие устройства работают на ISM. Для уменьшения интерференции между независимыми устройствами им предписывается комиссией FCC ограничивать излучаемую мощность (например, до одного ватта) и применять другие техники распределения сигналов по расширенному спектру. Кроме того, производителям устройств приходится задумываться о том, как избегать конфликтов с радиолокационными устройствами.

Конкретные диапазоны ISM в разных странах свои. Например, в США сетевые устройства могут использовать диапазоны, показанные на рис. 2.11, без получения лицензии FCC. Диапазон 900 МГц использовался для ранних версий устройств, работающих по стандарту 802.11, однако на сегодняшний момент он слишком загружен. Диапазон 2,4 ГГц работает в большинстве стран и применяется для передачи данных по стандартам 802.11b/g и Bluetooth, но подвержен помехам от микроволновых печей и радарных устройств. Часть спектра в районе 5 ГГц включает диапазоны U-NII (Unlicensed National Information Infrastructure, нелицензированная национальная информационная инфраструктура). Они находятся в начальной стадии своего развития, однако быстро завоевывают популярность благодаря широкой полосе пропускания и удобству для сетей 802.11a.

ISM band — диапазон ISM; U-NII bands — диапазоны U-NII. MHz — МГц; GHz — ГГц;

По аналогии с предыдущим изданием желательно добавить слева сверху «Ширина полосы», слева снизу «Частота».

Нелицензированные диапазоны в последнее десятилетие стали чрезвычайно популярными. Возможность свободно использовать спектр частот породила огромное количество инноваций в области беспроводных локальных и частных сетей, в частности дала толчок развитию технологий 802.11 и Bluetooth. Однако для продолжения развития необходимо больше частот. Большим шагом вперед в США стало решение FCC, принятое в 2009 году. Согласно этому решению, разрешено использовать незаполненные пространства вокруг частоты 700 МГц. Незаполненное (или свободное) пространство — это выделенные диапазоны частот, которые локально не используются.

Переход от аналогового к цифровому телевизионному вещанию в США в 2010 году освободил пространство вокруг частоты 700 МГц. Единственная сложность, связанная с этим, заключается в том, что нелицензированные устройства должны уметь распознавать все находящиеся поблизости лицензированные передатчики, включая беспроводные микрофоны, которые обладают преимущественным правом использования данного диапазона частот.

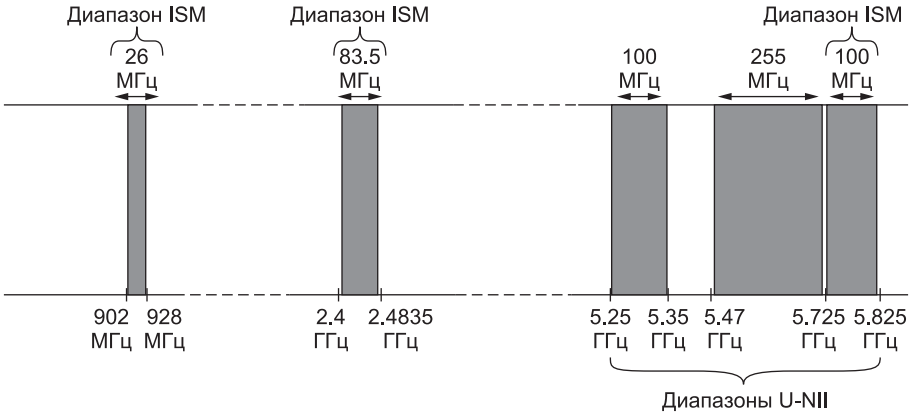


Рис. 2.11. Диапазоны ISM и U-NII, используемые в США беспроводными устройствами

Также суматоха наблюдается вокруг диапазона 60 ГГц. В 2001 году FCC открыла полосу от 57 до 64 ГГц для нелицензированного использования. Это огромная часть спектра, размер которой превышает все прочие диапазоны ISM вместе взятые, поэтому она вполне может справиться с обслуживанием высокоскоростных сетей и такими задачами, как беспроводная пересылка данных телевидения высокой четкости от одного устройства к другому в пределах вашей гостиной. На частоте 60 ГГц радиоволны поглощаются кислородом. Это означает, что сигналы неспособны распространяться далеко и, следовательно, возможна работа только в сетях малого радиуса действия. Высокие частоты (60 ГГц — это чрезвычайно высокая частота, EHF, или «миллиметровый» диапазон, прямо под инфракрасным излучением) первоначально были довольно сложными для освоения производителями оборудования, однако сегодня продукция уже завоевала рынок.

2.3.4. Передача в инфракрасном диапазоне

Инфракрасное излучение без использования кабеля широко применяется для связи на небольших расстояниях. Дистанционные пульты управления для телевизоров, видеомагнитофонов и стереоаппаратуры используют инфракрасное излучение. Они относительно направленные, дешевые и легко устанавливаемые, но имеют один важный недостаток: инфракрасное излучение не проходит сквозь твердые объекты (попробуйте встать между телевизором и пультом). Мы начали с рассмотрения длинных радиоволн и постепенно продвигаемся к видимому свету, и уже инфракрасные волны мало напоминают радиоволны и ведут себя, как свет.

С другой стороны, тот факт, что инфракрасные волны не проходят сквозь стены, является также и положительным. Ведь это означает, что инфракрасная система в одной части здания не будет интерферировать с подобной системой в соседней комнате, — вы, к счастью, не сможете управлять со своего пульта телевизором соседа. Кроме того, это повышает защищенность инфракрасной системы от прослушивания, по сравнению с радиосистемой. По этой причине для использования инфракрасной системы связи не требуется государственная лицензия, в отличие от радиосвязи (кроме диапазонов ISM). Связь в инфракрасном диапазоне применяется в настольных вычислительных системах (например, для связи ноутбуков с принтерами, поддерживающими стандарт IrDA (Infrared Data Association, ассоциация инфракрасной передачи данных)), но все же не играет значимой роли в телекоммуникации.

2.3.5. Связь в видимом диапазоне

Беспроводниковые оптические сигналы или оптические системы в свободном пространстве использовались в течение нескольких веков. Герой американской войны за независимость Пол Ревер (Paul Revere) в 1775 году в Бостоне использовал двоичные оптические сигналы, информируя с колокольни Старой Северной церкви (Old North Church) население о наступлении англичан. Более современным приложением является соединение локальных сетей в двух зданиях при помощи лазеров, установленных на крышах. Оптическая связь с помощью лазера является сугубо однонаправленной, поэтому для двусторонней связи необходимо на каждой стороне установить по лазеру и по фотодетектору. Такая технология позволяет организовать при очень низкой цене связь с очень хорошей пропускной способностью и относительно высокой безопасностью, так как перехватить узкий лазерный луч очень сложно. Кроме того, такая система довольно просто монтируется и, в отличие от микроволновой связи, не требует лицензии FCC (Федеральной комиссии связи США).

Сила и узкий луч являются сильными сторонами лазера, однако они создают и некоторые проблемы. Чтобы попасть миллиметровым лучом в мишень диаметром 1 мм на расстоянии 500 м, требуется снайперское искусство высочайшей пробы. Обычно на лазеры устанавливаются линзы для небольшой расфокусировки луча. Чтобы еще усложнить задачу, ветер и температурные изменения способны искажать луч. Кроме того, лазерные лучи не способны проходить сквозь дождь или густой туман, хотя в солнечные ясные дни они работают прекрасно. Однако многие из этих факторов теряют всякую значимость, когда речь заходит о передаче данных между двумя космическими станциями.

Один из авторов однажды присутствовал на конференции в современной европейской гостинице, где организаторы заботливо предоставили комнату, полную терминалов, чтобы участники конференции могли читать свою электронную почту во время скучных презентаций. Поскольку местная телефонная станция не желала устанавливать большое количество телефонных линий всего на три дня, организаторы установили лазер на крыше и нацелили его на здание университетского компьютерного центра, который находился на расстоянии нескольких километров. В ночь перед конференцией они проверили связь — она работала прекрасно. В 9 часов следующего утра, в ясный солнечный день связь была полностью потеряна и отсутствовала весь

день. Это повторилось и в последующие два дня. Когда конференция закончилась, организаторы обсудили эту проблему. Как выяснилось, в дневное время солнце нагревало крышу, горячий воздух от нее поднимался и отклонял лазерный луч, начинавший танцевать вокруг детектора (рис. 2.12). Этот эффект можно наблюдать невооруженным глазом в жаркий день на шоссе или над горячим радиатором автомобиля. Какой из этого можно извлечь урок? Прилагать усилия нужно не только для решения задачи в сложных условиях. Даже в хороших условиях беспроводниковые оптические системы связи необходимо проектировать с учетом возможных погрешностей.

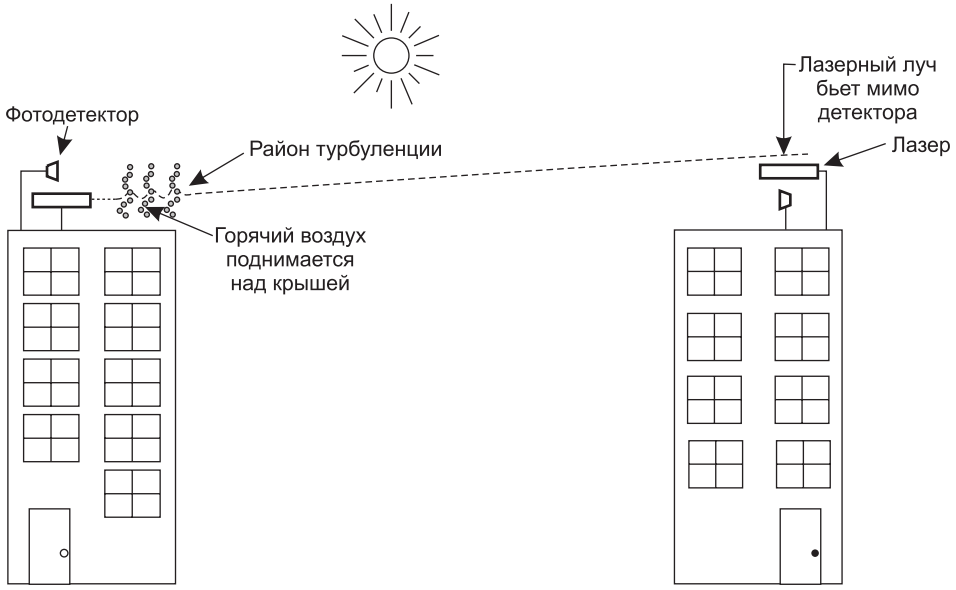


Рис. 2.12. Конвекционные потоки мешают работать лазерной системе связи. На рисунке изображена двунаправленная система с двумя лазерами

Беспроводниковая оптическая связь в современных условиях может казаться некой экзотической технологией, но у нее большой потенциал. Мы окружены камерами (распознающими свет) и дисплеями (излучающими свет с помощью светодиодов и других устройств). Обмен данными можно организовать на основе таких дисплеев. Светодиоды будут включаться и отключаться по определенному шаблону, излучая свет, который человеческий глаз увидеть не в состоянии. Коммуникация при помощи видимого света по своей природе безопасна; в непосредственной близости от дисплея создается удобная низкоскоростная сеть. Можно придумать множество причудливых сценариев с этой технологией в главной роли. Мигающие огни автомобилей экстренных служб могут передавать сигналы на ближайшие светофоры, помогая освобождать дорогу. На информационных табло можно выводить карты с данными в реальном времени. Даже праздничные гирлянды можно будет применить для воспроизведения музыки синхронно с миганием огоньков.

2.4. Спутники связи

В 1950-х и начале 60-х годов люди пытались организовать связь при помощи сигналов, отраженных от металлических метеозондов. К сожалению, мощность таких сигналов была слишком мала, и их практическое значение оказалось ничтожным. Затем ВМФ США обнаружил, что в небе постоянно висит некое подобие метеозонда — это была Луна. Была построена система для связи береговых служб с кораблями, в которой использовалось отражение сигналов от естественного спутника Земли.

Дальнейший прогресс в создании коммуникаций с помощью небесных тел на этом приостановился до запуска первого спутника связи. Ключевым отличием искусственной «луны» являлось то, что на спутнике было установлено оборудование, позволяющее усилить входящий сигнал перед отправкой его обратно на Землю. Это превратило космическую связь из забавного курьеза в мощную технологию.

Спутникам связи присущи определенные свойства, делающие их чрезвычайно привлекательными для самых разных областей применения. Проще всего представить себе спутник связи в виде своего рода огромного микроволнового повторителя, висящего в небе. Он включает в себя несколько **транспондеров**, каждый из которых настроен на определенную часть частотного спектра. Транспондеры усиливают сигналы и преобразуют их на новую частоту, чтобы при отправке на Землю отраженный сигнал не накладывался на прямой. Такой режим работы называется «узкая труба» (bent pipe). Можно добавить цифровую обработку, для того чтобы по отдельности манипулировать или перенаправлять потоки данных в доступном диапазоне. Кроме того, спутник может получать и пересылать дальше цифровую информацию. Такой способ восстановления сигнала улучшает общую производительность по сравнению с «узкой трубой», так как спутник не увеличивает количество шума в восходящем сигнале. Нисходящий луч может быть как широким, покрывающим огромные пространства на Земле, так и узким, который можно принять в области, ограниченной лишь несколькими сотнями километров.

В соответствии с законом Кеплера, период обращения спутника равен радиусу орбиты в степени 3/2. Таким образом, чем выше орбита, тем дольше период. Вблизи поверхности Земли период обращения вокруг нее составляет примерно 90 минут. Следовательно, спутники, расположенные на малой высоте, слишком быстро исчезают из вида приемо-передающих устройств, расположенных на Земле, поэтому необходимо организовывать непрерывные зоны покрытия и устанавливать для их отслеживания наземные антенны. На высоте 35 800 км период составляет 24 часа. А на высоте 384 000 км спутник будет обходить Землю целый месяц, в чем может убедиться любой желающий, наблюдая за Луной.

Конечно, период обращения спутника очень важно иметь в виду, но это не единственный критерий, по которому определяют, где его разместить. Необходимо принимать во внимание так называемые пояса Ван Аллена (Van Allen belts) — области скопления частиц с большим зарядом, находящихся в зоне действия магнитного поля Земли. Любой спутник, попав в такой пояс, довольно быстро будет уничтожен этими частицами. В результате учета этих факторов были выделены три зоны, в которых можно безопасно размещать искусственные спутники. Они изображены на рис. 2.13. Из этого же рисунка можно узнать о некоторых из их свойств. Мы вкратце рассмотрим спутники, размещаемые в каждой из этих трех зон.

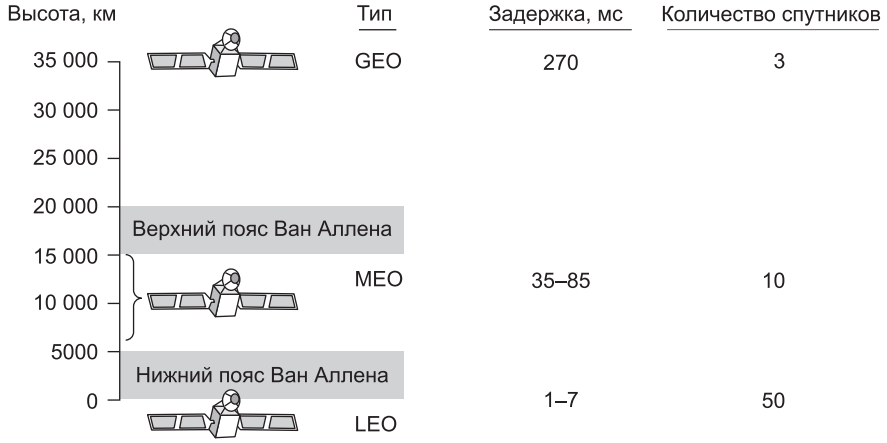


Рис. 2.13. Спутники связи и их свойства: высота орбиты, задержка, число спутников, необходимое для покрытия всей поверхности земного шара

2.4.1. Геостационарные спутники

В 1945 году писатель-фантаст Артур С. Кларк (Arthur S. Clarke) подсчитал, что спутник, расположенный на высоте 35 800 км на круговой экваториальной орбите, будет оставаться неподвижным относительно Земли. А значит, следить за ним будет гораздо проще (Clarke, 1945). Он развил свою мысль и описал целую коммуникационную систему, использующую такие (пилотируемые) **геостационарные спутники**. Он описал орбиты, солнечные батареи, радиочастоты и даже процедуры связи. К сожалению, в конце концов он пришел к неутешительному выводу, что такие спутники вряд ли будут иметь практическое значение, потому что на их борту невозможно разместить энергоемкие, хрупкие ламповые усилители. В связи с этим, Кларк не стал больше развивать свою идею, хотя и написал несколько фантастических рассказов о подобных искусственных спутниках.

Положение вещей изменило изобретение транзистора, и вот в июле 1962 года производится запуск первого в мире спутника связи Telstar.

С тех пор спутники связи стали многомиллиардным бизнесом и единственным прибыльным делом, связанным с космическими технологиями. Про спутники, вращающиеся на большой высоте, говорят, что они расположены на **геостационарной орбите** (GEO, Geostationary Earth Orbit).

Современные технологии таковы, что расположение спутников чаще, чем через каждые два градуса в 360-градусной экваториальной плоскости, является нерациональным. В противном случае возможна интерференция сигналов. Итак, если на каждые два градуса приходится один спутник, то всего их в экваториальной плоскости можно разместить 360/2 = 180. Сто восемьдесят спутников могут одновременно находиться в небе и вращаться в одной и той же плоскости на одной и той же высоте. Тем не менее у каждого транспондера есть возможность работы на разных частотах и с разной поляризацией, что позволяет увеличить максимальную пропускную способность всей системы.

Со временем возникла необходимость предотвращения беспорядочного использования околоземных орбит. Навести порядок в небе было поручено организации ITU. Процесс выделения орбит очень сильно связан с политикой, причем многие страны в борьбе за свой «кусок» неба напоминают далеких предков человека из каменного века. Это объясняется очень высокими потенциальными доходами, которые государство может извлечь, сдавая в аренду кусочки космоса. В то же время некоторые страны заявляют, что их государственные границы в высоту простираются до самой Луны и что использование орбит, проходящих над их территорией, иностранными государствами является нелегальным. Жаркие споры на эту тему подогревает еще и тот факт, что коммерческая связь — это далеко не единственное применение спутников связи, а значит, и их орбит. Ими пользуются операторы спутникового телевидения, правительственные структуры и военные.

Современные спутники могут быть довольно большими, весят более 5000 кг и потребляют до нескольких киловатт электроэнергии, вырабатываемой солнечными батареями. Эффекты гравитации, вызванные Солнцем, Луной и другими планетами, постепенно вызывают смещение с орбит и изменение ориентации. Приходится компенсировать это с помощью бортовых двигателей. Действия по сохранению параметров орбит спутников называются **позиционированием**. И все же приходит момент, когда топливо у бортовых двигателей заканчивается (обычно такое случается после десяти лет использования). Тогда спутник начинает беспомощно дрейфовать, постепенно сходя с орбиты. Понятно, что он перестает быть дееспособным и его нужно отключать. Обычно спутники связи заканчивают свою жизнь, постепенно входя в плотные слои атмосферы и сгорая там либо (крайне редко) падая на землю.

Участки орбит — это не единственный предмет, за который борются страны и отдельные компании. Разумеется, распределению между всеми желающими подлежат и рабочие диапазоны частот, поскольку нисходящие сигналы спутников могут вызывать помехи в работе микроволновых устройств. Поэтому ITU были выделены частотные диапазоны, предназначенные исключительно для спутников связи. Самые важные из них показаны в табл. 2.3.

Таблица 2.3. Основные частотные диапазоны спутников связи

Диапазон	Нисходящие сигналы, ГГц	Восходящие сигналы, ГГц	Ширина полосы, МГц	Проблемы
L	1,5	1,6	15	Узкая полоса; переполнен
S	1,9	2,2	70	Узкая полоса; переполнен
C	4,0	6,0	500	Наземная интерференция
Ku	11	14	500	Дождь
Ka	20	30	3500	Дождь, стоимость оборудования

Диапазон C был первой полосой частот, предназначенной для трафика коммерческих спутников. Он разбивается на два поддиапазона. Один из них предназначен для сигналов с Земли (восходящих), другой — для сигналов со спутника (нисходящих). Таким образом, для двусторонней передачи требуется сразу два канала. Они уже пере-

полнены пользователями, поскольку на тех же частотах работают наземные микроволновые устройства связи. В 2000 году, в соответствии с международным соглашением, было добавлено два дополнительных диапазона: S и L. Тем не менее они тоже весьма узки и уже заполнены.

Следующий высокочастотный диапазон коммерческой связи называется Ku (K under, то есть «под K»). Полоса пока еще не переполнена, и работающие на ее самых высоких частотах спутники могут располагаться на угловом расстоянии один градус друг от друга. У диапазона Ku имеется еще одна проблема: волны этих частот глушатся дождем. Вода очень плохо пропускает микроволновый сигнал. К счастью, очень сильные ливни обычно бывают весьма узко локализованы, поэтому проблему удается решить с помощью нескольких наземных установок, расположенных довольно далеко друг от друга. Цена, которую приходится платить за «проблему дождя», весьма высока: это дополнительные антенны, кабели и электронные устройства для быстро переключения станций. Наконец, самым высокочастотным диапазоном является Ka (K above, то есть, «над K»). Основной проблемой является очень высокая стоимость оборудования для работы на этих частотах. Помимо коммерческих диапазонов, существует также множество военных и правительственных.

На современном спутнике имеется порядка 40 транспондеров, чаще всего с полосами в 36 МГц. Обычно каждый транспондер работает по принципу «узкой трубы», однако недавно появились спутники, оснащенные бортовыми процессорами для обработки сигналов. В первых спутниках разделение транспондеров по каналам было статическим: весь доступный рабочий диапазон просто разделялся на несколько фиксированных полос. Теперь же сигнал транспондера разделяется на временные слоты, то есть каждому пользователю выделяется на передачу определенный промежуток времени. Ниже в этой главе мы изучим оба принципа (частотное и временное мультиплексирование) более подробно.

Первые геостационарные спутники связи имели один луч, который охватывал примерно 1/3 земной поверхности и назывался **точечным лучом**. Однако по мере удешевления, уменьшения размеров и энергоемкости микроэлектронных элементов, стали появляться более сложные стратегии. Стало возможно оборудовать каждый спутник несколькими антеннами и несколькими транспондерами. Каждый нисходящий луч сфокусировали на небольшой территории; таким образом, смогли осуществить одновременную передачу нескольких сигналов. Обычно эти так называемые **пятна** имеют форму овала и могут иметь относительно малые размеры — порядка нескольких сотен километров. Американский спутник связи охватывает широким лучом 48 штатов, а также имеет два узких луча для Аляски и Гавайских островов.

Новым витком развития спутников связи стало создание недорогих терминалов со сверхмалой апертурой — **VSAT** (Very Small Aperture Terminal) (Abramson, 2000). У этих небольших станций имеется антенна диаметром всего один метр (сравните с 10-метровой антенной GEO), их выходная мощность составляет примерно 1 Вт. Скорость работы в направлении Земля — спутник обычно составляет до 1 Мбит/с, зато связь спутник — Земля можно поддерживать до нескольких мегабит в секунду. Спутниковое широковещательное телевидение использует эту технологию для односторонней передачи сигнала.

Многим микростанциям VSAT не хватает мощности для того, чтобы связываться друг с другом (через спутник, разумеется). Для решения этой проблемы устанавли-

ваются специальные наземные концентраторы с большой мощной антенной. Концентратор (хаб, ретранслятор) распределяет трафик между несколькими VSAT, как показано на рис. 2.14. В таком режиме либо приемник, либо передатчик обязательно имеет большую антенну и мощный усилитель. Недостатком такой системы является наличие задержек. Достоинством — низкая цена за полноценную систему для конечного пользователя.

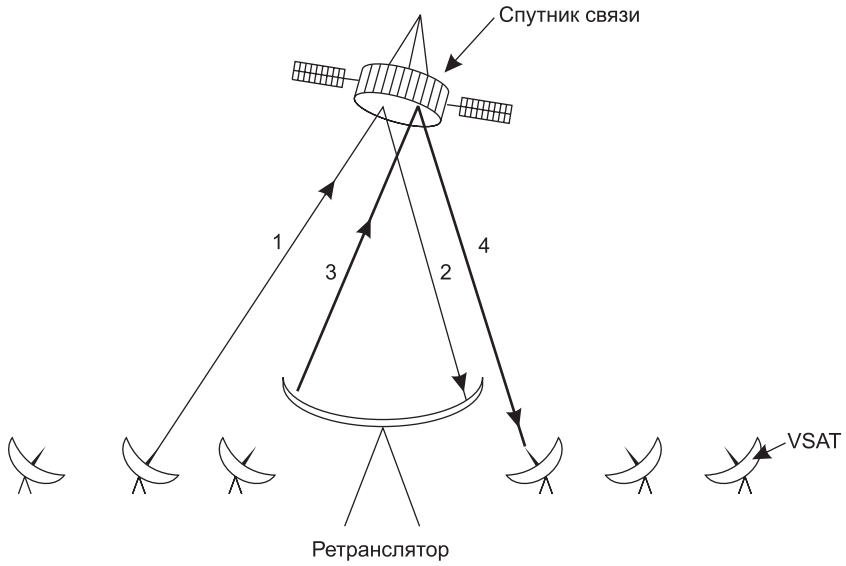


Рис. 2.14. Микростанция VSAT и наземный концентратор

Системы VSAT имеют большие перспективы использования в сельской местности. Об этом как-то не очень часто вспоминают, но половина населения земного шара живет минимум в часе ходьбы от ближайшего телефона. Протянуть телефонные линии ко всем селам и деревням не по карману большинству стран так называемого «третьего мира». Однако средств на установку тарелки VSAT, питающейся от солнечной батареи, может хватить не только у администрации региона, но и у частных лиц. Таким образом, VSAT — это технология, которая может позволить организовать связь в любой точке планеты.

Спутники связи обладают рядом свойств, которые радикально отличают их от любых наземных систем связи между абонентами. Во-первых, несмотря на предельно высокую скорость распространения сигнала (собственно, она практически равна скорости света — 300 000 км/с), расстояния между наземными приемо-передающими устройствами и спутниками таковы, что в технологии GEO задержки оказываются весьма значительными. В зависимости от взаимного расположения пользователя, наземной станции и спутника, время передачи может составлять 250–300 мс. Обычно оно составляет 270 мс (соответственно, в два раза больше — 540 мс — в системах VSAT, работающих через концентратор).

Для сравнения, сигнал в наземных микроволновых системах связи имеет задержку распространения примерно 3 мкс/км, а коаксиальный кабель и оптоволокно имеют

задержку порядка 5 мкс/км. Разность задержек здесь объясняется тем, что в твердых телах сигнал распространяется медленнее, чем в воздухе.

Еще одним важным свойством спутников является то, что они — исключительно широковещательное средство передачи данных. На отправку сообщения сотням абонентов, находящихся в зоне следа спутника, не затрачивается никаких дополнительных ресурсов по сравнению с отправкой сообщения одному из них. Для некоторых применений это свойство очень полезно. Например, можно представить себе кэширование на спутнике популярных веб-страниц, что резко повысит скорость их загрузки на сотни компьютеров, находящихся довольно далеко друг от друга. Конечно, широковещание симулируется обычными двухточечными сетями, однако спутниковое вещание в этом случае обходится значительно дешевле. С другой стороны, с точки зрения конфиденциальности данных, спутники — это прямо-таки беда: кто угодно может прослушивать абсолютно все. Здесь на защиту тех, кому важен ограниченный доступ к информации, встает криптография.

Спутники связи обладают еще одним замечательным свойством — независимостью стоимости передачи от расстояния между узлами. Звонок другу, живущему за океаном, стоит столько же, сколько звонок подружке, живущей в соседнем доме. Космические телекоммуникационные технологии, кроме того, обеспечивают очень высокую степень защиты от ошибок и могут быть развернуты на местности практически мгновенно, что очень важно для военных и служб МЧС.

2.4.2. Средневысотные спутники

На гораздо более низких высотах, нежели геостационарные спутники, между двумя поясами Ван Аллена, располагаются **средневысотные спутники** (**MEO**, Medium-Earth Orbite Satellites). Если смотреть на них с Земли, то будет заметно их медленное дрейфование по небосводу. Средневысотные спутники делают полный оборот вокруг нашей планеты примерно за 6 часов. Соответственно, наземным приемопередатчикам необходимо следить за их перемещением. Поскольку эти спутники находятся гораздо ниже, чем геостационарные, то и «засвечиваемое» ими пятно на поверхности Земли имеет более скромные размеры. Зато для связи с ними требуются менее мощные передатчики. Спутники MEO используются для поддержки навигационных систем, а не в телекоммуникациях, поэтому в дальнейшем мы не будем их рассматривать. Примерами средневысотных спутников являются около 30 спутников системы **GPS** (Global Positioning System, Глобальная система определения местонахождения), вращающихся вокруг Земли на высоте около 20 200 км.

2.4.3. Низкоорбитальные спутники

Снизим высоту еще больше и перейдем к рассмотрению **низкоорбитальных спутников** (**LEO**, Low-Earth Orbite Satellites). Для того чтобы создать целостную систему, охватывающую весь земной шар, нужно большое количество таких спутников. Причиной тому является, прежде всего, высокая скорость их движения по орбите. С другой стороны, благодаря относительно небольшому расстоянию между наземными передатчиками и спутниками, не требуется особо мощных наземных передатчиков,

а задержки составляют всего лишь несколько миллисекунд. Расходы на запуск тоже значительно ниже. В этом разделе мы рассмотрим два примера спутниковых группировок, применяемых для голосовой связи: Iridium и Globalstar.

Iridium

В течение первых 30 лет существования спутников связи низкоорбитальные спутники использовались очень мало, поскольку они появлялись и исчезали из зоны видимости передатчика слишком быстро. В 1990 году фирма Motorola совершила большой прорыв в этой области, попросив FCC разрешить ей запустить 77 спутников связи для нового проекта **Iridium** (77-м элементом таблицы Менделеева является Иридиум). Впрочем, планы вскоре изменились, и было решено использовать только 66 спутников, поэтому проект следовало бы переименовать в Dysprosium, но это название звучало бы менее благозвучно. Идея состояла в том, что на место исчезающего из вида спутника будет тотчас приходить следующий. Этакая карусель. Такое предложение породило новую волну безумной конкуренции среди коммуникационных компаний. Каждая из них захотела «повесить» в небе свою цепочку низкоорбитальных спутников.

После семи лет притирки компаний друг к другу и решения вопросов финансирования совместными усилиями удалось, наконец, запустить спутники. Услуги связи начали предоставляться с ноября 1998 года. К сожалению, коммерческий спрос на большие и тяжелые телефоны спутниковой связи оказался незначительным, потому что за семь лет конкурентной борьбы, которые прошли до запуска проекта Iridium, сотовая связь шагнула очень далеко вперед. В результате Iridium практически не приносил прибыли, и в августе 1999 его пришлось объявить банкротом — это было одно из самых эффектных корпоративных фиаско в истории. Спутники, как и другое имущество (стоимостью порядка $5 миллиардов), были проданы инвестору за $25 миллионов в качестве своего рода космической распродажи старого барахла. Другие предприятия по предоставлению услуг спутниковой связи вскоре последовали печальному примеру.

Проект Iridium был вновь запущен в марте 2001 года и продолжает расти. Эта система предоставляет связь с любой точкой земного шара при помощи ручных устройств, связывающихся напрямую со спутниками. Можно передавать речь, данные, факсы, информацию для пейджеров, а также навигационную информацию. И все это работает и на суше, и на море, и в воздухе! Основными клиентами Iridium являются судоходные, авиационные компании, фирмы, занимающиеся поиском нефти, а также частные лица, путешествующие в местах, где отсутствует телекоммуникационная инфраструктура (например, пустыни, горы, Южный полюс, некоторые страны третьего мира).

Спутники Iridium вращаются по околоземной круговой полярной орбите на высоте 750 км. Они составляют ожерелье, ориентированное вдоль линий долготы (по одному спутнику на 32° долготы). Шесть таких ожерелий опоясывают Землю, как показано на рис. 2.15. У каждого спутника максимум 48 ячеек (пятен от лучей сигналов), а полоса пропускания вмещает 3840 каналов. Некоторые каналы используются пейджинговыми компаниями и для навигации, остальные — для передачи данных и речи.

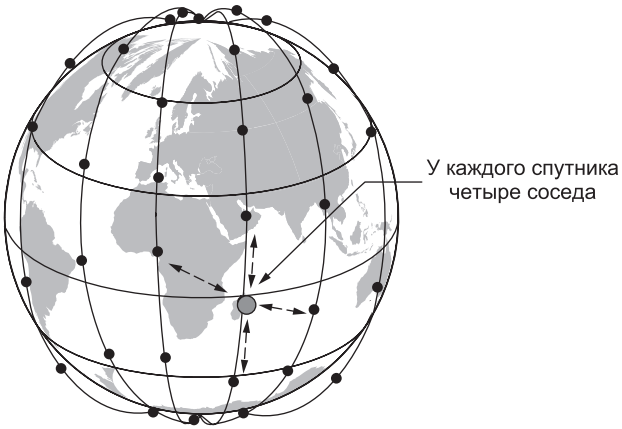


Рис. 2.15. Шесть ожерелий Земли из спутников Iridium

Шесть ожерелий позволяют покрыть всю поверхность Земли, как видно на рисунке. Интересным свойством Iridium является то, что в этой системе обмен данными между очень удаленными друг от друга абонентами происходит в космосе, как показано на рис. 2.16, *а*. Абонент на Северном полюсе напрямую связывается со спутником, который находится над ним. У каждого спутника четыре соседа, с которыми он может обмениваться данными: два в том же ожерелье (см. рис. 2.15) и два в соседних. Спутники передают голосовые данные по этой сетке, и в результате речь первого абонента достигает второго абонента на Южном полюсе.

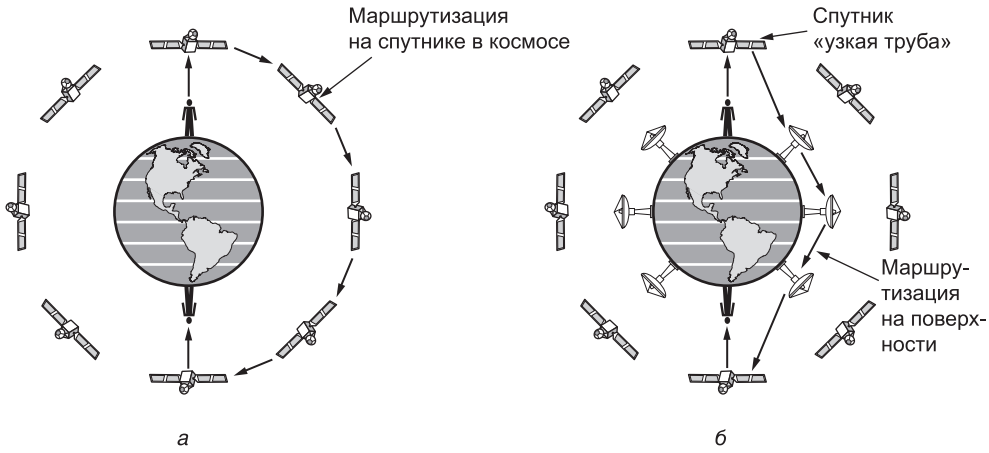


Рис. 2.16. Пересылка данных в космосе (*а*); пересылка данных наземными станциями (*б*)

Globalstar

Альтернативой проекту Iridium является система **Globalstar**. Она построен на 48 низкоорбитальных спутниках, но имеет иную схему ретрансляции сигналов. Если

в Iridium в качестве маршрутизаторов используются сами спутники, передающие по цепочке сигнал (что требует наличия на них довольно сложного оборудования), то в Globalstar применяется обычный принцип «узкой трубы». Допустим, звонок приходит на спутник с Северного полюса (рис. 2.16, *б*). Принятый сигнал отправляется обратно на Землю и захватывается крупной наземной приемопередающей станцией рядом с домиком Санта-Клауса. Маршрутизация производится между такими станциями, разбросанными по всему миру. Наземная цель сигнала — ближайший ко второму абоненту наземный маршрутизатор. Через находящийся рядом с ним спутник вызов поступает к абоненту. Преимуществом такой схемы является то, что наиболее сложное оборудование устанавливается на поверхности Земли, а здесь работать с ним гораздо проще, чем на орбите. К тому же использование мощных наземных антенн позволяет принимать слабый сигнал со спутника; значит, можно уменьшить потребную мощность телефонов. В результате телефоны передают сигналы с мощностью всего несколько милливатт, и наземные антенны получают очень слабый сигнал, даже после его усиления спутником. Тем не менее такой мощности хватает для нормальной работы.

За год в космос запускается около 20 спутников, включая спутники-гиганты, весящие более 5000 кг. Организации, которые не могут позволить себе огромных бюджетных трат, используют крошечные спутники. Для того чтобы сделать исследование космического пространства более доступным, академики из Калифорнийского политехнического института и Стэнфорда в 1999 году придумали стандарт миниатюрных спутников и подходящей пусковой установки, значительно снижающие стоимость запуска спутника (Nugent и др., 2008). CubeSat — это спутник, представляющий собой куб со стороной 10 см. Каждый такой спутник весит не более одного килограмма, а стоимость его запуска не превышает 40 000 долларов. Пусковая установка прикрепляется к летательным аппаратам, которые отправляются в коммерческие полеты. Фактически пусковая установка — это труба, вмещающая до трех миниатюрных спутников. На орбиту они выстреливаются при помощи пружины. Уже запущено около 20 спутников CubeSat, и еще очень много проектов находятся в активной разработке. Большинство таких спутников общаются с наземными станциями в диапазонах УВЧ (UHF) и ОВЧ (VHF).

2.4.4. Спутники против оптоволокна

Такое сравнение не только уместно, но и поучительно. Всего лишь 25 лет назад люди смогли осознать, что будущее телекоммуникационных систем — за спутниками связи. В конце концов, телефонная система не особо менялась последние 100 лет, и похоже, что не изменится и еще через 100 лет. Такая стабильность вызвана, в том числе, и мощной регулятивной средой, которая обязывала телефонные компании предоставлять качественный сервис за разумные деньги и взамен предлагала гарантированную прибыль за счет инвестиций. Для тех, кому требовалось передавать не только речь, но и данные, сделали модемы на 1200 бит/с. Собственно, это все, что долгое время предоставляла телефонная система.

В 1984 году в США и чуть позднее — в Европе стала возникать конкурентная борьба в области связи, которая все поставила с ног на голову. Телефонные компании занялись прокладкой оптического волокна для междугородной телефонии и стали предоставлять услуги высокоскоростного доступа в Интернет, например, по ADSL

(Asymmetric Digital Subscriber Line, Асимметричная цифровая абонентская линия). Наконец-то стали снижаться искусственно завышенные тарифы на дальнюю связь, за счет которых долгое время удерживались низкие тарифы на местные переговоры. Довольно неожиданно оптоволоконные кабели стали победителями среди средств связи.

Тем не менее у спутников имеются свои области применения, в которых оптоволокно, увы, бессильно. Во-первых, если речь идет о быстром развертывании, преимущество спутников бесспорно. Быстрая реакция крайне важна для военных целей, особенно во время войны, а в мирное время — для служб МЧС. После серьезного землетрясения и последующего цунами в Суматре в декабре 2004 года восстановить связь удалось всего за 24 часа — и все благодаря спутникам. В этом регионе существует рынок поставщиков услуг спутниковой связи, на котором крупные игроки, такие как компания Intelsat, владеющая более чем пятьюдесятью спутниками, арендуют мощности там, где это необходимо. Для клиентов, которые обслуживаются в существующих сетях спутниковой связи, в любой точке земного шара можно быстро развернуть микростанцию VSAT, обеспечив связь со скоростью до мегабита в секунду.

Вторая область применения спутников — связь в регионах с плохо развитой наземной инфраструктурой. Сегодняшние пользователи хотят иметь возможность общаться в любых уголках мира. Сети мобильной связи хорошо покрывают регионы с большой плотностью населения, но в других местах (например, в море или пустыне) они почти недоступны. Iridium же предоставляет услуги голосовой связи по всему миру, даже на Южном полюсе. Кроме того, развертывание наземной инфраструктуры стоит недешево и в большой степени зависит от территориальных условий и прав собственности. В Индонезии используется собственный спутник для внутреннего телефонного трафика. Приобрести его оказалось дешевле, чем проложить подводный кабель между всеми 13 667 островами архипелага.

Третья интересующая нас область — широковещание. Пакет данных, отправленный со спутника, одновременно принимается тысячами наземных станций. Поэтому со спутников вещают многие сетевые телеканалы. На современном рынке даже есть услуга прямого вещания со спутника: спутниковый телевизионный или радиоприемник устанавливается прямо в доме или автомобиле пользователя. Рассылать таким способом можно любое содержимое, не только телепрограммы. Например, передавать данные о стоимости акций, облигаций и ценах на товары тысячам дилеров может оказаться дешевле с применением спутника. Развернуть широковещание с помощью наземных средств будет и дороже, и сложнее.

В целом, основным средством телекоммуникаций на Земле, вероятно, будет комбинация оптоволокна и сотовой радиосвязи, но для некоторых специальных применений будет использоваться спутниковая система. Однако есть одно «но», которое может приостановить развитие всего этого: экономика. Хотя оптоволоконные кабели обладают очень высокой пропускной способностью, беспроводные системы, как наземные, так и спутниковые, вероятно, будут вести очень жесткую политику ценовой конкуренции. Если будет продолжаться удешевление спутниковых систем (например, если какие-нибудь будущие космические корабли будут способны выводить на орбиту одновременно десятки спутников связи), а низкоорбитальные спутники постепенно будут все больше использоваться в телекоммуникациях, то не исключено, что оптоволоконные сети уйдут с ведущих ролей на большинстве рынков.

2.5. Цифровая модуляция и мультиплексирование

Теперь, когда мы изучили свойства проводных и беспроводных каналов, мы обращаем наше внимание к проблеме пересылки цифровой информации. Провода и беспроводные каналы переносят аналоговые сигналы, такие как непрерывно меняющиеся напряжение, интенсивность света или интенсивность звука. Чтобы послать цифровую информацию, мы должны разработать аналоговые сигналы, которые будут представлять биты. Процесс преобразования между битами и сигналами, которые их представляют, называют **цифровой модуляцией**.

Мы начнем со схем, которые непосредственно преобразовывают биты в сигнал. Эти схемы приводят к **передаче в основной полосе частот**, в которой сигнал занимает частоты от нуля до максимума, который зависит от сигнального уровня. Это характерно для проводов. Затем мы рассмотрим схемы, которые регулируют амплитуду, фазу или частоту несущего сигнала для передачи битов. Эти схемы приводят к **передаче в полосе пропускания**, в которой сигнал занимает полосу частот вокруг несущей сигнала. Это характерно для беспроводных и оптических каналов, для которых сигналы должны находиться в заданном диапазоне частот. Каналы часто совместно используются несколькими сигналами. В конце концов, намного более удобно использовать один провод, чтобы перенести несколько сигналов, чем проложить провод для каждого сигнала. Этот вид совместного использования называют **мультиплексированием**. Это может быть достигнуто несколькими различными способами. Мы рассмотрим методы временного, частотного мультиплексирования и мультиплексирования с кодовым разделением.

Все методы модуляции и мультиплексирования, которые мы описываем в этом разделе, широко используются для проводов, оптоволокна, наземного радио и спутниковых каналов. В следующих разделах мы рассмотрим примеры сетей, чтобы увидеть эти методы в действии.

2.5.1. Низкочастотная передача

Самая простая форма цифровой модуляции — использовать положительное напряжение, чтобы представить 1 и отрицательное напряжение, чтобы представить 0. Для оптоволокна присутствие света могло бы представить 1 и отсутствие света могло бы представить 0. Эту схему называют **NRZ (Non-Return-to-Zero, без возвращения к нулю)**. Это странное название возникло по историческим причинам и просто означает, что сигнал следует за данными. Пример показан на рис. 2.17, *б*.

Посланный сигнал NRZ отправляется по проводу. С другой стороны, приемник преобразовывает его в биты, выбирая сигнал равномерно по времени.

Сигнал не будет походить на сигнал, который был послан. Он будет ослаблен и искажен каналом и шумом в приемнике. Чтобы расшифровать биты, приемник отображает образцы сигнала в самые близкие символы. Для NRZ положительное напряжение будет указывать на то, что было послано 1, отрицательное напряжение — что был послан 0.

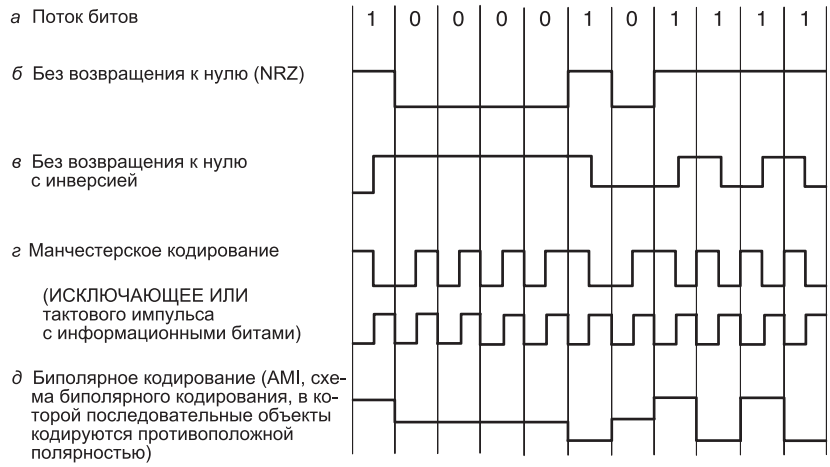


Рис. 2.17. Линейные коды (коды для линии связи): *а* — биты, *б* — NRZ; *в* — NRZI; *г* — Манчестер; *д* — биполярный или AMI

NRZ — хорошая начальная точка для наших исследований, потому что это просто, но практически эта схема редко используется отдельно. Более сложные схемы могут преобразовать биты в сигналы, которые лучше отвечают техническим соображениям. Эти схемы называют **линейными кодами (кодами для линии связи)**. Ниже, мы описываем линейные коды, которые помогают эффективно использовать пропускную способность, восстанавливать синхронизацию и баланс постоянного тока.

Эффективность использования полосы частот

С NRZ сигнал может циклически повториться между положительными и отрицательными уровнями для каждых двух битов (в случае чередования единиц и нулей). Это означает, что когда скорость битов — B бит/с, необходима полоса, по крайней мере, $B/2$ Гц. Данное отношение получается из скорости Найквиста. Это — фундаментальный предел, таким образом, мы не можем выполнить NRZ быстрее, не используя большей полосы. Полоса является ограниченным ресурсом даже для проводных каналов, сигналы высокой частоты сильнее затухают, делая их менее полезными, а также требуют более скоростной обработки.

Одна из стратегий более эффективного использования ограниченной полосы состоит в том, чтобы использовать больше чем два сигнальных уровня. При использовании четырех уровней напряжения, например, мы можем послать два бита сразу как один **символ**. Эта схема будет работать, пока сигнал, поступающий в приемник, достаточно силен, чтобы эти четыре уровня различались. Скорость, с которой изменяется сигнал, является тогда половиной битрейта, таким образом, необходима меньшая полоса.

Мы называем скорость, с которой сигнал изменяет уровень, **символьной скоростью**, чтобы отличить ее от понятия **битрейт**. Битрейт — скорость символа, умноженная на число битов в символе. Более старое название символьной скорости, особенно в контексте устройств, названных телефонными модемами, которые передают циф-

ровые данные по телефонным линиям — **бодрейт** (скорость в бодах). В литературе термины *битрейт* и *бодрейт* часто путают.

Заметьте, что количество уровней сигнала не обязательно должно быть степенью двойки. Часто это не так, некоторые из уровней используются для защиты от ошибок и упрощения конструкции приемника.

Синхронизация

Для всех схем, которые кодируют биты в символы, приемник, чтобы правильно расшифровывать биты, должен знать, когда символ заканчивается и начинается следующий символ. С NRZ, в котором символы — просто уровни напряжения, несколько подряд нулей или единиц оставляют сигнал неизменным. Через некоторое время трудно отделить биты, поскольку 15 нулей очень напоминают 16 нулей, если у вас нет очень точных часов.

Точные часы помогли бы с этой проблемой, но они — дорогое решение для массового оборудования. Помните, мы распределяем биты во времени по линиям связи, которые работают на высоких скоростях (много мегабит в секунду), таким образом, часы должны были бы отклоняться меньше чем на долю микросекунды в течение долгой работы. Это могло бы быть разумно для медленных каналов или коротких сообщений, но это не общее решение.

Одна из стратегий состоит в том, чтобы послать в приемник отдельный сигнал часов. Отдельная линия для часов — это не сложно для компьютерных шин или коротких кабелей, в которых есть много параллельных линий, но это расточительно для большинства сетевых каналов: если бы у нас была еще одна линия, чтобы послать сигнал, то мы могли бы использовать ее, чтобы послать данные. Здесь следует исхитриться смешать сигнал часов с сигналом данных и объединить их с помощью «исключающего или» так, чтобы никакая дополнительная линия не была необходима. Результат показан на рис. 2.17, *г*. Часы выполняют передачу сигнала часов в каждый момент прохождения бита, таким образом, это работает на скорости два битрейта. Когда сигнал часов объединяется с уровнем 0, происходит переход низкий-к-высокому, который является просто часами. Этот переход — логический 0. Когда он объединяется с уровнем 1, инвертируется и происходит переход от высокого к низкому. Этот переход — логическая 1. Эту схему называют **Манчестерским кодированием**, она использовалась для классического Ethernet.

Недостаток Манчестерского кодирования в том, что из-за часов требуется вдвое бóльшая полоса пропускания, чем для NRZ, а мы знаем, что эта полоса часто важна. Другая стратегия основана на идее, что мы должны кодировать данные, чтобы гарантировать, что в сигнале есть достаточно много переходов. Предположим, что у NRZ будут проблемы синхронизации только для долгих последовательностей нулей или единиц. Если переходы будут происходить часто, то приемнику будет легко остаться синхронизированным с поступающим потоком символов.

В качестве шага в правильном направлении, мы можем упростить ситуацию, кодируя 1 как переход и 0 как отсутствие перехода, или наоборот. Это кодирование называют **NRZI (Без возвращения к нулю с инверсией)**. Пример показан на рис. 2.17, *в*. Популярный стандарт соединения компьютера с периферийными устройствами —

USB (Универсальная последовательная шина) — использует NRZI. В этом случае длинные последовательности единиц не вызывают проблему.

Конечно, длинные последовательности нулей все еще вызывают проблему, которую мы должны решить. Если бы мы были телефонной компанией, то могли бы просто потребовать, чтобы отправитель не передал слишком много нулей подряд. Более старые цифровые телефонные линии в США, названные **линиями T1**, действительно фактически требовали, чтобы по ним посылали не больше чем 15 последовательных нулей, чтобы работать правильно. Чтобы действительно решить проблему, мы можем разбить последовательности нулей, отображая небольшие группы битов, которые будут переданы так, чтобы группы с последовательным нулями были отображены на немного более длинные образцы, у которых нет слишком длинных последовательностей нулей.

Один из известных кодов для этого называется **4B/5B**. Каждые 4 бита отображены в 5-битовый образец с заданной таблицей преобразования. Эти пять комбинаций двоичных разрядов выбраны так, чтобы никогда не встречались больше трех последовательных нулей. Отображение показано в табл. 2.4. Эта схема добавляет 25 % накладных расходов, что лучше чем 100 % при Манчестерском кодировании. Так как имеется 16 входных комбинаций и 32 выходных комбинации, некоторые из выходных комбинаций не используются. За вычетом комбинаций со слишком многими последовательными нулями остаются еще некоторые. В качестве награды мы можем использовать эти коды, не соответствующие данным, чтобы представить управляющие сигналы физического уровня. Например, иногда «11111» обозначает свободную линию, а «11000» обозначает начало фрейма.

Таблица 2.4. Отображение 4B/5B

Данные (4B)	Ключевое слово (5B)	Данные (4B)	Ключевое слово (5B)
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

Альтернативный подход, который должен заставить данные выглядеть случайными, известен как скремблирование. В этом случае, вероятно, что будут частые переходы. **Скремблер** объединяет с помощью «исключающего или» данные с псевдослучайной последовательностью прежде, чем они будут переданы. Это смешивание делает данные столь же случайными, как псевдослучайная последовательность (предполагается, что они независимы от псевдослучайной последовательности). Приемник тогда применяет «исключающее или» к поступающим битам с той же самой псевдослучайной

последовательностью, чтобы получить реальные данные. Для того чтобы это было практично, псевдослучайную последовательность должно быть легко создать. Обычно это доверяется простому генератору случайных чисел.

Скремблирование привлекательно, потому что оно не добавляет требований к полосе пропускания или времени на служебные данные. Фактически оно используется для создания дополнительных требований к сигналу — энергетические составляющие не должны попадать на основные гармоники (возникающие из-за повторяющихся пакетов данных), поскольку это может привести к возникновению интерференции. Скремблирование идеально подходит для этого случая, потому что случайные сигналы весьма похожи на «белый шум», или их энергия «размазана» по частоте.

Однако скремблирование не гарантирует, что не потребуется никакой длительной обработки. Иногда оно может быть неудачным. Если данные будут похожи на псевдослучайную последовательность, то в результате применения «исключающего или» они превратятся в нули. Такого не произойдет с длинной псевдослучайной последовательностью, которую трудно предсказать. Однако для коротких или предсказуемых последовательностей существует возможность подобрать комбинацию двоичных разрядов, которые вызывают длинные последовательности нулей после шифрования и «портят» связь. В ранних версиях стандартов для отправки IP-пакетов по каналам SONET в телефонной сети этот дефект присутствовал (Malis и Simpson, 1999). Пользователи могли послать определенные «пакеты убийцы», которые гарантировано вызывали проблемы.

Симметричные сигналы

Сигналы, у которых есть столько же положительного напряжения, сколько и отрицательного напряжения даже за короткие периоды времени, называют **симметричными сигналами**. Они составляют в среднем ноль, это означает, что у них нет никакой составляющей постоянного тока. Отсутствие такой компоненты является преимуществом, потому что в некоторых каналах, таких как коаксиальный кабель или линии с трансформаторами, сигнал сильно затухает в силу физических свойств. Кроме того, один из методов присоединения приемника к каналу, называемый **емкостная связь**, пропускает только часть сигнала с переменным током. В любом случае, если мы посылаем сигнал, среднее число которого не ноль, мы тратим впустую энергию, поскольку компонент постоянного тока будет отфильтрована.

Балансирование помогает обеспечить передачу для сигналов синхронизации, так как представляет собой сочетание положительных и отрицательных напряжений. Оно также обеспечивает простой способ калибровать приемники, потому что среднее значение сигнала может быть измерено и использоваться в качестве порога решения для расшифровки символов. В несимметричном сигнале среднее значение может далеко отклониться от истинного решения, например, из-за плотности единиц, в результате больше символов было бы расшифровано с ошибками.

Простейший способ создать симметричный код состоит в том, чтобы использовать два уровня напряжения для представления логической 1, (скажем +1 В или −1 В), и 0 В для представления логического нуля. Чтобы послать 1, передатчик чередует уровни между +1 В и −1 В так, чтобы они всегда давали среднее. Эту схему называют **биполяр-**

ным кодированием. В телефонных сетях ее называют **AMI (Alternate Mark Inversion, схема биполярного кодирования, в которой последовательные объекты кодируются противоположной полярностью)**, основываясь на старой терминологии, в которой 1 называют «маркой» и 0 называют «пробелом». Пример показан на рис. 2.17, *д*.

Биполярное кодирование добавляет уровень напряжения, чтобы достигнуть баланса. Или мы можем использовать для достижения баланса отображение, такое как 4B/5B (так же как переходы для синхронизации). Пример симметричного кода — линейный код **8B/10B**. Он отображает 8 входных бит на 10 выходных бит, таким образом, это на 80 % эффективно, точно так же, как код линии 4B/5B. 8 бит разделены на группу из 5 бит, которые отображаются на 6 бит, и группу из 3 бит, которые отображаются на 4 бита. 6-битовые и 4-битовые символы таким образом связаны. В каждой группе некоторые входные образцы могут быть отображены на симметричные выходные образцы, у которых одинаковое число нулей и единиц. Например, «001» отображен на симметричный «1001». Но комбинаций недостаточно много для того, чтобы все выходные образцы были сбалансированы. Поэтому каждый входной образец отображен на два выходных образца. У каждого будет один с дополнительной 1 и один с дополнительным 0. Например, «000» отображен и на «1011» и на его дополнение «0100». Когда происходит отображение входных битов, кодирующее устройство помнит **неравенство** предыдущего символа. Неравенство — общее количество нулей или единиц, которые делают сигнал не сбалансированным. Кодирующее устройство выбирает или выходной образец, или его альтернативу, чтобы уменьшить неравенство. Для кода 8B/10B самое большое неравенство будет составлять 2 бита. Таким образом, сигнал никогда не будет сильно не сбалансирован. И в нем никогда не будет больше чем пяти последовательных единиц или нулей, что облегчит синхронизацию.

2.5.2. Передача в полосе пропускания

Часто для отправления информации по каналу мы хотим использовать диапазон частот, который начинается не в нуле. Для беспроводных каналов непрактично посылать сигналы на очень низких частотах, потому что размер антенны зависит от длины волны сигнала и становится огромным. В любом случае, выбор частот обычно диктуют регулирующие ограничения и потребность избежать помех. Даже для проводов полезно помещать сигнал в заданный диапазон частот, чтобы позволить различным видам сигналов сосуществовать на канале. Этот вид передачи называют передачей в полосе пропускания, потому что для передачи используется произвольная полоса частот.

К счастью, во всех фундаментальных результатах, изложенных ранее в этой главе, фигурировала полоса пропускания или ширина диапазона частот. Абсолютные значения частоты не имеют значения для полосы пропускания. Это означает, что мы можем взять **низкочастотный сигнал**, который занимает диапазон от 0 до B Гц, и сместить его, чтобы занять **полосу пропускания** от S до $S + B$ Гц, не изменяя количество информации, которую он может перенести, даже при том, что сигнал будет выглядеть по-другому. Чтобы обработать сигнал в приемнике, мы можем сместить его обратно в область низких частот, где более удобно определять символы.

В цифровой модуляции передача в полосе пропускания достигается, регулируя или модулируя сигнал несущей, которая находится в полосе пропускания. Мы можем

модулировать амплитуду, частоту или фазу сигнала несущей. Каждый из этих методов имеет соответствующее название. В **ASK (Amplitude Shift Keying, амплитудная манипуляция)**, чтобы представить 0 и 1, используются две различные амплитуды. Пример с ненулевым и нулевым уровнем показан на рис. 2.18, *б*. Для представления большего числа символов может использоваться большее число уровней. Аналогично, при **FSK (Frequency Shift Keying, частотная манипуляция)** используется два или несколько различных тона. Пример на рис. 2.18, *в* использует только две частоты. В самой простой форме **PSK (Phase Shift Keying, фазовая манипуляция)** несущая систематически подворачивается на 0 или 180 градусов через определенные интервалы времени. Поскольку используются две фазы, этот вид носит название **BPSK (Binary Phase Shift Keying, бинарная фазовая манипуляция)**. «Бинарный» здесь означает два символа, а не то, что символы представляют 2 бита. Пример показан на рис. 2.18, *г*. Улучшенный вариант, который использует полосу канала более эффективно, состоит в том, чтобы использовать четыре сдвига, например на 45, 135, 225 или 315 градусов для передачи двух бит информации за один временной интервал. Этот вариант называют **QPSK (Quadrature Phase Shift Keying, квадратурная фазовая манипуляция)**.

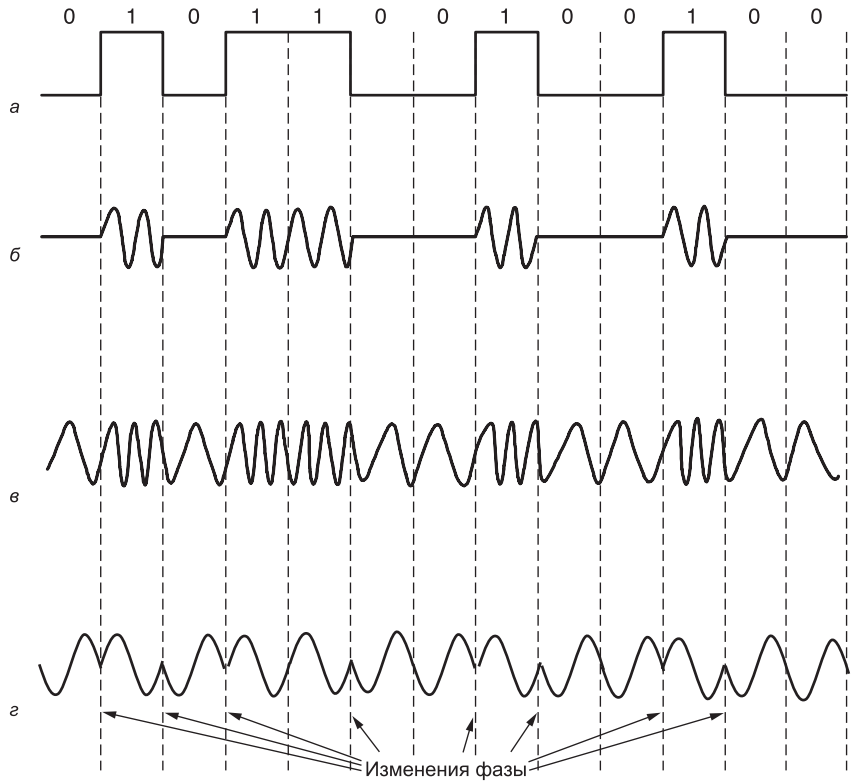


Рис. 2.18. Сигналы: *а* — бинарные; *б* — амплитудная манипуляция; *в* — фазовая манипуляция; *г* — квадратурная фазовая манипуляция

Мы можем объединить эти схемы и использовать больше уровней, чтобы передать больше битов за символ. Одновременно может быть промодулирована только либо

частота, либо фаза, потому что они связаны, частота является скоростью изменения фазы во времени. Обычно комбинируются амплитудная и фазовая модуляция. На рис. 2.19 показана три примера. В каждом примере точки показывают комбинации амплитуды фазы каждого символа. На рис. 2.19, *а* изображены точки, расположенные под углами 45, 135, 225 и 315 градусов, с постоянным уровнем амплитуды (это видно по расстоянию до них от начала координат. Фаза этих точек равна углу, который линия, проведенная через точку и начало координат, составляет с положительным направлением горизонтальной оси. Амплитуда точки — расстояние от начала координат. Этот рисунок — представление QPSK.

Такие диаграммы называются **диаграммами созвездий**. На рис. 2.19, *б* изображен другой комбинированный метод модуляции, с более плотным сигнальным созвездием, использующий 16 комбинаций амплитудных и фазовых сдвигов. С его помощью можно передавать уже 4 бита на символ. Такая схема называется **квадратурной амплитудной модуляцией**, или **QAM-16 (Quadrature Amplitude Modulation)**. На рис. 2.23, *в* изображена еще более плотная схема модуляции. С помощью 64 комбинаций можно в один символ поместить 6 бит. Метод называется **QAM-64**. Используются схемы QAM и более высоких порядков. Как вы можете предположить из этих диаграмм, легче создать электронику, чтобы произвести символы как комбинацию значений на каждой оси, чем как комбинацию значений фазы и амплитуды. Именно поэтому образцы похожи на квадраты, а не концентрические круги.

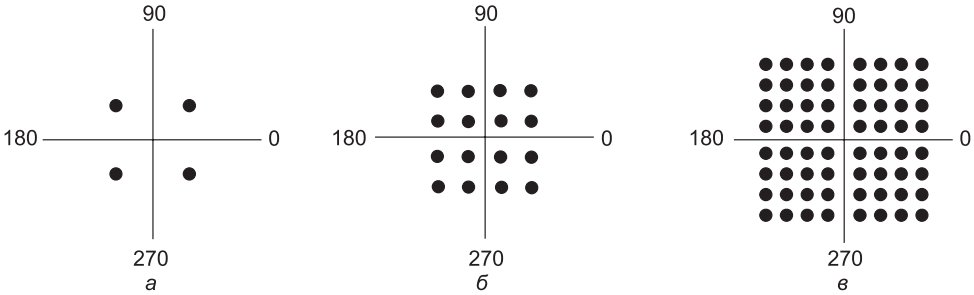
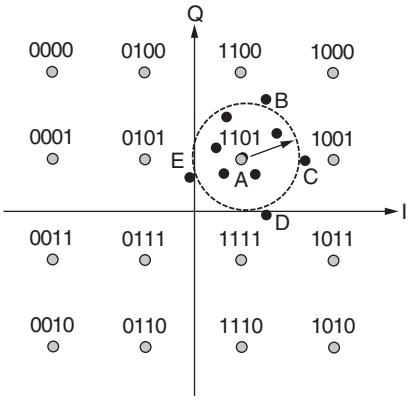


Рис. 2.19. Модуляция: *а* — QPSK; *б* — QAM-16; *в* — QAM-64

Созвездия, которые мы видели до сих пор, не показывают, как биты назначаются символам. При этом важно, чтобы небольшой скачок шума в приемнике не приводил ко многим битовым ошибкам. Это могло бы произойти, если бы мы назначали последовательные битовые значения смежным символам. Используя, например, QAM-16, когда один символ — 0111 и соседний символ — 1000, если приемник по ошибке выбирает смежный символ, все биты будут неправильными. Лучшее решение состоит в том, чтобы отобразить биты на символы так, чтобы смежные символы отличались только по одной позиции двоичного разряда. Это отображение называют **кодом Грэя**. Рисунок 2.20 показывает созвездие QAM-16, к которому применено кодирование Грэя. Теперь, если приемник расшифрует символ ошибочно, он сделает только одну битовую ошибку в ожидаемом случае, что расшифрованный символ близок к переданному символу.



Когда отправлено 1101

Точка	Расшифровано как	Битовых ошибок
A	1101	0
B	1100	1
C	1001	1
D	1111	1
E	0101	1

Рис. 2.20. QAM-16 с применением кодирования Грея

2.5.3. Частотное уплотнение

Схемы модуляции, которые мы рассмотрели, позволяют посылать один сигнал для передачи данных по проводу или беспроводному каналу. Однако экономия ресурсов играет важную роль в сетях передачи данных. Стоимость прокладки и обслуживания магистрали с высокой пропускной способностью и низкокачественной линии практически одна и та же (то есть львиная доля этой стоимости уходит на рытье траншей, а не на сам медный или оптоволоконный кабель). Вследствие этого были развиты схемы мультиплексирования для совместно использования линий многими сигналами.

FDM (Frequency Division Multiplexing, **мультиплексирование с разделением частоты, частотное уплотнение)** использует передачу в полосе пропускания, чтобы совместно использовать канал. Спектр делится на диапазоны частот, каждый пользователь получает исключительное владение некоторой полосой, в которой он может послать свой сигнал. AM-радиовещание иллюстрирует FDM. Выделенный спектр составляет приблизительно 1 МГц, примерно от 500 до 1500 кГц. Другие частоты выделены другим логическим каналам (станциям), каждая станция действует в части спектра, с межканальным разделением, достаточно большим, чтобы предотвратить помехи.

Более подробный пример на рис. 2.21 показывает, как три речевых телефонных канала могут объединяться в одну линию с использованием частотного уплотнения. Фильтры ограничивают используемую полосу частот примерно 3100 Гц на каждый речевой канал. Когда одновременно мультиплексируется множество каналов, на каждый выделяется полоса 4000 Гц. Избыток называют **защитной полосой**. Она сохраняет каналы хорошо отделенными. Для начала сигналы повышаются по частоте, причем для разных каналов величины сдвигов разные. После этого их можно суммировать, поскольку каждый канал теперь сдвинут в свою область спектра. Заметьте, что даже при том, что между смежными каналами благодаря защитным полосам есть промежутки, имеется некоторое наложение. Это связано с тем, что у реальных фильтров нет идеального резкого края. Это означает, что сильный всплеск в одном канале может ощущаться как нетермальный шум в соседнем канале.

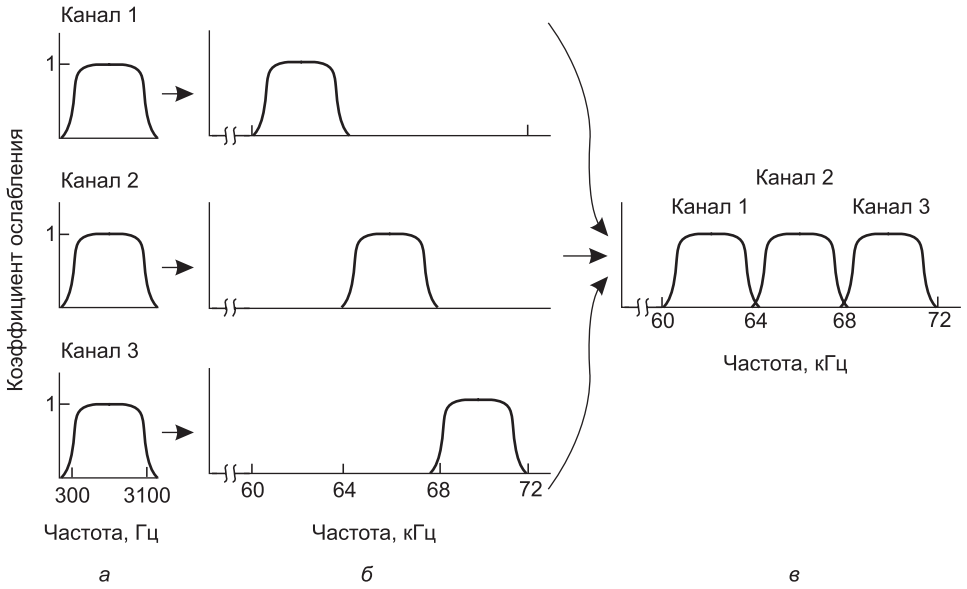


Рис. 2.21. Частотное уплотнение: *а* — исходные спектры сигналов; *б* — спектры, сдвинутые по частоте; *в* — уплотненный канал

Эта схема много лет использовалась для мультиплексирования звонков в телефонной сети, но теперь предпочтение отдается мультиплексированию по времени. Однако FDM продолжает использоваться в телефонных сетях, а также в сотовых, наземных радио и спутниковых сетях на более высоком уровне разбиения.

При отправке цифровых данных возможно эффективно разделить спектр, не используя защитные полосы. В **OFDM (Orthogonal Frequency Division Multiplexing, мультиплексирование с ортогональным частотным разделением)** полоса канала разделена на многие *поднесущие*, которые независимо передают данные (например, с квадратурной амплитудной модуляцией). Поднесущие плотно упакованы вместе в частотной области. Таким образом, сигналы от каждой поднесущей простираются в смежные. Однако, как видно на рис. 2.22, частотная характеристика каждой поднесущей разработана так, чтобы в центре смежных поднесущих это был ноль. Поднесущая поэтому может быть выбрана в своей центральной частоте без помех от соседних. Чтобы это работало, необходим защитный интервал времени, чтобы повторить часть символьных сигналов во времени так, чтобы у них была желаемая частотная характеристика. Однако эти служебные издержки намного меньше, чем при большом количестве защитных полос.

Идея OFDM существовала уже давно, но только в прошлое десятилетие она была широко принята, после того как стало возможно осуществить OFDM эффективно с точки зрения преобразований Фурье цифровых данных по всем поднесущим (вместо того, чтобы отдельно модулировать каждую поднесущую). OFDM используется в 802.11, кабельных сетях и сетях линии электропередачи. Запланировано ее применение в сотовых системах четвертого поколения. Обычно один высокоскоростной

поток цифровой информации разделен на многие потоки с низкой скоростью, которые передаются на поднесущие параллельно. Это разделение значимо, потому что с дефектами канала легче справиться на уровне поднесущих; некоторые поднесущие могут быть очень ухудшены и исключены в пользу поднесущих, которые принимаются хорошо.

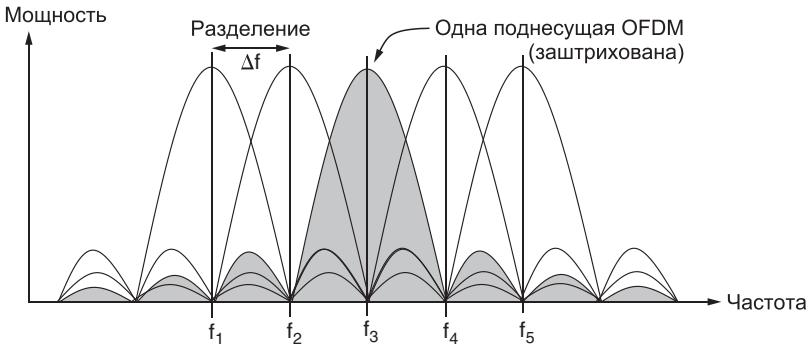


Рис. 2.22. Мультиплексирование с ортогональным частотным разделением (OFDM)

2.5.4. Мультиплексирование с разделением времени

Альтернатива FDM — **TDM (Time Division Multiplexing**, **Мультиплексирование с разделением времени, временно́е уплотнение**). Здесь пользователи сменяются (по кругу), каждый периодически получая всю полосу пропускания на небольшой отрезок времени. Пример трех потоков, мультиплексированных с помощью TDM, показан на рис. 2.23. Биты от каждого входного потока взяты в фиксированный **временной слот** и выведены в совокупный поток. Этот поток движется с суммарной скоростью отдельных потоков. Для этого потоки должны быть синхронизированы по времени. Маленькие **защитные интервалы**, аналогичные защитной полосе частот, могут быть добавлены, чтобы компенсировать небольшие отклонения синхронизации.

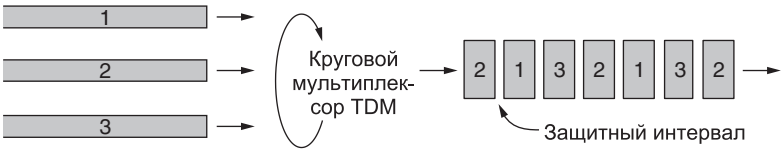


Рис. 2.23. Мультиплексирование с разделением времени (TDM)

TDM широко используется в телефонных сетях и сетях сотовой связи. Чтобы избежать путаницы, давайте отметим, что это очень отличается от альтернативного метода **STDM (Мультиплексирование со статистическим временны́м разделением)**. Слово «статистическое» указывает, что отдельные потоки поступают в мультиплексный поток *не* по фиксированному распорядку, а согласно статистике их запросов. STDM — это другое название для пакетной коммутации.

2.5.5. CDM — кодовое разделение каналов

Третий вид мультиплексирования, которое работает совершенно иначе, чем FDM и TDM, — это **CDM (Мультиплексирование с кодовым разделением, кодовое разделение каналов)**. Оно является формой коммуникации **распределенного спектра**, в которой узкополосный сигнал распределяется по более широкому диапазону частот. Это делает его более терпимым к помехам, а также позволяет нескольким сигналам от различных пользователей совместно использовать общий диапазон частот. Поскольку мультиплексирование с кодовым разделением главным образом используется для этой последней цели, его обычно называют **CDMA (Code Division Multiple Access, множественный доступ с кодовым разделением)**.

В CDMA каждая станция может при передаче все время пользоваться полным спектром частот. Одновременный множественный доступ обеспечивается за счет применения теории кодирования. Прежде чем разбирать алгоритм работы, рассмотрим следующую аналогию. Представьте себе зал ожидания в аэропорту. Множество пар оживленно беседуют. Временное уплотнение можно сравнить с ситуацией, когда все пары людей говорят по очереди. Частотное уплотнение мы сравним с ситуацией, при которой люди говорят на разной высоте звука: одни на высокой, другие низкой, так что все ведут свои разговоры одновременно, но независимо от других. Для CDMA лучше всего подходит сравнение с ситуацией, когда все говорят одновременно, но каждая пара говорящих использует свой язык общения. Франкоговорящие промывают косточки всем остальным, воспринимая чужие разговоры как шум. Таким образом, ключевой идеей CDMA является выделение полезного сигнала при игнорировании всего остального. Далее следует слегка упрощенное описание технологии CDMA.

В CDMA каждый битовый интервал разбивается на m коротких периодов, называемых **элементарными сигналами**, или **чипами (chip)**. Обычно в битовом интервале помещаются 64 или 128 элементарных сигналов. В нашем примере мы будем допускать, что битовый интервал содержит только 8 элементарных сигналов на бит, для упрощения. Каждой станции соответствует уникальный m-битный код, называющийся **элементарной последовательностью**. Из педагогических соображений удобнее использовать биполярную запись в виде последовательности –1 и +1. В скобках будем показывать элементарные последовательности.

Чтобы передать один бит, станция посылает свою элементарную последовательность. Чтобы передать бит со значением 0, нужно отправить вместо элементарной последовательности ее дополнение (все единицы последовательности меняются на нули, а все нули — на единицы). Никакие другие комбинации передавать не разрешается. Таким образом, если $m = 8$ и если станции A соответствует последовательность (–1 –1 –1 +1 +1 –1 +1 +1), она может послать бит «1», передав элементарную последовательность, а бит «0», передав (+1 +1 +1 –1 –1 +1 –1 –1). Здесь посылаются настоящие сигналы с такими уровнями напряжения, но нам достаточно думать о них как о последовательностях чисел.

Увеличить количество информации, которое необходимо передавать (чтобы скорость составила b бит/с, нужно отправлять mb элементарных сигналов в секунду), можно только за счет увеличения в m раз пропускной способности. Таким образом, для CDMA нужна в m раз большая пропускная способность, чем для станции не

применяющей CDMA (предполагая, что никаких изменений в методах модуляции и кодирования не производилось). Если имеется полоса шириной 1 МГц, на которой работают 100 станций, то при частотном уплотнении каждая из них получила бы свои 10 кГц и работала бы со скоростью 10 Кбит/с (предположим, используется 1 бит/Гц). При CDMA каждая станция использует всю ширину диапазона (1 МГц), так что скорость передачи элементарных сигналов достигает сотни и «размазывает» пропускную способность станции 10 Кбит/с на все каналы.

На рис. 2.24, *а* и *б* мы покажем элементарные последовательности четырех станций и сигналы, которыми они представляются.

Каждая станция имеет собственную уникальную элементарную последовательность. Обозначим символом S вектор длины m для станции S, а символом $\overline{S}$ — дополнение S. Все элементарные последовательности попарно **ортогональны**. Мы имеем в виду, что нормированное скалярное произведение двух различных элементарных последовательностей S и T (пишется $S \bullet T$) равно 0. Известно, как генерировать такие последовательности с помощью метода, известного как **коды Уолша**. Используя математическую запись, можно выразить вышесказанное таким образом:

$$S \bullet T \equiv \frac{1}{m}\sum_{i=1}^{m} S_i T_i = 0. \tag{2.3}$$

Попросту говоря, сколько всего пар, столько и разных пар. Это свойство ортогональности мы строго докажем чуть позже. Обратите внимание: если $S \bullet T = 0$, то и $S \bullet \overline{T}$ также равно 0. Нормированное скалярное произведение любой элементарной последовательности на саму себя равно 1:

$$S \bullet S = \frac{1}{m}\sum_{i=1}^{m} S_i S_i = \frac{1}{m}\sum_{i=1}^{m} S_i^2 = \frac{1}{m}\sum_{i=1}^{m} (\pm 1)^2 = 1.$$

Это действительно так, поскольку каждое из m слагаемых суммы равно 1, поэтому вся сумма равна m. Обратите также внимание на то, что $S \bullet S = -1$.

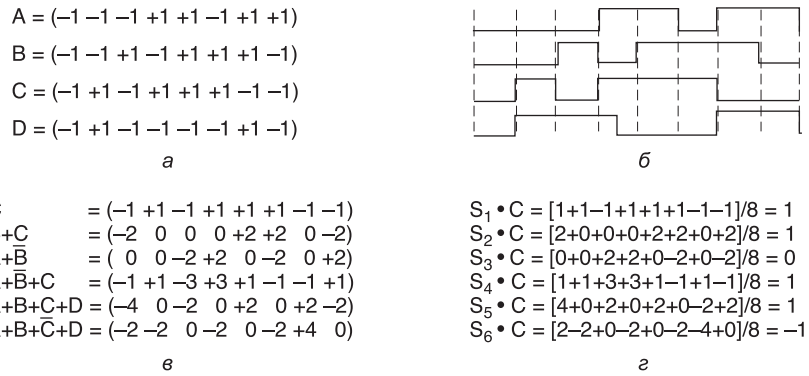


Рис. 2.24. Последовательность: *а* — двоичные элементарные последовательности для четырех станций; *б* — биполярные элементарные двоичные последовательности; *в* — шесть примеров передачи; *г* — восстановление сигнала станции C

В течение каждого битового интервала станция может либо передавать 1, посылая свою элементарную последовательность, либо передавать 0, посылая дополнение к последовательности, либо может молчать и ничего не передавать. Предположим, что все станции синхронизировались во времени, то есть все последовательности начали передаваться в один и тот же момент.

Когда две или более станции пытаются осуществить одновременную передачу, их биполярные сигналы линейно складываются. Например, если при передаче одного элементарного сигнала три станции послали +1, а одна послала –1, то в результате получится +2. Можно рассматривать это как сложение напряжений: три станции имеют на выходе +1 В, а одна имеет на выходе –1 В. В результате сложения получаем +2 В.

На рис. 2.24, *в* изображено шесть примеров передачи, в которой одновременно принимают участие одна или несколько станций. В первом примере *C* передает единичный бит, поэтому мы просто получаем элементарную последовательность этой станции. Во втором примере и *B* и *C* передают единичные биты, в результате чего мы получаем сумму их биполярных последовательностей, а именно:

$$(-1 -1 +1 -1 +1 +1 +1 -1) + (-1 +1 -1 +1 +1 +1 -1 -1) =$$
$$= (0\ -2\ \ 0\ \ 0\ \ 0\ +2\ +2\ \ 0\ -2).$$

Чтобы восстановить исходный битовый поток каждой из станций, приемник должен заранее знать элементарные последовательности всех передатчиков, с которыми он работает. Восстановление осуществляется путем вычисления нормированного скалярного произведения принятой последовательности (то есть линейной суммы сигналов всех станций) и элементарной последовательности той станции, чей исходный сигнал восстанавливается. Если принята элементарная последовательность *S* и приемник пытается понять, что передала станция с элементарной последовательностью *C*, то производится вычисление нормированного скалярного произведения $S \bullet C$.

Чтобы понять, как это все работает, давайте представим себе эти две станции, *A* и *C*. Пусть обе передают единичный бит в то время, как станция *B* передает нулевой бит. Приемник получает сумму сигналов, которая равна: $S = A + \overline{B} + C$, и вычисляет произведение:

$$S \bullet C = (A + \overline{B} + C) \bullet C = A \bullet C + \overline{B} \bullet C + C \bullet C = 0 + 0 + 1 = 1.$$

Первые два слагаемых равны нулю, потому что все пары элементарных последовательностей тщательно подбирались такими, чтобы они были ортогональными, см. выражение (2.3). Теперь понятно, почему это условие должно быть наложено на элементарные последовательности.

Обратимся снова к шести примерам, показанным на рис. 2.24, *в*. Конкретный результат декодирования этих последовательностей представлен на рис. 2.24, *г*. Допустим, приемник заинтересован в извлечении потока битов, посланного станцией *C*, из всех шести последовательностей S_1–S_6. Для этого он вычисляет каждый бит путем суммирования парных произведений принятой последовательности (*S*) и вектора *C* (см. рис. 2.24, *б*), затем деления результата на 8 (так как $m = 8$ в данном случае). Как видите, каждый раз находится верный бит. Это так же просто, как говорить по-французски!

В идеальной системе CDMA без учета шумов, которую мы здесь изучили, допустимое количество станций, которые передают сигналы одновременно, может быть сколь угодно большим, при этом будут использоваться более длинные последовательности чипов. Для 2^n станции коды Уолша могут обеспечить 2^n ортогональных последовательностей длины 2^n. Однако имеется одно существенное ограничение — то, что мы предположили, что все чипы синхронизированы по времени в приемнике. Эта синхронизация даже приближенно не выполняется в таких приложениях, как сотовая связь (где CDMA был широко внедрен начиная с 1990-х).

Мы вернемся к этой теме позже и опишем, чем асинхронный CDMA отличается от синхронного CDMA.

Кроме сотовой связи CDMA используется спутниками и кабельными сетями. Мы в нашем кратком представлении пропустили много усложняющих факторов. Инженеры, которые хотят получить глубокое понимание CDMA, должны читать Viterbi (1995) и Lee and Miller (1998). Чтение этих материалов, впрочем, требует довольно мало знаний в коммуникационной инженерии.

2.6. Коммутируемая телефонная сеть общего пользования

Когда между двумя компьютерами, принадлежащими одной компании и расположенными недалеко друг от друга, необходимо установить связь, часто проще всего оказывается проложить между ними кабель. Подобным образом работают локальные сети. Однако, когда расстояния велики, или компьютеров очень много, или кабель надо прокладывать поперек шоссе или еще какой-либо государственной магистрали, цена прямого кабельного соединения становится недоступно высокой. Кроме того, почти во всех странах мира законом запрещено протягивать частные линии связи над или под объектами государственной собственности. Поэтому проектировщики сетей должны рассчитывать на имеющиеся средства телекоммуникации.

Подобные средства связи, в частности **коммутируемая телефонная сеть общего пользования** (PSTN, Public Switched Telephone Network), были созданы много лет назад с совершенно иной целью — передать человеческий голос в более-менее узнаваемом виде. Их применимость для соединения друг с другом компьютеров весьма незначительна.

Чтобы увидеть размер проблемы, предположим, что дешевый потребительский кабель, работающий между двумя компьютерами, может передать данные со скоростью 1 Гбит/с или больше. При этом типичный ADSL, прекрасная быстрая альтернатива телефонному модему, работает в пределах 1 Мбит/с. Это различие — как различие между полетом на самолете и неторопливой прогулкой.

Тем не менее телефонная сеть плотно переплетена с глобальными компьютерными сетями, поэтому стоит посвятить некоторое время тому, чтобы изучить это подробно. Ограничивающим фактором для организации сети оказывается «последняя миля», по которой соединяются клиенты, а не магистрали и коммутаторы телефонной сети.

Эта ситуация меняется с внедрением оптоволоконной связи и цифровых технологий на краю сети, но это требует времени и денег. В течение долгого времени

проектировщики компьютерных систем, привыкшие работать с устройствами, производительность которых была на три порядка выше, посвятили много времени и сил, чтобы выяснить, как использовать телефонную сеть эффективно.

В следующих разделах мы опишем телефонную систему и покажем, как она работает. Дополнительные сведения о телефонной системе см. в книге (Bellamy, 2000).

2.6.1. Структура телефонной системы

Вскоре после того, как Александр Грэхем Белл (Alexander Graham Bell) в 1876 году (всего на несколько часов раньше своего конкурента, Элиша Грея (Elisha Gray)) запатентовал телефон, на его изобретение появился огромный спрос. Вначале этот рыночный сектор занимался торговлей телефонными аппаратами, которые продавались парами. Задача протягивания между ними единственного провода была возложена на покупателя. Если владелец телефона хотел поговорить с n другими владельцами телефонов, ему приходилось протягивать отдельные провода ко всем n домам. За первый год существования такой телефонной сети города оказались опутанными настоящей сетью из проводов, тянущихся над домами и деревьями в полнейшем беспорядке. Стало очевидно, что модель соединения телефонов «каждый с каждым» работать не будет (рис. 2.25, *а*).

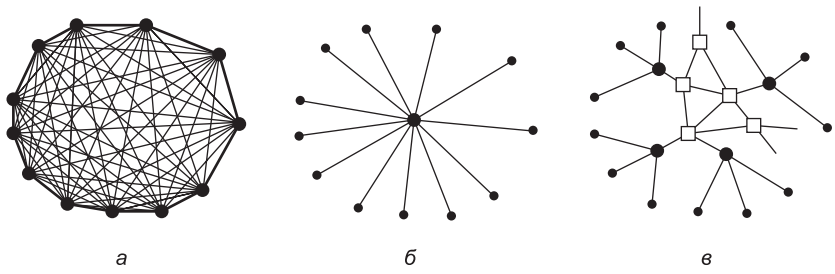


Рис. 2.25. Сеть «каждый с каждым» (*а*); централизованный коммутатор (*б*); двухуровневая иерархия (*в*)

К чести Белла, он заметил это и основал телефонную компанию Bell Telephone Company, открывшую свой первый офис в 1878 году в Нью-Хэйвене, штат Коннектикут. Компания прокладывала провод к каждому дому или офису пользователя. Чтобы позвонить, пользователь должен был покрутить ручку телефона, при этом в офисе телефонной компании звенел звонок, привлекающий внимание оператора, который вручную соединял звонившего с требуемым номером, втыкая разъем в нужное гнездо. Структура телефонной сети с одним коммутатором изображена на рис. 2.25, *б*.

Довольно скоро подобные офисы компании Bell System стали появляться повсюду, кроме того, возник спрос на междугородную связь, поэтому Bell System стала соединять свои офисы. Вскоре они столкнулись опять с той же проблемой: задача соединения каждого офиса с каждым очень быстро стала невыполнимой, поэтому были созданы офисы второго уровня (рис. 2.29, *в*). Через некоторое время количество офисов второго уровня также стало слишком большим. В конце концов, иерархия разрослась до пяти уровней.

К 1890 году были созданы три основные части телефонной системы — коммутаторные телефонные станции, провода, соединяющие пользователей с ними (теперь уже сбалансированные, изолированные витые пары, а не проволока с землей вместо второго провода), и линии междугородной связи, соединяющие отдельные телефонные станции. Краткую техническую историю телефонной системы см. в книге (Hawley, 1991).

Хотя в каждой из этих областей с тех пор производились улучшения, основа модели Bell System осталась неизменной на протяжении более 100 лет. Приведенное ниже описание сильно упрощено; тем не менее оно передает суть дела. Каждый телефон соединен при помощи двух медных проводов с ближайшей **оконечной телефонной станцией** (также называемой **местной центральной станцией**). Расстояние от телефона до ближайшего коммутатора обычно от 1 до 10 км, в городах меньше, чем в сельской местности.

В одних только Соединенных Штатах насчитывается около 22 000 оконечных телефонных станций. Двухпроводное соединение между телефоном каждого абонента и оконечной телефонной станцией называется **местной линией связи** (или локальным контуром). Если местные линии связи всего мира соединить последовательно в одну линию, то их можно будет протянуть до Луны и обратно 1000 раз.

Одно время 80 % капитала компании AT&T было вложено в медные провода местных линий связи. Таким образом, компания AT&T представляла собой самую большую в мире медную шахту. К счастью, этот факт не был широко известен в сообществе инвесторов. В противном случае какие-нибудь корпорации могли скупить AT&T, выкопать все провода и продать их медно-обогатительным комбинатам с целью получения быстрой прибыли, тем самым прекратив всю телефонную связь в США.

Если абонент, подключенный к оконечному коммутатору, позвонит другому абоненту, подключенному к тому же коммутатору, то коммутирующий механизм установит между ними прямое электрическое соединение. Это соединение будет сохраняться в течение всего разговора.

Если же абоненты подключены к разным оконечным станциям, то должна использоваться другая процедура. У каждой оконечной станции имеется несколько линий к одному или нескольким коммутационным центрам, называемым **пригородно-междугородными станциями** (или, если они расположены в одной области, **транзитными станциями**). Соединяющие их линии называются **междугородными**. Число различных видов центров коммутации и их топология различна в разных странах в зависимости от телефонной плотности страны.

Если оба абонента подключены к одной и той же междугородной станции (что вполне вероятно, если они находятся недалеко друг от друга), то связь может быть установлена этой междугородной станцией. На рис. 2.25, *в* показана телефонная сеть, состоящая только из телефонных аппаратов (маленькие точки), оконечных коммутаторов (большие точки) и междугородных станций (квадраты).

Если у абонентов нет общей междугородной станции, то связь между ними будет установлена на более высоком иерархическом уровне. Междугородные станции связываются друг с другом высокоскоростными **межстанционными линиями**. Вплоть до 1984 года, когда распалась корпорация AT&T, телефонная система представляла собой многоуровневую иерархическую маршрутизацию, для поиска пути заходящую на

более высокие уровни иерархии до тех пор, пока не обнаружится общий коммутатор. Эта схема была заменена более гибкой, неиерархической маршрутизацией. Рисунок 2.26 показывает, как могло бы быть направлено дальнее соединение.

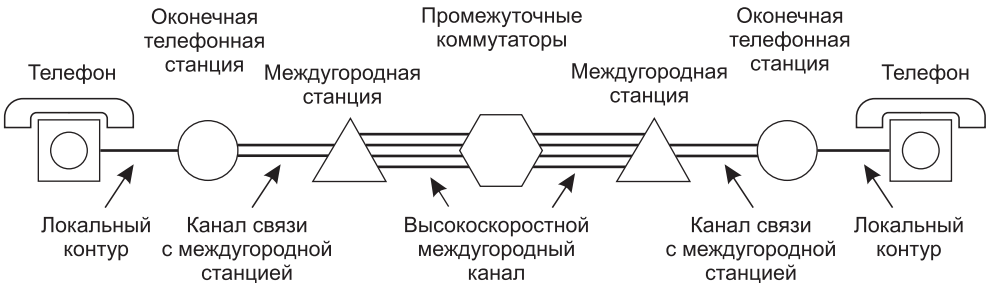


Рис. 2.26. Типичный маршрут связи при большой дистанции между абонентами

В телекоммуникациях применяется широкий спектр сред передачи данных. В отличие от современных офисных зданий, где проводка обычно категории 5, местные линии связи в дома по большей части состоят из витых пар категории 3, а волокно только начинает там появляться. Для связи между коммутаторами широко используются коаксиальные кабели, микроволновая связь и особенно оптоволоконные кабели.

В прошлом телефонная система была аналоговой, в виде электрического напряжения передавался сам голосовой сигнал. С появлением оптического волокна, цифровой электроники и компьютеров стала возможной цифровая передача сигнала на всех уровнях иерархии, кроме местных линий связи — последнего аналогового звена цепи. Цифровая связь предпочтительней потому, что нет необходимости в точности воспроизводить форму аналогового сигнала после того, как он проходит через множество коммутаторов и усилителей. Достаточно распознать лишь одно из двух состояний линии: 0 или 1. Это свойство делает цифровую передачу более надежной, чем аналоговая. К тому же она дешевле и проще в обслуживании.

В целом, телефонная система состоит из следующих трех главных компонентов.

1. Местные линии связи (аналоговые витые пары, подводящиеся в дома и офисы).
2. Магистральные каналы (цифровая связь на базе оптоволокна между коммутационными станциями).
3. Коммутационные станции (в них вызовы переадресуются с одних магистралей на другие).

После того как мы коснемся темы политики телефонии, мы вернемся к более подробному рассмотрению каждого из этих трех компонентов. Местные линии связи — это то, что соединяет каждого конкретного абонента со всей остальной системой, поэтому этот пункт чрезвычайно важен. К сожалению, это и самое слабое место в системе. Междугородным магистралям присуща фундаментальная проблема соединения множества вызовов в один и отправка его по единому кабелю. Это называется *уплотнением*, или мультиплексированием канала, и мы применим для этого частотное и временно́е уплотнение. Мы изучим три стратегии мультиплексирования. Наконец, есть два принципиально разных способа коммутации, которые мы также рассмотрим ниже.

2.6.2. Политика телефонии

На протяжении десятков лет, вплоть до 1984 года, корпорация Bell System предоставляла услуги в области как локальной, так и междугородной связи на большей части Соединенных Штатов. В 1970-х годах правительство США пришло к мнению, что такая монополия является незаконной, и подала судебный иск на компанию AT&T. Правительство выиграло судебный процесс, и 1 января 1984 года компания AT&T была разделена на AT&T Long Lines, 23 местных телефонных компании **BOC** (Bell Operating Company) и еще несколько компаний. 23 компании BOC были объединены в 7 региональных RBOC для повышения их экономической жизнеспособности. По решению суда (это не было актом Конгресса США) за одну ночь вся структура телекоммуникаций в США была изменена.

Детали этой процедуры были описаны в документе под названием **MFJ**. (Modified Final Judgment — измененное окончательное судебное решение. Своего рода оксюморон: если решение могло быть изменено, значит, оно никак не окончательное.) Это событие привело к повышению конкуренции, снижению цен на дальнюю связь, как для частных, так и для корпоративных абонентов. Однако одновременно с этим возросли цены на местную связь, поскольку был нарушен экономический баланс: раньше местные тарифы удерживались низкими именно за счет высоких междугородных тарифов. Во многих других странах сейчас рассматривается вопрос о конкуренции между аналогичными линиями.

Прямое отношение к нашему разговору имеет то, что новая конкурентоспособная структура вызвала ключевую техническую особенность, которая будет добавлена к архитектуре телефонной сети. Чтобы стало понятно, кто чем теперь должен заниматься, Соединенные Штаты были разделены на 164 области, называемые **LATA (Local Access and Transport Area — область локального доступа и транспорта)**. Области LATA приблизительно совпадали с областями, имеющими один и тот же телефонный код. В пределах LATA имелся один локальный оператор связи **LEC (Local Exchange Carrier — местная телекоммуникационная компания)**, обладавший монополией на традиционные телефонные услуги в пределах области LATA. Наиболее важными операторами связи LEC являются компании BOC, хотя в некоторых областях LATA в роли LEC выступает одна из независимых телефонных компаний, общее число которых превышает 1500.

Новым было и то, что связь между областями LATA поддерживалась другой компанией: оператором линии дальней связи **IXC (IntereXchange Carrier)**. Изначально компания AT&T Long Lines была единственным крупным оператором линий дальней связи, однако сегодня корпорации WorldCom и Sprint составляют ей серьезную конкуренцию. Одной из главных проблем при разделении корпорации AT&T было гарантировать, что все владельцы линий дальней связи обеспечат одинаковое качество связи, тарифы и количество цифр в номерах телефонов. Общий вид структуры изображен на рис. 2.27. На нем приведено три примера областей LATA с несколькими оконечными телефонными станциями в каждой из них. Области LATA 2 и 3 включают в себя также небольшую иерархическую структуру с транзитными станциями (внутренними для LATA междугородными станциями).

Любой владелец линий дальней связи может создать свой коммутатор, называемый **POP (Point of Presence — точка присутствия)**, в области LATA для обработки

звонков, исходящих из этой области. Подразумевается, что местный оператор связи LEC соединит каждого владельца линий дальней связи с каждой оконечной станцией либо напрямую, как в случае LATA 1 и 3 (см. рис. 2.27), либо через транзитные коммутаторы, как в случае LATA 2. Кроме того, условия соединения, как технические, так и финансовые, должны быть идентичными для каждого владельца линий дальней связи. В этом случае абонент, скажем, области LATA 1 сможет выбирать себе владельца линий дальней связи для звонков, например, в область LATA 3.

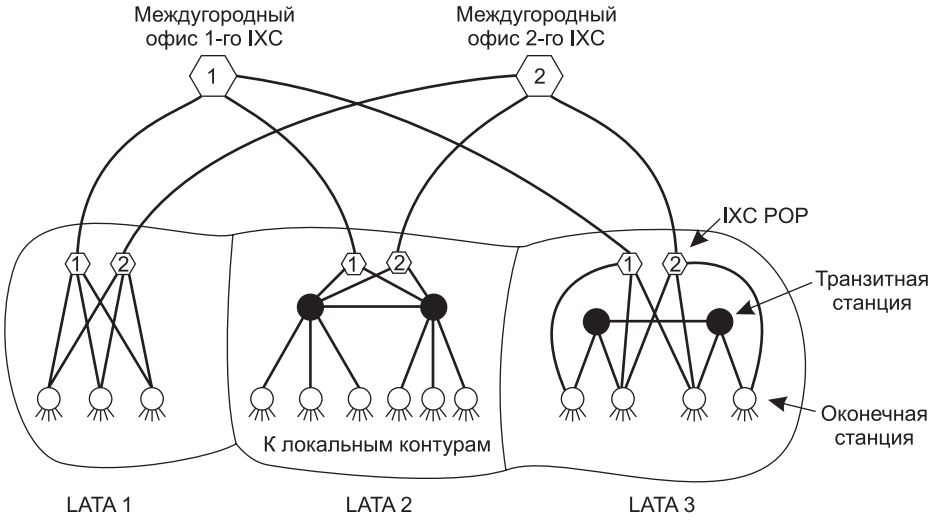


Рис. 2.27. Взаимоотношение между LATA, LEC и IXC. Кружками обозначены коммутаторы LEC. Шестиугольники принадлежат IXC, номер которых указан внутри

Одним из требований документа MFJ было запрещение владельцам линий дальней связи предоставлять услуги в области локальной связи, а операторам связи — в области междугородной связи. Однако никаких других ограничений наложено не было, и все эти компании могли, например, торговать жареными цыплятами и т. п. В 1984 году такой закон звучал вполне недвусмысленно. Однако развивающиеся технологии сыграли довольно злую шутку с законом, в результате чего он оказался устаревшим. Дело в том, что ни кабельное телевидение, ни сотовая телефонная связь не фигурировали в решении суда. По мере того как кабельное телевидение из одностороннего превращалось в двустороннее, а сотовые телефоны становились все популярнее, как локальные операторы связи, так и владельцы линий дальней связи начали сливаться с операторами кабельной и сотовой связи или скупать их.

В 1995 году Конгресс США счел невозможным дальнейшие попытки поддерживать различия между компаниями различного типа и издал проект закона, разрешающий компаниям, занимающимся кабельным телевидением, местным телефонным компаниям, операторам дальней связи и сотовым операторам предоставлять любые услуги этих смежных областей связи. Идея заключалась в том, чтобы позволить компаниям предоставлять клиентам единый интегрированный пакет услуг, включающий в себя кабельное телевидение, телефон и информационные сервисы. Таким образом, компании смогли бы соревноваться в качестве и цене услуг. В феврале 1996 года законо-

проект стал законом, в результате чего многие BOC превратились в IXC, а компании, занимавшиеся, например, кабельным телевидением, стали также предоставлять услуги местной телефонной связи, соперничая с LEC.

Одно интересное предписание можно найти в законе 1996 года: операторы местной связи должны были реализовать **переносимость локальных номеров телефона**. Это означало, что абонент мог сменить оператора, не меняя своего номера телефона.

Переносимость для номеров мобильного телефона (и между стационарными и мобильными телефонами) последовала в 2003 году. Эти условия удалили огромное препятствие для многих людей, делая их намного более склонными сменить местную телекоммуникационную компанию. В результате телекоммуникационный ландшафт США вновь подвергся значительной реструктуризации. Этому примеру снова последовали многие страны. Часто так и происходит: мировое сообщество наблюдает за экспериментами, проводящимися в Соединенных Штатах, и если их результаты оказываются положительными, то многие страны перенимают опыт. Если же результаты оставляют желать лучшего, то каждая страна пробует сделать что-то свое.

2.6.3. Местные линии связи: модемы, ADSL, беспроводная связь

Пришло время вплотную заняться изучением принципов работы телефонной системы.

Начнем с той части телефонной системы, с которой большинство людей знакомо очень хорошо. Итак, имеется двухпроводная линия, идущая от оконечной телефонной станции в дома и небольшие организации. Эта часть называется иногда **последней милей**, хотя длина местной линии на местности может составлять и несколько миль. На этом отрезке вот уже более 100 лет используется аналоговая связь, и похоже, что в ближайшие несколько лет ситуация не изменится (из-за высокой стоимости перехода на цифровые линии).

Много усилий было предпринято по сжатию сети передачи данных из медных проводов местных сетей, которые уже существуют. Телефонные модемы посылают цифровые данные между компьютерами по узкому каналу, который телефонная сеть предусматривает для голосового вызова. Они когда-то широко использовались, но были в значительной степени замещены широкополосными технологиями, такими как ADSL, которые тоже используют местную сеть, чтобы послать цифровые данные от клиента до оконечной станции, где они выходят в Интернет. И модемы и ADSL должны иметь дело с ограничениями старых местных сетей: относительно узкая пропускная способность, ослабление и искажение сигналов и восприимчивость к электрическому шуму, такому как перекрестные наводки.

В некоторых местах местная линия была модернизирована с помощью оптоволокна. За оптоволокном будущее. Такие линии поддерживают компьютерные сети изначально. Ограничивающий фактор — деньги. Вопрос только в том, готовы ли люди платить за дополнительные возможности.

В данном разделе мы рассмотрим как традиционный подход к построению местных линий, так и новые тенденции, наблюдающиеся в этой области. Мы обсудим телефонные модемы, ADSL, оптоволоконное подключение.

Модемы

Чтобы послать биты по местной линии связи или любому другому физическому каналу, с этой целью они должны быть преобразованы в аналоговые сигналы, которые могут быть переданы по каналу. Это преобразование достигается с использованием методов для цифровой модуляции, которую мы изучили в предыдущем разделе. В другом конце канала аналоговый сигнал преобразуется обратно в биты.

Устройство, принимающее последовательный поток битов и преобразующее его в выходной сигнал, модулируемый одним или несколькими из приведенных способов, а также выполняющий обратное преобразование, называется **модемом** (сокращение от «*мод*улятор-*дем*одулятор»). Существуют различные разновидности модемов: телефонные модемы, модемы DSL, кабельные модемы, беспроводные модемы и т. д. Модем может быть встроен в компьютер (это теперь обычная практика для телефонных модемов) или быть отдельным устройством (что типично для DSL и кабельных модемов). Логически, модем вставлен между (цифровым) компьютером и (аналоговой) телефонной линией, как видно на рис. 2.28.

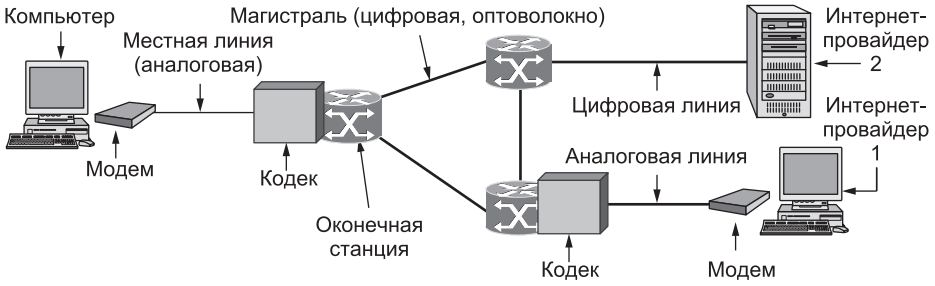


Рис. 2.28. Одновременное использование аналоговой и цифровой связи для соединения компьютеров. Преобразования осуществляются модемами и кодеками

Модем используется для пересылки битов между компьютерами по речевой (аналоговой) телефонной линии, вместо разговора. Основная трудность при этом состоит в том, что речевая телефонная линия ограничена 3100 Гц, чего достаточно, чтобы передать разговор. Эта пропускная способность — на четыре порядка величины меньше, чем пропускная способность, которая используется для Ethernet или 802.11 (WiFi). Неудивительно, скорости передачи данных телефонных модемов также на четыре порядка величины меньше чем у Ethernet и 802.11.

Давайте посчитаем, почему так происходит. Теорема Найквиста утверждает, что даже при наличии идеального канала с частотой 3000 Гц (каковым телефонная линия не является) невозможно передавать отсчеты сигнала чаще, чем с частотой 6000 Гц. На практике большинство модемов делают 2400 отсчетов в секунду, или 2400 бод, стремятся к повышению числа бит на отсчет, допуская одновременный двусторонний трафик (используя для разных направлений разные частоты).

Скромный модем со скоростью 2400 бод использует 0 вольт для передачи логического ноля и 1 вольт для передачи логической единицы, с одним битом на символ. Одно улучшение — можно использовать 4 разных символа, как в четырех фазах QPSK, тогда при двух битах на символ будет достигнута битовая скорость 4800 бит/с.

С развитием технологии был достигнут большой прогресс и высокие скорости. Большие скорости требовали больший набор символов или **совокупность**. При большом количестве символов даже слабый шум при детектировании амплитуды или фазы может привести к ошибке. Для уменьшения этой вероятности были разработаны стандарты, подразумевающие включение в состав каждого отсчета несколько дополнительных битов коррекции. Такие схемы называются **решетчатым кодированием**, или **TCM (Trellis-Coded Modulation)**. (Ungerboeck, 1987.)

Так, например, стандарт V.32 имеет 32 точки на диаграмме для передачи 4 бит данных и один контрольный бит на символ для линии 2400 бод, что позволяет достигнуть скорости 9600 бит/с с коррекцией ошибок. Следующим шагом после скорости 9600 бит/с стала скорость 14 400 бит/с. Новый стандарт был назван **V.32 bis** и передает 6 бит данных и один контрольный бит на отсчет при частоте дискретизации 2400 бод. Совокупность имела теперь уже тысячи символов. Последние модемы этой серии были сделаны в соответствии со стандартом **V.34 bis** и использовали 14 бит/символ при 2400 бод, за счет чего была достигнута скорость 33 600 бит/с.

Почему это предел? Скорость модемов ограничена 33 600 бит/с в связи с тем, что ограничение Шеннона определяет для телефонных линий максимум в 35 Кбит/с для средней длины местных соединений и качества линий. Дальнейшее ускорение невозможно в силу законов термодинамики.

Однако есть способ улучшить ситуацию. На оконечной станции телефонной компании данные для передачи по телефонной сети преобразуются в цифровую форму (ядро телефонной сети преобразовано из аналогового в цифровое уже давно). Для ситуации, когда есть две местные линии связи, по одной в каждом конце, предел — 35 Кбит/с. Каждая из них добавляет шум к сигналу. Если бы мы могли избавиться от одной из этих местных линий, мы увеличили бы ОТНОШЕНИЕ СИГНАЛ/ШУМ, и максимальная скорость была бы удвоена.

Данный подход был реализован в модемах на 56 Кбит/с. Одна сторона как правило, провайдер, получает высококачественную цифровую передачу от ближайшей оконечной станции. Когда хотя бы на одном конце соединения отсутствует «последняя миля» (а большинство провайдеров сейчас уже пользуются только цифровыми каналами), максимальная скорость передачи повышается до 70 Кбит/с. Если же соединение устанавливается между двумя обычными пользователями, каждый из которых передает данные по аналоговой местной линии с помощью модема, максимальная скорость ограничена 33,6 Кбит/с.

Причина, по которой используются модемы со скоростью 56 Кбит/с (а не 70 Кбит/с), связана с теоремой Найквиста. По телефонной линии передаются цифровые отсчеты.

Телефонная линия имеет полосу пропускания около 4000 Гц (включая защитные полосы). Таким образом, максимальное число отсчетов в секунду, необходимых чтобы восстановить сигнал, — 8000. Число бит на отсчет, используемое в США, равно 8, причем 1 бит является контрольным, что позволяет передавать пользовательские данные с битовой скоростью 56 000 бит/с. В Европе все 8 бит являются информационными, поэтому, в принципе, максимальная скорость может достигать 64 000 бит/с, однако международным соглашением установлено ограничение в 56 000 бит/с.

В результате появились стандарты модемов **V.90** и **V.92**. Они предоставляют пользователю возможность передачи данных в сторону провайдера со скоростью

33,6 Кбит/с и 48 Кбит/с, а в обратную сторону — со скоростью 56 Кбит/с. Причиной такой асимметрии является тот факт, что трафик от абонента обычно во много раз меньше трафика от провайдера. Из этого также следует, что большая часть полосы может быть выделена под поток данных от провайдера, тем самым увеличивая его шанс работать действительно со скоростью 56 Кбит/с.

Цифровые абонентские линии

Когда скорость связи по телефонным линиям достигла значения 56 Кбит/с, создалось впечатление, что развитие на этом остановится. Между тем, каналы кабельного телевидения стали предлагать своим абонентам 10 Мбит/с при работе по общему кабелю. Доступ к Интернету стал неотъемлемой частью бизнеса этих компаний, и телефонные компании (прежде всего, уровня LEC) поняли, что необходимы более конкурентоспособные системы. Для этого нужно было предоставить цифровые услуги конечным пользователям, используя местные линии.

Вскоре появилось множество предложений, носящих общее название **xDSL (Digital Subscriber Line — цифровая абонентская линия)**, где вместо буквы *x* могли стоять другие буквы. Системы, использующие каналы с расширенной пропускной способностью, иногда называют **широкополосными сетями**, хотя такой термин является скорее коммерческим, а не техническим. Ниже мы обсудим самую популярную в этой области — **ADSL (Asymmetric DSL — асимметричная DSL)**. Мы будем использовать термины DSL и xDSL для краткого обозначения для всех разновидностей.

Причиной, по которой модемы являются такими медленными устройствами, является то, что телефонные линии, по которым они традиционно работали, были изобретены для передачи человеческой речи и вся система была направлена на оптимизацию именно в этом аспекте. Данные всегда оставались пасынками телефонной системы. В той точке, где заканчивается местная линия (оконечная телефонная станция), сигнал подвергается фильтрации, при которой вырезаются частоты ниже 300 Гц и выше 3400 Гц. Отсечка не является прямоугольной — оба края имеют уровень 3 дБ, — поэтому полоса пропускания считается равной 4000 Гц, хотя на самом деле диапазон между двумя этими краями составляет только 3100 Гц. Таким образом, цифровым данным приходится пробираться по этому узкому каналу.

Хитрость, за счет которой работает xDSL, заключается в том, что ее абоненты подключаются к особому коммутатору, на котором отсутствует описанный выше фильтр. Таким образом, на передачу данных отводится вся полоса пропускания местной линии. Лимитирующим фактором в этом случае становится сама физическая природа линии, а не искусственно вырезанный кусок диапазона в 3100 Гц.

К сожалению, емкость местных линий достаточно быстро падает с увеличением длины линии, и сигнал все более и более ухудшается вдоль провода.

Емкость также зависит от толщины и общего качества витой пары. График зависимости потенциально достижимой пропускной способности от длины линии приведен на рис. 2.29. Здесь предполагается, что все остальные факторы являются оптимальными (новые провода, аккуратные кабели и т. д.).

Реализация такой зависимости создает определенные проблемы для телефонной компании. Когда заявляется определенная скорость работы, автоматически ограничивается радиус, за пределами которого данное предложение не может быть реализовано.

Это означает, что когда приходит клиент, живущий достаточно далеко от телефонной станции, ему говорят: «Спасибо за проявленный интерес, но вы живете на 100 метров дальше от нас, чем нужно для того, чтобы стать нашим абонентом. Не могли бы вы переехать поближе?» Чем ниже предлагаемая скорость, тем больше радиус охвата и количество клиентов. Но, вместе с тем, чем ниже скорость, тем менее привлекательным выглядит предложение и тем меньше абонентов согласится выкладывать свои деньги. Это такой перекресток, на котором встречаются бизнес и технология.

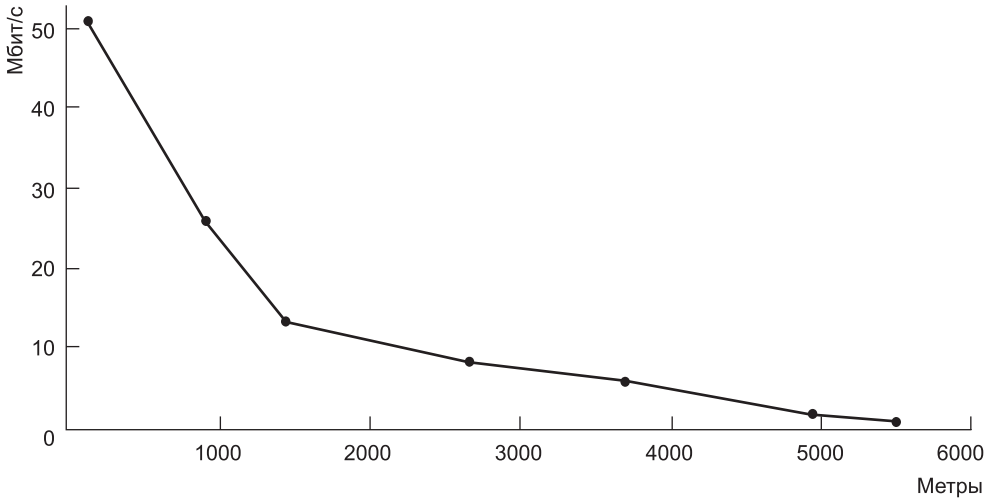


Рис. 2.29. Зависимости пропускной способности от расстояния для DSL по UTP категории 3

Службы xDSL разрабатывались с определенной целью. Во-первых, они должны были работать на существующих местных линиях, представляющих собой витые пары категории 3. Во-вторых, они не должны были влиять на работу аппаратуры абонента вроде телефона и факса. В-третьих, скорость работы должна была быть выше 56 Кбит/с. Наконец, в-четвертых, они должны были предоставлять постоянное подключение, и услуги при этом должны были оплачиваться только в виде фиксированной ежемесячной абонентской платы, но никак не поминутно.

Чтобы удовлетворить техническим целям, доступный спектр на 1,1 МГц на местной линии разделен на 256 независимых каналов 4312,5 Гц каждый. Это распределение показано на рис. 2.30. Первое предложение ADSL исходило от компании AT&T и работало за счет разделения спектра местной линии, который составляет примерно 1,1 МГц, на три частотных диапазона.

Схема OFDM, которую мы рассматривали в предыдущем разделе, посылала данные по этим каналам, поэтому получила, в контексте ADSL, название **дискретная мультитональная модуляция, DMT (Discrete MultiTone)**. Канал 0 используется для **POTS (Plain Old Telephone Service — обычной телефонной сети)**. Каналы с 1 по 5 не используются, чтобы голосовой сигнал не имел возможности интерферировать с информационным. Из оставшихся 250 каналов один занят контролем передачи в сторону провайдера, один — в сторону пользователя, а все прочие доступны для передачи пользовательских данных.

Рис. 2.30. Работа ADSL с использованием дискретной мультитональной модуляции

В принципе, каждый из свободных каналов может быть использован для полнодуплексной передачи, однако из-за помех, взаимной интерференции и так далее практически это не реализуется. Провайдер может самостоятельно определять, сколько каналов использовать для входящего трафика, сколько для исходящего. Технически возможно осуществлять такое разделение в пропорции 50/50, но фактически большинство провайдеров предоставляет 80–90 % пропускной способности для передачи в сторону абонентов, так как большинство пользователей скачивают гораздо больше данных, чем загружают. Обычно под исходящий трафик пользователю отводится 32 канала, по всем остальным информационным каналам он может принимать данные. В целях увеличения пропускной способности можно несколько последних каналов сделать дуплексными, однако это потребует введения в строй дополнительных схем, исключающих образование эха.

В 1999 году был одобрен международный стандарт ADSL, известный как **G.dmt**. Он позволяет принимать входящий трафик со скоростью 8 Мбит/с и отправлять исходящий со скоростью 1 Мбит/с. Его сменило в 2002 году второе поколение, названное ADSL2, с различными усовершенствованиями, что дало скорость целых 12 Мбит/с для входящего трафика и 1 Мбит/с для исходящего. Теперь существует ADSL2+, который удваивает нисходящую скорость (от провайдера к клиенту) до 24 Мбит/с, удваивая пропускную способность, чтобы использовать полосу 2,2 МГц по витой паре.

Однако числа, приведенные здесь, являются скоростями лучшего случая для хороших линий близко (в пределах 1–2 км) к оконечной телефонной станции. Немного линий поддерживают эти скорости, и немного провайдеров предлагают эти скорости. Как правило, провайдеры предлагают приблизительно 1 Мбит/с к клиенту и 256 Кбит/с от клиента (стандартный сервис), соответственно 4 Мбит/с и 1 Мбит/с (улучшенный сервис) и 8 Мбит/с и 2 Мбит/с (премиум-сервис).

В пределах каждого канала модуляция QAM используется на скорости примерно 4000 символов/с. Качество линии в каждом канале постоянно проверяется, и скорость передачи данных корректируется при использовании большей или меньшей совокупности, как на рис. 2.19. У различных каналов могут быть различные скорости передачи данных, начиная с 15 бит на символ, посланными на канале с высоким отношением сигнал/шум, и снижаться до 2, 1 или 0 бит на символ, посланный на канале с низким отношением сигнал/шум в зависимости от стандарта.

Типичная организация ADSL-линии показана на рис. 2.31. На схеме видно, что телефонная компания установила в помещении у абонента специальное **устройство**

сопряжения с сетью, **NID (Network Interface Device)**. Эта маленькая пластмассовая коробочка маркирует окончание зоны владений телефонной компании и начало частной собственности абонента. Недалеко от этого устройства (а иногда вообще в одном блоке с ним) расположен **разветвитель**, который представляет собой аналоговый фильтр, отделяющий полосу POTS (0–4000 Гц) от каналов данных. Сигналы, проходящие по POTS, передаются на имеющийся телефон или факс, а все остальные отправляются на ADSL-модем, который использует цифровой сигнальный процессор для выполнения OFDM. Поскольку большинство модемов ADSL — внешние, то требуется организовать высокоскоростное соединение модема с системным блоком компьютера. Обычно это делается с помощью сетевой карты Ethernet, USB-кабеля или 802.11.

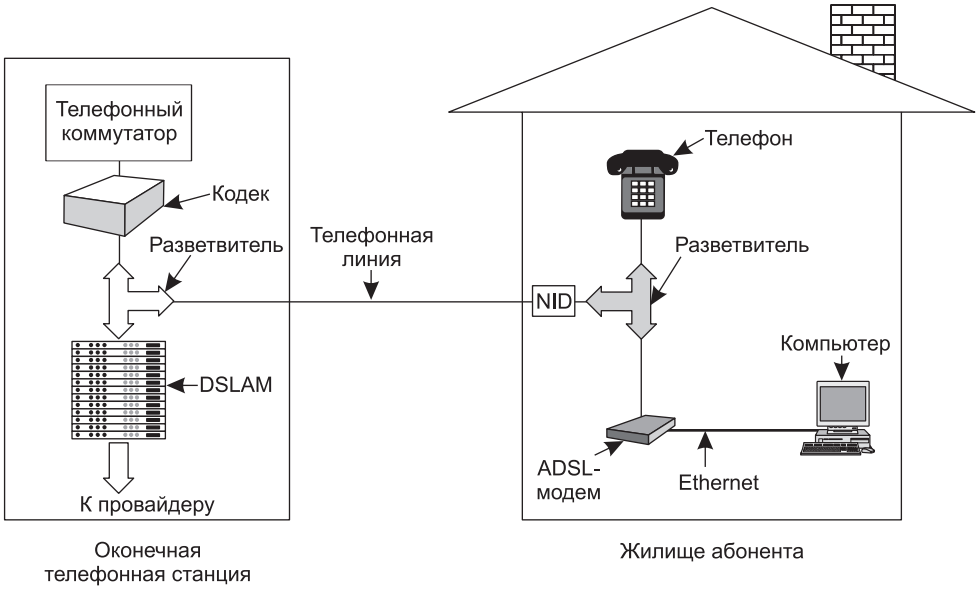


Рис. 2.31. Типичная конфигурация оборудования ADSL

На противоположном конце кабеля, на оконечной коммутационной станции, также установлен разветвитель. Здесь голосовая составляющая сигнала отделяется от информационной и пересылается на обычный телефонный коммутатор. Сигнал, передающийся на частотах, превышающих 26 кГц, отправляется на устройство нового типа, которое называется **мультиплексором доступа к DSL, DSLAM (Digital Subscriber Line Access Multiplexer)**, в состав которого в качестве ADSL-модема входит сигнальный процессор того же типа, что и у абонента. По мере восстановления по цифровому сигналу битовой последовательности формируются пакеты, отсылающиеся провайдеру.

Полное разделение голосовой связи и системы передачи данных позволило телефонным компаниям без особых проблем внедрить ADSL. Требуется всего лишь приобрести DSLAM и разветвитель и подсоединить абонентов ADSL к этому разветвителю. Прочие системы с высокой пропускной способностью (например, ISDN) требуют

гораздо больших усилий для их внедрения и согласования с имеющимся коммутационным оборудованием.

Одним из недостатков представленной на рис. 2.31 системы является наличие NID и разветвителя в жилище пользователя. Установить это оборудование может только технический специалист телефонной компании, что выражается, прежде всего, в высокой стоимости выездных услуг. По этой причине был стандартизован другой вариант комплектации, неофициально названный **G.lite**, в котором отсутствует разветвитель. Фактически это та же самая показанная на рис. 2.31 схема, но без разветвителя у пользователя. Имеющаяся телефонная линия используется как есть. Единственное, что пользователю необходимо сделать, это вставить в разъем каждого телефонного аппарата специальный микрофильтр, который в итоге должен оказаться в схеме между телефоном или ADSL-модемом и телефонной линией. Телефонный микрофильтр вырезает сигналы, частоты которых превышают 3400 Гц. Что же касается фильтра для ADSL-модема, то он, напротив, пропускает только высокие частоты, вырезая диапазон от 0 до 26 кГц. Однако система с разветвителем является более надежной, поэтому максимальная скорость работы G.lite — только 1,5 Мбит/с (против 8 Мбит/с в системах с разветвителем). Более подробно об ADSL можно узнать в Starr (2003).

Волокно до дома

Развернутые медные местные линии ограничивают производительность ADSL и телефонных модемов. Чтобы позволить им обеспечивать более быстрый и качественный сетевой сервис, телефонные компании обновляют местные линии при любой возможности, устанавливая оптоволокно на всем пути до зданий и офисов. Результат называют **FTTH (Fiber To The Home — Волокно до дома)**. Хотя технология FTTH была уже доступна в течение некоторого времени, развертывание только начало расти в 2005 году с ростом спроса на высокоскоростной Интернет у клиентов, привыкших к DSL и кабелю, которые хотели загрузить фильмы. Приблизительно 4 % американских зданий теперь соединены с FTTH со скоростями доступа к Интернету до 100 Мбит/с.

Существуют несколько разновидностей форм «FTTX» (где *X* — это подвал, бордюр или окрестности). Они используются, чтобы отметить, что волокно может приблизиться к дому. В этом случае медь (витая пара или коаксиальный кабель) обеспечивает достаточно быструю скорость на последнем коротком отрезке. Выбор того, как далеко положить волокно, — экономический, необходимо балансировать стоимость и ожидаемый доход. В любом случае дело в том, что оптоволокно пересекло традиционный барьер «последней мили». Мы сосредоточимся в нашем обсуждении на FTTH.

Как медные провода перед этим, волоконный кабель местной линии пассивен. Это означает, что не требуется никакое оборудование с электропитанием для усиления или какой-либо иной обработки сигналов. Волокно просто переносит сигналы между домом и оконечной станцией. Это, в свою очередь, уменьшает стоимость и улучшает надежность.

Обычно волоконные кабели из зданий объединены так, чтобы для группы до 100 зданий только один кабель достигал оконечной станции. В исходящем направлении оптические разделители делят сигнал из оконечной станции так, чтобы он достигал всех зданий. Шифрование необходимо для безопасности, если только один дом должен быть в состоянии расшифровать сигнал. Во входящем направлении оп-

тические объединители сливают сигналы из зданий в единственный сигнал, который получает оконечная станция.

Эту архитектуру называют **PON (Passive Optical Network — пассивная оптическая сеть)**, она показана на рис. 2.32. Обычно всеми зданиями совместно используется одна длина волны для исходящей передачи и другая для входящей.

Даже с разделением огромная пропускная способность и низкое ослабление волокна означают, что PON может предоставить пользователям высокие скорости на расстояниях до 20 км. Фактические скорости передачи данных и другие детали зависят от типа пассивной сети. Распространены два вида. Сети **GPON (Гигабитные PON)** пришли из мира телекоммуникаций, поэтому они определены стандартом ITU. Сети **EPON (Ethernet PON)** больше соответствуют миру сетей, они определены стандартом IEEE. Оба вида имеют скорость около гигабита и могут перенести трафик для различных сервисов, включая Интернет, видео и голос. Например, GPON предоставляют 2,4 Гбит/с исходящего и 1,2 или 2,4 Гбит/с входящего потока.

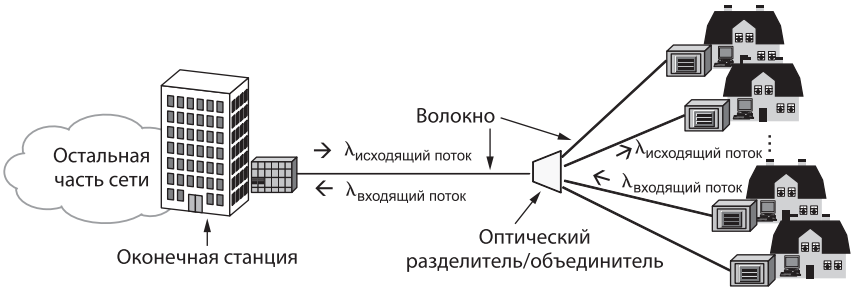


Рис. 2.32. Пассивная оптическая сеть для Волокна до дома

Для того чтобы несколько зданий могли совместно использовать пропускную способность единственного волокна оконечной станции необходим некий протокол. В исходящем направлении это легко. Оконечная станция, может послать сообщения в каждый дом в любом порядке, в котором захочет. В обратном направлении, однако, сообщения из различных зданий нельзя послать в одно время, иначе различные сигналы столкнутся. Здания также не могут услышать передачи друг друга, таким образом, они не могут слушать перед передачей. Решение состоит в том, что оборудование в зданиях запрашивает и получает время, когда оно может использовать оборудование оконечной станции. Чтобы эта система заработала, используется процесс ранжирования, позволяющий настроить время передачи так, чтобы все сигналы, полученные в оконечной станции, были синхронизированы. Этот способ подобен кабельным модемам, которые мы рассмотрим позже в этой главе. Для получения дополнительной информации о будущем пассивных оптоволоконных сетей см. работу (Grobe и Elbers, 2008).

2.6.4. Магистрали и мультиплексирование

Магистрали в телефонной сети не только намного быстрее, чем местные линии, они отличаются в двух других отношениях. Ядро телефонной сети переносит цифровую информацию, а не аналоговую; то есть биты, не голос. Это требует преобразования

в оконечной станции к цифровой форме для передачи по магистралям на большие расстояния. Магистрали переносят тысячи, даже миллионы звонков одновременно. Это совместное использование важно для достижения экономии за счет роста производства, так расходы на установку и поддержку магистрали высокой пропускной способности и магистрали низкой пропускной способности (между двумя коммутаторами) практически одинакова. Это достигается с помощью частотного и временного мультиплексирования.

Ниже мы кратко исследуем как голосовые сигналы оцифрованы так, чтобы они могли быть транспортированы телефонной сетью. После этого мы увидим, как используется временное мультиплексирование, чтобы перенести биты на магистраль, включая систему TDM, используемую для волоконной оптики (SONET). Затем мы обратимся к FDM, поскольку она применяется в волоконной оптике, где называется мультиплексированием разделения длины волны.

Оцифровка голосовых сигналов

На ранних этапах развития телефонной сети ядро обрабатывало голосовые вызовы как аналоговую информацию. Методы FDM много лет применялись для мультиплексирования голосовых 4000 Гц (3100 Гц плюс защитная полоса) каналов в большие и еще бóльшие блоки. Например, 12 звонков в диапазоне от 60 до 108 кГц известны как **группа**, а пять групп (в общей сложности 60 звонков) известны как **супергруппа** и т. д. Эти методы FDM все еще используются для некоторых медных проводов и микроволновых каналов. Однако FDM требует аналоговой схемы и не поддается тому, чтобы быть реализованным компьютером. TDM, напротив, может быть обработан полностью цифровой электроникой, таким образом, этот вид уплотнения стал намного более широко распространенным в последние годы. Так как TDM может использоваться только для цифровых данных, а местные линии производят аналоговые сигналы, необходимо преобразование из аналогового в цифровой сигнал в оконечной станции, куда приходят все отдельные местные линии, чтобы быть объединенными на исходящие магистрали.

Аналоговые сигналы оцифровываются в оконечной станции конца устройством, названным **кодеком** (сокращение для «кодер-декодера»). Кодер-декодер делает 8000 отсчетов в секунду (125 мкс/отсчет), потому что теорема Найквиста говорит, что этого достаточно, чтобы получить всю информацию от телефонного канала с полосой пропускания 4 кГц. При более низкой скорости осуществления выборки была бы потеряна информация; при более высокой не была бы получена никакая дополнительная информация. Каждый отсчет амплитуды сигнала квантуется к 8-битовому числу.

Этот метод называют **PCM** (**Pulse Code Modulation**, **импульсно-кодовая модуляция**). Она образует сердце современной телефонной сети. Как следствие, фактически все временны́е интервалы в пределах телефонной сети — числа, кратные 125 мкс. Стандартная несжатая скорость передачи данных для телефонного звонка, таким образом, составляет 8 бит на каждые 125 мкс, или 64 Кбит/с.

В другом конце звонка аналоговый сигнал восстанавливается из квантованных образцов, проигрывая и сглаживая их по времени. Результат не будет точно таким

же, как оригинальный аналоговый сигнал, даже если мы действовали на скорости Найквиста, потому что образцы квантовались. Чтобы уменьшить ошибку из-за квантизации, уровни квантизации расположены с неравными интервалами. Используется логарифмическая шкала, что дает относительно больше битов меньшим амплитудам сигнала и относительно меньше битов бóльшим амплитудам сигнала. Таким образом, ошибка пропорциональна амплитуде сигнала.

Широко используются две версии квантизации: **μ-закон (μ-law)**, используемый в Северной Америке и Японии, и **A-закон (A-law)**, используемый в Европе и остальной части мира. Обе версии определены в стандарте ITU G.711. Представьте, что динамический диапазон сигнала (или отношение между самыми большими и самыми маленькими значениями) сжимается прежде, чем сигнал будет (равномерно) квантоваться, и затем расширяется, когда аналоговый сигнал воссоздается. По этой причине данный способ называют **компандированием (companding)**. Также возможно сжать отсчеты после того, как они оцифрованы, так чтобы они потребовали намного меньше, чем 64 Кбит/с. Однако мы оставим эту тему до того момента, когда будем исследовать аудиоприложения, такие как IP-телефония.

Мультиплексирование с разделением времени

Мультиплексирование с разделением времени, основанное на импульсно-кодовой модуляции, используется, чтобы передать несколько голосовых вызовов по магистралям, посылая выборку из каждого звонка каждые 125 мкс. Когда цифровая передача стала реальностью, ITU (CCITT) был неспособен достигнуть соглашения по международному стандарту для импульсно-кодовой модуляции. В результате в разных странах используется множество несовместимых схем.

Методом, используемым в Северной Америке и Японии, является канал **T1**, формат кадра которого изображен на рис. 2.33. (Технически говоря, формат называется DS1, а канал называется T1, но мы не будем делать здесь это тонкое различие, следуя широко распространенной промышленной традиции.) Линия T1 состоит из 24 голосовых каналов, мультиплексированных вместе. Каждый из этих 24 каналов, в свою очередь, вставляет в выходной поток 8 бит.

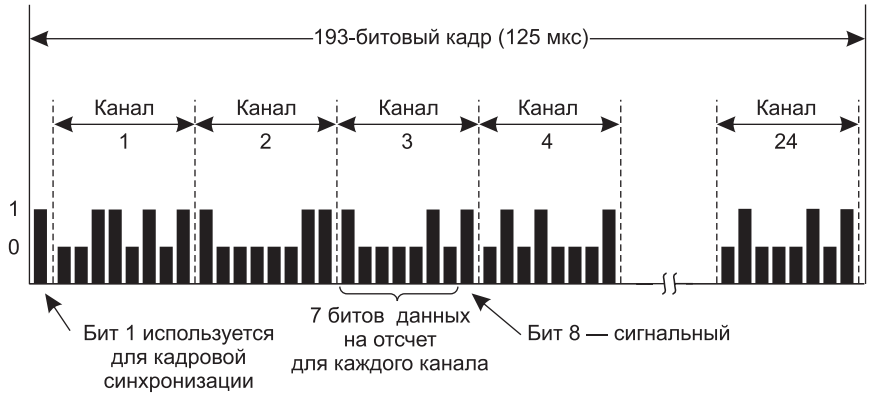


Рис. 2.33. Канал T1 (1,544 Мбит/с)

Кадр состоит из 24 × 8 = 192 бита плюс один дополнительный бит в целях управления, итого 193 бита каждые 125 мкс. В результате это дает огромную суммарную скорость передачи данных в 1,544 Мбит/с. 193-й бит используется для синхронизации кадров и сигналов. В одном варианте 193-й бит используется после группы из 24 кадров и называется **расширенный суперфрейм**. Шесть битов, в 4-й, 8-й, 12-й, 16-й, 20-й и 24-й позиции, взятые из образца 001011 . . . Обычно приемник постоянно проверяет этот образец, чтобы убедиться, не потерял ли он синхронизацию. Если это вдруг происходит, то приемник сканирует принятые данные, отыскивая кадровый бит и с его помощью восстанавливая синхронизацию. Еще 6 битов используются для того, чтобы послать код проверки ошибок, чтобы помочь приемнику подтвердить, что он синхронизирован. Если он теряет синхронизацию, приемник может просмотреть на образец и подтвердить код проверки ошибок, чтобы повторно синхронизироваться. Оставшиеся 12 бит используются для получения информации для управления и поддержки сети, такой как выполнение сообщений от удаленного конца.

У формата T1 есть несколько разновидностей. Более ранние версии послали сигнальную информацию **в полосе**, то есть в том же самом канале, что и данные, используя некоторые из битов данных. Эта схема — одна из форм **сигнализации, ассоциированной с каналом**, потому что у каждого канала есть свой собственный сигнальный подканал. В одном из способов младший значащий бит из 8-битового образца на каждом канале используется в каждом шестом кадре. Этот вариант красочно называется **сигнализация с ограблением бита**. Идея состоит в том, что несколько украденных битов не будут иметь значения для голосовых вызовов. Никто не услышит различия.

Для данных, однако, ситуация другая. Отправка неправильных битов, по меньшей мере, бесполезна. Если используются более старые версии T1, только 7 из 8 битов, или 56 Кбит/с, могут использоваться в каждом из этих 24 каналов. Вместо этого более новые версии T1 обеспечивают чистые каналы, в которых все биты могут использоваться, чтобы послать данные. Чистые каналы — то, что хотят фирмы, которые арендуют линию T1, когда они посылают данные через телефонную сеть вместо голосовых отсчетов. Сигнализация для любых голосовых вызовов тогда обработана **вне полосы**, то есть в отдельном от данных канале. Часто сигнализация сделана с **общим сигнализирующим каналом**, в котором есть совместно используемый сигнальный канал. Один из 24 каналов может использоваться с этой целью.

Вне Северной Америки и Японии вместо T1 используется 2048 Мбит/с канал **E1**. У этого канала есть 32 8-битовых образца данных, упакованные в основные 125 мкс кадры. Тридцать из каналов используются для информации и до двух используются для сигнализации. Каждая группа из четырех кадров обеспечивает 64 сигнальных бита, половина которых используются для того, чтобы сигнализировать (или связаны с каналом, или общий канал), и половина используются для синхронизации кадра или зарезервированы для каждой страны для использования по желанию.

Мультиплексирование с разделением времени позволяет нескольким каналам T1 быть мультиплексированными в каналы высшего порядка. Рисунок 2.34 показывает, как это может быть сделано. Слева мы видим четыре канала T1, объединяемых в один канал T2. Мультиплексирование в канале T2 и каналах более высоких порядков выполняется побитно, а не побайтно, причем 24 голосовых канала составляют один кадр T1. Четыре потока T1 по 1,544 Мбит/с образуют 6,176 Мбит/с, однако реально

в канале T2 используется скорость передачи, равная 6,312 Мбит/с. Дополнительные биты используются для синхронизации кадров и восстановления в случае сбоя канала. T1 и T3 активно используются рядовыми пользователями, тогда как T2 и T4 можно найти только внутри телефонной системы, поэтому они не столь известны.

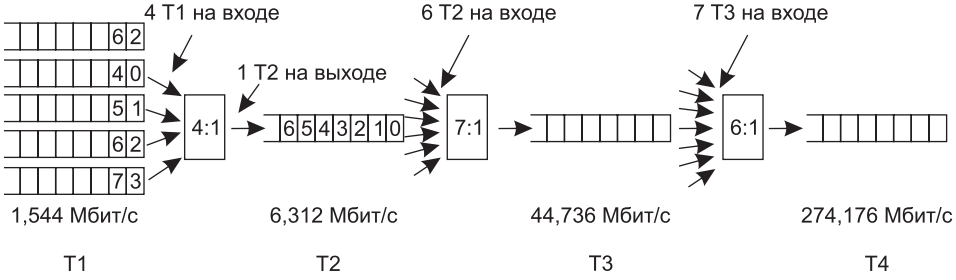


Рис. 2.34. Мультиплексирование потоков T1 на каналах высших порядков

На следующем уровне семь каналов T2 объединяются побитно в канал T3. Затем шесть потоков T3 формируют поток T4. На каждом этапе добавляется небольшое количество избыточной информации для синхронизации кадров.

Между США и остальным миром нет почти никаких договоренностей по поводу основного канала, а также о том, каким образом они должны мультиплексироваться в каналы более высоких уровней. Американская схема объединения по 4, 7 и 6 не затрагивает остальных, и, например, стандарты ITU предполагают мультиплексирование по четыре потока в один поток на каждом уровне. Кроме того, данные о структурировании и восстановлении отличаются в США и в стандартах ITU. Иерархия ITU объединяет по 32, 128, 512, 2048 и 8192 канала, соответственно работающих на скоростях 2,048, 8,848, 34,304, 139,264 и 565,148 Мбит/с.

SONET/SDH

Когда оптоволоконная связь только появилась, у каждой телефонной компании была своя собственная система мультиплексирования с разделением времени. После раздела в 1984 году корпорации AT&T местным телефонным компаниям пришлось подключаться к различным междугородным линиям с различными оптическими системами TDM. Появилась очевидная потребность в стандартизации. В 1985 году исследовательское подразделение региональных телефонных компаний Bellcore начало разработку стандарта **SONET (Synchronous Optical Network — синхронная оптическая сеть)**.

Позднее к этой работе подключился ITU, что привело в 1989 году к созданию стандарта SONET, а также набора параллельных рекомендаций ITU (G.707, G708 и G709). Рекомендации ITU, получившие название **SDH (Synchronous Digital Hierarchy — синхронная цифровая иерархия)**, отличаются от стандарта SONET небольшими деталями. Практически все телефонные линии дальней связи в США и во многих других странах сегодня используют SONET на физическом уровне. Дополнительную информацию см. в книгах (Bellamy, 2000; Goralsky, 2000; Shepard, 2001).

При разработке системы SONET ставились четыре главные цели. Во-первых, SONET должен был обеспечивать объединение сетей, построенных на различных

носителях. Для достижения этой цели потребовалось определить общий стандарт, описывающий длины волн, синхронизацию, структуру кадра и другие вопросы.

Во-вторых, требовалось средство объединения цифровых систем США, Европы и Японии, построенных на 64 Кбит/с каналах с импульсно-кодовой модуляцией, но использующих эти каналы различными (и не совместимыми друг с другом) способами.

В-третьих, SONET должен был предоставить способ объединения нескольких цифровых каналов. Во время разработки системы SONET наиболее быстрым широко используемым в США каналом был T3 со скоростью 44,736 Мбит/с. Стандарт T4 уже был описан, но мало использовался, а стандарта выше T4 определено не было. Одной из задач SONET было продолжить иерархию до скоростей, измеряющихся в гигабитах в секунду. Также нужен был стандартный способ объединения нескольких медленных каналов в один канал SONET.

В-четвертых, SONET должен был обеспечить поддержку операций, администрирования и обслуживания (OAM, Operation, Administration, Maintenance). Предыдущие системы справлялись с этой задачей не слишком хорошо.

Вначале было решено реализовать SONET на основе традиционной системы мультиплексирования с разделением времени, при этом вся пропускная способность оптоволоконного кабеля выделялась одному каналу, который разбивался на интервалы времени, выделяемые подканалам. SONET как таковая является синхронной системой. Интервалы между посылаемыми битами управляются таймером с точностью 10^{-9}. Биты отсылаются в линию также в строго определенные моменты времени, контролируемые главным таймером.

Обычный кадр SONET представляет собой блок из 810 байт, выдаваемых каждые 125 мкс. Поскольку SONET является синхронной системой, кадры выдаются независимо от наличия какой-либо полезной информации, которую необходимо переслать. Скорость 8000 кадров в секунду очень точно соответствует частоте дискретизации каналов PCM, используемых в телефонии.

Проще всего описать кадр SONET из 810 байт в виде прямоугольника из 9 рядов по 90 колонок. Тогда очевидно, что $8 \times 810 = 6480$ бит передаются 8000 раз в секунду, что дает скорость передачи 51,84 Мбит/с. Это основной канал SONET, называющийся **STS-1 (Synchronous Transport Signal — синхронный транспортный сигнал)**. Все магистрали SONET кратны STS-1.

Первые три колонки каждого кадра зарезервированы под системную управляющую информацию, как показано на рис. 2.35. Первые три ряда содержат заголовок раздела, следующие шесть рядов — заголовок линии. Заголовок секции генерируется и проверяется в начале и конце каждого раздела, тогда как заголовок линии генерируется и проверяется в начале и конце каждой линии.

Передатчик SONET посылает соседние кадры по 810 байт без межкадровых промежутков, даже если данных для передачи нет (в этом случае посылаются фиктивные байты). С точки зрения приемника это выглядит как бесконечный битовый поток. Как же он узнает, где находится граница каждого кадра? Дело в том, что первые два байта кадра содержат фиксированную последовательность, которую приемник и старается найти. Если в большом количестве последовательно принятых кадров обнаруживается одна и та же комбинация нулей и единиц, то логично предположить,

что это граница кадра и приемник считает себя синхронизированным с передатчиком. Теоретически пользователь может регулярно вставлять служебную последовательность в поток, но на практике это не может сбить с толку приемник, так как данные, отправляемые несколькими пользователями, подвергаются уплотнению и по другим причинам.

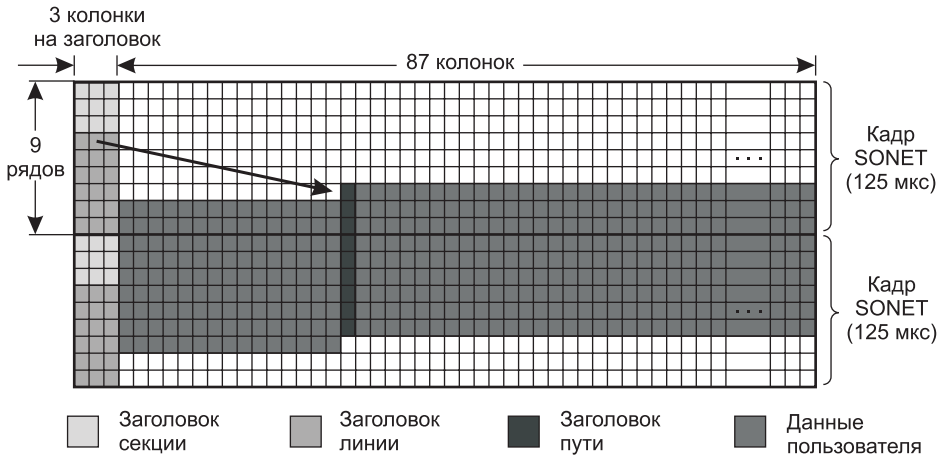


Рис. 2.35. Два соседних кадра системы SONET

В оставшихся 87 столбцах содержатся данные пользователя. Они передаются со скоростью $87 \times 9 \times 8 \times 8000 = 50{,}112$ Мбит/с. Эти данные могут быть и голосовыми отсчетами, которые T1 и другие каналы берут целиком, и пакетами. SONET — это просто удобный контейнер для транспортировки битов. На самом деле данные пользователя, называемые **синхронным полезным пакетом**, **SPE (Synchronous Payload Envelope)**, не всегда начинаются с первой строки и четвертой колонки. SPE может начинаться где угодно в пределах кадра. А указатель на его первый байт хранится в первой строке заголовка линии. Первой колонкой SPE является заголовок пути (то есть заголовок для сквозного протокола подуровня).

Возможность начинать SPE в любом месте кадра SONET и даже занимать соседние два кадра, как показано на рис. 2.35, придает системе дополнительную гибкость. Например, если данные пользователя прибывают на источник, в то время как пустой кадр SONET уже передается, то они могут быть вставлены в текущий кадр, а не ждать начала следующего кадра.

Иерархическая система мультиплексирования SONET/SDH показана в табл. 2.5. Определены скорости для синхронных транспортных сигналов от STS-1 до STS-768, в пределах примерно от канала T3 до 40 Гбит/с. Еще более высокие скорости будут, конечно, определены со временем, канал OC-3072 на 160 Гбит/с будет следующей моделью, когда это станет технологически выполнимым. Оптический канал (OC, Optical Carrier), соответствующий n-му синхронному транспортному сигналу (STS-n), называется OC-n и совпадает с STS-n с точностью до бита, с той разницей, что для синхронизации требуется некоторая перестановка битов. Названия SDH отличаются — они начинаются с OC-3, так как системы на основе рекомендаций ITU не имеют

стандартизованной скорости 51,84 Мбит/с. Мы показали обычные скорости, которые, начиная с OC-3, кратны 4. В общую скорость потоков данных включены все управляющие сигналы. В скорость передачи полезной нагрузки SPE не входят заголовки линий и разделов. В скорость передачи данных пользователя включаются только 87 полезных колонок кадра.

Если какой-нибудь из данных каналов, например OC-3, не является мультиплексным, а переносит данные от одного источника, то к его названию добавляется латинская буква «*c*», означающая *concatenated* (объединенный). Таким образом, OC-3 означает 155,52-Мбитный канал, состоящий из трех отдельных каналов OC-1, а OC-3c означает передачу потока данных от одного источника со скоростью 155,52 Мбит/с. Три потока OC-1 в составе потока OC-3c разделяются одной колонкой. Первая колонка 1 отделяет поток 1, затем колонка 1 — поток 2, колонка 1 — поток 3, затем колонка 2 — поток 1 и так далее до конца кадра шириной 270 колонок и глубиной 9 строк.

Таблица 2.5. Скорости мультиплексирования SONET и SDH

SONET		SDH	Скорость передачи данных, Мбит/с		
Электрические	**Оптические**	**Оптические**	**Общая**	**SPE**	**Пользователя**
STS-1	OC-1		51,84	50,112	49,536
STS-3	OC-3	STM-1	155,52	150,336	148,608
STS-9	OC-9	STM-3	466,56	451,008	445,824
STS-12	OC-12	STM-4	622,08	601,344	594,432
STS-48	OC-48	STM-16	2488,32	2405,376	2377,728
STS-192	OC-192	STM-64	9953,28	9621,504	9510,912
STS-768	OC-768	STM-256	39813,12	38486,06	38043,648

Спектральное уплотнение

В оптоволоконных каналах используется особый вариант частотного уплотнения. Он называется **спектральным уплотнением (WDM, Wavelength-Division Multiplexing)**. Способ реализации частотного уплотнения в оптоволоконных линиях показан на рис. 2.36. Здесь четыре кабеля подходят к одному сумматору, и по каждому из них идет сигнал со своей энергией в своем частотном диапазоне. Четыре луча объединяются и дальше распространяются по одному волокну. На противоположном конце они расщепляются разветвителем. На каждом выходном кабеле имеется короткий специальный участок внутреннего слоя, который является фильтром, пропускающим сигнал только одной длины волны. Получающиеся сигналы могут быть направлены их месту назначения или повторно объединены разными способами для дальнейшей мультиплексной передачи.

Данный метод не представляет собой ничего нового. Это просто частотное уплотнение на очень высоких частотах. Этот способ работы является просто мультиплексированием разделения частоты в очень высоких частотах, в терминах WDM оптические

волоконные каналы описываются их длиной волны или «цветом», а не частотой. Поскольку каждый сигнал передается в своем частотном диапазоне, и эти диапазоны успешно разделяются, подобный вариант мультиплексирования может применяться для передачи на большие расстояния. Единственным отличием от электрического частотного уплотнения является в данном случае то, что система, использующаяся для уплотнения, то есть призма или дифракционная решетка, является абсолютно пассивным, а следовательно, чрезвычайно надежным элементом.

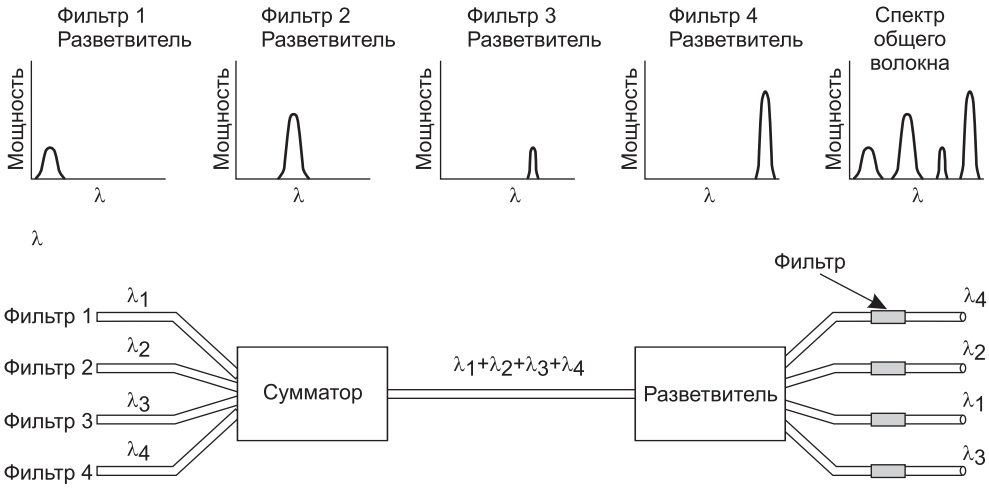


Рис. 2.36. Спектральное уплотнение

Причина, по которой спектральное уплотнение является популярным, в том, что один оптический кабель обычно работает на частоте не более нескольких гигагерц из-за невозможности более быстрого преобразования электрических сигналов в оптические и обратно. Однако, объединяя сигналы разных длин волн на одном кабеле, можно получить суммарную пропускную способность, линейно зависящую от числа каналов. Полоса пропускания одного волокна составляет 25 000 Гц (см. рис. 2.6), следовательно, теоретически даже при 1 бит/Гц можно разместить 2500 каналов по 10 Гбит/с (и более высокие скорости тоже возможны).

Технология WDM развивается с такой скоростью, что компьютерным технологиям остается только стыдиться перед ней своих темпов развития. Она была изобретена примерно в 1990 году. Первые коммерческие системы использовали 8 каналов по 2,5 Гбит/с на канал. К 1998 году на рынке появились уже 40-канальные системы с такой же пропускной способностью канала. В 2006 году уже были системы из 192 каналов по 10 Гбит/с и из 64 каналов по 40 Гбит/с. Такой емкости достаточно, чтобы передавать 80 полнометражных DVD-фильмов в секунду. Каналы также плотно размещены в волокне, разделенные 200, 100 или всего 50 ГГц. Эксперименты компаний, проведенные в лабораторных условиях, показали десятикратное превосходство этой технологии, но путь от лаборатории до внедрения обычно занимает несколько лет. Когда число каналов очень велико, а длины волн отличаются на очень малые величины, системы называют **плотными WDM**, или **DWDM** (Dense WDM).

Еще одной новой разработкой является оптический усилитель. Раньше необходимо было через каждые 100 км разбивать сигнал на каналы, преобразовывать оптические каналы в электрические и усиливать последние традиционным способом, после чего выполнять обратное преобразование и объединение. Теперь же любые оптические усилители могут регенерировать объединенный сигнал целиком через каждые 1000 км, при этом нет необходимости в оптоэлектрических преобразованиях.

В примере на рис. 2.36 изображена система с постоянными длинами волн. Данные из входного кабеля 1 попадают на выходной кабель 3, а из кабеля 2 — в кабель 1 и т. д. Однако можно построить и коммутируемые WDM-системы. В таком устройстве выходные фильтры настраиваются с помощью интерферометров Фабри—Перо или Маха—Цандера. Эти устройства позволяют выбранным частотам быть измененными динамически управляющим компьютером. Эта способность обеспечивает большую гибкость, чтобы обеспечить много путей с различной длиной волны через телефонную сеть от фиксированного набора волокон. Чтобы узнать больше про оптические сети и спектральное уплотнение, читайте Ramaswami и др. (2009).

2.6.5. Коммутация

С точки зрения среднего телефонного инженера, телефонная система состоит из двух частей: внешнего оборудования (местных телефонных линий и магистралей, вне коммутаторов) и внутреннего оборудования (коммутаторов), расположенного на телефонной станции. Мы рассмотрели внешнее оборудование. Теперь пора уделить внимание внутреннему.

В телефонных системах используется два различных приема: коммутации каналов и коммутации пакетов. Традиционная телефонная система основана на коммутации каналов, но коммутация пакетов начинает распространяться с распространением технологии IP-телефонии. Мы несколько подробнее обсудим коммутацию каналов и ее отличие от коммутации пакетов. Оба вида коммутации достаточно важны, и мы еще вернемся к ним в дальнейшем на сетевом уровне.

Коммутация каналов

Когда вы (или ваш компьютер) снимаете телефонную трубку и набираете номер, коммутирующее оборудование телефонной системы отыскивает физический путь, состоящий из кабелей и ведущий от вашего телефона к телефону того, с кем вы связываетесь. Такая система, называемая **коммутацией каналов**, схематически изображена на рис. 2.37, *а*. Каждый из шести прямоугольников представляет собой коммутирующую станцию (оконечную или междугородную). В данном примере каждая станция имеет три входных и три выходных линии. Когда звонок проходит через коммутационную станцию, между входной и выходной линиями устанавливается физическое соединение (показано пунктирными линиями).

На заре телефонии соединение устанавливалось вручную телефонным оператором, который замыкал две линии проводом с двумя штекерами на концах. С изобретением автоматического коммутатора связана довольно забавная история. Автоматический коммутатор изобрел в XIX веке владелец похоронного бюро Алмон Б. Строуджер

(Almon B. Strowger) вскоре после изобретения телефона. Когда кто-либо умирал, родственник умершего звонил городскому телефонному оператору и говорил: «Соедините меня, пожалуйста, с похоронным бюро». К несчастью для мистера Строуджера, в его городе было два похоронных бюро, и жена владельца конкурирующей фирмы как раз работала телефонным оператором. Мистер Строуджер быстро понял, что либо он изобретет автоматический телефонный коммутатор, либо ему придется закрывать дело. Он выбрал первое. На протяжении почти 100 лет используемое во всем мире оборудование для коммутации каналов называлось **искателем Строуджера**. (История не упоминает, не устроилась ли жена его конкурента, уволенная с работы телефонного оператора, в телефонное справочное агентство сообщать телефонный номер своего похоронного бюро всем желающим.)

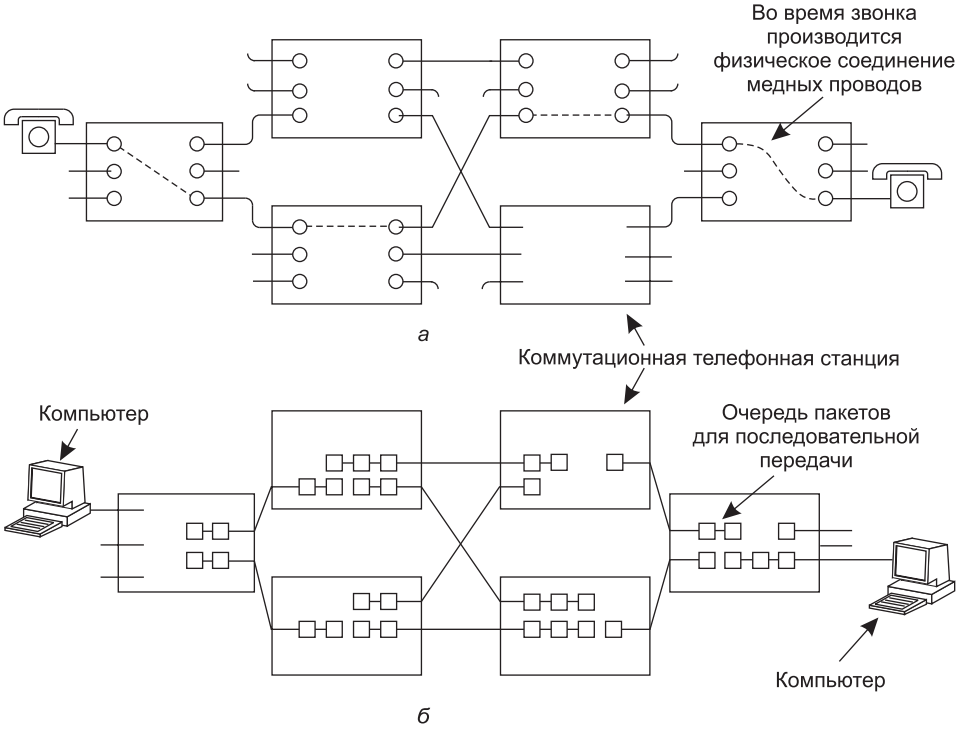


Рис. 2.37. Коммутация: *а* — каналов; *б* — пакетов

Модель, изображенная на рис. 2.37, *а*, конечно, сильно упрощена, поскольку канал, соединяющий двух абонентов телефонной линии, на самом деле может быть не только медным проводом, но и, например, микроволновой или оптоволоконной магистралью, на которой объединены тысячи телефонных абонентов. Тем не менее основная идея остается той же самой: когда один абонент звонит другому, устанавливается определенный путь, связывающий их, и этот путь остается неизменным до конца разговора.

Важным свойством коммутации каналов является необходимость установления сквозного пути от одного абонента до другого *до* того, как будут посланы данные.

Именно поэтому время от конца набора номера до начала разговора может занимать около 10 с и более для междугородных или международных звонков. В течение этого интервала времени телефонная система ищет путь, изображенный на рис. 2.38, *а*. Обратите внимание на то, что еще до начала передачи данных сигнал запроса на разговор должен пройти весь путь до пункта назначения и там быть распознан. Для многих компьютерных приложений (например, при проверке кредитной карточки клиента кассовым терминалом) длительное время установления связи является нежелательным.

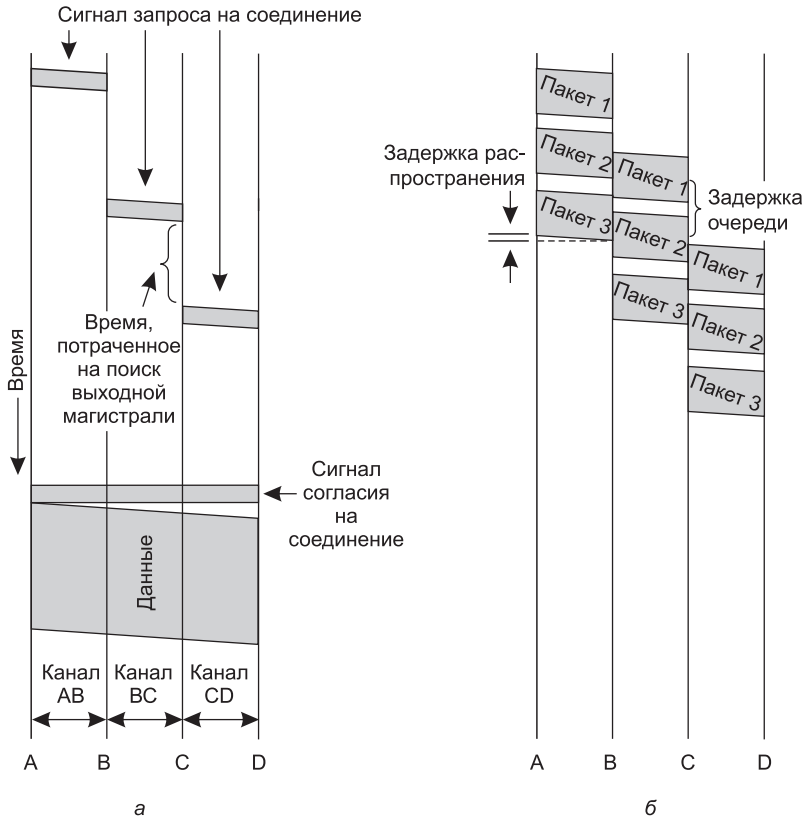


Рис. 2.38. Затраты времени при коммутации каналов (*а*) и коммутации пакетов (*б*)

В результате при установлении физического соединения между абонентами, как только этот путь установлен, единственной задержкой для распространения сигнала будет скорость распространения электромагнитного сигнала, то есть около 5 мс на каждые 1000 км. Еще одним свойством такой системы является то, что после начала разговора линия уже не может вдруг оказаться занятой, хотя она может быть занятой до установки соединения (например, благодаря отсутствию соответствующей возможности у коммутатора или магистрали). Также следствием установленного пути является отсутствие опасности скопления, то есть как только звонок был передан, вы никогда не получите сигналы «занято». Конечно, такой сигнал вы могли получить прежде, чем соединение было установлено из-за нехватки объема линии или коммутатора.

Коммутация пакетов

Альтернативным способом коммутации является **коммутация пакетов**, которая схематически изображена на рис. 2.37, *б* и описана в главе 1. При использовании такой формы коммутации отдельные пакеты пересылаются по мере готовности. В отличие от коммутации каналов, при коммутации пакетов нет необходимости устанавливать связь между двумя абонентами до начала передачи данных. Маршрутизаторы должны использовать передачу с промежуточной буферизацией, чтобы самостоятельно послать каждый пакет, продвигающийся к месту назначения. Эта процедура не похожа на коммутацию каналов, когда результат установки соединения — резервирование пропускной способности на всем пути от отправителя до приемника. Все данные в канале следуют этим путем. Среди других результатов следования по данному пути — гарантия прибытия данных в нужном порядке. При коммутации пакетов нет никакого неизменного пути, поэтому различные пакеты могут следовать различными путями, в зависимости от сетевых условий в тот момент, когда они были отправлены, и могут прибыть не по порядку.

Сети с пакетной коммутацией устанавливают низкую верхнюю границу размера пакетов. Это гарантирует, что ни один пользователь не может монополизировать линию передачи очень долго (например, много миллисекунд), так что сети с пакетной коммутацией могут обработать интерактивный трафик. Это также уменьшает задержку, так как первый пакет длинного сообщения может быть переправлен прежде, чем второй полностью прибыл. Однако задержка промежуточной буферизации пакета в памяти маршрутизатора, прежде чем он будет переслан к следующему маршрутизатору, превышает задержку при коммутации каналов. При коммутации каналов биты текут по проводу непрерывно.

Коммутация пакетов и каналов отличается и другими особенностями. Поскольку никакая пропускная способность при коммутации пакетов не зарезервирована, пакетам, вероятно, придется ждать, чтобы быть отправленными. Это вводит **задержку очереди** и скопление, если много пакетов послано в одно время. С другой стороны, нет никакой опасности застать сигнал «занято» и не иметь возможности использовать сеть. Таким образом, скопление происходит в разное время при коммутации каналов (во время установки) и пакетной коммутации (когда пакеты посланы).

Если канал был зарезервирован для отдельного пользователя и нет никакого трафика, его пропускная способность потрачена впустую. Она не может использоваться для другого трафика. Пакетная коммутация не тратит впустую пропускную способность и таким образом более эффективна с системной точки зрения. Понимание этого выбора крайне важно для понимания различия между коммутацией каналов и пакетной коммутацией. Выбор между гарантируемым сервисом и тратой ресурсов против не гарантированного сервиса и не траты ресурсов.

Пакетная коммутация более устойчива к сбоям. На самом деле, именно это свойство стало причиной изобретения данного метода. Если, например, выходит из строя один из коммутаторов, то все линии, подключенные к нему, также выходят из строя. Но при коммутации пакетов данные могут быть отправлены в обход «умершего» коммутатора.

Наконец, еще одним различием между двумя способами коммутации является политика оплаты услуг. Системы с коммутацией каналов традиционно взимают плату за

расстояние передачи и время на линии. В мобильных телефонах расстояние роли не играет (кроме международных звонков), а время играет не очень значительную роль (ну, например, тариф с 2000 бесплатных минут дороже, чем тариф с 1000 бесплатных минут, причем иногда звонки в ночное время и в выходные являются льготными). В случае коммутации пакетов время на линии вообще не принимается в расчет, однако иногда взимается плата за трафик. С обычных пользователей провайдеры иногда берут просто ежемесячную абонентскую плату, поскольку это проще для обеих сторон, однако магистральные транспортные службы взимают с местных провайдеров плату за объем трафика.

Все различия сведены в табл. 2.6.

Таблица 2.6. Сравнительные характеристики коммутации каналов и коммутации пакетов

Параметр	Коммутация каналов	Коммутация пакетов
Установка соединения	Требуется	Не требуется
Выделенный «медный» путь	Да	Нет
Каждый пакет перемещается по одному и тому же пути	Да	Нет
Пакеты приходят в правильном порядке	Да	Нет
Критичность выхода из строя коммутатора	Да	Нет
Доступная пропускная способность	Фиксированная	Динамическая
Возможность занятости линии	Во время установки соединения	Для каждого пакета
Возможность простоя линии	Да	Нет
Передача с промежуточным хранением	Нет	Да
Оплата	За время на линии	За трафик

Традиционно телефонные сети использовали схему с коммутацией каналов, чтобы обеспечить высококачественные телефонные звонки, а компьютерные сети использовали пакетную коммутацию для простоты и эффективности. Однако есть известные исключения. Некоторые более старые компьютерные сети имели внутреннюю схему коммутации каналов (например, X.25), а некоторые более новые телефонные сети используют пакетную коммутацию в технологии IP-телефонии. Для пользователя это выглядит точно так же, как стандартный телефонный звонок, но в сети происходит коммутация пакетов голосовых данных. Этот подход позволил возникнуть рынку дешевых международных звонков с помощью телефонных карточек, хотя, возможно, с более низким качеством звонка, чем у должностных лиц.

2.7. Мобильная телефонная система

Традиционная телефонная система (даже если она в один прекрасный день полностью перейдет на многогигабитный оптоволоконный кабель) никогда не сможет удовлетворить потребности огромной группы пользователей — людей, находящихся в пути.

Люди сейчас хотят звонить, а также использовать телефон для просмотра электронной почты и веб-серфинга буквально везде: в автомобиле, самолете, бассейне, во время пробежки в парке. Следовательно, имеется огромный интерес к беспроводной телефонии. В следующих параграфах мы рассмотрим подробности, касающиеся этой темы.

Мобильные телефоны используются в широкой области передачи голоса и данных.

На данный момент существует уже три разных поколения **мобильных (сотовых) телефонов.** Эти поколения называют **1G**, **2G** и **3G**.

1. Аналоговая голосовая связь.
2. Цифровая голосовая связь.
3. Цифровая голосовая связь и обмен данными (Интернет, электронная почта и т. д.).

(Мобильные телефоны не следует путать с беспроводными телефонами, состоящими из базовой станции и одной или нескольких переносных трубок. Они предназначены для использования внутри жилья или в непосредственной близости от него. Их никогда не объединяют в сети, поэтому в дальнейшем мы их рассматривать не будем.)

Хотя большая часть нашего обсуждения будет посвящена техническому устройству этих систем, нельзя не отметить тот интересный факт, что огромное влияние на процесс развития технологий этого типа оказали политические и экономические решения. Первая мобильная система была предложена американской компанией AT&T, которая, с согласия комиссии FCC, установила мобильную связь на всей территории Соединенных Штатов. В результате целая страна обрела единую (аналоговую) систему связи, и мобильный телефон, купленный, например, в Калифорнии, успешно работал в Нью-Йорке. А в Европе все получилось наоборот: когда туда пришла мобильная связь, каждая страна бросилась разрабатывать собственные системы, в результате чего проиграли все.

Однако Европа чему-то научилась на своих ошибках, и с появлением цифровых систем государственные телефонные службы объединились, чтобы создать единый стандарт (GSM), так что любой европейский мобильный телефон мог работать везде в Европе. К тому времени в США государство вышло из бизнеса, связанного со стандартизацией, поэтому новые цифровые мобильные системы стали заботой коммерческих структур. Это привело к тому, что разные производители стали выпускать разнотипные мобильные телефоны, и в США появились две основные абсолютно несовместимые цифровые мобильные телефонные системы и еще несколько небольших систем.

Несмотря на изначальное лидерство США, Европа сейчас обошла Штаты по популярности мобильной связи. Наличие единой системы, которая работает где угодно в Европе и с любым провайдером, является одной из причин, но есть и другая. Второе отличие США от Европы заключалось в скромном вопросе телефонных номеров. В США не различаются номера мобильных и стационарных телефонов. Поэтому нет никакой возможности узнать, набирая номер, например, (212) 234-5678, попадете вы на городской телефон (дешевый или вообще бесплатный звонок) или на сотовый (дорогой звонок). Чтобы люди не нервничали каждый раз, гадая, куда они звонят, телефонные компании заставили абонентов сотовой связи платить за входящие

звонки. Но многих такое решение отпугнуло — люди стали бояться потратить большую сумму денег на один только прием входящих звонков. В Европе у мобильных телефонов номер начинается с особого кода (обычно это число в районе 800–900), поэтому его сразу можно узнать. Значит, можно установить обычное правило, принятое в телефонии: платит звонящий (за исключением международных звонков, где платят оба).

Третий фактор, оказавший большое влияние на популярность мобильных систем, — это широкое распространение телефонов с предоплатой разговоров (до 75 % в некоторых регионах). Их можно купить во многих магазинах, и это не сложнее, чем купить цифровой фотоаппарат. Они могут быть заряжены, например, на 20 или 50 евро, а при снижении баланса до нуля их можно «перезарядить» с помощью секретного PIN-кода. Теперь такие мобильные телефоны есть у любого подростка, и родители могут быть на связи со своим чадом, не опасаясь при этом, что он самостоятельно наговорит по телефону на кругленькую сумму. Если мобильный телефон используется лишь эпизодически, то это обходится практически бесплатно, поскольку почти всегда можно найти тариф, на котором отсутствует абонентская плата или плата за входящие звонки.

2.7.1. Мобильные телефоны первого поколения: аналоговая передача речи

Однако хватит о политике и бизнесе. Поговорим о технологиях. Начнем наше рассмотрение с самых первых из них. Мобильные радиотелефоны эпизодически применялись в морском судоходстве и военной связи во время первых десятилетий XX века. В 1946 году в Сент-Луи была установлена первая система автомобильных телефонов. Она имела один большой передатчик, расположенный на крыше высокого здания, и единственный канал приема и передачи данных. Чтобы начать разговор, нужно было нажать на кнопку, которая включала передатчик и отключала приемник. Такие системы, известные как **тангентные**, существовали в некоторых городах в конце 50-х годов. СВ-радио, системы, используемые в такси и полицейских машинах, часто используют эту же технологию.

В 1960-х годах появилась **усовершенствованная система мобильной телефонной связи (IMTS, Improved Mobile Telephone System)**. Она также использовала мощный (200 Вт) передатчик, установленный на вершине горы, но уже имела два частотных канала: один для отправки, другой — для приема данных. Поэтому микрофонная кнопка уже была не нужна. Благодаря разделению входящих и исходящих каналов пользователи мобильных телефонов не могли слышать чужие разговоры (в отличие от тангентных систем, используемых в такси).

IMTS поддерживала 23 канала в диапазоне от 150 до 450 МГц. Из-за небольшого числа каналов пользователям часто подолгу приходилось ждать освобождения линии. К тому же из-за сильной мощности передатчика смежные системы должны были располагаться на расстоянии нескольких сотен километров друг от друга во избежание интерференции сигналов. В общем, из-за низкой емкости эта система была признана непрактичной.

Усовершенствованная мобильная телефонная связь (AMPS)

Все изменилось с появлением системы **усовершенствованной мобильной телефонной связи** (**AMPS**, **Advanced Mobile Phone System**), изобретенной компанией Bell Labs и впервые установленной в США в 1982 году. Она также использовалась в Англии, где называлась TACS, и в Японии — под именем MCS-L1. Несмотря на то что такая система формально перестала использоваться в 2008 году, мы все же ее рассмотрим, чтобы понять контекст систем 2G и 3G, которые ее улучшили.

В любой мобильной телефонной системе географический регион охвата делится на **соты** (отсюда иногда применяемое название — «сотовые телефоны»). В AMPS размер сот составляет обычно от 10 до 20 км; в цифровых системах соты еще мельче. Каждая сота работает на своих частотах, не пересекающихся с соседними. Лежащая в основе телефонной системы AMPS идея разбиения территории на относительно небольшие ячейки и использования одних и тех же частот в различных (но не соседних) ячейках дает этой системе значительно большие возможности по сравнению с более ранними системами. В то время как в системе IMTS на территории диаметром 100 км для каждого звонка требовалась своя частота, система AMTS в той же области могла состоять из ста десятикилометровых сот и поддерживать от 5 до 10 звонков на одной и той же частоте в сильно удаленных друг от друга ячейках. Кроме того, небольшие размеры сот означают меньшую мощность, требующуюся для передатчиков, а значит, и меньшую стоимость устройств.

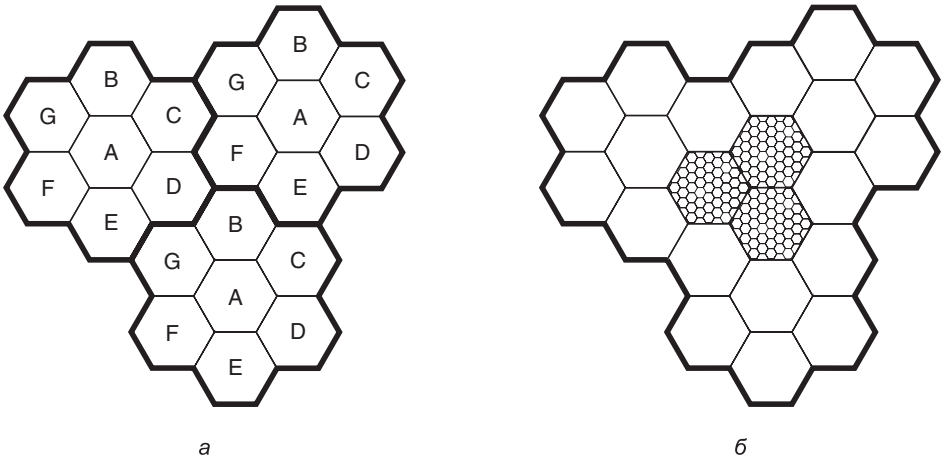


Рис. 2.39. В соседних сотах используются различные частоты (*а*); чтобы увеличить количество пользователей, можно использовать соты меньшего размера (*б*)

Идея повторного использования частоты проиллюстрирована на рис. 2.39, *а*. Соты имеют форму, близкую к окружности, однако на модели их легче представить в виде шестиугольников. На 2.39, *а* все соты одного размера. Они объединены в группы по семь сот. Каждая буква соответствует определенному набору частот. Обратите внимание на то, что между ячейками с одинаковыми наборами частот располагается буфер примерно в две ячейки шириной, в котором данные частоты не используются, — это

обеспечивает хорошее разделение сигналов одинаковых частот и низкий уровень помех.

Главная задача заключается в том, чтобы найти подходящие возвышенности для размещения антенн базовых станций. Для решения этой проблемы многие операторы связи заключили договоры с Римско-католической церковью, поскольку последней принадлежит существенное количество высоких строений в различных странах. Удобно и то, что все они находятся под единым управлением.

Если в каком-нибудь регионе количество пользователей вырастает настолько, что система переполняется, то мощность передатчиков уменьшается, а переполненные соты разбиваются на соты меньшего размера (**микросоты**), как показано на рис. 2.39, *б*. Телефонные компании иногда создают временные микросоты, используя переносные вышки со спутниковой связью во время больших спортивных соревнований, концертов и в других местах, где большое количество пользователей сотовой связи оказывается сконцентрировано в течение нескольких часов.

В центре каждой ячейки располагается базовая станция (БС), с которой связываются все телефоны, находящиеся в ее зоне действия. Базовая станция состоит из компьютера и приемника/передатчика, соединенного с антенной. В небольших системах все базовые станции соединены с одним устройством, называемым **MTSO** (**Mobile Telephone Switching Office** — **коммутатор мобильных телефонов**) или **MSC** (**Mobile Switching Center** — **мобильный коммутационный центр**). Большой системе может потребоваться несколько коммутаторов, которые соединяются с коммутатором второго уровня и т. д. Коммутаторы мобильных телефонов являются аналогами оконечных телефонных станций и в самом деле соединяются хотя бы с одним оконечным коммутатором обычной телефонной системы. Коммутаторы мобильных телефонов связываются с базовыми станциями, друг с другом и с коммутируемой телефонной сетью общего пользования, используя коммутацию пакетов.

В каждый момент времени мобильный телефон логически находится в зоне действия одной ячейки и управляется базовой станцией этой ячейки. Когда телефон физически покидает ячейку, его базовая станция замечает ослабление сигнала и опрашивает все окружающие станции, насколько хорошо они слышат сигнал этого телефона. Затем базовая станция передает управление данным телефоном ячейке, получающей от него наиболее сильный сигнал, таким образом определяя ячейку, в которую переместился мобильный телефон. После этого телефон информируется о переходе в ведение новой БС, и если в этот момент ведется разговор, телефону будет предложено переключиться на новый канал (поскольку в соседних сотах одинаковые частотные каналы не используются). Подобный процесс называется **передачей** (**handoff**) и занимает около 300 мс. Назначение канала осуществляет коммутатор мобильных телефонов, являющийся центральным нервом системы. Базовые станции представляют собой всего лишь радиоретрансляторы.

Каналы

Система AMPS использует для разделения каналов частотное уплотнение (FDM). Она использует 832 дуплексных канала, каждый из которых состоит из пары симплексных каналов. Такую структуру называют **FDD** (**Frequency Division Duplex** — **дуплексный**

режим с разделением частоты). 832 симплексных канала передачи располагаются в диапазоне от 824 до 849 МГц, и еще 832 симплексных канала приема — от 869 до 894 МГц. Ширина каждого канала составляет 30 кГц.

Все 832 канала можно разделить на четыре категории.

1. Управляющие каналы (от базовой станции к мобильному телефону) для управления системой.
2. Пейджинговые каналы (от базовой станции к мобильному телефону) для передачи сообщений мобильным пользователям.
3. Каналы доступа (двунаправленные) для установления соединения и назначения каналов.
4. Каналы данных (двунаправленные) для передачи голоса, факса или данных.

Для управления резервируется 21 канал. Поскольку одни и те же частоты не могут использоваться в соседних сотах, то число голосовых каналов, доступных в пределах одной ячейки, значительно меньше 832 — обычно около 45.

Управление вызовом

Каждый мобильный телефон в системе AMPS снабжается 32-разрядным порядковым номером и 10-значным телефонным номером, которые записываются в ППЗУ телефона. Телефонный номер состоит из 3-значного кода области, занимающего 10 бит, и 7-значного номера абонента, занимающего 24 бит. При включении телефон сканирует запрограммированный список из 21 управляющего канала, в котором он ищет наиболее сильный сигнал. Затем телефон передает в эфир свой 32-разрядный порядковый номер и 34-разрядный телефонный номер. Как и вся управляющая информация в системе AMPS, этот пакет посылается в цифровой форме, несколько раз, с применением помехоустойчивого кодирования, хотя сами голосовые каналы являются аналоговыми.

Когда базовая станция слышит этот сигнал, она передает сообщение коммутатору мобильного центра, который фиксирует появление нового пользователя, а также информирует «домашний» коммутатор абонента о его новом местоположении. Обычно мобильный телефон регистрируется примерно каждые 15 минут.

Чтобы позвонить с мобильного телефона, его владелец включает телефон, вводит номер и нажимает клавишу SEND. При этом телефон посылает набранный телефонный номер вместе со своими идентификаторами по каналу доступа. Если при этом происходит коллизия, то телефон повторяет попытку позже. Когда базовая станция получает запрос, она информирует об этом коммутатор. Если звонящий является клиентом оператора связи, которому принадлежит данный коммутатор (или одного из ее партнеров), тогда коммутатор ищет для него свободный канал. Если такой канал находится, то номер этого канала посылается обратно по управляющему каналу. Затем мобильный телефон автоматически переключается на выбранный голосовой канал и ждет, пока тот, кому звонят, ответит.

Входящие звонки обрабатываются иначе. Находящиеся в режиме ожидания телефоны постоянно прослушивают пейджинговый канал, ожидая адресованных им сообщений. Когда поступает звонок на мобильный телефон (с обычного или другого

мобильного телефона), то пакет посылается на «домашний» коммутатор вызываемого, которому должно быть известно текущее местонахождение абонента. Этот пакет пересылается на базовую станцию в его текущей ячейке, которая посылает по пейджинговому каналу сообщение типа: «Элемент 14, вы здесь?» При этом телефон, которому звонят, по управляющему каналу отвечает: «Да». Тогда базовая станция ему сообщает: «Элемент 14, вам звонок по каналу 3». После этого сотовый телефон переключается на канал 3 и начинает издавать звуковые сигналы (или проигрывать мелодию, которую владельцу подарили на день рождения).

2.7.2. Второе поколение мобильных телефонов: цифровая передача голоса (G2)

Первое поколение сотовых телефонных систем было аналоговым. Второе поколение является цифровым. Оно обеспечивало увеличение пропускной способности, позволяя голосовым сигналам быть оцифрованными и сжатыми. Сжатие и шифровка голоса и управляющих сигналов улучшает безопасность. Это, в свою очередь, защищает от мошенничества и подслушивания, как от намеренного поиска, так и эха других звонков из-за распространения радиоволн. Наконец, это позволяет предоставлять новые сервисы, такие как обмен текстовыми сообщениями.

Как не было никаких четких стандартов в первом поколении мобильных телефонов, так они и не появились ко второму поколению. Было разработано несколько различных систем, три из которых были широко распространены.

D-AMPS (Digital Advanced Mobile Phone System — цифровая усовершенствованная система мобильного телефона) является цифровой версией AMPS, которая сосуществует с AMPS и использует уплотнение с разделением времени, чтобы поместить несколько вызовов в один и тот же частотный канал. Она описана в Международном стандарте IS-54 и следующем за ним IS-136. **GSM (Global System for Mobile communications — глобальная система для мобильной связи,)** стала доминирующей системой, и хотя она медленно завоевывает популярность в США, она теперь используется фактически всюду в мире. Как и D-AMPS, GSM основан на соединении частотного и временного уплотнения. **CDMA (Code Division Multiple Access — множественный доступ с кодовым разделением каналов)**, описанный в международном стандарте IS-95, является абсолютно другим видом системы и не основан ни на частотном, ни на временно́м уплотнении. Хотя CDMA не является доминирующей системой второго поколения, эта технология стала основанием для систем третьего поколения.

Название **PCS (Personal Communications Services — Персональная служба связи)** иногда используется в литературе по маркетингу и означает систему второго поколения (цифровую, разумеется). Изначально так назывался телефон, работающий в диапазоне 1900 МГц, впрочем, сейчас различия почти стерлись.

Далее мы опишем GSM, так как это наиболее распространенная система 2G. В следующем разделе мы более подробно рассмотрим CDMA, когда будем говорить о 3G.

GSM — Глобальная система мобильной связи

GSM появилась в 1980-х годах как попытка создать единственный европейский стандарт второго поколения. Задача была возложена на телекоммуникационную компанию, название которой на французском языке Groupe Specialé Mobile. Первые системы GSM были развернуты начиная с 1991 года и имели быстрый успех. Скоро стало ясно, что GSM будет пользоваться огромным успехом, простираясь даже до Австралии, поэтому он был переименован, чтобы иметь больше мирового обращения.

GSM и другие системы мобильной связи, которые мы изучим, сохраняют от систем первого поколения конструкцию, основанную на сотах, повторном использовании частоты в сотах и мобильности передач при перемещении пользователей. Отличаются только детали. Ниже мы рассмотрим лишь основные свойства GSM. Печатный вариант стандарта GSM занимает свыше 5000 (sic!) страниц. Основная часть текста относится к описанию инженерных аспектов системы, в частности устройства приемников, обрабатывающих многолучевое распространение сигналов, синхронизации приемников и передатчиков. Ни о том, ни о другом мы не сможем рассказать в этой книге.

Рисунок 2.40 показывает, что архитектура GSM подобна архитектуре AMPS, хотя у компонентов другие названия. Мобильный телефон теперь разделен на телефонную трубку и сменный чип с информацией об учетной записи и абоненте, названной **Симкартой** (**SIM card**, что является сокращением для **Subscriber Identity Module** — **модуль идентичности абонента**). Сим-карта активирует телефонную трубку и содержит секретную информацию, которая позволяют мобильному телефону и сети идентифицировать друг друга и шифровать переговоры. Сим-карта может быть удалена и включена в другой телефонной трубке, чтобы превратить эту телефонную трубку, с точки зрения сети, в ваш мобильный телефон.

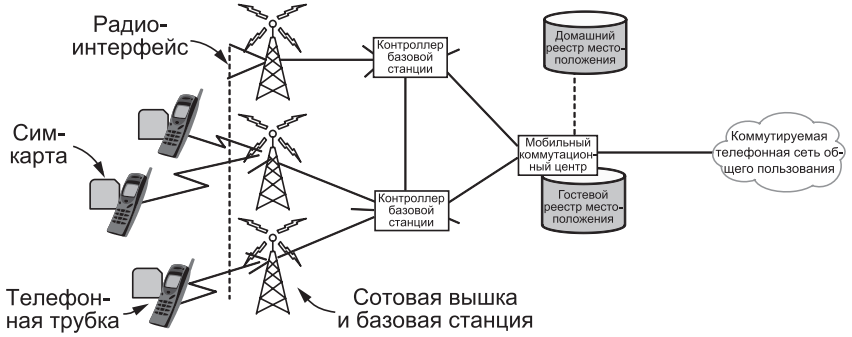


Рис. 2.40. Архитектура мобильной сети GSM

Мобильные телефоны связываются с сотовыми базовыми станциям по **радиоинтерфейсу**, который мы опишем далее. Каждая базовая станция соты соединена с **BSC** (**Base Station Controller** — **Контроллер базовой станции**), который управляет радиоресурсами сот и обрабатывает передачу. Контроллер базовой станции, в свою очередь, соединен с мобильным коммутационным центром (как в AMPS), который направляет звонки и соединяется с **PSTN** (**Public Switched Telephone Network** — **коммутируемая телефонная сеть общего пользования**).

Чтобы направлять звонки, мобильный коммутационный центр должен знать, где мобильные телефоны в настоящее время находятся. Он поддерживает базу данных находящихся вблизи него мобильных телефонов, связанных с сотами, которыми управляет центр. Эту базу данных называют **VLR (Visitor Location Register — гостевой реестр местоположения)**. Есть также база данных в мобильной сети, которая дает последнее известное местоположение каждого мобильного телефона. Она называется **HLR (Home Location Register — домашний реестр местоположения)**. Эта база данных используется, чтобы направить входящие вызовы к правильным местоположениям. Обе базы данных должны постоянно обновляться, поскольку мобильные телефоны перемещаются из соты в соту.

Теперь мы опишем радиоинтерфейс подробнее. GSM работает в одном диапазоне частот во всем мире, включая 900, 1800 и 1900 МГц. Больший чем в AMPS диапазон частот выделен для того, чтобы поддержать огромное количество пользователей. GSM, как и AMPS, — сотовая система с дуплексом путем разделения частот. Таким образом, каждый мобильный телефон передает на одной частоте и получает на другой, более высокой частоте (на 55 МГц выше для GSM, на 80 МГц выше для AMPS). Однако в отличие от AMPS в GSM отдельная пара частот разделена мультиплексированием с разделением времени на временные слоты. Таким образом она совместно используется несколькими мобильными телефонами.

Чтобы управлять несколькими мобильными телефонами, каналы GSM значительно шире AMPS (200 кГц против 30 кГц). Каждая полоса частот имеет ширину 200 кГц, как показано на рис. 2.41. Система GSM в диапазоне 900 МГц имеет 124 пары симплексных каналов. Полоса пропускания каждого симплексного канала составляет 200 кГц. Канал поддерживает 8 отдельных соединений при помощи временного уплотнения. Каждой активной в данный момент базовой станции назначен один кадровый интервал на пару каналов. Теоретически каждая сота может иметь до 992 каналов, однако многие из них сознательно делают недоступными во избежание конфликтов с соседними сотами. На рис. 2.41 восемь заштрихованных кадровых интервалов принадлежат одному и тому же соединению, по четыре в каждом направлении. Прием и передача происходит в разных интервалах, поскольку аппаратура GSM не может работать одновременно в двух режимах и на перестройку требуется некоторое время. Если мобильной станции присвоен диапазон 890,4/935,4 МГц и кадровый интервал 2 хочет осуществить передачу на базовую станцию, он воспользуется нижним набором заштрихованных интервалов (а также последующими), размещая в каждом из них порцию данных. Так будет продолжаться до тех пор, пока не будут посланы все данные.

Интервалы TDM, изображенные на рис. 2.41, являются частью сложной иерархической системы кадров. Каждый интервал имеет специфическую структуру, как и их группы. Упрощенная иерархия изображена на рис. 2.42. Мы видим здесь, что интервал TDM состоит из 148-битного кадра данных, который занимает канал на 577 мкс (включая защитный интервал длиной 30 мкс). Кадры данных начинаются и заканчивается тремя нулями, это делается для их разграничения. В них также входят 57-битные информационные (*Information*) поля, в каждом из которых присутствует контрольный бит проверки содержимого (голос/данные). Между информационными полями имеется 26-битное поле синхронизации (*Sync*), которое используется приемником для синхронизации с границей кадра передатчика.

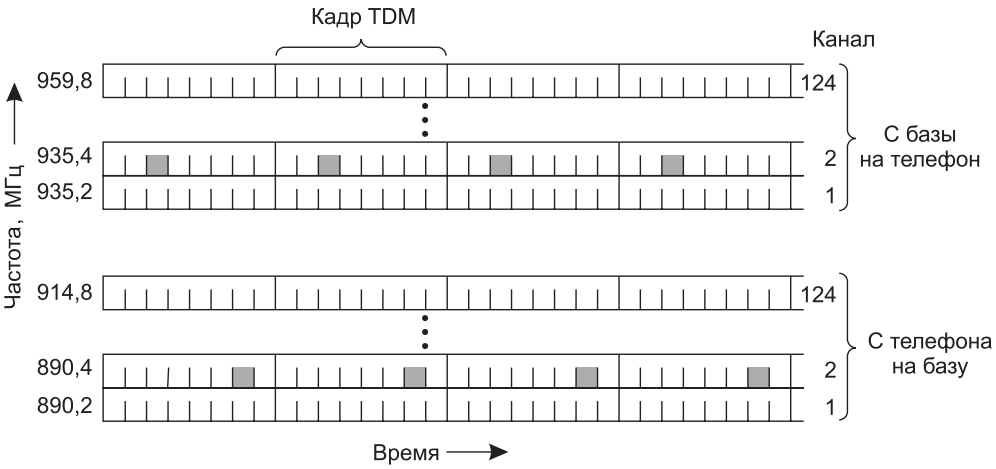


Рис. 2.41. GSM имеет 124 частотных канала, в каждом из них 8-интервальная система с разделением времени

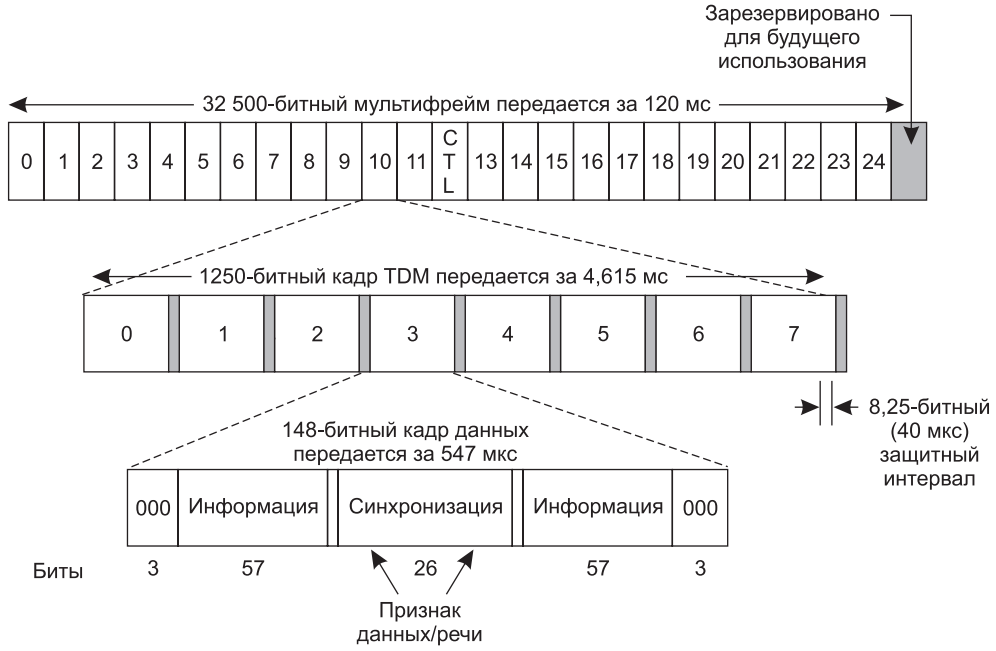


Рис. 2.42. Часть иерархической структуры кадров GSM

Кадр данных передается за 547 мкс, но передатчику разрешается посылать данные только через каждые 4,615 мс, поскольку он делит канал с семью другими станциями. Общая скорость каждого канала составляет 270 883 бит/с. Она делится между 8 пользователями. Тем не менее, как и в AMPS, на накладные расходы тратится

большая часть пропускной способности, и в итоге на одного пользователя приходится 24,7 Кбит/с (перед началом исправления ошибок). После исправления ошибок остается 13 Кбит/с, с помощью которых нужно передать голос. Хотя это существенно меньше, чем 64 Кбит/с импульсно-кодовой модуляции для несжатых голосовых сигналов в неподвижной телефонной сети, сжатие на мобильном устройстве может достигнуть этого уровня с небольшой потерей качества.

Как видно на рис. 2.42, 8 кадров данных образуют один кадр TDM, а 26 кадров TDM образуют 120-миллисекундный мультикадр (мультифрейм). В мультифрейме двенадцатый интервал используется для служебных целей, а двадцать пятый зарезервирован для будущего использования, поэтому для пользовательского трафика остается только 24 интервала.

Тем не менее в дополнение к 26-интервальному мультифрейму, показанному на рис. 2.42, используется еще и 51-интервальный мультифрейм (не показан на рисунке). Некоторые интервалы нужны для управляющих каналов. **Широковещательный управляющий канал** представляет собой непрерывный поток, исходящий от базовой станции, в котором содержится ее идентификационная информация и статус канала. Все мобильные устройства производят мониторинг мощности сигнала, по которому они определяют моменты перехода в ведение новой соты.

Выделенный управляющий канал используется для поиска мобильного телефона, обновления информации о нем, регистрации и установки соединения. В частности, каждая БС содержит базу данных телефонов, находящихся в текущий момент под ее управлением. Информация, необходимая для обновления этой базы, передается по выделенному управляющему каналу.

Наконец, есть еще **общий управляющий канал**, разделяемый на три логических подканала. Первый из них — **пейджинговый канал**, с помощью которого базовая станция сообщает о входящих звонках. Каждый мобильный телефон постоянно прослушивает его в ожидании звонка, на который он должен ответить. Второй — **канал случайного доступа**, позволяющий пользователям запросить интервал в выделенном управляющем канале. Если два запроса сталкиваются (коллизия), они искажаются и им приходится впоследствии осуществлять повторные попытки. Используя выделенный управляющий канал, мобильный телефон может инициировать исходящий звонок. Присвоенный интервал объявляется при помощи третьего подканала — **канала предоставления доступа**.

Наконец, GSM отличается от AMPS тем, как обрабатывается передача. В AMPS MSC управляет этим полностью без помощи от мобильных устройств. С временными слотами в GSM большую часть времени мобильный телефон ни посылает, ни получает. Неактивные слоты — возможность для мобильного телефона измерить качество сигнала от других соседних базовых станций. Он производит эти измерения и посылает эту информацию в BSC. BSC может использовать ее, чтобы определить, когда мобильный телефон покидает одну ячейку и входит в другую так, что может выполнить передачу. Эта схема называется **MAHO (Mobile Assisted HandOff)**.

2.7.3. Мобильные телефоны третьего поколения: цифровая речь и данные

Первое поколение мобильных телефонов было голосовым аналоговым, второе было голосовым цифровым. Третье поколение, которое называют **3G**, представляет цифровую передачу речи *и* данных.

Развитием этой отрасли движет большое количество факторов. Во-первых, объем передаваемых данных уже превышает объем передаваемой речи в стационарных сетях, и первый показатель растет экспоненциально, тогда как последний растет довольно вяло. Многие эксперты предрекают такое же будущее и мобильным сетям: трафик данных превысит голосовой трафик. Во-вторых, компьютерная индустрия и индустрии телефонии и развлечений уже стали полностью цифровыми и быстро объединяются. Многие восхищаются компактностью и малым весом портативного устройства, которое выступает в качестве телефона, проигрывателя компакт-дисков, DVD-проигрывателя, терминала для электронной почты, обладает веб-интерфейсом, возможностями текстового редактора, включает в себя электронные игры и многое другое, и все это с международной беспроводной высокоскоростной связью с Интернетом.

iPhone фирмы Apple — хороший пример 3G-устройства. С ним люди подключаются к беспроводным информационным службам, и объемы беспроводных данных компании AT&T резко растут с популярностью iPhone. Проблема в том, что iPhone использует сеть 2,5G — улучшенная 2G сеть, но не настоящая сеть 3G, и имеет недостаточную информационную емкость, чтобы пользователи были счастливы. Мобильная телефония 3G обеспечит достаточное количество беспроводной пропускной способности, чтобы сделать будущих пользователей счастливыми.

Еще в 1992 году международный союз телекоммуникаций, ITU, сделал попытку конкретизировать и реализовать эти мечты и выпустил проект под названием **IMT-2000**, где IMT означало «**Международная мобильная связь**» (**International Mobile Telecommunications**).

Вот основные сервисы, для предоставления которых задумывалась сеть IMT-2000.

1. Высококачественная передача речи.
2. Обмен сообщениями (замена e-mail, факса, SMS, чата и т. д.).
3. Мультимедиа (проигрывание музыки, видео, фильмов, телевидения и т. д.).
4. Доступ в Интернет (включая просмотр страниц с аудио- и видеоинформацией).

В качестве дополнительных услуг могут быть видеоконференции, телепрезентации, групповые электронные игры, мобильная коммерция (использование мобильного телефона для оплаты покупок). Более того, все эти сервисы должны быть доступны по всему миру (с автоматическим соединением через спутник в тех местах, где стационарная сеть отсутствует), на основе постоянного подключения и с гарантированным качеством обслуживания.

ITU задумывал IMT-2000 как единую технологию, чтобы производители могли выпустить универсальное устройство, которое можно было бы продавать по всему миру (как компьютеры и проигрыватели компакт-дисков и не в пример мобильным телефонам и телевизорам). Одна стандартная технология сильно упрощает жизнь операторам связи и привлекает клиентов. Война форматов (так получилось с Betamax и VHS в мире видеозаписи) неблагоприятно сказывается на бизнесе.

Оказалось, что это было слишком оптимистично. Номер 2000 обозначал три вещи: (1) год, когда это, как предполагалось, будет работать, (2) частота, на которой это, как предполагалось, будет работать (в МГц), и (3) пропускная способность, которую сервис должен иметь (в Кбит/с). Это не было достигнуто ни по одному из трех пунктов. К 2000 году ничего не было осуществлено.

ITU рекомендовал правительствам всех стран зарезервировать частоту 2 ГГц для международного роуминга. Рекомендации последовал только Китай. Наконец, в какой-то момент осознали, что невозможно выделить каждому пользователю пропускную способность в 2 Мбит/с, особенно учитывая *повышенную* мобильность многих из них (просто нереально с достаточно высокой скоростью осуществлять передачу с одной базовой станции на другую). Более реалистично выглядит выделение 2 Мбит/с стационарным абонентам, которые сидят дома (в этом случае такая система будет серьезным конкурентом ADSL), 384 Кбит/с для людей, которые не спеша прогуливаются по парку, и 144 Кбит/с для связи с абонентами, движущимися в автомобилях.

Несмотря на эти начальные неудачи, с тех пор было многое достигнуто. Было выдвинуто несколько технических предложений, впоследствии некоторые отсеялись и остались две основные технологии. Первая из них называется **WCDMA — широкополосный CDMA (Wideband CDMA)**, была предложена фирмой Ericsson и продвинута Европейским союзом, который назвал ее **UMTS (Universal Mobile Telecommunications System — универсальная система мобильной связи)**. Вторым претендентом стала система **CDMA2000**, предложенная Qualcomm.

У этих систем больше сходств, чем различий. Базовый принцип обеих систем — это CDMA. WCDMA использует полосу пропускания 5 МГц, а CDMA2000 — 1,25 МГц. Если бы инженеров из Ericsson и Qualcomm посадили за стол переговоров и поставили задачу выработать единую систему, они, наверное, справились бы с этим довольно быстро. Беда в том, что настоящей проблемой, как всегда, является отнюдь не инженерное решение, а политика. Европе требовалась система, умеющая работать с GSM; Соединенным Штатам нужна была система, совместимая с одной из уже существующих там систем (IS-95). Каждая сторона поддерживала свою компанию (Ericsson находится в Швеции, Qualcomm — в Калифорнии). В конце концов, обе компании оказались вовлечены во множественные тяжбы, связанные с патентами на технологию CDMA.

Во всем мире 10—15 % абонентов мобильной связи уже используют технологии 3G. В Северной Америке и Европе, приблизительно одна треть мобильных абонентов — 3G. Япония была ранним последователем, и теперь почти все мобильные телефоны в Японии — 3G. Эти числа включают и UMTS, и CDMA2000, и 3G продолжает быть одним большим котлом деятельности, поскольку рынок трясет. Чтобы добавить беспорядка, UMTS стал единственным стандартом 3G с разнообразными несовместимыми опциями, включая CDMA2000. Это изменение было попыткой объединить конфликтующие стороны, но оно только сглаживает технические различия. Мы будем говорить UMTS, подразумевая WCDMA, а не CDMA2000.

Мы ограничим наше обсуждение использованием CDMA в сотовых связях, поскольку это — отличительный признак обеих систем. В CDMA не происходит ни временно́го, ни частотного разделения каналов, но осуществляется соединение, при котором каждый пользователь работает на том же диапазоне частот в то же самое время. Когда это было предложено для сотовых систем впервые, промышленность

отреагировала приблизительно так же, как королева Изабелла на предложение Колумба достигнуть Индии, направляясь на запад. Однако благодаря упорству компании Qualcomm CDMA преуспел как 2G система (IS-95) и окреп до такой степени, что стал техническим основанием для 3G.

Заставить CDMA работать в мобильном телефоне требует большего, чем основной метод CDMA, который мы описали в предыдущем разделе. А именно, мы описали синхронный CDMA, в котором элементарные последовательности являются строго ортогональными. Эта схема работает, когда все пользователи синхронизированы на времени начала элементарных последовательностей, как в случае передачи от базовой станции к мобильным телефонам. Базовая станция может передать элементарные последовательности, начинающиеся в одно время, так чтобы сигналы были ортогональными и могли быть отделены. Однако трудно синхронизировать передачи независимых мобильных телефонов. Без специальных усилий их передачи достигли бы базовой станции в разное время, без гарантии ортогональности. Чтобы мобильные телефоны могли передавать сигналы базовой станции без синхронизации, нужны кодовые последовательности, которые будут ортогональны при всех возможных смещениях.

Хотя и невозможно найти последовательности, которые являются ортогональными для этого общего случая, длинные псевдослучайные последовательности достаточно близки к этому. У них есть характерная особенность — низкая **взаимная корреляция** друг с другом при любых смещениях. Это означает, что когда одна последовательность умножена на другую последовательность, в итоге внутреннее произведение будет маленьким; это был бы ноль, если бы они были ортогональными. (Интуитивно, случайные последовательности должны всегда отличаться друг от друга. Их перемножение должно давать случайный сигнал низкого уровня.) Это позволяет приемнику фильтровать нежелательные помехи из полученного сигнала. Кроме того, **автокорреляция** псевдослучайных последовательностей также является малой, кроме случая нулевого смещения. Это означает, что, когда одна последовательность умножена на свою отсроченную копию и просуммирована, результат будет маленьким, кроме тех случаев, когда задержка — ноль. (Интуитивно, отсроченная случайная последовательность похожа на другую случайную последовательность, и мы вернулись к случаю взаимной корреляции.) Это позволяет приемнику обнаруживать начало требуемой передачи в полученном сигнале.

Использование псевдослучайных последовательностей позволяет базовой станции получать сообщения CDMA от несинхронизированных мобильных телефонов. Однако неявное предположение в нашем обсуждении CDMA — то, что уровни мощности всех мобильных телефонов в приемнике одинаковы. В противном случае маленькая взаимная корреляция с сильным сигналом может превзойти большую автокорреляцию со слабым сигналом. Таким образом, уровнем сигнала передачи мобильных телефонов нужно управлять, чтобы минимизировать интерференцию между конкурирующими сигналами. Именно это вмешательство ограничивает пропускную способность систем CDMA.

Уровни мощности, полученные на базовой станции, зависят от того, как далеко находятся передатчики, а также с какой мощностью они передают. Может быть много мобильных станций на различных расстояниях от базовой станции. Хороший эвристический алгоритм, позволяющий уравнять полученную мощность, — использовать

инверсию уровня мощности сигнала базовой станции. Другими словами, мобильная станция, получающая слабый сигнал от базовой станции, будет использовать большую мощность, чем получающая мощный сигнал. Для большей точности базовая станция также дает каждой мобильной станции обратную связь, чтобы увеличить, уменьшить или считать устойчивым передаваемый уровень сигнала. Обратная связь передается часто (1500 раз в секунду), потому что хорошее управление важно, чтобы минимизировать интерференцию.

Другое усовершенствование базовой схемы CDMA, которую мы описали ранее, должно позволить различным пользователям посылать данные на различных скоростях. Эта уловка достигнута естественно в CDMA, фиксируя скорость, на которой передаются чипы и назначая пользовательские последовательности чипов различной длины. Например, в WCDMA скорость чипов — 3,84 Мчипов/с, а кодовые последовательности содержат от 4 до 256 чипов. При использовании кода с 256 чипами после исправления ошибок остается приблизительно 12 Кбит/с, и эта скорость достаточна для голосового вызова. Для кода с 4 чипами пользовательская скорость передачи данных близка к 1 Мбит/с. Коды промежуточной длины дают промежуточные уровни скорости; чтобы достигнуть больших мегабит в секунду, мобильный телефон должен использовать более одного канала шириной 5 МГц.

Теперь давайте опишем преимущества CDMA, учитывая, что мы имели дело с проблемами заставляя его работать. У CDMA есть три основных преимущества. Во-первых, CDMA может улучшить пропускную способность, используя в своих интересах маленькие периоды, когда некоторые передатчики молчат. В вежливых голосовых вызовах одна сторона молчит в то время, когда другая говорит. В среднем занято только 40 % времени. Однако паузы могут быть маленькими и трудно предсказуемыми. При использовании временного или частотного уплотнения невозможно повторно назначить время или каналы частоты достаточно быстро, чтобы извлечь выгоду из этих маленьких пауз. Однако в CDMA просто отсутствие передачи от одного пользователя понижает влияние на других пользователей, и вероятно, что некоторая часть пользователей не будет передавать в занятой соте в данный момент времени. Таким образом CDMA использует в своих интересах ожидаемые паузы, чтобы произвести большее число одновременных звонков.

Во-вторых, с CDMA каждая сота использует одни и те же частоты. В отличие от GSM и AMPS, частотное уплотнение для отделения передач разных пользователей не требуется. Это устраняет сложные задачи планирования частоты и улучшает пропускную способность. Это также облегчает для базовой станции использование разнонаправленных антенн, или **секторные антенны**, вместо всенаправленной антенны. Направленные антенны концентрируют сигнал в намеченном направлении и уменьшают сигнал, и следовательно, интерференцию в других направлениях. Это, в свою очередь, увеличивает пропускную способность. Распространены три способа разделения на секторы. Базовая станция должна отследить мобильный телефон, когда он перемещается из сектора в сектор. В CDMA это легко, потому что все частоты используются во всех секторах.

В-третьих, CDMA облегчает **мягкую передачу** (**soft handoff**), при которой мобильный телефон обнаруживается новой базовой станцией раньше, чем отключится предыдущая. Таким образом, нет никакой потери непрерывности. Мягкая пере-

дача показана на рис. 2.43. Это легко в CDMA, потому что все частоты используются в каждой соте. Альтернатива — **жесткая передача (hard handoff)**, при которой предыдущая базовая станция обрывает вызов до того, как его возьмет новая. Если новая станция неспособна принять вызов (например, потому что нет доступной частоты), вызов обрывается. Пользователи замечают это, но недостаток связан с текущей конструкцией. Жесткая передача — норма при использовании частотного уплотнения, чтобы избежать затрат мобильной передачи или приема на двух частотах одновременно.

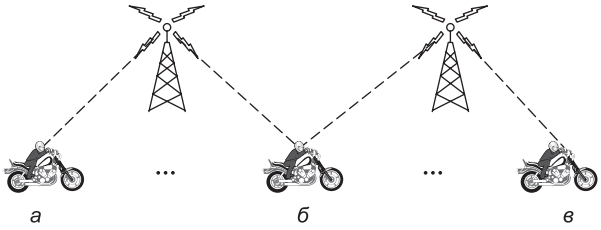


Рис. 2.43. Мягкая передача: *а* — перед; *б* — во время и *в* — после

О системах 3G написано много, причем отзывы в основном восторженные. Большинство пишет о третьем поколении мобильной связи в том духе, что это самое большое достижение со времен изобретения хлеборезки. Тем временем некоторые операторы связи уже делают первые осторожные шаги в направлении 3G, предлагая, что называется, **2,5G**, хотя более точно было бы назвать это 2,1G. Одна такая система называется **EDGE (Enhaced Data rates for GSM Evolution — повышенные скорости передачи для развития GSM)** и представляет собой обычный GSM с увеличенным числом бит на символ. Проблема состоит в том, что чем больше бит на символ используется, тем больше вероятность ошибок. Поэтому в EDGE применяются девять различных схем модуляции и коррекции ошибок. Отличаются они друг от друга процентом пропускной способности, выделяемым на исправление ошибок, возникающих вследствие повышенной скорости.

Технология EDGE — это один шаг вдоль эволюционного пути, который разделяет GSM от WCDMA. Точно так же есть эволюционный путь, определенный для операторов, чтобы обновить сети от IS-95 к CDMA2000.

Даже при том, что сети 3G еще не полностью развернуты, некоторые исследователи расценивают 3G как решенное дело. Эти люди уже работают над системами четвертого поколения под названием **LTE (Long Term Evolution)**. Некоторые из предложенных особенностей 4G: высокая пропускная способность; вездесущность (связь всюду); плавная интеграция с другими проводными и беспроводными IP-сетями, включая точки доступа 802.11; адаптивный ресурс и управление спектром; и высокое качество сервиса для мультимедиа. Для получения дополнительной информации см. Astely et al. (2009) и Larmo et al. (2009).

Тем временем беспроводные сети с уровнем пропускной способности 4G уже доступны. Основной пример **802.16**, так же известный как **WiMAX**. Краткий обзор мобильного WiMAX читайте в Ахмади (2009). Сказать, что промышленность стремительно развивается — огромное преуменьшение. Увидим, что произойдет через несколько лет.

2.8. Кабельное телевидение

Мы уже изучили более или менее подробно стационарные и беспроводные телефонные системы. Они, безусловно, будут играть важную роль в сетевых технологиях будущего. Тем не менее все популярнее становится альтернативная стационарная сетевая система, а именно кабельное телевидение. Многие уже получают доступ в Интернет и телефонные услуги по кабельным сетям. В следующих разделах мы будем обсуждать кабельное телевидение как сетевую структуру и как альтернативу телефонной системе, которую мы только что изучили. Дополнительную информацию по этой теме можно получить в изданиях Donaldson и Jones (2001), Dutta-Roy (2001) и Fellows и Jones (2001).

2.8.1. Абонентское телевидение

Кабельное телевидение впервые появилось в конце 1940-х годов и было способом улучшить прием сигнала в отдаленных поселках и горной местности. Система изначально состояла из большой антенны, расположенной на вершине холма и улавливающей телевизионный сигнал усилителя, называющегося **распределительным устройством**, и коаксиального кабеля, по которому сигнал доставлялся непосредственно к абонентам, как показано на рис. 2.44.

Рис. 2.44. Первая система кабельного телевидения

Вначале такая система называлась **абонентским телевидением**, или телевидением с коллективной антенной. Ее могло содержать даже какое-нибудь маленькое частное семейное предприятие. Любой предприниматель, немножко знакомый с электроникой, мог установить у себя в населенном пункте оборудование и ему оставалось только найти клиентов, готовых оплачивать услуги. По мере роста числа абонентов необходимо было добавлять кабели и усилители. Передача была исключительно односторонней: от распределителя к пользователям. К 1970 году появились тысячи независимых систем.

В 1974 году корпорация Time основала новый канал под названием «Домашняя билетная касса», который представлял собой кабельное кино. Затем появились другие подобные тематические каналы: спортивный, кулинарный, новостной и т. д. Это

привело к двум изменениям в данной отрасли. Во-первых, крупные корпорации стали скупать существующие кабельные системы и прокладывать свои кабели для привлечения новых клиентов. Во-вторых, со временем появилась необходимость в объединении систем, зачастую расположенных в различных городах, с целью основания новых кабельных каналов. Различные кабельные компании стали объединять свои сети, организуя единые региональные и национальные сети. Примерно то же самое происходило восьмьюдесятью годами ранее с телефонными сетями. Изолированные друг от друга телефонные станции стали объединяться, что позволило организовывать междугородные звонки.

2.8.2. Кабельный Интернет

В течение долгих лет кабельная система расширялась, и обычные кабели между городами стали заменяться оптоволоконными с высокой пропускной способностью. Примерно то же самое стало происходить в телефонной сети. Система, использующая оптическое волокно на длинных магистралях и коаксиальный кабель для подвода сигнала к домам, получила название **HFC (Hybrid Fiber Coax — Комбинированная оптокоаксиальная кабельная система)**. Электрооптические преобразователи, реализующие интерфейс между оптической и электрической частями сети, называются **оптоузлами**. Поскольку пропускная способность оптических кабелей гораздо выше, чем коаксиальных, один оптоузел может обслуживать несколько низкоскоростных линий. Часть современной системы HFC показана на рис. 2.45, *а*.

В последнее время многие операторы кабельных сетей решили, что пора начать проникновение в Интернет. Некоторые, впрочем, захотели заняться также кабельной телефонией. Технические различия кабельного телевидения и телефонии определили инженерные задачи, которые предстояло решить. Прежде всего необходимо было заменить все односторонние усилители двухсторонними, чтобы поддерживать передачу данных в двух направлениях. В то время как это происходило, ранний Интернет по кабельным системам использовал сеть кабельного телевидения для передачи данных пользователю и связь через телефонный модем для передачи в обратном направлении. Это было умное обходное решение, но небольшая часть сети по сравнению с тем, как это могло быть.

Между тем, есть еще одно существенное различие между HFC (см. рис. 2.45, *а*) и телефонной системой (рис. 2.45, *б*), которое устранить гораздо сложнее. Кабель может быть один на несколько домов, а телефонный провод местной линии в каждую квартиру подводится свой. Когда речь идет о широковещательном телевидении, особой разницы нет. Все телепрограммы распространяются по кабелю, и не важно, 10 или 10 000 абонентов будут подключены к нему. Но когда один и тот же кабель используется для доступа в Интернет, то один клиент, скачивающий очень большой файл, потенциально может тем самым отнимать существенную часть пропускной способности у всех остальных. Чем больше пользователей, тем жестче конкуренция между ними в этом смысле. В телефонной системе такого нет: передача большого файла по каналу ADSL никак не влияет на пропускную способность соседнего канала. С другой стороны, пропускная способность коаксиального кабеля много выше, чем витой пары, поэтому вам повезло, если ваши соседи не очень много пользуются Интернетом.

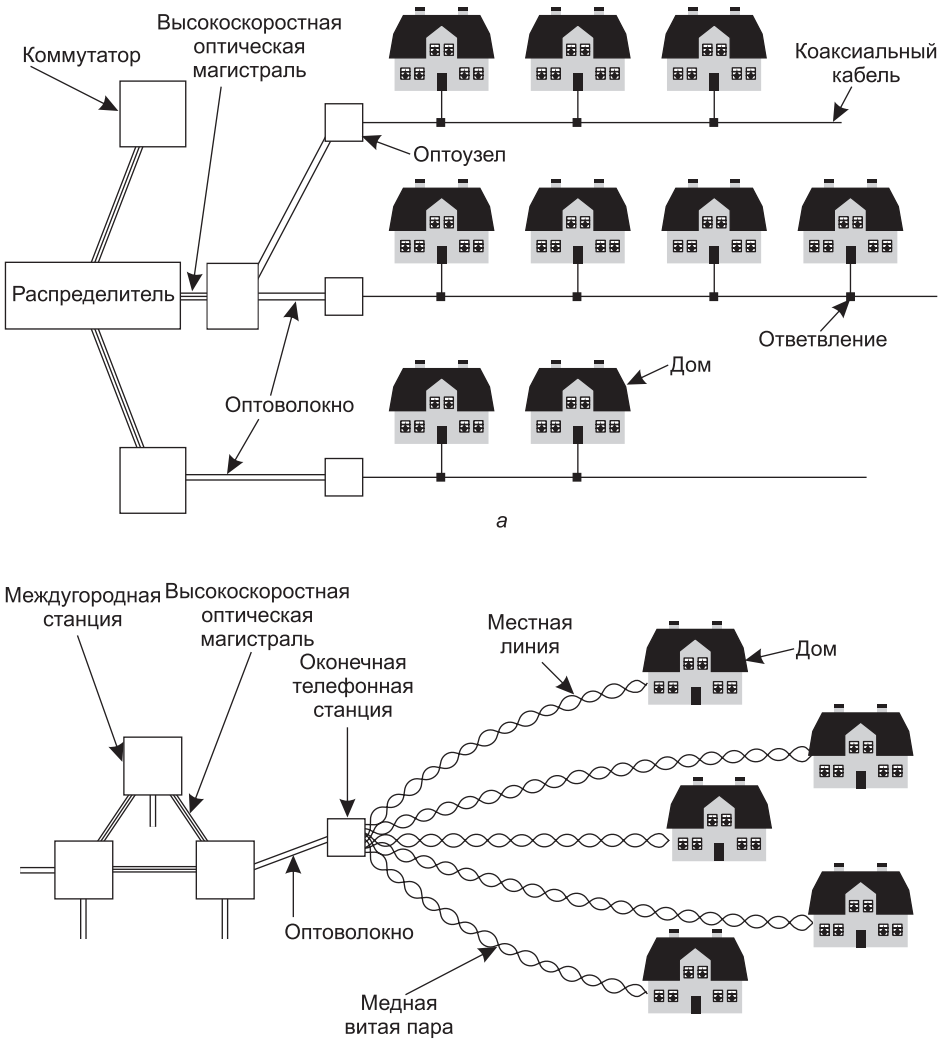


Рис. 2.45. Кабельное телевидение (*а*); стационарная телефонная система (*б*)

Эта проблема была решена: длинные кабели были разделены на короткие участки, напрямую подключаемые к оптоузлу. Доступная полоса пропускания на участке от распределителя до каждого оптоузла очень велика, и поскольку в одном сегменте кабеля обычно не бывает большого числа абонентов, трафик вполне управляем. Обычный кабельный сегмент охватывает 500–2000 домов, однако все больше людей подключается к кабельному Интернету, поэтому иногда требуется более мелкое разбиение, что приводит к появлению дополнительных оптоузлов.

2.8.3. Распределение частот

Если выкинуть все телевизионные каналы и использовать кабельную инфраструктуру исключительно для доступа в Интернет, это приведет к появлению большого числа недовольных пользователей, поэтому никто так не делает. Более того, в большинстве городов существуют определенные ограничения, не позволяющие так сделать, даже если какая-нибудь компания и захочет. Значит, нужно было найти какой-то способ совместного существования телевизионного сигнала и цифровых данных на одном кабеле.

Решение было основано на частотном уплотнении. Кабельное телевидение в Северной Америке традиционно занимает частоты с 54 до 550 МГц (за исключением диапазона с 88 до 108 МГц, отданного FM-радио). Ширина полосы каждого канала составляет 6 МГц, включая защитные полосы, и передает один традиционный аналоговый телеканал или несколько цифровых телеканалов. В Европе нижний предел обычно ограничен 65 МГц, а каналы имеют ширину полосы 6–8 МГц, что позволяет увеличить разрешение, требуемое системам PAL и SECAM, однако это не очень принципиально. Нижняя часть спектра не используется. Современные кабели хорошо работают на частотах свыше 550 МГц, часто до 750 МГц и выше. Было принято решение выделить под исходящие каналы частоты 5–42 МГц (чуть выше в Европе), а высокие частоты использовать для входящих каналов. Распределение спектра в кабельных системах показано на рис. 2.46.

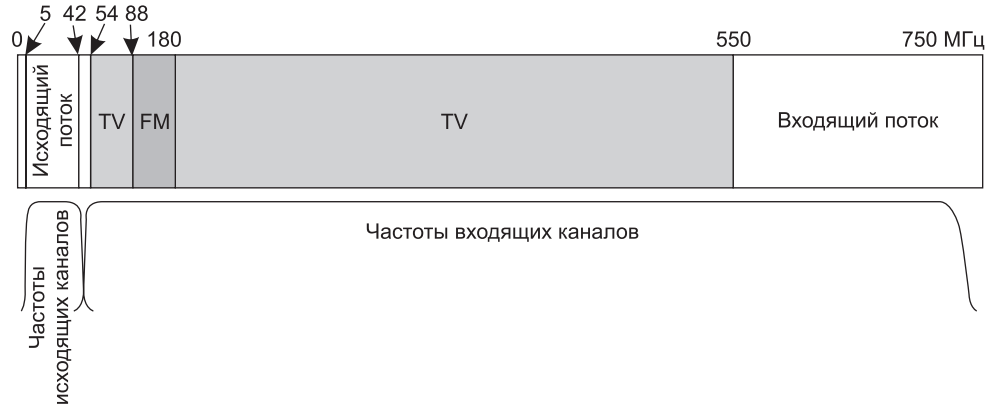


Рис. 2.46. Распределение частот в типичной системе кабельного телевидения, используемой для доступа в Интернет

Обратите внимание: поскольку телевизионный сигнал целиком идет только в одном направлении (входящем), можно использовать исходящие усилители, работающие только в диапазоне 5–42 МГц, а входящие — в диапазоне 54 МГц и выше, как показано на рисунке. Итак, входящий и исходящий спектры имеют сильный дисбаланс благодаря тому, что основная доля трафика приходится на телевидение и входящие интернет-каналы. И операторы кабельного телевидения, и компании, занимающиеся кабельным Интернетом, остались довольны таким распределением. Как мы уже го-

ворили, телефонные компании часто предлагают ассиметричный DSL-сервис, хотя у них нет особых технических оснований, чтобы так делать.

Кроме обновления усилителей, оператору нужно обновить и распределительное устройство на входе системы. Вместо латентного усилителя нужно поставить интеллектуальное цифровое вычислительное устройство с высокоскоростным оптоволоконным интерфейсом к провайдеру. Иногда обновляется даже имя этого устройства: вместо распределителя его называют **CMTS (Cable Modem Termination System — Оконечное устройство кабельного модема)**. Далее мы воздержимся от столь значительного обновления и будем по-прежнему называть распределитель распределителем.

2.8.4. Кабельные модемы

Для доступа в Интернет нужен кабельный модем — устройство, имеющее два интерфейса: один к компьютеру, второй — к кабельной сети. В первые годы существования кабельного Интернета у оператора связи были свои модемы, которые устанавливались у абонента специалистом службы технической поддержки. Однако затем стало понятно, что открытый стандарт может позволить создать рынок конкурентоспособных кабельных модемов, снизить цены на них и тем самым привлечь клиентов. Более того, возможность купить кабельный модем в обычном магазине и установить его самостоятельно (как пользователи всегда устанавливали беспроводные точки доступа) позволит избежать больших расходов на оплату выезда специалиста.

В результате многие операторы кабельных сетей объединились с фирмой CableLabs с целью выработки стандарта на кабельные модемы и тестирования продукции на совместимость. Модемы появившегося стандарта **DOCSIS (Data Over Cable Service Interface Specification — Спецификация передачи данных по кабельному интерфейсу)** практически заменили собственные модемы операторов. Версия DOCSIS 1.0 вышла в 1997 году, и скоро, в 2001 году, за ней — DOCSIS 2.0. Она увеличила скорости загрузки, чтобы лучше поддерживать симметричные сервисы, такие как IP-телефония. Наиболее современная версия стандарта — DOCSIS 3.0 — вышла в 2006 году. Она использует бóльшую полосу пропускания, чтобы увеличить скорости в обоих направлениях. Европейская версия стандарта называется **EuroDOCSIS**. Однако не всем операторам нравится идея свободной продажи стандартных кабельных модемов — слишком уж хорошие деньги они получают за сдачу в аренду модемов своим захваченным в плен клиентам. Открытый стандарт, породивший десятки фирм-производителей кабельных модемов, продающих их в магазинах, ведет к концу подобной практики.

Интерфейс между модемом и компьютером довольно традиционен. Обычно это Ethernet или, иногда, USB. Второй интерфейс более сложный, так как он использует FDM, TDM и CDMA для разделения между пользователями пропускной способности кабеля.

При включении кабельного модема он начинает прослушивать входящий канал в поисках специального пакета, время от времени посылаемого распределителем, чтобы получить системные параметры для модемов, только что включившихся в работу. После обнаружения данного пакета новый модем объявляет о своем появлении по одному из исходящих каналов. Распределитель отвечает, присваивая модему входящий и исходящий канал. Это распределение каналов может быть дина-

мически изменено распределителем, если он решит, что необходимо сбалансировать нагрузку.

Использование каналов на 8 МГц или на 6 МГц — дело частотного уплотнения. Каждый кабельный модем посылает данные по одному восходящему и одному нисходящему каналу или по нескольким каналам при DOCSIS 3.0. Обычная схема состоит в том, чтобы взять каждые 6 (или 8) МГц нисходящего канала и промодулировать их QAM-64 или, если качество кабеля исключительно хорошо, QAM-256. С каналом на 6 МГц и QAM-64 мы получаем приблизительно 36 Мбит/с. За вычетом служебных сигналов, полезная нагрузка сети — приблизительно 27 Мбит/с. С QAM-256 полезная нагрузка сети — приблизительно 39 Мбит/с. Европейские значения на 1/3 больше.

Для восходящих каналов имеется больше радиочастотного шума, потому что система не была первоначально разработана для данных, и шум от многих пользователей направляется к распределителю, поэтому используется более консервативная схема. Она колеблется от QPSK до QAM-128, где некоторые из символов используются для защиты от ошибок с решетчатой кодированной модуляцией (Треллис-модуляцией). С меньшим количеством битов на символ для восходящих данных, асимметрия между скоростью восходящего и нисходящего каналов намного выше, чем показано рис. 2.46.

Чтобы совместно использовать полосу пропусканя для восходящих данных многих пользователей, используется уплотнение с разделением времени. Иначе их передачи столкнулись бы в распределителе. Время работы делится на **мини-слоты (minislots)**, и разные пользователи отправляют данные в разные мини-слоты. Для этого модем определяет, на каком расстоянии от распределителя он находится. Для этого посылается специальный пакет и высчитывается время, через которое приходит ответ. Этот процесс называется **измерением дальности (ranging)**. Модему необходимо знать эти данные, чтобы правильно синхронизироваться. Каждый исходящий пакет должен умещаться в один или несколько соседних мини-слотов. Распределитель анонсирует каждое начало цикла мини-слотов, однако этот «стартовый выстрел» модемы слышат не одновременно, поскольку они находятся на разных расстояниях. Зная свое удаление от распределителя, модем может вычислить, когда на самом деле был послан принятый им сигнал начала мини-слота. Длина мини-слота зависит от сети. Обычно объем полезной информации в нем равен 8 байт.

Во время инициализации распределитель также присваивает модему мини-слот для запроса полосы пропускания восходящего канала. Когда компьютер хочет отослать пакет данных, он передает его модему, который запрашивает необходимое количество мини-слотов для него. Если запрос принят, то распределитель посылает подтверждение по нисходящему каналу. После этого пакет отправляется, начиная с первого «своего» мини-слота. Используя специальное поле заголовка, можно сообщить о необходимости передать дополнительные пакеты.

Как правило, одному и тому же мини-слоту запроса соответствует несколько модемов, что приводит к конкуренции между ними. Для решения этой проблемы существует две возможности. Первая — использовать множественный доступ с кодовым разделением каналов (CDMA), чтобы пользователи совместно использовали мини-слот. Это решает проблему конкуренции, потому что все пользователи с помощью CDMA могут посылать данные одновременно, хотя и с меньшей скоростью. Вторая возможность — не использовать CDMA, в этом случае может не быть никакого

подтверждения запроса, из-за столкновения. Тогда модем может повторить попытку только через случайный промежуток времени. Если при повторной попытке снова возникла коллизия, то случайный промежуток удваивается. (Для читателей, уже немного знакомых с сетевыми технологиями: это интервальный метод ALOHA с экспоненциальной двоичной отсрочкой передачи. Ethernet не может использоваться в качестве кабельного интерфейса, поскольку станции не могут прослушивать линию. Мы вернемся к этим вопросам в главе 4.)

Нисходящие каналы управляются не так, как восходящие. Во-первых, отправитель в этом случае только один — распределитель, поэтому не возникает никакой борьбы за линию и нет необходимости в мини-слотах, которые, на самом деле, являются разновидностью статистического временного уплотнения. Во-вторых, трафик нисходящего канала обычно гораздо выше, чем восходящего, поэтому используются пакеты фиксированного размера — 204 байта. Часть пакета — код коррекции ошибок Рида—Соломона, плюс еще некоторая служебная информация. Собственно данные занимают в пакете 184 байта. Эти числа были выбраны из соображений совместимости с цифровым телевидением, использующим MPEG-2, так что телевизионный и входящий информационный каналы имеют один и тот же формат. Логическая структура соединения показана на рис. 2.47.

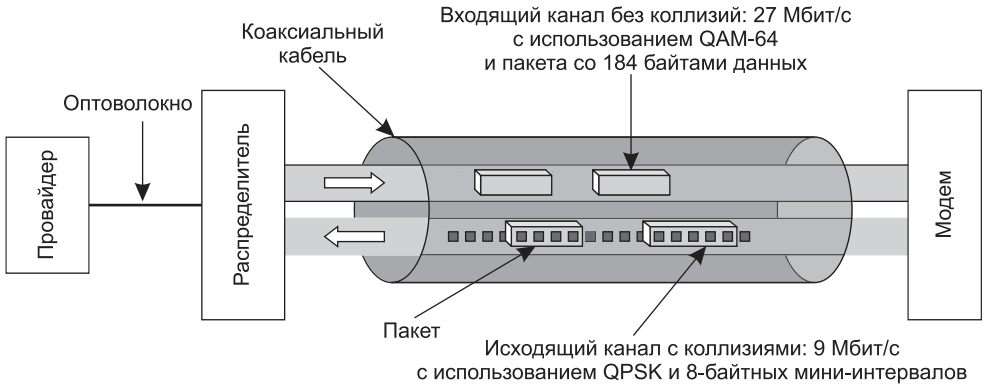


Рис. 2.47. Типичная схема входящего и исходящего каналов, принятая в США

2.8.5. ADSL или кабель?

Что лучше, ADSL или кабельная сеть? С тем же успехом можно спорить о том, какая операционная система лучше. Или какой язык. Или какая религия. Ответ зависит от того, кого вы спрашиваете. Давайте сравним ADSL и кабельные сети по нескольким параметрам. И та и другая система в качестве магистрального носителя использует оптическое волокно, однако на его концах используются разные типы носителей. В кабельных сетях это коаксиал, в ADSL — витая пара. Теоретически пропускная способность коаксиального кабеля в сотни раз выше, чем у витой пары. Тем не менее полная пропускная способность все равно недоступна пользователям кабельных систем, потому что большая часть полосы пропускания занята совершенно бесполезными вещами — телевизионными программами.

На практике довольно трудно говорить о реальной эффективной емкости каналов. Провайдеры ADSL заявляют некоторую пропускную способность (например, 1 Мбит/с по нисходящему каналу, 256 Кбит/с по восходящему) и обычно достигают примерно 80 % от нее. Провайдеры кабельных сетей могут искусственно ограничивать пропускную способность каждому пользователю, чтобы помочь предсказать качество работы, но они не могут действительно дать гарантии, потому что эффективная емкость зависит от того, сколько людей в настоящее время активно на кабельном сегменте пользователя. Иногда скорость будет выше, чем в ADSL, иногда — ниже. Раздражает в данном случае непредсказуемость. Если сейчас все «летает», это не означает, что через минуту будет так же, потому что не исключено, что именно сейчас самый большой свинтус в районе, занимающий своим трафиком всю пропускную способность сегмента, включает свой компьютер.

По мере привлечения в ADSL все большего числа пользователей, качество обслуживания практически не снижается, поскольку каждый абонент имеет выделенное соединение. В кабельной системе каждый новый пользователь сегмента снижает качество обслуживания в целом. Единственный выход из данной ситуации — разбивать загруженные участки на более мелкие и подсоединять их напрямую к оптическому кабелю. Это стоит довольно дорого, поэтому операторы всячески стараются избежать таких ситуаций.

Мы, между прочим, уже изучили одну систему с совместно используемым каналом — это мобильная телефонная система. Там тоже имеются группы пользователей, находящихся в одной ячейке, каждый из которых занимает какую-то часть пропускной способности. Обычно существует жесткое разделение используемых ресурсов, для этого применяется частотное и временно́е уплотнение, потому как речевой трафик обычно довольно ровный. Однако применять жесткое разделение ресурсов при передаче данных оказывается крайне неэффективным, потому что зачастую канал вообще простаивает, тогда зарезервированные ресурсы тратятся просто так. С кабелем используется более динамическое средство для распределения совместно используемой пропускной способности. Несмотря на все это, в этом смысле кабельная система гораздо ближе к мобильной телефонии, чем к стационарным системам.

Доступность — это параметр, по которому ADSL и кабельные сети отличаются друг от друга. У каждого есть телефон, но не каждый живет достаточно близко к оконечной ADSL-станции, чтобы установить соответствующую систему. С другой стороны, не у всех есть кабель в доме или в районе, но если уж он есть, то удаленность от оптоузла или распределительного устройства большой роли не играет. Стоит также отметить, что поскольку кабельные системы начались с кабельного телевидения, корпоративных клиентов у них очень мало.

Будучи двухточечной системой, ADSL является более защищенной, чем кабельная сеть. Любой абонент последней может запросто считывать все пакеты, проходящие мимо него. По этой причине любой приличный оператор кабельной сети предлагает шифрование трафика обоих направлений. И все же, даже если пакет перехвачен в зашифрованном виде, это менее безопасно, чем полное отсутствие возможности перехвата.

Телефонная система, вообще говоря, надежнее кабеля. Например, существует система резервного питания, которая позволяет телефонной сети работать даже при временных отключениях электричества. Если же отключится питание какого-либо

усилителя кабельной сети, все пользователи, находящиеся в его ведении, сразу потеряют соединение.

Наконец, существует большой выбор провайдеров ADSL. Иногда это даже форсируется специальными законодательными актами. Этого не скажешь про операторов кабельных сетей. Далеко не всегда есть какой-либо выбор.

Вывод такой: ADSL и кабельные сети имеют много общего и отличаются друг от друга не так уж сильно. Они предлагают сравнимое качество обслуживания и, по мере роста конкуренции между ними, по-видимому, будут предлагать сравнимые цены.

Резюме

Физический уровень является базовым для сетей любого типа. Природа носителей информации накладывает два фундаментальных ограничения на все каналы, и это определяет их допустимую пропускную способность. Первое ограничение носит имя Найквиста и имеет отношение к идеальным бесшумным каналам. Второе ограничение, ограничение Шеннона, говорит о каналах с шумом.

Среда передачи данных может быть проводниковой или беспроводной. Основными проводниковыми средами являются витая пара, коаксиальный кабель и оптоволоконный кабель. Среди беспроводных сред следует выделить радио, микроволны, инфракрасное и лазерное излучение распространяющиеся по воздуху и спутниковую связь.

Цифровые методы модуляции посылают биты по проводниковым и беспроводным средам как аналоговые сигналы. Линейные коды работают в основной полосе частот, и сигналы могут быть помещены в полосу пропускания путем модулирования амплитуды, частоты и фазы несущей частоты. Каналы могут быть совместно использованы пользователями путем временного, частотного мультиплексирования времени и мультиплексирования с кодовым разделением.

Ключевым элементом большинства глобальных сетей является телефонная система. Ее главные компоненты — это местные линии, магистрали и коммутаторы. ADSL имеет скорость до 40 Мбит/с, достигая ее путем разделения местной линии на множество виртуальных каналов, которые работают параллельно. Это существенно превосходит скорости телефонных модемов. Пассивные оптические сети доставляют волокно до жилища, для еще больших скоростей доступа чем ADSL.

Магистрали переносят цифровые данные. В них применяются различные способы уплотнения. Спектральное уплотнение (WDM) обеспечивает много каналов большой емкости по отдельным волокнам. Временное (TDM) уплотнение распределяет каждую высокоскоростную связь между пользователями. Важны технологии как коммутации каналов, так и коммутации пакетов.

Для мобильных применений обычная стационарная телефонная сеть не подходит. Мобильные телефоны сейчас очень широко распространены в качестве средства передачи речи, однако растет и их роль как средства передачи данных. Первое поколение мобильных телефонных систем, 1G, было аналоговым, доминировала система AMPS. Второе поколение было цифровым, системы стандарта GSM сейчас наиболее широко распространены в мире. Сейчас разворачивается третье поколение, оно также цифровое и базируется на широкополосном CDMA, с WCDMA а также CDMA2000.

Альтернативной сетевой системой является кабельное телевидение. Оно сильно видоизменилось с коаксиального кабеля до гибридной оптокоаксиальной сети, от телевидения до телевидения и Интернета. В принципе, данная система обладает высокой пропускной способностью, но реальное качество обслуживания сильно зависит от числа и деятельности активных пользователей.

Вопросы

1. Сосчитайте коэффициенты Фурье для функции $f(t) = t$ $(0 \leqslant t \leqslant 1)$.
2. По бесшумному каналу с полосой пропускания 4 кГц каждую 1 мс передаются отсчеты сигнала. Какова будет максимальная скорость передачи данных? Как изменится максимальная скорость передачи данных, если канал будет иметь шум с отношением сигнал/шум 30 дБ?
3. Ширина телевизионных каналов составляет 6 МГц. Сколько бит в секунду может быть передано по такому каналу при использовании четырехуровневых цифровых сигналов? Предполагается, что шума в канале нет.
4. Какова максимально допустимая скорость передачи данных при передаче двоичного сигнала по каналу с полосой пропускания 3 кГц и отношением сигнал/шум в 20 дБ?
5. Какое отношение сигнал/шум требуется для использования линии с полосой пропускания в 50 кГц в качестве канала T1?
6. Каковы преимущества волоконной оптики как среды передачи по сравнению с медью? Есть ли какая-либо обратная сторона использования волоконной оптики вместо меди?
7. Какова пропускная способность полосы спектра в 0,1 мкм для длины волны в 1 мкм?
8. Требуется переслать последовательность компьютерных экранных изображений по оптоволоконному кабелю. Размеры экрана 2560 × 1600 пикселов, каждый пиксел по 24 бита. Требуется передавать 60 экранов в секунду. Какая необходима для этого пропускная способность, а также какая часть спектра (в микронах) будет использована при условии, что передача осуществляется на длине волны 1,30 микрон?
9. Верна ли теорема Найквиста для высококачественного одномодового оптоволоконного кабеля или только для медного провода?
10. Радиоантенны обычно лучше всего работают при размере антенны, равном длине волны радиосигнала. Диаметр антенны варьируется в пределах от 1 см до 5 метров. Какому диапазону частот это соответствует?
11. Лазерный луч диаметром 1 мм нацелен на детектор диаметром 1 мм, установленный на крыше здания на расстоянии 100 м. На какой угол должен отклониться лазерный луч, чтобы он промахнулся мимо детектора?
12. 66 низкоорбитальных спутников проекта Iridium образуют шесть ожерелий вокруг Земли. Период их обращения составляет 90 минут. Каков средний интервал, необходимый наземному передатчику для осуществления передачи (handoff)?
13. Вычислите общее время передачи пакета для спутников GEO (высота: 35 800 км), MEO (высота: 18 000 км) и LEO (высота: 750 км).
14. Каково время ожидания вызова, сделанного на Северном полюсе до достижения Южного полюса, если звонок направлен через спутники Iridium? Предположите, что время коммутации в спутниках составляет 10 мкс и радиус земли составляет 6371 км.
15. Какая минимальная полоса пропускания необходима, чтобы достигнуть скорости передачи B бит/с, если сигнал передан с использованием NRZ, MLT-3 и манчестерского кодирования? Обоснуйте ваш ответ.

16. Докажите, что в кодировании 4B/5B переход сигнала произойдет, по крайней мере, через каждые четыре бита.
17. Сколько кодов оконечных телефонных станций существовало до 1984 года, когда доступ к каждой из станций осуществлялся через трехзначный код региона и первые три цифры местного номера? Коды регионов начинались с одной из цифр из диапазона 2–9, вторая цифра всегда была 1 или 0, а третья цифра могла быть любой. Первые две цифры местного номера были из диапазона 2–9; третья цифра могла быть любой.
18. Простая телефонная система состоит из двух оконечных коммутаторов и одного междугородного коммутатора, с которым оконечные коммутаторы соединены дуплексным кабелем с полосой пропускания в 1 МГц. За восьмичасовой рабочий день с одного телефона производится в среднем 4 звонка. Средняя продолжительность одного разговора составляет 6 мин. 10 процентов звонков являются междугородными (то есть проходят через междугородный коммутатор). Каково максимальное количество телефонов, которое может поддерживать оконечный коммутатор? (Предполагается 4 кГц на канал.) Объясните, почему телефонная компания может решить поддерживать меньшее число телефонов, чем этот максимум для оконечного коммутатора.
19. У местной телефонной компании 10 млн абонентов. Все телефоны подключены к центральному коммутатору медными витыми парами. Средняя длина витых пар составляет 10 км. Сколько стоит медь местных телефонных линий? Предполагается, что провода круглые в сечении, диаметром 1 мм. Плотность меди равна 9,0 г/см^3, а цена меди — 6 долларов за килограмм.
20. Какой системой является нефтепровод — симплексной, полудуплексной, дуплексной или вообще не вписывается в эту классификацию? Что можно сказать о реке или о соединении типа walkie-talkie?
21. Стоимость мощных микропроцессоров упала настолько, что теперь возможна их установка в каждый модем. Как это отразилось на обработке ошибок в телефонной линии? Отрицает ли это необходимость обнаружения и исправлении ошибок на уровне 2?
22. Амплитудно-фазовая диаграмма модема того же типа, что изображен на рис. 2.19, состоит из точек с координатами: (1, 1), (1, –1), (–1, 1) и (–1, –1). Сколько бит в секунду сможет передавать такой модем на скорости 1200 бод?
23. Какова максимальная скорость передачи битов, достижимая в стандартном V.32 модеме, если скорость двоичной передачи (в бодах) составляет 1200 и коррекция ошибок не применяется?
24. Сколько частот использует полнодуплексный модем с модуляцией QAM-64?
25. Десять сигналов, каждому из которых требуется полоса 4000 Гц, мультиплексируются в один канал с использованием частотного уплотнения (FDM). Какова должна быть минимальная полоса уплотненного канала? Ширину защитных интервалов считать равной 400 Гц.
26. Почему период дискретизации импульсно-кодовой модуляции был выбран равным 125 мкс?
27. Каков процент накладных расходов в канале T1, то есть какой процент от пропускной способности в 1,544 Мбит/с недоступен для конечного потребителя? Как это соотносится с процентом для каналов OC-1 и OC-768?
28. Сравните максимальную пропускную способность бесшумных каналов с полосой пропускания 4 кГц, использующих: а) аналоговое кодирование с двумя битами на отсчет; б) систему T1 с импульсно-кодовой модуляцией.
29. При потере синхронизации система T1 пытается восстановить синхронизацию при помощи первого бита каждого кадра. Сколько кадров должно быть исследовано для восстановления синхронизации с вероятностью ошибки 0,001?

30. В чем отличие, если оно есть, между демодуляторной частью модема и кодирующей частью кодека? (Оба преобразуют аналоговый сигнал в цифровой.)
31. В сети SONET точность системных часов составляет 10–9. Сколько времени понадобится, чтобы дрейф часов составил 1 бит? Есть ли практическое следствие из этих расчетов? Какое?
32. Сколько потребуется времени, чтобы передать файл размером 1 Гбайт от одного VSAT до другого, используя концентратор, как показано на рис. 2.14? Предположите, что скорость работы в направлении Земля — спутник 1 Мбит/с, скорость спутник — Земля 7 Мбит/с и используется коммутация каналов со временем настройки канала 1,2 с.
33. Вычислите время передачи в предыдущей задаче, если используется коммутация пакетов. Предположите, что размер пакета составляет 64 Кбайт, задержка коммутации спутника и концентратора составляет 10 мкс и размер заголовка пакета составляет 32 байт.
34. В табл. 2.5 скорость пользователя для канала OC-3 составляет 148,608 Мбит/с. Покажите, как это число может быть получено из параметров канала OC-3 системы SONET. Каково будут общее количество, SPE, и пользовательские скорости передачи данных линии OC-3072?
35. Для работы со скоростями передачи данных ниже STS-1 SONET имеет систему виртуального согласования (VT). Это часть полезной нагрузки, которая вставляется в кадр STS-1 и комбинируется с другими частями полезной нагрузки, заполняя весь кадр данных. Например, VT1.5 использует три колонки, VT2 использует 4 колонки, VT3 — 6 колонок, VT6 — 12 колонок кадра STS-1. Какие типы VT могут помочь согласовать скорость со следующими системами:

    1) Служба DS-1 (1,544 Мбит/с).

    2) Европейская служба CEPT-1 (2,048 Мбит/с).

    3) Служба DS-2 (6,312 Мбит/с).
36. Какова доступная для пользователя полоса пропускания в канале OC-12c?
37. Три сети с коммутацией пакетов состоят из n узлов каждая. Топология первой сети представляет собой звезду, с центральным коммутатором, вторая является двунаправленным кольцом, а в третьей все узлы соединены друг с другом. Каким будет наименьшее, среднее и наибольшее расстояние между узлами каждой сети в прыжках?
38. Сравните задержку при передаче сообщения длиной в x бит по пути из k прыжков в сети с коммутацией каналов и в (слабо загруженной) сети с коммутацией пакетов. Время установки канала составляет s секунд, задержка распространения сигнала равна d секунд на прыжок, размер пакета равен p бит, а скорость передачи данных составляет b бит/с. При каких условиях сеть с коммутацией пакетов будет обладать меньшим временем задержки? Объясните, при каких условиях следует отдать предпочтение сети с коммутацией пакетов, а не сети с коммутацией каналов.
39. Предположим, что x бит данных пользователя передается по пути в k прыжков в сети с коммутацией пакетов в виде серии пакетов, каждый из которых состоит из p бит данных и h бит заголовка, причем $x >> p + h$. Скорость передачи линий составляет b бит/с, а задержкой распространения сигнала можно пренебречь. Каким должно быть значение p, чтобы значение суммарной задержки было минимальным?
40. В обычной сотовой телефонной системе с шестигранными ячейками запрещено использовать одинаковые частотные диапазоны в соседних ячейках. Если доступно 840 частот, сколько из них может быть использовано в одной ячейке?
41. Реальная форма набора ячеек редко бывает такой правильной, как показано на рис. 2.39. Даже отдельные ячейки почти всегда имеют неправильную форму. Выскажите свои пред-

положения относительно причин этого явления. Как такая неправильная форма влияет на частоты, назначенные каждой ячейке?

42. Сколько микроячеек системы PCS диаметром 100 м потребуется, чтобы покрыть ими Сан-Франциско (120 км^2)?
43. Когда пользователь сотовой телефонной системы пересекает границу между сотами, в некоторых случаях разговор прерывается, несмотря на то что все приемники и передатчики функционируют нормально. Почему?
44. Пусть A, B и C одновременно передают нулевые биты, используя систему CDMA и элементарные последовательности, показанные на рис. 2.24, *а*. Как будет выглядеть результирующая элементарная последовательность?
45. Рассмотрим еще один подход к вопросу свойства ортогональности элементарных последовательностей CDMA. Каждый бит в паре последовательностей может совпадать или не совпадать. Выразите свойство ортогональности в терминах совпадений и несовпадений парных битов.
46. Приемник CDMA получает элементарную последовательность: (–1 + 1 – 3 + 1 – 1 – 3 + + 1 + 1). Предполагая, что исходные последовательности такие, как показано на рис. 2.24, *б*, какие станции посылали сигналы и какие именно?
47. На рис. 2.24 пользовательская пропускная способность для OC-3 составляет 148 608 Мбит/с. Покажите, как эта величина может быть рассчитана из параметров SONET OC-3.
48. Топология телефонной системы в части, включающей оконечный коммутатор, соединенный с телефонами абонентов, представляет собой звезду. Кабельное телевидение, напротив, состоит из единого длинного кабеля, объединяющего все дома в одной местности. Предположим, что в кабельном телевидении будущего вместо медного кабеля будет применяться оптоволоконный с пропускной способностью 10 Гбит/с. Сможет ли подобная линия воспроизвести работу телефонной линии и обеспечить каждому абоненту отдельную линию до оконечного коммутатора? Если да, то сколько телефонов может быть подключено к одному кабелю?
49. Оператор кабельной сети предоставляет доступ в Интернет в районе, состоящем из 5000 домов. Компания использует коаксиальный кабель и распределяет спектр таким образом, что полоса пропускания нисходящего потока для каждого кабеля составляет 100 Мбит/с. Чтобы привлечь клиентов, компания объявила, что каждому дому будет предоставлено 2 Мбит/с для нисходящего трафика в любое время. Опишите, что нужно компании, чтобы сдержать слово.
50. Используя распределение спектра, показанное на рис. 2.46, а также данную в тексте информацию, подсчитайте, сколько мегабит в секунду отводится в кабельной системе на нисходящий и восходящий каналы.
51. С какой скоростью пользователь кабельной сети может принимать данные, если остальные пользователи пассивны? Рассмотрите варианты интерфейса пользователя:

    1) 10 Мбит/с, Ethernet;

    2) 100 Мбит/с, Ethernet;

    3) беспроводная связь, 54 Мбит/с.
52. Мультиплексирование потоков данных STS-1 играет важную роль в технологии SONET. Мультиплексор 3:1 уплотняет три входных потока STS-1 в один выходной поток STS-3. Уплотнение производится побайтно, то есть первые три выходных байта соответствуют первым байтам входных потоков 1, 2 и 3 соответственно. Следующие три байта — вторым байтам потоков 1, 2 и 3 и т. д. Напишите программу, симулирующую работу мультиплексора

3:1. В программе должно быть пять процессов. Главный создает четыре других процесса (для трех входных потоков и мультиплексора). Каждый процесс входного потока считывает в кадр STS-1 данные из файла в виде последовательности из 810 байт. Затем кадры побайтно отсылаются процессу мультиплексора. Мультиплексор принимает потоки и выводит результирующий кадр STS-3 (снова побайтно), записывая его на стандартное устройство вывода. Для взаимодействия между процессами используйте метод конвейеров (pipes).

53. Напишите программу для реализации CDMA. Предположите, что длина элементарной последовательности равна 8 и число передающих станций 4. Ваша программа состоит из трех наборов процессов: четыре процесса передатчика (t0, t1, t2 и t3), один соединяющий процесс и четыре процесса приемника (r0, r1, r2 и r3). Основная программа, которая также действует как соединяющий процесс, сначала читает четыре элементарные последовательности (биполярная запись) из стандартного ввода и последовательности из 4 бит (по 1 бит на процесс передатчика, который будет передан) и порождает четыре пары процессов передатчика и приемника. Каждой паре процессов передатчика/приемника (t0, r0; t1, r1; t2, r2; t3, r3) соответствует одна элементарная последовательность, и каждому процессу передатчика назначают 1 бит (первый бит — t0, второй бит — t1 и т. д.). Затем каждый процесс передатчика вычисляет сигнал, который будет передан (последовательность из 8 битов), и посылает его в соединяющий процесс. После получения сигналов от всех четырех процессов передатчика соединяющий процесс комбинирует сигналы и посылает объединенный сигнал в четыре процесса приемника. Каждый процесс приемника вычисляет полученный бит и печатает его на стандартное устройство вывода. Для взаимодействия между процессами используйте метод конвейеров (pipes).

Глава 3
Канальный уровень

В этой главе мы рассмотрим принципы построения второго уровня нашей модели — канального уровня (иногда его называют также уровнем передачи данных). Мы обсудим алгоритмы, обеспечивающие надежную, эффективную пересылку целых блоков информации, называемых кадрами (сравните с физическим уровнем, задачей которого является передача отдельных битов), между двумя компьютерами. Мы будем рассматривать две машины, физически связанные каналом связи, действующим подобно проводу (например, коаксиальным кабелем, телефонной линией или беспроводным каналом). Основное свойство канала, которое делает его подобным проводу, заключается в том, что биты принимаются точно в том же порядке, в каком передаются.

На первый взгляд, можно подумать, что данная проблема настолько проста, что и изучать тут нечего — машина *A* просто посылает биты в канал, а машина *B* их оттуда извлекает. К сожалению, в каналах связи иногда случаются ошибки при передаче данных. Кроме того, скорость передачи данных ограничена, а время распространения сигнала отлично от нуля. Все эти ограничения оказывают серьезное влияние на эффективность передачи данных. Использующиеся для связи протоколы должны учитывать все эти факторы. Данным протоколам и посвящена эта глава.

После знакомства с ключевыми аспектами устройства канального уровня мы изучим его протоколы, рассмотрев природу ошибок и методы их обнаружения и исправления. Затем мы обсудим ряд протоколов, начиная с простых и далее рассматривая все более сложные протоколы. Каждый следующий протокол будет решать все более сложные проблемы канального уровня. Наконец, мы приведем несколько примеров протоколов передачи данных на канальном уровне.

3.1. Ключевые аспекты организации канального уровня

Канальный уровень использует определенные службы физического уровня для отправки и получения битов по коммуникационным каналам. У него есть ряд специфических функций. К ним относятся:

- обеспечение строго очерченного служебного интерфейса для сетевого уровня;
- обработка ошибок передачи данных;
- управление потоком данных, исключающее затопление медленных приемников быстрыми передатчиками.

Для этих целей канальный уровень берет пакеты, полученные с сетевого уровня, и вставляет их в специальные **кадры** (также часто называемые **фреймами** — **frames**) для передачи. В каждом кадре содержится заголовок, поле данных и концевик. Структура кадра показана на рис. 3.1. Управление кадрами — это основа деятельности канального уровня. В следующих разделах мы более подробно изучим обозначенные выше вопросы.

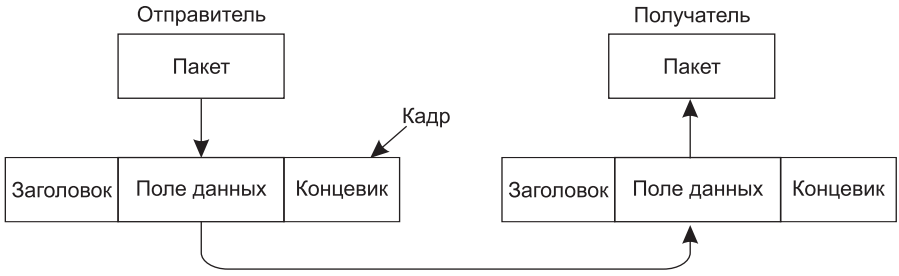


Рис. 3.1. Соотношение между пакетами и кадрами

Хотя эта глава и посвящена детальному рассмотрению канального уровня и соответствующих протоколов, многие вопросы, обсуждаемые здесь, такие как контроль ошибок и контроль потока, относятся также к транспортным и другим протоколам. Обеспечение надежности — это общая цель, для достижения которой слаженно работать должны все уровни. На самом деле, во многих сетях эти функции являются прерогативой верхних уровней и вообще не относятся к канальному уровню, который выполняет лишь простейшие функции. С другой стороны, не так уж это важно, потому что принципы все равно остаются неизменными. Аргументом в пользу рассмотрения их именно в свете канального уровня является то, что здесь они предстают в наиболее простой форме и их легко показать в деталях.

3.1.1. Сервисы, предоставляемые сетевому уровню

Задача канального уровня заключается в предоставлении сервисов сетевому уровню. Основным сервисом является передача данных от сетевого уровня передающей машины сетевому уровню принимающей машины. На передающей машине работает некий процесс, который передает биты с сетевого уровня на канальный уровень для передачи их по назначению. Работа канального уровня заключается в передаче этих битов на принимающую машину, так чтобы они могли быть переданы сетевому уровню принимающей машины, как показано на рис. 3.2, *а*. В действительности данные передаются по пути, показанному на рис. 3.2, *б*, однако проще представлять себе два канальных уровня, связывающихся друг с другом при помощи протокола передачи данных. По этой причине на протяжении этой главы будет использоваться модель, изображенная на рис. 3.2, *а*.

Канальный уровень может предоставлять различные сервисы. Их набор может быть разным в разных протоколах. Обычно возможны следующие варианты, которые мы рассмотрим далее по очереди.

1. Сервис без подтверждений, без установки соединения.
2. Сервис с подтверждениями, без установки соединения.
3. Сервис с подтверждениями, ориентированный на соединение.

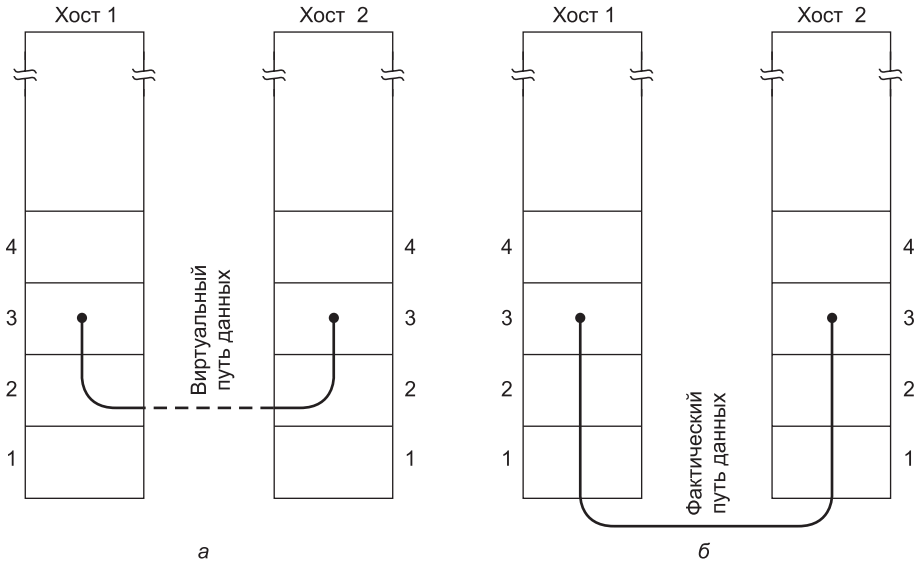


Рис. 3.2. Виртуальное соединение (*а*); реальное соединение (*б*)

Сервис без подтверждений и без установки соединения заключается в том, что передающая машина посылает независимые кадры принимающей машине, и принимающая машина не посылает подтверждений о приеме кадров. Хороший пример канального уровня, предоставляющего сервис такого класса, — Ethernet. Никакие соединения заранее не устанавливаются и не разрывается после передачи кадров. Если какой-либо кадр теряется из-за шума в линии, то на канальном уровне не предпринимается никаких попыток восстановить его. Данный класс сервисов приемлем при очень низком уровне ошибок. В этом случае вопросы, связанные с восстановлением потерянных при передаче данных, могут быть оставлены верхним уровням. Он также применяется в линиях связи реального времени, таких как передача речи, в которых лучше получить искаженные данные, чем получить их с большой задержкой.

Следующим шагом в сторону повышения надежности является сервис с подтверждениями, без установки соединения. При его использовании соединение также не устанавливается, но получение каждого кадра подтверждается. Таким образом, отправитель знает, дошел ли кадр до пункта назначения в целости или потерялся. Если в течение установленного интервала времени подтверждения не поступает, кадр посылается снова. Такой сервис полезен в случае использования каналов с большой

вероятностью ошибок, например, в беспроводных системах. Среди сервисов такого класса можно назвать, например, 802.11 (WiFi).

Вероятно следует отметить, что предоставление подтверждений является скорее оптимизацией, чем требованием. Сетевой уровень всегда может послать пакет и ожидать подтверждения его доставки удаленной машине. Если за установленный период времени подтверждение не будет получено отправителем, сообщение может быть выслано еще раз. Проблема при использовании данной стратегии заключается в том, что она зачастую оказывается неэффективной. Кадры обычно имеют жесткое ограничение максимальной длины, связанное с аппаратными требованиями, а также существует определенная задержка доставки. На сетевом уровне эти параметры неизвестны. Сетевой уровень может разбивать сообщения, скажем, на 10 кадров. В среднем, два из них потеряются по дороге. Передача сообщения таким методом может занять очень много времени. Если подтверждать получение отдельных кадров, то ошибки можно будет исправлять напрямую и гораздо быстрее. В таких надежных каналах, как, например, оптоволоконный кабель, накладные расходы на подтверждения на канальном уровне только снизят пропускную способность канала, однако для беспроводной связи (ненадежной по своей природе) такие расходы окупятся и уменьшат время передачи длинных сообщений.

Наиболее сложным сервисом, который может предоставлять канальный уровень, является ориентированный на соединение сервис с подтверждениями. При использовании данного метода источник и приемник, прежде чем передать друг другу данные, устанавливают соединение. Каждый посылаемый кадр нумеруется, а канальный уровень гарантирует, что каждый посланный кадр действительно принят на другой стороне канала связи. Кроме того, гарантируется, что каждый кадр был принят всего один раз и что все кадры были получены в правильном порядке. Таким образом, ориентированный на соединение сервис предоставляет процессам сетевого уровня эквивалент надежного потока битов. Он подходит для длинных ненадежных связей, таких как спутниковый канал или междугородное телефонное соединение. В службе без установления соединения возможно, что при потере подтверждения один и тот же кадр будет послан несколько раз и, следовательно, несколько раз получен. Это лишняя нагрузка на канал и неразумное расходование полосы пропускания.

При использовании ориентированного на соединение сервиса передача данных состоит из трех различных фаз. В первой фазе устанавливается соединение, при этом обе стороны инициализируют переменные и счетчики, необходимые для слежения за тем, какие кадры уже приняты, а какие — еще нет. Во второй фазе передаются кадры данных. Наконец, в третьей фазе соединение разрывается, и при этом освобождаются все переменные, буферы и прочие ресурсы, использовавшиеся во время соединения.

3.1.2. Формирование кадра

Для обслуживания сетевого уровня канальный уровень должен использовать сервисы, предоставляемые ему физическим уровнем. Физический уровень принимает необработанный поток битов и пытается передать его по назначению. Если канал зашумлен, как обычно бывает с беспроводными и большинством проводных соединений, то на физическом уровне добавляются избыточные сигналы, чтобы снизить

количество ошибок до допустимого уровня. Однако поток битов, получаемый на уровне передачи данных, не застрахован от ошибок. У некоторых битов могут быть другие значения, количество принятых бит может быть меньше, равно или больше числа переданных бит. Канальный уровень должен обнаружить ошибки и, если нужно, исправить их.

Обычно канальный уровень разбивает поток битов на отдельные кадры и вычисляет для каждого кадра короткий маркер, называемый контрольной суммой. Контрольная сумма добавляется в кадр перед тем, как он пересылается дальше. (Алгоритмы подсчета контрольных сумм будут обсуждаться ниже в этой главе.) Когда кадр прибывает в пункт назначения, его контрольная сумма подсчитывается снова. Если она отличается от содержащейся в кадре, то канальный уровень понимает, что при передаче кадра произошла ошибка, и принимает меры (например, игнорирует испорченный кадр и посылает передающей машине сообщение об ошибке).

Разбиение потока битов на отдельные кадры представляет собой более сложную задачу, чем это может показаться на первый взгляд. В хорошей системе приемник с легкостью находит отметки начала новых кадров, минимально нагружая полосу пропускания. Мы рассмотрим четыре метода маркировки границ кадров.

1. Подсчет количества байтов.
2. Использование сигнальных байтов с символьным заполнением.
3. Использование сигнальных битов с битовым заполнением.
4. Использование запрещенных сигналов физического уровня.

Первый метод формирования кадров использует поле в заголовке для указания количества байтов в кадре. Когда канальный уровень на принимающем компьютере видит это поле, он узнает, сколько байтов последует, и таким образом определяет, где находится конец кадра. Этот прием проиллюстрирован на рис. 3.3, *а* для четырех небольших кадров размером 5, 5, 8 и 8 байтов соответственно.

Недостаток такой системы в том, что при передаче может быть искажен сам счетчик. Например, если размер второго кадра из числа 5 станет из-за ошибки в единственном бите числом 7, как показано на рис. 3.3, *б*, то принимающая машина потеряет синхронизацию и не сможет правильно обнаружить начало следующего кадра. Даже если контрольная сумма не совпадет (скорее всего) и принимающий компьютер поймет, что кадр принят неверно, то он все равно не сможет определить, где начало следующего кадра. Запрашивать повторную передачу кадра также бесполезно, поскольку принимающий компьютер не знает, сколько байтов нужно пропустить до начала повторной передачи. По этой причине метод подсчета байтов отдельно от других теперь практически не применяется.

Второй метод формирования кадров решает проблему восстановления синхронизации после сбоя при помощи маркировки начала и конца каждого кадра специальными байтами. Зачастую в качестве разделителя используется один и тот же байт, называемый **флаговым**. Он устанавливается в начале и в конце кадра. Этот байт помечен на рис. 3.4, *а* как FLAG. Два соседних флаговых байта говорят о том, что кончился один кадр и начался другой. Таким образом, если приемник теряет синхронизацию, ему необходимо просто найти два флаговых байта, с помощью которых он распознает конец текущего кадра и начало следующего.

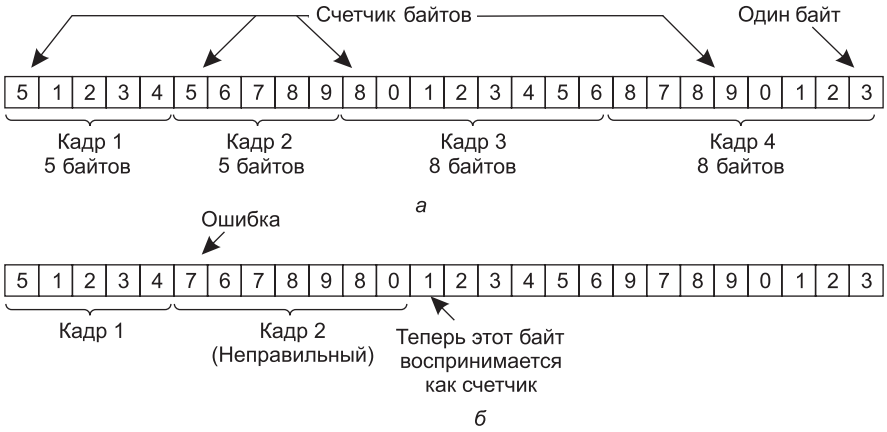


Рис. 3.3. Поток байтов: *а* — без ошибок; *б* — с одной ошибкой

Однако одна проблема все же остается. В передаваемых данных, особенно если это двоичные данные, такие как фотографии или музыка, запросто может встретиться последовательность, используемая в качестве флагового байта. Возникновение такой ситуации, скорее всего, собьет синхронизацию. Одним из способов решения проблемы является добавление специального escape-символа (знака переключения кода, ESC) непосредственно перед случайно совпавшим флаговым байтом внутри кадра. Таким образом, настоящий флаг можно отличить от «подложного» по наличию или отсутствию перед ним ESC. Канальный уровень получателя вначале убирает эти escape-символы, затем передает кадр на сетевой уровень. Такой метод называется **символьным заполнением (byte stuffing)**.

Следующий логичный вопрос: а что если и символ ESC случайно окажется среди прочих данных? Решение такое же: вставить перед этим фиктивным escape-символом настоящий. На стороне получателя первый символ ESC будет удален, а следующий байт данных останется, даже если это еще один байт ESC или флаговый байт. Некоторые примеры показаны на рис. 3.4, *б*. В любом случае, байтовая последовательность после ее очищения от вставных символов в точности совпадает с исходной. Найти границу кадра можно все так же по двум последовательным флаговым байтам до удаления дополнительных символов ESC.

Схема символьного заполнения, показанная на рис. 3.4, — это немного упрощенная модель протокола **PPP (Point-to-Point Protocol, протокол «точка–точка»)**, с помощью которого пакеты передаются по коммуникационным каналам. Мы изучим PPP в конце этой главы.

Третий метод разделения потока битов на кадры позволяет обойти недостатки символьного заполнения, которое обязывает использовать исключительно 8-битные байты. Делить данные на кадры можно на уровне бит, причем кадры могут содержать произвольное число бит и состоять из блоков любого размера. Данный метод был разработан для некогда популярного протокола **HDLC (High-level Data Link Control — высокоуровневый протокол управления каналом передачи данных)**. Каждый кадр начинается и завершается специальной последовательностью бит, 01111110 (или 0x7E

в шестнадцатеричной системе). Это все тот же флаговый байт. Если в битовом потоке передаваемых данных встретится пять идущих подряд единиц, уровень передачи данных автоматически вставит в выходной поток нулевой бит. **Битовое заполнение (bit stuffing)** аналогично символьному, при котором в кадр перед случайно встретившимся среди данных флагом вставляется escape-символ. Он также гарантирует минимальную плотность передачи, помогающую сохранять синхронизацию на физическом уровне. По этой причине битовое заполнение применяется в протоколе USB (**Universal Serial Bus — универсальная последовательная шина**).

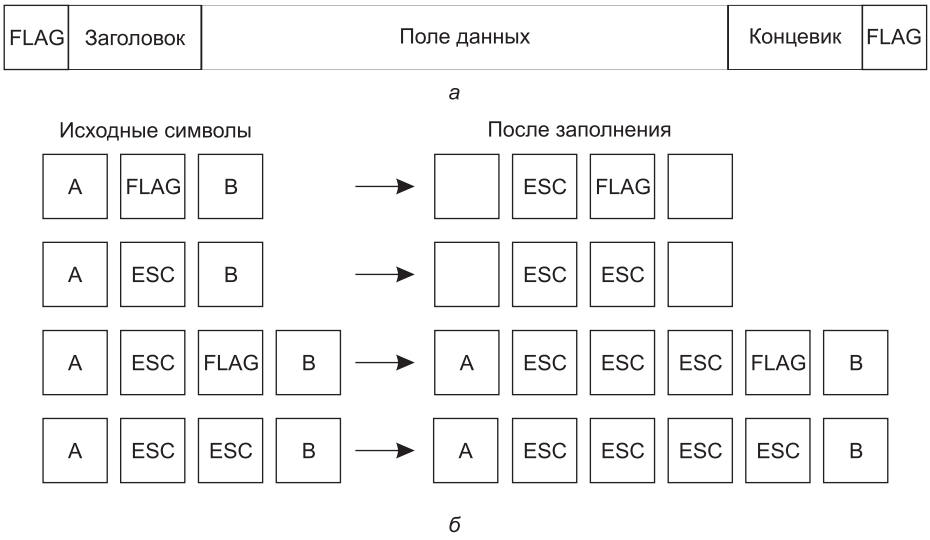


Рис. 3.4. Кадр, ограниченный флаговыми байтами (*а*); четыре примера байтовых последовательностей до и после символьного заполнения (*б*)

Когда принимающая сторона встречает пять единиц подряд, за которыми следует ноль, она автоматически удаляет этот ноль. Битовое заполнение, как и символьное, является абсолютно прозрачным для сетевого уровня обоих компьютеров. Если флаговая последовательность битов (01111110) встречается в данных пользователя, она передается в виде 011111010, но в памяти принимающего компьютера сохраняется опять в исходном виде: 01111110. На рис. 3.5 приведен пример битового заполнения.

Благодаря битовому заполнению границы между двумя кадрами могут быть безошибочно распознаны с помощью флаговой последовательности. Таким образом, если приемная сторона потеряет границы кадров, ей нужно всего лишь отыскать в полученном потоке битов флаговый байт, поскольку он встречается только на границах кадров и никогда — в данных пользователя.

Побочный эффект как битового, так и байтового заполнения заключается в том, что длина кадра зависит от содержимого, то есть от входящих в него данных. Например, если в данных флаговых байт нет, то 100 байт данных можно передать в кадре размером приблизительно 100 байт. Если же данные состоят исключительно из флаговых байт, то перед каждым таким байтом вставляется специальный символ, и длина кадра

увеличивается примерно до 200 байт. С битовым заполнением увеличение составляет около 12,5 %, так как к каждому байту добавляется 1 бит.

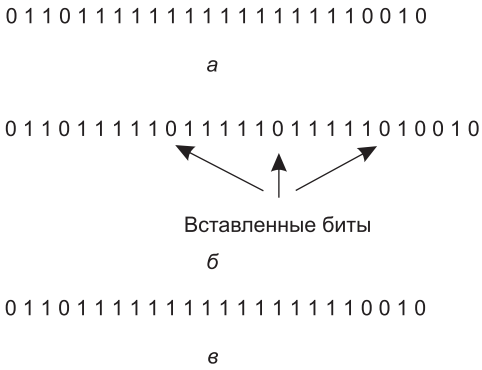


Рис. 3.5. Битовое заполнение: *а* — исходные данные; *б* — данные на линии; *в* — данные, сохраненные в памяти после удаления вставных битов

Последний метод формирования кадров напрямую связан с особенностями физического уровня. В главе 2 мы узнали, что при кодировании битов в виде сигналов для облегчения работы получающей стороны добавляются избыточные данные. Это означает, что в самих данных некоторые сигналы не появляются. Например, в линии 4B/5B четыре бита данных сопоставляются с пятью сигнальными битами, для того чтобы гарантировать удовлетворительную передачу битов. Таким образом, 16 из 32 возможных сигналов не используются. Некоторые из зарезервированных сигналов можно применять для обозначения начальной и конечной границы кадров. Фактически для разграничения кадров можно применять «запрещенные сигналы». Прелесть этого метода в том, что использование зарезервированных сигналов делает поиск начальной и конечной границы кадра чрезвычайно простым, отменяя необходимость заполнять данные дополнительными байтами или битами.

Во многих протоколах канального уровня для повышения безопасности используются различные сочетания описанных методов. В протоколах Ethernet и 802.11 кадр часто начинается с четко определенного шаблона, называемого **преамбулой** (**preamble**). Этот шаблон может быть довольно длинным (для 802.11 обычно 72 бита), что упрощает адресату задачу приема пакета. Вслед за преамбулой в заголовке передается поле длины (счетчик битов). Он нужен для обнаружения конца кадра.

3.1.3. Обработка ошибок

Решив проблему маркировки начала и конца кадра, мы сталкиваемся с новой проблемой: как гарантировать доставку сетевому уровню принимающей машины всех кадров и при этом расположить их в правильном порядке. Предположим, что у адресата есть возможность понять, содержит полученный кадр правильную или ошибочную информацию (подробнее о кодах, позволяющих распознать и исправить ошибки передачи, мы поговорим чуть далее). В линиях, где подтверждение передачи не требуется, отправитель просто посылает кадры, не заботясь о том, дошли ли они до

адресата. Но для ориентированного на соединение сервиса с подтверждениями этого недостаточно.

Обычно для гарантирования надежной доставки отправителю посылается информация о том, что происходит на другом конце линии. Протокол требует от получателя посылать обратно специальные управляющие кадры, содержащие позитивные или негативные сообщения о полученных кадрах. Получив позитивное сообщение, отправитель узнает, что посланный им кадр успешно получен на том конце линии. Негативное сообщение, напротив, означает, что с кадром что-то случилось и его нужно передать снова.

Кроме того, посланный кадр может из-за неисправности оборудования или какой-нибудь помехи (например, шума) пропасть полностью. В этом случае принимающая сторона его просто не получит и, соответственно, никак не прореагирует. Аналогично, если кадр подтверждения теряется, то отправитель также не знает, как ему действовать дальше. Очевидно, что протокол, в котором отправитель отсылает кадр, а затем начинает ожидать подтверждения (положительного или отрицательного ответа), зависнет навсегда, если кадр потеряется из-за, например, сбоя оборудования или коммуникационного канала.

Чтобы избежать зависаний сети в случае полной потери кадров, используются таймеры канального уровня. После посылки кадра включается таймер, отсчитывая интервал времени, достаточный для получения принимающим компьютером этого кадра, его обработки и посылки обратно подтверждения. В нормальной ситуации кадр правильно принимается, а подтверждение посылается назад и вручается отправителю, прежде чем истечет установленный интервал времени, и только после этого таймер отключается.

Однако если либо кадр, либо подтверждение теряется по пути, установленный интервал времени истечет, и отправитель получит сообщение о возможной проблеме. Самым простым решением для отправителя будет послать кадр еще раз. Однако при этом возникает опасность получения одного и того же кадра несколько раз на канальном уровне принимающего компьютера и повторной передачи его сетевому уровню. Чтобы этого не случилось, необходимо последовательно пронумеровать отсылаемые кадры, так чтобы получатель мог отличить повторно переданные кадры от оригиналов.

Вопрос управления таймерами и порядковыми номерами, гарантирующими, что каждый кадр доставлен сетевому уровню принимающего компьютера ровно один раз, не больше и не меньше, является очень важной частью задачи, решаемой канальным уровнем (и более высокими уровнями). Ниже в этой главе мы подробно изучим методы этого управления на серии постепенно усложняющихся примеров.

3.1.4. Управление потоком

Еще один важный аспект разработки канального уровня (а также более высоких уровней) связан с вопросом о том, что делать с отправителем, который постоянно желает передавать кадры быстрее, чем получатель способен их получать. Такая ситуация может возникнуть, если у передающей стороны оказывается более быстрый и мощный компьютер, чем у принимающей. Представьте, например, смартфон, запрашивающий веб-страницу с высокотехнологичного сервера. Мощнейшая машина развертывает

пожарный шланг, наводит его на бедный телефон и бомбардирует его данными до тех пор, пока тот не захлебнется. Даже при идеально работающей линии связи в определенный момент получатель просто не сможет продолжать обработку все прибывающих кадров и начнет их терять.

Очевидно, что для предотвращения подобной ситуации следует что-то предпринять. В настоящее время применяются два подхода. При первом, называющемся **управлением потоком с обратной связью (feedback-based flow control)**, получатель отсылает отправителю информацию, разрешающую последнему продолжить передачу или, по крайней мере, сообщающую о том, как идут дела у получателя. При втором подходе, **управлении потоком с ограничением (rate-based flow control)**, в протокол встраивается механизм, ограничивающий скорость, с которой передатчики могут передавать данные. Обратная связь с получателем отсутствует.

В этой главе мы рассмотрим только подход с обратной связью, в основном потому, что подход с ограничением используется только на транспортном уровне (подробнее об этом — в главе 5). Управление потоком с обратной связью применяется на канальном уровне, но чаще — на более высоких уровнях. При этом оборудование канального уровня работает достаточно быстро, чтобы потери информации не происходило. Например, про аппаратную реализацию этого уровня в виде карт **NIC (Network Interface Card — сетевая интерфейсная карта)** говорят, что она работает со скоростью передачи данных, то есть кадры обрабатываются с той же скоростью, с какой они прибывают. Канальный уровень не отвечает за переполнение, эти проблемы решаются на более высоких уровнях.

Известны различные схемы контроля потока с обратной связью, но большинство из них использует один и тот же принцип. Протокол содержит четко определенные правила, определяющие, когда отправитель может посылать следующий кадр. Эти правила часто запрещают пересылку кадра до тех пор, пока получатель не даст разрешения, либо явно, либо неявно. Например, при установке соединения получатель может сказать: «Вы можете послать мне сейчас *n* кадров, но не посылайте следующие кадры, пока я не попрошу вас продолжать». В данной главе мы рассмотрим различные механизмы, основанные на этом принципе.

3.2. Обнаружение и исправление ошибок

Как было показано в главе 2, у каналов передачи данных большой диапазон характеристик. В некоторых каналах, таких как оптическое волокно в телекоммуникационных сетях, вероятность ошибки крайне низка, поэтому потеря данных происходит исключительно редко. Но количество ошибок, например, в беспроводных сетях или старых местных сетях в десятки раз больше. Здесь ошибки передачи вообще считаются нормой. Для того чтобы полностью исключить их, потребуются слишком большие затраты в терминах производительности. Отсюда следует вывод: ошибки при передаче данных останутся важным фактором еще на долгие годы. Сейчас мы приступаем к изучению методов их обнаружения и устранения.

Разработчики сетей создали две основные стратегии для борьбы с ошибками. Каждый метод основывается на добавлении к передаваемым данным некоторой избыточ-

ной информации. В одном случае этой информации должно быть достаточно, чтобы принимающая сторона могла выявить, какие данные должны были прийти. В другом случае избыточной информации должно быть достаточно только для того, чтобы получатель понял, что произошла ошибка (без указания ее типа), и запросил повторную передачу. Первая стратегия использует коды, называющиеся **корректирующими** или **кодами с исправлением ошибок (error-correcting codes)**. Вторая — **коды с обнаружением ошибок (error-detecting codes)**. Использование кода с исправлением ошибок часто называют **прямым исправлением ошибок (Forward Error Correction — FEC)**.

Каждая стратегия занимает свою, так сказать, экологическую нишу. В высоконадежных каналах, таких как оптоволокно, дешевле использовать код с обнаружением ошибок и просто заново передавать случайные поврежденные блоки. Однако, скажем, беспроводные соединения, в которых может возникать множество ошибок, чаще используют коды с избыточностью, достаточной для того, чтобы приемник мог определить, какие данные должны были прийти. Прямое исправление ошибок применяется в шумных каналах, так как вероятность ошибки при повторной передаче так же велика, как и при первой.

Для того чтобы определить, какой метод лучше подойдет в конкретной ситуации, нужно понять, какой тип ошибок более вероятен. Ни код с исправлением ошибок, ни код с обнаружением ошибок не позволят справиться со всеми возможными ошибками, поскольку лишние биты, передаваемые для повышения надежности, также могут быть повреждены в пути. Хорошо бы, если бы каналы передачи данных могли отличать дополнительные биты от битов данных, но это невозможно. Для канала все биты одинаковы. Это означает, что для того чтобы избежать необнаруженных ошибок, необходимо использовать достаточно надежные коды, чтобы успешно справляться со всеми обнаруженными.

В одной модели считается, что причина ошибок — экстремально высокие значения термального шума, которые изредка на короткие промежутки времени перекрывают сигнал, порождая изолированные однобитные ошибки. Вторая модель предполагает, что ошибки чаще возникают целыми последовательностями, а не по одиночке. Объясняется это физическими процессами, вызывающими неполадки, такими как глубокое замирание беспроводного канала или временная электрическая помеха в кабельном канале.

Обе модели имеют практическую значимость, но у каждой свои преимущества и недостатки. Почему последовательность ошибок может быть лучше одиночных? Компьютер всегда отправляет данные блоками. Предположим, что размер блока равен 1000 бит, а вероятность ошибки равна 0,001 на один бит. Если бы ошибки были независимыми, то почти в каждом блоке обнаруживалась бы ошибка. Однако если возникнет целая последовательность ошибок, то в среднем из ста блоков только один будет поврежден. С другой стороны, последовательность ошибок исправить намного сложнее, чем изолированные ошибки.

Существуют и другие типы ошибок. Иногда местоположение ошибки известно. Например, физический уровень получает аналоговый сигнал, значение которого намного отличается от ожидаемого нуля или единицы, и объявляет, что бит потерян. Такой канал называется **каналом со стиранием (erasure channel)**. В каналах со стиранием ошибки исправлять проще, чем в каналах, где значения битов меняются на

противоположные: даже если значение бита утеряно, по крайней мере, нам известно, где притаилась ошибка. Тем не менее воспользоваться преимуществами стирающих каналов удается нечасто.

Далее мы рассмотрим коды с исправлением ошибок и коды с обнаружением ошибок. Прошу вас только не забывать о двух вещах. Во-первых, мы изучаем этот вопрос на канальном уровне, так как это первое место, где перед нами встает проблема надежной пересылки группы битов. Однако коды используются весьма широко, так как вопрос надежности важен всегда и везде. Коды исправления ошибок можно встретить на физическом уровне, особенно когда речь идет о зашумленных каналах, и на более высоких уровнях, особенно при рассылке мультимедийной информации в режиме реального времени. Коды обнаружения ошибок применяются на канальном, сетевом и транспортном уровнях.

Помимо этого, следует помнить, что коды ошибок относятся к прикладной математике. Если только вы не крупный специалист по полям Галуа или свойствам слабо заполненных матриц, используйте надежные коды, полученные из проверенных источников, и не пытайтесь конструировать собственные. В действительности, так делается во многих стандартных протоколах; одни и те же коды будут встречаться вам снова и снова. Далее мы подробно изучим простой код, а затем коснемся нескольких более сложных. Так вы сможете лучше понять преимущества и недостатки различных кодов и познакомиться с кодами, применяемыми на практике.

3.2.1. Коды с исправлением ошибок

Мы рассмотрим четырех разных кода с исправлением ошибок:

1. Коды Хэмминга.
2. Двоичные сверточные коды.
3. Коды Рида—Соломона.
4. Коды с малой плотностью проверок на четность.

Все эти коды добавляют к пересылаемой информации избыточные данные. Кадр состоит из m бит данных (то есть информационных бит) и r избыточных или контрольных бит. В **блочном коде** r контрольных бит вычисляются как простая функция связанных с ними m бит данных, как если бы для этих m бит данных r контрольных бит находились по огромной таблице соответствий. В **систематическом коде** m бит данных пересылаются напрямую вместе с контрольными битами и не кодируются перед отправкой. В **линейном коде** r контрольных бит вычисляются как линейная функция от m бит данных. Очень часто используется функция исключающего ИЛИ (XOR) или суммирование по модулю 2. Это означает, что для кодирования используются такие операции, как умножение матриц или простые логические схемы. Далее в этом разделе, если не указано иное, речь пойдет о линейных систематических блочных кодах.

Пусть полная длина кадра равна n (то есть $n = m + r$). Будем обозначать это как код (n,m). Набор из n бит, содержащий информационные и контрольные биты, часто называют n-битовым **кодовым словом** или кодовой комбинацией. **Кодовая норма (code rate)** или просто норма — это часть кодового слова, несущая неизбыточную информацию, или m/n. На практике значения нормы могут сильно отличаться. На-

пример, для шумного канала обычной нормой считается 1/2, то есть половина полученной информации будет избыточной. В хороших каналах норма близка к единице, и к большим сообщениям добавляются лишь несколько контрольных бит.

Для того чтобы понять, как исправляются ошибки, сначала необходимо познакомиться с самим понятием ошибки. Если рассмотреть два кодовых слова, например 10001001 и 10110001, можно определить число отличающихся в них соответствующих разрядов. В данном примере отличаются 3 бита. Для нахождения этого числа нужно сложить два кодовых слова по модулю 2 (операция «исключающее или») и сосчитать количество единиц в результате, например:

$$\begin{array}{r} 10001001 \\ + \\ \underline{10110001} \\ 00111000 \end{array}$$

Количество бит, которыми отличаются два кодовых слова, называется **кодовым расстоянием** или расстоянием между кодовыми комбинациями в смысле Хэмминга (Hamming, 1950). Смысл этого числа состоит в том, что если два кодовых слова находятся на кодовом расстоянии d, то для преобразования одного кодового слова в другое понадобится d ошибок в одиночных битах.

Зная алгоритм формирования контрольных разрядов, можно построить полный список всех допустимых кодовых слов и в этом списке найти такую пару кодовых слов, кодовое расстояние между которыми будет минимальным. Это расстояние называется **минимальным кодовым расстоянием** кода, или расстоянием всего кода в смысле Хэмминга.

В большинстве приложений передачи данных все 2^m возможных сообщений являются допустимыми, однако благодаря использованию контрольных бит не все 2^n возможных кодовых слов используются. В действительности, если контрольных битов r, то допустимыми считаются лишь $2^m/2^n$ или $1/2^r$ кодовых слов, а не все возможные 2^m. Именно разреженность данных в пространстве кодовых слов позволяет получателю распознавать и исправлять ошибки.

Способности блочного кода по обнаружению и исправлению ошибок зависят от его минимального кодового расстояния. Для надежного обнаружения d ошибок необходим код с минимальным кодовым расстоянием, равным $d + 1$, поскольку d однобитовых ошибок не смогут изменить одну допустимую комбинацию так, чтобы получилась другая допустимая комбинация. Когда приемник встречает запрещенную кодовую комбинацию, он понимает, что при передаче произошла ошибка. Аналогично, для исправления d ошибок требуется код с минимальным кодовым расстоянием, равным $2d + 1$, так как в данном случае даже при d однобитовых ошибках результат окажется ближе к исходному кодовому слову, чем к любому другому и, следовательно, его можно будет однозначно восстановить. Это означает, что исходное кодовое слово можно восстановить уникальным образом, основываясь на предположении, что возникновение большого числа ошибок менее вероятно.

В качестве простейшего примера корректирующего кода рассмотрим код, у которого есть всего четыре допустимые кодовые комбинации:

0000000000, 0000011111, 1111100000 и 1111111111

Этот код имеет расстояние, равное 5, это означает, что он может исправлять двойные ошибки и обнаруживать четверные ошибки. Если приемник получит кодовое слово 0000000111, ожидая только однобитные или двухбитные ошибки, он поймет, что оригинал должен быть равен 0000011111. Однако если тройная ошибка изменит 0000000000 на 0000000111, ошибка будет исправлена неверно. Если ожидать все перечисленные ошибки, то их можно распознавать. Ни одно из полученных кодовых слов не входит в список допустимых. Следовательно, произошла ошибка. Очевидно, что в данном примере невозможно одновременно исправлять двойные ошибки и распознавать четверные, потому что для этого полученное кодовое слово нужно будет интерпретировать двумя разными способами.

В нашем примере задачу декодирования, то есть поиск допустимого кодового слова, наиболее похожего на полученное, можно выполнить простым осмотром. К сожалению, в более общем случае, когда в качестве кандидатов приходится оценивать все кодовые слова, это заняло бы очень много времени. Вместо этого разрабатываются практические коды, которые по определенным подсказкам ищут наиболее подходящее исходное кодовое слово.

Попробуем создать код, состоящий из m информационных и r контрольных бит, способный исправлять одиночные ошибки. Каждому из 2^m допустимых сообщений будет соответствовать n недопустимых кодовых слов, отстоящих от сообщения на расстояние 1. Их можно получить инвертированием каждого из n бит n-битового кодового слова. Таким образом, каждому из 2^m допустимых сообщений должны соответствовать $n + 1$ кодовых комбинаций. Поскольку общее количество возможных кодовых комбинаций равно 2^n, получается, что $(n + 1)2^m \leqslant 2^n$. Так как $n = m + r$, это требование может быть преобразовано к виду:

$$(m + r + 1) \leqslant 2^r. \tag{3.1}$$

При заданном m данная формула описывает нижний предел требуемого количества контрольных бит для возможности исправления одиночных ошибок.

Этот теоретический нижний предел может быть достигнут на практике с помощью метода Хэмминга (1950). В **кодах Хэмминга** биты кодового слова нумеруются последовательно слева направо, начиная с 1. Биты с номерами, равными степеням 2 (1, 2, 4, 8, 16 и т. д.), являются контрольными. Остальные биты (3, 5, 6, 7, 9, 10 и т. д.) заполняются m битами данных. Такой шаблон для кода Хэмминга (11,7) с 7 битами данных и 4 контрольными битами показан на рис. 3.6. Каждый контрольный бит обеспечивает сумму по модулю 2 или четность некоторой группы бит, включая себя самого. Один бит может входить в несколько вычислений контрольных бит. Чтобы определить, в какие группы контрольных сумм будет входить бит данных в k-й позиции, следует разложить k по степеням числа 2. Например, 11 = 8 + 2 + 1, а 29 = 16 + 8 + 4 + 1. Каждый бит проверяется только теми контрольными битами, номера которых входят в этот ряд разложения (например, 11-й бит проверяется битами 1, 2 и 8). В нашем примере контрольные биты вычисляются для проверки на четность в сообщении, представляющем букву «A» в кодах ASCII.

Расстояние этого кода равно 3, то есть он позволяет исправлять одиночные ошибки (или распознавать двойные). Причина такой сложной нумерации бит данных и контрольных бит становится очевидной, если рассмотреть процесс декодирования. Когда

прибывает кодовое слово, приемник повторяет вычисление контрольных бит, учитывая значения полученных контрольных бит. Их называют результатами проверки. Если контрольные биты правильные, то для четных сумм все результаты проверки должны быть равны нулю. В таком случае кодовое слово принимается как правильное.

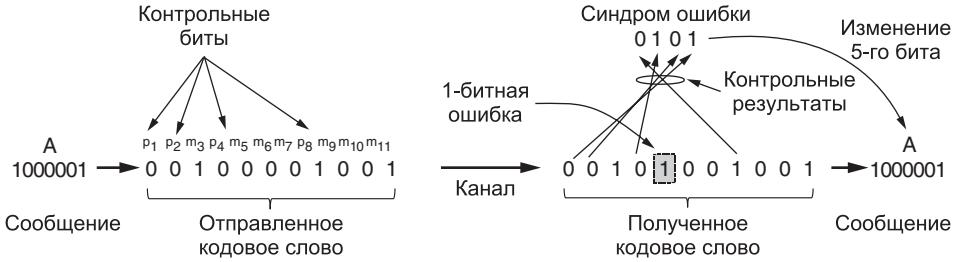


Рис. 3.6. Пример кода Хэмминга (11,7), исправляющего однобитную ошибку

Если не все результаты проверки равны нулю, это означает, что обнаружена ошибка. Набор результатов проверки формирует **синдром ошибки (error syndrome)**, с помощью которого выделяется и исправляется ошибка. На рис. 3.6 в канале произошла ошибка в одном бите. Результаты проверки равны 0, 1, 0 и 1 для k = 8, 4, 2 и 1 соответственно. Таким образом, синдром равен 0101 или 4 + 1 = 5. Согласно схеме, пятый бит содержит ошибку. Поменяв его значение (а это может быть контрольный бит или бит данных) и отбросив контрольные биты, мы получим правильное сообщение: букву «A» в кодах ASCII.

Кодовые расстояния полезны для понимания блочных кодов, а коды Хэмминга применяются в самокорректирующейся памяти. Однако в большинстве сетей используются более надежные коды. Второй тип кода, с которым мы познакомимся, называется **сверточным кодом**. Из всех рассмотренных в данной книге только он не относится к блочному типу. Кодировщик обрабатывает последовательность входных бит и генерирует последовательность выходных бит. В отличие от блочного кода, никакие ограничения на размер сообщения не накладываются, также не существует грани кодирования. Значение выходного бита зависит от значения текущего и предыдущих входных бит — если у кодировщика есть возможность использовать память. Число предыдущих бит, от которого зависит выход, называется длиной кодового ограничения для данного кода. Сверточные коды описываются в терминах кодовой нормы и длины кодового ограничения.

Сверточные коды широко применяются в развернутых сетях. Например, входят в мобильную телефонную систему GSM, спутниковые сети и сети 802.11. В качестве примера можно рассмотреть популярный сверточный код, показанный на рис. 3.7. Он называется сверточным кодом NASA с $r = 1/2$ и $k = 7$ (впервые этот код был использован при подготовке космического полета станции Voyager, запущенной в 1977 году). С тех пор он постоянно используется в других приложениях, таких как сети 802.11.

На рис. 3.7 для каждого входного бита слева создается два выходных бита справа. Эти выходные биты получаются путем применения операции исключающего ИЛИ к входному биту и внутреннему состоянию. Так как кодирование работает на уровне бит и использует линейные операции, это двоичный линейный сверточный код. По-

скольку один входной бит создает два выходных бита, кодовая норма r равна 1/2. Этот код не систематический, так как входные биты никогда не передаются на выход напрямую, без изменений.

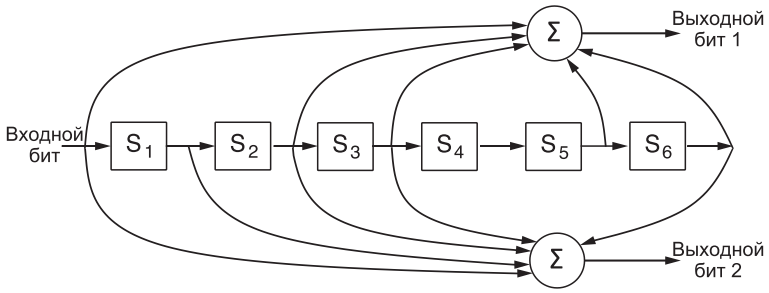


Рис. 3.7. Двоичный сверточный код NASA применяется в сетях 802.11

Внутреннее состояние хранится в шести регистрах памяти. При поступлении на вход очередного бита значения в регистрах сдвигаются вправо. Например, если на вход подается последовательность 111, а в первоначальном состоянии в памяти только нули, то после подачи первого, второго и третьего бита оно будет меняться на 100000, 110000 и 111000 соответственно. На выходе получатся значения 11, затем 10 и затем 01. Для того чтобы полностью подавить первоначальное состояние регистров (тогда оно не будет влиять на результат), требуется семь сдвигов. Длина кодового ограничения для данного кода k, таким образом, равна 7.

Декодирование сверточного кода осуществляется путем поиска последовательности входных битов, с наибольшей вероятностью породившей наблюдаемую последовательность выходных битов (включающую любые ошибки). Для небольших значений k это делается с помощью широко распространенного алгоритма, разработанного Витерби (Forney, 1973). Этот алгоритм проходит по наблюдаемой последовательности, сохраняя для каждого шага и для каждого возможного внутреннего состояния входную последовательность, которая породила бы наблюдаемую последовательность с минимальным числом ошибок. Входная последовательность, которой соответствует минимальное число ошибок, и есть наиболее вероятное исходное сообщение.

Сверточные коды были очень популярны для практического применения, так как в декодировании очень легко учесть неопределенность значения бита (0 или 1). Например, предположим, что –1 В соответствует логическому уровню 0, а +1 В соответствует логическому уровню 1. Мы получили для двух бит значения 0,9 В и –0,1 В. Вместо того чтобы сразу определять соответствие — 1 для первого бита и 0 для второго, — можно принять 0,9 В за «очень вероятную единицу», а –0,1 В за «возможный нуль» и скорректировать всю последовательность. Для более надежного исправления ошибок к неопределенностям можно применять различные расширения алгоритма Витерби. Такой подход к обработке неопределенности значения бит называется **декодированием с мягким принятием решений (soft-decision decoding)**. И наоборот, если мы решаем, какой бит равен нулю, а какой единице, до последующего исправления ошибок, то такой метод называется **декодированием с жестким принятием решений (hard-decision decoding)**.

Третий тип кода с исправлением ошибок, с которым мы познакомимся, называется **кодом Рида—Соломона**. Как и коды Хэмминга, коды Рида—Соломона относятся к линейным блочным кодам; также многие из них систематические. Но в отличие от кодов Хэмминга, которые применяются к отдельным битам, коды Рида—Соломона работают на группах из m-битовых символов. Разумеется, математика здесь намного сложнее, так что применим аналогию.

Коды Рида-Соломона основываются на том факте, что каждый многочлен n-й степени уникальным образом определяется $n + 1$ точкой. Например, если линия задается формулой $ax + b$, то восстановить ее можно по двум точкам. Дополнительные точки, лежащие на той же линии, излишни, но они полезны для исправления ошибок. Предположим, что две точки данных представляют линию. Мы отправляем по сети эти две точки данных, а также еще две контрольные точки, лежащие на той же линии. Если в значение одной из точек по пути закрадывается ошибка, то точки данных все равно можно восстановить, подогнав линию к полученным правильным точкам. Три точки окажутся на линии, а одна, ошибочная, будет лежать вне ее. Обнаружив правильное положение линии, мы исправили ошибку.

В действительности, коды Рида—Соломона определяются как многочлены на конечных полях. Для m-битовых символов длина кодового слова составляет $2^m - 1$ символов. Очень часто выбирают значение $m = 8$, то есть одному символу соответствует один байт. Тогда длина кодового слова — 255 байт. Широко используется код (255,233), он добавляет 32 дополнительных символа к 233 символам данных. Декодирование с исправлением ошибок выполняется при помощи алгоритма, разработанного Берлекэмпом и Мэсси, который эффективно осуществляет подгонку для кодов средней длины (Massey, 1969).

Коды Рида—Соломона широко применяются на практике благодаря хорошим возможностям по исправлению ошибок, особенно последовательных. Они используются в сетях DSL, в кабельных и спутниковых сетях, а также для исправления ошибок на компакт-дисках, дисках DVD и Blu-ray. Поскольку работа идет на базе m-битных символов, однобитные ошибки и m-битные последовательные ошибки обрабатываются одинаково — как одна символьная ошибка. При добавлении $2t$ избыточных символов код Рида—Соломона способен исправить до t ошибок в любом из переданных символов. Это означает, например, что код (255,233) с 32 избыточными символами исправляет до 16 символьных ошибок. Так как символы могут быть последовательными, а размер их обычно составляет 8 бит, то возможно исправление последовательных ошибок в 128 битах. Ситуация выглядит еще лучше, если модель ошибок связана с удалением данных (например, царапина на компакт-диске, уничтожившая несколько символов). В таком случае возможно исправление до $2t$ ошибок.

Коды Рида—Соломона часто используют в сочетании с другими кодами, например сверточными. И вот почему. Сверточные коды эффективно обрабатывают изолированные однобитные ошибки, но с последовательностью ошибок они, скорее всего, не справятся, особенно если ошибок в полученном потоке бит слишком много. Добавив внутрь сверточного кода код Рида—Соломона, вы сможете очистить поток бит от последовательностей ошибок. Таким образом, получившийся составной код обеспечит надежную защиту как от одиночных, так и от массовых ошибок.

Наконец, взглянем на код **LDPC (Low-Density Parity Check, код с малой плотностью проверок на четность)**. Коды LDPC — это линейные блочные коды, изобретенные Робертом Галлагером и описанные в его докторской диссертации (Gallagher, 1962). Как и о большинстве других диссертаций, об этой вскоре позабыли. Однако в 1995 году, когда вычислительные мощности сделали огромный скачок вперед, представленный в ней код изобрели заново.

В коде LDPC каждый выходной бит формируется из некоторого подмножества входных бит. Это приводит нас к матричному представлению кода с низкой плотностью единиц — отсюда и название. Полученные кодовые слова декодируются алгоритмом аппроксимации, который последовательно улучшает наилучшее приближение, составленное из полученных данных, пока не получает допустимое кодовое слово. Так осуществляется устранение ошибок.

Коды LDPC удобно применять для блоков большого размера. Они превосходно справляются с ошибками — лучше, чем на практике удается многим другим кодам (включая те, с которыми мы уже познакомились). По этой причине коды LDPC быстро добавляются в новые протоколы. Они являются частью стандарта цифрового телевидения, сетей Ethernet 10 Гбит/с, сетей, работающих по линиям электропитания, а также последней версии 802.11. Очевидно, что эти коды обязательно будут использоваться и в новых сетях, пока что находящихся на стадии разработки.

3.2.2. Коды с обнаружением ошибок

Коды с исправлением ошибок широко применяются в беспроводных системах связи, которые славятся зашумленностью среды передачи и, следовательно, большим количеством ошибок по сравнению с системами на основе оптоволоконного кабеля. Передать что-либо, не используя исправление ошибок, было бы практически невозможно. Однако в системах, где информация передается по оптоволокну или высококачественному медному проводу, уровень ошибок гораздо ниже, поэтому обнаружение ошибок и повторная передача являются более подходящим методом.

Мы рассмотрим три кода с обнаружением ошибок. Все они относятся к линейным систематическим блочным кодам:

1. Код с проверкой на четность.
2. Код с контрольными суммами.
3. Циклический избыточный код.

Для того чтобы понять, в каких ситуациях обнаружение ошибок может быть эффективнее исправления, возьмем первый из перечисленных кодов. К отправляемым данным присоединяется единственный **бит четности (parity bit)**, который выбирается так, чтобы число единичных битов в кодовом слове было четным (или нечетным). Это эквивалентно вычислению бита четности в виде суммы по модулю 2 для битов данных (или применению операции исключающего ИЛИ). Например, если отправляется кодовое слово 1011010 и число единиц должно быть четным, то в конце добавляется ноль, и последовательность превращается в 10110100. Если же число единиц должно быть нечетным, то последовательность устанавливается такая: 10110101. Кодовое рас-

стояние кода с единственным битом четности равно двум, так как любая однобитная ошибка меняет четность кодового слова на неправильную. Это означает, что данный код позволяет распознавать однобитные ошибки.

Рассмотрим канал с изолированными ошибками, возникающими с вероятностью 10^{-6} на бит. Такое значение может показаться очень небольшим, но для длинного кабельного канала, в котором распознавать ошибки довольно сложно, оно в лучшем случае считается допустимым. Типичные локальные сети характеризуются вероятностью ошибки 10^{-10}. Пусть блок данных состоит из 1000 бит. Для создания кода, исправляющего однократные ошибки в 1000-битном блоке, как видно из представленного выше уравнения (3.1), потребуется 10 контрольных бит. Для 1 Мбит данных это составит 10 000 проверочных бит. Чтобы просто обнаруживать одиночную 1-битовую ошибку, достаточно одного бита четности на блок. На каждые 1000 блоков будет выявляться одна ошибка, и придется переслать повторно еще один блок (1001 бит), чтобы исправить ее. Таким образом, суммарные накладные расходы на обнаружение ошибки и повторную передачу составят всего 2001 бит на 1 Мбит данных против 10 000 бит, необходимых для кода Хэмминга.

Проблема данной схемы заключается в том, что если к блоку добавлять всего один бит четности, то гарантированно распознаваться будет только одна однобитная ошибка в блоке. В случае возникновения последовательности ошибок вероятность обнаружения ошибки будет всего лишь 0,5, что абсолютно неприемлемо. Этот недостаток может быть исправлен, если рассматривать каждый посылаемый блок как прямоугольную матрицу n бит шириной и k бит высотой (принцип ее построения был описан выше). Если вычислить и отправить один бит четности для каждой строки, то гарантированно обнаружить можно будет до k однобитных ошибок, при условии, что в каждой строке будет не большое одной ошибки.

Однако можно сделать кое-что еще, чтобы повысить уровень защиты от последовательностей ошибок — биты четности можно вычислять в порядке, отличном от того, в котором данные отправляются. Этот способ называется **чередованием (interleaving)**. В нашем примере мы будет вычислять бит четности для каждого из n столбцов, но биты данных отправляться будут в виде k строк, в обычном порядке: сверху вниз и слева направо. В последней строке отправим n бит четности. На рис. 3.8 порядок пересылки показан для $n = 7$ и $k = 7$.

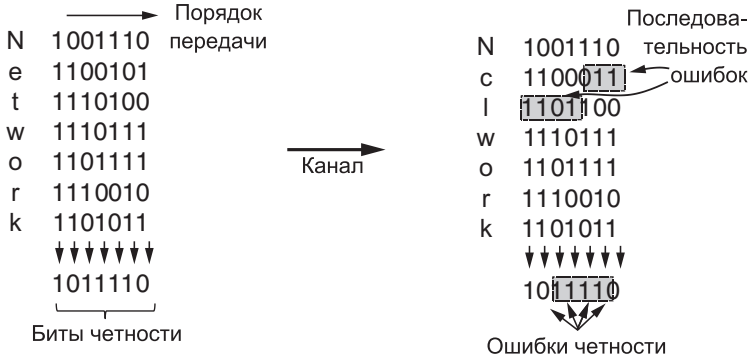


Рис. 3.8. Чередование битов четности для обнаружения последовательностей ошибок

Чередование представляет собой общую технику преобразования кода, способного обнаруживать (или исправлять) изолированные ошибки, в код, обнаруживающий (или исправляющий) последовательности ошибок. На рис. 3.8, там, где присутствует последовательность ошибок длиной $n = 7$, мы видим, что ошибочные биты находятся в разных столбцах (последовательность ошибок не означает, что все биты в ней неправильные; это всего лишь подразумевает, что, по меньшей мере, первый и последний биты сбойные. На рис. 3.8 из семи сбойных бит на самом деле изменено значение только четырех). В каждом из n столбцов повреждено будет не больше одного бита, поэтому биты четности этих столбцов помогут выявить ошибку. В данном методе n бит четности в блоках из kn битов данных применяются для обнаружения одной последовательности ошибок длиной n бит или меньше.

Последовательность ошибок длиной $n + 1$ не будет обнаружена, если будут инвертированы первый и последний биты, а все остальные биты останутся неизменными. Если в блоке при передаче возникнет длинная последовательность ошибок или несколько коротких, вероятность того, что четность любого из n столбцов будет верной (или неверной), равна 0,5, поэтому вероятность необнаружения ошибки будет равна 2^{-n}.

Второй тип кода с обнаружением ошибок, **код с использованием контрольной суммы**, весьма напоминает группу кодов, применяющих биты четности. Под «контрольной суммой» часто подразумевают любую группу контрольных бит, связанных с сообщением, независимо от способа их вычисления. Группа бит четности — также один из примеров контрольной суммы. Однако существуют и другие, более надежные контрольные суммы, основанные на текущей сумме бит данных в сообщении. Контрольная сумма обычно помещается в конец сообщения, в качестве дополнения функции суммирования. Таким образом, ошибки можно обнаружить путем суммирования всего полученного кодового слова: бит данных и контрольной суммы. Если результат равен нулю, значит, ошибок нет.

Один из примеров контрольной суммы — это 16-битная контрольная сумма, которая используется во всех пакетах протокола IP при пересылке данных в Интернете (Braden и др., 1988). Она представляет собой сумму бит сообщения, поделенного на 16-битные слова. Так как данный метод работает со словами, а не с битами (как при использовании битов четности), то ошибки, при которых четность не меняется, все же изменяют значение суммы, а значит, могут быть обнаружены. Например, если бит младшего разряда в двух разных словах меняется с 0 на 1, то проверка четности этих битов не выявит ошибку. Однако при добавлении к 16-битной контрольной сумме две единицы дадут другой результат, и ошибка станет очевидной.

Контрольная сумма, применяемая в Интернете, вычисляется с помощью обратного кода или арифметики с дополнением до единицы, а не как сумма по модулю 2^{16}. В арифметике обратного кода отрицательное число представляет собой поразрядное дополнение своего положительного эквивалента. Большинство современных компьютеров работают на арифметике с дополнением до двух, в которой отрицательное число является дополнением до единицы плюс один. На компьютере с арифметикой с дополнением до двух сумма с дополнением до единицы эквивалентна сумме по модулю 2^{16}, причем любое переполнение старших бит добавляется обратно к младшим битам. Такой алгоритм обеспечивает единообразный охват данных битами контроль-

ной суммы. В противном случае при сложении двух старших бит переполнение может быть утеряно без изменения суммы. Но есть и еще одно преимущество. У дополнения до единицы может быть два представления нуля: все нули и все единицы. Таким образом, одно значение (например, все нули) указывает, что контрольной суммы нет и дополнительное поле для этого не требуется.

Десятилетиями существовало мнение, что кадры, для которых вычисляется контрольная сумма, содержат случайные значения бит. Анализ алгоритмов вычисления контрольных сумм всегда проводился с учетом именно такого предположения. Изучение фактических данных, выполненное Партриджем и другими в 1995 году, показало, что данное предположение неверно. Следовательно, нераспознанные ошибки проскальзывают в некоторых случаях намного чаще, чем полагали раньше.

В частности, контрольная сумма для Интернета, несмотря на эффективность и простоту, в определенных ситуациях слабо защищает от ошибок именно потому, что это простая сумма. Она не позволяет распознать удаление или добавление нулевых данных, а также случаи, когда части сообщения меняются местами или расщепляются таким образом, что склеенными оказываются части двух разных пакетов. Может казаться, что подобные ошибки вряд ли произойдут в случайных процессах, но они вполне вероятны в сетях с неправильно работающим оборудованием.

Намного лучшим выбором считается **контрольная сумма Флетчера** (Fletcher, 1982). Она включает компонент, отвечающий за позицию: произведение значения данных и соответствующей позиции добавляется к текущей сумме. Это обеспечивает лучшие возможности по обнаружению изменений в положении данных.

Хотя две приведенные выше схемы в некоторых случаях могут быть приемлемыми на более высоких уровнях, на практике на канальном уровне широко используется другой, более надежный метод обнаружения ошибок — **полиномиальный код**, так же известный, как **CRC (Cyclic Redundancy Check — циклический избыточный код)**. В основе полиномиальных кодов лежит представление битовых строк в виде многочленов с коэффициентами, равными только 0 или 1. Кадр из k бит рассматривается как список коэффициентов многочлена степени $k - 1$, состоящего из k членов от x^{k-1} до x^0. Старший (самый левый) бит кадра соответствует коэффициенту при x^{k-1}, следующий бит — коэффициенту при x^{k-2} и т. д. Например, число 110001 состоит из 6 бит и, следовательно, представляется в виде многочлена пятой степени с коэффициентами 1, 1, 0, 0, 0 и 1: $x^5 + x^4 + x^0$.

С данными многочленами осуществляются арифметические действия по модулю 2 в соответствии с алгебраической теорией поля. При этом перенос при сложении и заем при вычитании не производится. И сложение, и вычитание эквивалентны исключающему ИЛИ (XOR).

10011011	00110011	1110000	01010101
$\pm$ 11001010	$+$ 11001101	$-$ 10100110	$-$ 10101111
01010001	11111110	01010110	11111010

Деление чисел осуществляется в точности так же, как и деление обычных двоичных чисел, с той разницей, что вычитание производится снова по модулю 2. Говорят, что делитель «уходит» в делимое, если в делимом столько же бит, сколько в делителе.

При использовании циклического кода отправитель и получатель должны сначала договориться насчет **образующего многочлена**, $G(x)$. Старший и младший биты образующего многочлена должны быть равны 1. Для вычисления CRC для некоторого кадра из m бит, соответствующего полиному $M(x)$, необходимо, чтобы этот кадр был длиннее образующего многочлена. Идея состоит в добавлении CRC в конец кадра таким образом, чтобы получившийся многочлен делился на образующийся многочлен $G(x)$ без остатка. Получатель, приняв кадр, содержащий контрольную сумму, пытается разделить его на $G(x)$. Ненулевой остаток от деления означает ошибку.

Алгоритм вычисления CRC при этом может быть следующим:

1. Пусть r — степень многочлена $G(x)$. Добавим r нулевых бит в конец кадра так, чтобы он содержал $m + r$ бит и соответствовал многочлену $x^rM(x)$.
2. Разделим по модулю 2 битовую строку, соответствующую многочлену $x^rM(x)$, на битовую строку, соответствующую образующему многочлену $G(x)$.
3. Вычтем по модулю 2 остаток от деления (он должен быть не более r бит) из битовой строки, соответствующей многочлену $x^rM(x)$. Результат и будет передаваемым кадром, который мы будем называть многочленом $T(x)$.

На рис. 3.9 показаны вычисления для кадра 1101011111 и образующего многочлена $G(x) = x^4 + x + 1$.

Должно быть очевидно, что многочлен $T(x)$ делится (по модулю 2) на $G(x)$ без остатка. В любом случае, если вы уменьшите делимое на остаток, то результат должен делиться без остатка. Например, в десятичной системе счисления, если разделить 210 278 на 10 941, то остаток от деления будет равен 2399. Если вычесть 2399 из 210 278, то результат (207 879) будет делиться на 10 941 без остатка.

Теперь проанализируем возможности этого метода. Какие ошибки сможет он обнаруживать? Представим, что произошла ошибка при передаче кадра, так что вместо многочлена $T(x)$ получатель принял $T(x) + E(x)$. Каждый единичный бит многочлена $E(x)$ соответствует инвертированному биту в пакете. Если в многочлене $E(x)$ k бит равны 1, это означает, что произошло k единичных ошибок. Единичный пакет ошибок характеризуется первой единицей, смесью нулей и единиц и конечной единицей, а остальные биты равны 0.

Получатель делит принятый кадр с контрольной суммой на образующий многочлен $G(x)$, то есть он вычисляет $[T(x) + E(x)]/G(x)$. $T(x)/G(x)$ равно 0, поэтому результат вычислений просто равен $E(x)/G(x)$. Ошибки, которые случайно окажутся кратными образующему многочлену $G(x)$, останутся незамеченными, остальные ошибки будут обнаружены.

Если происходит единичная ошибка, то $E(x) = x^i$, где i означает номер ошибочного бита. Если образующий многочлен $G(x)$ содержит два или более члена, то $E(x)$ никогда не будет делиться на него без остатка, поэтому будут обнаружены все единичные ошибки.

В случае двух изолированных однобитовых ошибок $E(x) = x^i + x^j$, где $i > j$, это можно также записать в виде: $E(x) = x^j(x^{i-j} + 1)$. Если предположить, что образующий многочлен $G(x)$ не делится на x, то достаточным условием обнаружения всех двойных ошибок будет неделимость на $G(x)$ многочлена $x^k + 1$ для любого k от 1 до максимального значения $i - j$, то есть до максимальной длины кадра. Известны простые

многочлены с небольшими степенями, обеспечивающие защиту для длинных кадров. Например, многочлен $x^{15} + x^{14} + 1$ не является делителем для $x^k + 1$ для любого k от 1 до 32 768.

Рис. 3.9. Пример вычисления CRC

Если ошибка затронет нечетное количество бит в кадре, многочлен $E(x)$ будет содержать нечетное число членов (например, $x^5 + x^2 + 1$, но не $x^2 + 1$). Интересно, что в системе арифметических операций по модулю 2 многочлены с нечетным числом членов не делятся на $x + 1$. Если в качестве образующего выбрать многочлен, делящийся на $x + 1$, то с его помощью можно обнаружить все ошибки, состоящие из нечетного количества инвертированных битов.

И наконец, что наиболее важно, полиномиальный код с r контрольными битами будет обнаруживать все пакеты ошибок длиной $\leqslant r$. Пакет ошибок длиной k может быть представлен в виде многочлена $x^i(x^{k-1} + ... + 1)$, где i определяет, насколько далеко от правого конца кадра располагается пакет ошибок. Если образующий многочлен $G(x)$ содержит член x^0, то x^i не будет его множителем, поэтому если степень выражения в скобках меньше степени $G(x)$, то остаток от деления никогда не будет нулевым.

Если длина пакета ошибок равна $r + 1$, то остаток от деления будет нулевым тогда и только тогда, когда пакет ошибок будет идентичен $G(x)$. По определению пакета или последовательности ошибок, его первый и последний биты должны быть равны 1, поэтому будет ли он совпадать с образующим многочленом, будет зависеть от $r - 1$ промежуточных битов. Если все комбинации считать равновероятными, то вероятность такой нераспознаваемой ошибки будет равна $(1/2)^{r-1}$.

Также можно показать, что при возникновении пакета ошибок длиннее $r + 1$ битов или нескольких коротких пакетов вероятность пропуска ошибки составляет $(1/2)^r$ при условии, что все комбинации битов равновероятны.

Некоторые образующие многочлены стали международными стандартами. Вот, например, полином, использующийся в IEEE 802 (он основан на многочлене, который первоначально предлагался для стандартов Ethernet):

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1.$$

Среди других его полезных свойств имеется и такое: этот многочлен позволяет определяться любые пакеты ошибок длиной не более 32 бит и пакеты, дающие нечетное число бит. Начиная с 1980-х годов он применяется очень широко. Тем не менее его нельзя назвать наилучшим выбором. Выполнив обстоятельные компьютерные вычисления, Кастаноли (Castagnoli и др., 1993) и Купман (Koopman, 2002) обнаружили наилучшие коды CRC. Расстояние Хэмминга, соответствующее сообщениям обычной длины, равно для них 6, в то время как у CRC-32 стандарта IEEE расстояние Хэмминга равно всего 4.

Хотя алгоритм вычисления CRC может показаться сложным, Питерсон (Peterson) и Браун (Brown) в 1961 году показали, что может быть создана простая схема для аппаратной проверки и подсчета CRC на основе сдвигового регистра. Эта схема до сих пор повсеместно применяется на практике. Десятки сетевых стандартов работают на основе кодов CRC, включая почти все локальные сети (такие как Ethernet, 802.11) и двухабонентские системы (пакеты, пересылаемые по связям SONET).

3.3. Элементарные протоколы передачи данных на канальном уровне

Знакомство с протоколами мы начнем с рассмотрения трех протоколов возрастающей сложности. Прежде чем приступить к изучению протоколов, полезно высказать некоторые допущения, лежащие в основе данной модели связи.

Для начала мы предполагаем, что на физическом, канальном и сетевом уровнях находятся независимые процессы, общающиеся с помощью передачи друг другу сообщений. Типичная реализация показана на рис. 3.10. Процессы физического уровня и часть процессов канального уровня работают на специальном оборудовании, которое называется сетевой интерфейсной картой (Network Interface Card или NIC). Остальные процессы канального уровня и процессы сетевого уровня — на центральном процессоре. Они являются частью операционной системы, причем программное обеспечение процесса канального уровня зачастую принимает форму **драйвера устройства**. Однако другие варианты реализации также возможны (например, три процесса, выполняющиеся на специальном устройстве, называемом сетевым ускорителем, или на ЦП с частотой, определяемой программно). В действительности, оптимальная реализация в каждый период развития технологий своя и зависит от имеющихся технических возможностей. В любом случае, представление трех уровней в виде отдельных процессов будет служить поддержанию концептуальной чистоты обсуждения, а также подчеркнет независимость уровней.

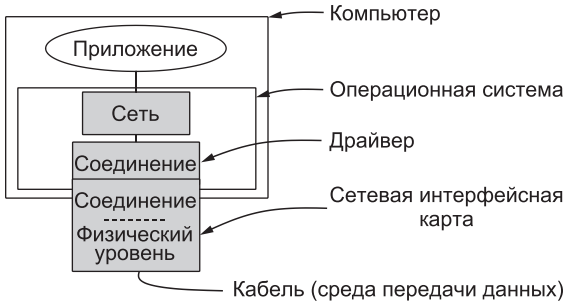


Рис. 3.10. Реализация физического, канального и сетевого уровней

Другим ключевым допущением будет то, что машина *A* хочет послать на машину *B* длинный поток данных, используя надежный, ориентированный на соединение сервис. Позднее мы рассмотрим случай, при котором одновременно машина *B* также хочет послать данные на машину *A*. Предполагается, что у машины *A* имеется бесконечный источник данных, готовых к отправке, и что ей никогда не требуется ждать готовности данных. Когда канальный уровень машины *A* запрашивает данные, сетевой уровень всегда готов их ему предоставить. (Это ограничение также будет потом отброшено.)

Также предполагается, что компьютеры не выходят из строя. При передаче могут возникать ошибки, но не проблемы, связанные с поломкой оборудования или случайной перезагрузкой.

При рассмотрении канального уровня пакет, передаваемый ему по интерфейсу сетевым уровнем, рассматривается как чистые данные, каждый бит которых должен быть доставлен сетевому уровню принимающей машины. Тот факт, что сетевой уровень принимающей машины может интерпретировать часть этого пакета как заголовок, не касается канального уровня.

Получив пакет, канальный уровень формирует из пакета кадры, добавляя заголовок и концевик (см. рис. 3.1). Таким образом, кадр состоит из внедренного пакета, некоторой служебной информации (в заголовке) и контрольной суммы (в концевике). Затем кадр передается канальному уровню принимающей машины. Мы будем предполагать наличие соответствующих библиотечных процедур, например `to_physical_layer` для отправки кадра и `from_physical_layer` для получения кадра. Эти процедуры вычисляют и добавляют или проверяют контрольную сумму (обычно это делается аппаратно), так что протоколы, о которых мы говорим в этом разделе, могут не беспокоиться об этом. Они могут применять, например, алгоритм циклических кодов, обсуждавшийся в предыдущем разделе.

Вначале получатель ничего не должен делать. Он просто сидит без дела, ожидая, что что-то произойдет. В приводимых в данной главе примерах протоколов ожидание событий уровнем передачи данных обозначается вызовом процедуры `wait_for_event(&event)`. Эта процедура возвращает управление, только когда что-то происходит (например, прибывает кадр). При этом переменная `event` сообщает, что именно случилось. Наборы возможных событий отличаются в разных протоколах и поэтому будут описываться для каждого протокола отдельно. Следует заметить, что в действительности канальный уровень не находится в холостом цикле ожидания событий, как

мы предположили, а получает прерывание, когда это событие происходит. При этом он приостанавливает свои текущие процессы и обрабатывает пришедший кадр. Тем не менее для простоты мы проигнорируем эти детали и предположим, что канальный уровень все свое время посвящает работе с одним каналом.

Листинг 3.1. Общие объявления для последующих протоколов. Объявления располагаются в файле protocol.h

```
#define MAX_PKT 1024   /* определяет размер пакета в байтах */

typedef enum {false, true} boolean;     /* тип boolean */
typedef unsigned int seq_nr;            /* порядковые номера кадров или подтверждений */
typedef struct {unsigned char data[MAX_PKT];} packet;  /* определение пакета */
typedef enum {data, ack, nak} frame_kind;  /* определение типа пакета */

typedef struct {                /* кадры, транспортируемые на данном уровне*/
  frame_kind kind;              /* тип кадра */
  seq_nr seq;                   /* порядковый номер */
  seq_nr ack;                   /* номер подтверждения */
  packet info;                  /* пакет сетевого уровня */
} frame;

/* ожидать события и вернуть тип события в переменной event */
void wait_for_event(event_type *event);

/* получить пакет у сетевого уровня для передачи по каналу */
void from_network_layer(packet *p);

/* передать информацию из полученного пакета сетевому уровню */
void to_network_layer(packet *p);

/* получить пришедший пакет у физического уровня и скопировать его в r */
void from_physical_layer(frame *r);

/* передать кадр физическому уровню для передачи */
void to_physical_layer(frame *s);

/* запустить таймер и разрешить событие timeout */
void start_timer(seq_nr k);

/* остановить таймер и запретить событие timeout */
void stop_timer(seq_nr k);

/* запустить вспомогательный таймер и разрешить событие ack_timeout */
void start_ack_timer(void);

/* остановить вспомогательный таймер и запретить событие ack_timeout */
void stop_ack_timer(void);

/* разрешить сетевому уровню инициировать событие network_layer_ready */
void enable_network_layer(void);
```

продолжение ⇩

Листинг 3.1 (*продолжение*)

```
/* запретить сетевому уровню инициировать событие network_layer_ready */
void disable_network_layer(void);

/* макрос inc развертывается прямо в строке: Циклически увеличить переменную k */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Когда принимающая машина получает кадр, контрольная сумма вычисляется заново. Если контрольная сумма в кадре неверна (то есть при передаче возникли ошибки), то канальный уровень получает соответствующую информацию (`event=cksum_err`). Если кадр прибывает в целости, канальному уровню об этом также сообщается (`event=frame_arrival`), после чего он может получить этот кадр у физического уровня с помощью процедуры `from_physical_layer`. Получив неповрежденный кадр, канальный уровень проверяет управляющую информацию, находящуюся в заголовке кадра, и если все в порядке, часть этого кадра передается сетевому уровню. Заголовок кадра не передается сетевому уровню ни при каких обстоятельствах.

Для запрета передачи сетевому уровню любой части заголовка кадра есть веская причина: поддержание полного разделения сетевого и канального уровней. До тех пор пока сетевой уровень ничего не знает о формате кадра и протоколе канального уровня, изменения формата и протокола не потребуют изменений программного обеспечения сетевого уровня. Это происходит при установке в компьютер новой сетевой карты. Поддержание строгого интерфейса между сетевым и канальным уровнями значительно упрощает разработку программ, так как протоколы различных уровней могут развиваться независимо.

В листинге 3.1 показаны некоторые объявления (на языке C), общие для многих протоколов, обсуждаемых ниже. Определены пять типов данных: `boolean`, `seq_nr`, `packet`, `frame_kind` и `frame`. Тип `boolean` представляет собой перечисляемый тип, переменные которого могут принимать значения `true` или `false`. Тип `seq_nr` является целым без знака, используемым для нумерации кадров. Эти последовательные номера могут принимать значения от 0 до числа `MAX_SEQ` включительно, которое определяется в каждом протоколе, использующем его. Тип `packet` является единицей информации, используемой при обмене информацией между сетевым и канальным уровнями одной машины или двумя равноранговыми сетевыми уровнями. В нашей модели пакет всегда состоит из `MAX_PKT` байт, хотя на практике он обычно имеет переменную длину.

Структура `frame` состоит из четырех полей: `kind`, `seq`, `ack` и `info`, первые три из которых содержат управляющую информацию, а последнее может содержать данные, которые необходимо передать. Эти три управляющих поля вместе называются **заголовком кадра (frame header)**.

Поле `kind` сообщает о наличии данных в кадре, так как некоторые протоколы отличают кадры, содержащие только управляющую информацию, от кадров, содержащих также и данные. Поля `seq` и `ack` используются соответственно для хранения последовательного номера кадра и подтверждения. Подробнее их использование будет описано ниже. Поле данных кадра, `info`, содержит один пакет. В управляющем кадре поле `info` не используется. В реальной жизни используется поле `info` переменной длины, полностью отсутствующее в управляющих кадрах.

Важно понимать взаимоотношения между пакетом и кадром. Сетевой уровень создает пакет, принимая сообщение от транспортного уровня и добавляя к нему за-

головок сетевого уровня. Этот пакет передается канальному уровню, который включает его в поле info исходящего кадра. Когда кадр прибывает на пункт назначения, канальный уровень извлекает пакет из кадра и передает его сетевому уровню. Таким образом, сетевой уровень может действовать так, как будто машины обмениваются пакетами напрямую.

В листинге 3.1 также перечислен ряд процедур. Это библиотечные процедуры, детали которых зависят от конкретной реализации, и их внутреннее устройство мы рассматривать не будем. Как уже упоминалось ранее, процедура wait_for_event представляет собой холостой цикл ожидания какого-нибудь события. Процедуры to_network_layer и from_network_layer используются канальным уровнем для передачи пакетов сетевому уровню и для получения пакетов от сетевого уровня соответственно. Обратите внимание: процедуры from_physical_layer и to_physical_layer используются для обмена кадрами между канальным и физическим уровнями, тогда как процедуры to_network_layer и from_network_layer применяются для передачи пакетов между канальным и сетевым уровнями. Другими словами, процедуры to_network_layer и from_network_layer относятся к интерфейсу между уровнями 2 и 3, тогда как процедуры from_physical_layer и to_physical_layer относятся к интерфейсу между уровнями 1 и 2.

В большинстве протоколов предполагается использование ненадежного канала, который может случайно потерять целый кадр. Для обработки подобных ситуаций передающий канальный уровень, посылая кадр, запускает таймер. Если за установленный интервал времени ответ не получен, таймер воспринимает это как тайм-аут, и канальный уровень получает сигнал прерывания.

В наших примерах протоколов этот сигнал реализован в виде значения event=timeout, возвращаемого процедурой wait_for_event. Для запуска и остановки таймера используются процедуры start_timer и stop_timer соответственно. Событие timeout может произойти, только если запущен таймер, но stop_timer еще не была вызвана. Процедуру start_timer разрешается запускать во время работающего таймера. Такой вызов просто переинициализирует часы, чтобы можно было начать отсчет заново (до нового тайм-аута, если таковой будет иметь место).

Процедуры start_ack_timer и stop_ack_timer используются для управления вспомогательными таймерами при формировании подтверждений в особых обстоятельствах.

Процедуры enable_network_layer и disable_network_layer используются в более сложных протоколах, где уже не предполагается, что у сетевого уровня всегда есть пакеты для отправки. Когда канальный уровень разрешает работу сетевого уровня, последний получает также разрешение прерывать работу первого, когда ему нужно послать пакет. Такое событие мы будем обозначать как event=network_layer_ready. Когда сетевой уровень отключен, он не может инициировать такие события. Тщательно следя за включением и выключением сетевого уровня, канальный уровень не допускает ситуации, в которой сетевой уровень заваливает его пакетами, для которых не осталось места в буфере.

Последовательные номера кадров всегда находятся в пределах от 0 до MAX_SEQ (включительно). Число MAX_SEQ различно в разных протоколах. Для увеличения последовательного номера кадров на 1 циклически (то есть с обнулением при достижении числа MAX_SEQ) используется макрос inc. Он определен в виде макроса, поскольку используется прямо в строке в тех местах программы, где быстродействие является критичным. Как мы увидим позднее в этой книге, производительность сети часто

ограничена быстродействием протоколов. Определение простых операций в виде макросов не снижает удобочитаемости программы, увеличивая при этом ее быстродействие.

Объявления в листинге 3.1 являются частью всех последующих протоколов. Для экономии места и удобства ссылок они были извлечены и собраны вместе, но, по идее, они должны быть объединены с протоколами. В языке C такое объединение производится с помощью директивы препроцессора `#include` с указанием ссылки на файл `protocol.h`, в котором помещаются данные определения.

3.3.1. Симплексный протокол «Утопия»

В качестве первого примера мы рассмотрим самый простой протокол. Данные передаются только в одном направлении, и он даже не задумывается о том, что где-то может произойти ошибка. Сетевой уровень на передающей и приемной стороне находится в состоянии постоянной готовности. Временем обработки можно пренебречь. Размер буфера неограничен. И что лучше всего, канал связи между канальными уровнями никогда не теряет и не искажает кадры. Этот совершенно нереальный протокол, который мы назовем «Утопия», показан в листинге 3.2. Он всего лишь демонстрирует базовую структуру, необходимую для построения настоящего протокола.

Протокол состоит из двух процедур, `sender1` (отправитель) и `receiver1` (получатель). Процедура `sender1` работает на канальном уровне посылающей машины, а процедура `receiver1` — на канальном уровне принимающей машины. Ни последовательные номера, ни подтверждения не используются, поэтому `MAX_SEQ` не требуется. Единственным возможным событием является `frame_arrival` (то есть прибытие неповрежденного кадра).

Процедура `sender1` представляет собой бесконечный цикл, начинающийся с оператора `while`, посылающий данные на линию с максимально возможной скоростью. Тело цикла состоит из трех действий: получения пакета (всегда обязательное) с сетевого уровня, формирования исходящего пакета с помощью переменной `s` и отсылки пакета адресату. Из служебных полей кадра данный протокол использует только поле `info`, поскольку другие поля относятся к обработке ошибок и управлению потоком, которые в данном протоколе не применяются.

Процедура принимающей стороны ничуть не сложнее. Вначале она ожидает, пока что-нибудь произойдет, причем единственно возможным событием в данном протоколе может быть получение неповрежденного пакета. Когда пакет появляется, процедура `wait_for_event` возвращает управление, при этом переменной `event` присваивается значение `frame_arrival` (которое все равно игнорируется). Обращение к процедуре `from_physical_layer` удаляет вновь прибывший кадр из аппаратного буфера и помещает его в переменную `r`. Наконец, порция данных передается сетевому уровню, а канальный уровень отправляется ждать следующий кадр.

Листинг 3.2. Неограниченный симплексный протокол «Утопия»

```
/* Протокол 1 ("Утопия") обеспечивает только одностороннюю передачу данных — от
отправителя к получателю. Предполагается, что в канале связи нет ошибок и что получатель
способен мгновенно обрабатывать получаемые данные. Соответственно, отправитель в цикле
передает данные на линию с максимально доступной для него скоростью. */
```

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender1(void)
{
  frame s;                          /* буфер для исходящего кадра */
  packet buffer;                    /* буфер для исходящего пакета */

  while (true) {
      from_network_layer(&buffer);  /* получить у сетевого уровня пакет для передачи */
      s.info = buffer;              /* скопировать его в кадр s для передачи */
      to_physical_layer(&s);        /* послать кадр s */
  }                                 /* Мы дни за днями шепчем "Завтра, завтра".
                                       Так тихими шагами жизнь ползет
                                       К последней недописанной странице.
                                       Макбет, V, v */
}

void receiver1(void)
{
  frame r;
  event_type event; /* заполняется процедурой ожидания событий, но не используется здесь
*/

  while (true) {
      wait_for_event(&event); /* единственное возможное событие — прибытие кадра, событие
frame_arrival */
      from_physical_layer(&r);      /* получить прибывший кадр */
      to_network_layer(&r.info);    /* передать данные сетевому уровню */
  }0
}
```

Протокол «Утопия» абсолютно нереалистичен, так как он не умеет ни управлять потоком данных, ни исправлять ошибки. Принцип его работы схож с сервисом без подтверждения и без установки соединения, который надеется, что все эти проблемы решаются на более высоких уровнях. Однако даже такой сервис все же обладает некоторыми способностями распознавать ошибки.

3.3.2. Симплексный протокол с ожиданием для канала без ошибок

Теперь мы отбросим самое нереальное предположение, использованное в протоколе 1, — способность получающего сетевого уровня мгновенно обрабатывать приходящие данные. Очень часто возникают ситуации, когда отправитель посылает данные слишком быстро, и получатель не успевает обработать их. Следовательно, очень важно предотвращать такие заторы. Сохраняется предположение о том, что в канале связи нет ошибок. Линия связи остается симплексной.

Одно из решений — сконструировать получатель таким образом, чтобы его мощности хватало на обработку непрерывного потока последовательных кадров (или же определить канальный уровень так, чтобы информация пересылалась достаточно медленно, не вызывая перегрузки получателя). У получателя должен быть буфер

большого объема, а его скорость должна быть не ниже скорости пересылки данных. Кроме того, он должен уметь быстро отдавать кадры сетевому уровню. Это наихудшее из возможных решений. Оно требует специального оборудования, и если линия загружена слабо, то ресурсы расходуются зря. Кроме того, он всего лишь перекладывает проблему слишком быстрой пересылки кадров на чужие плечи: в данном случае ее приходится решать сетевому уровню.

Лучшим решением данной проблемы является обратная связь со стороны получателя. Передав пакет сетевому уровню, получатель посылает небольшой управляющий пакет отправителю, разрешая тем самым посылать следующий кадр. Отправитель, отослав кадр, должен ждать разрешения на отправку следующего кадра. Подобная задержка — это простейший пример протокола с управлением потоком.

Протоколы, в которых отправитель посылает один кадр, после чего ожидает подтверждения, называются **протоколами с ожиданием (stop-and-wait)**. В листинге 3.3 приведен пример симплексного протокола с ожиданием.

Листинг 3.3. Симплексный протокол с ожиданием

```
/* Протокол 2 (с ожиданием) также обеспечивает только одностороннюю передачу данных, от
отправителя к получателю. Снова предполагается, что в канале связи нет ошибок. Однако
на этот раз емкость буфера получателя ограничена, и, кроме того, ограничена скорость
обработки данных получателем. Поэтому протокол должен не допускать отправления данных
быстрее, чем получатель способен их обработать. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;           /* буфер для исходящего кадра */
  packet buffer;     /* буфер для исходящего пакета */
  event_type event;  /* единственное возможное событие — прибытие кадра
                        (событие frame_arrival)*/

  while (true) {
      from_network_layer(&buffer);  /* получить у сетевого уровня пакет для
                                       передачи */
      s.info = buffer;              /* скопировать его в кадр s для передачи */
      to_physical_layer(&s);        /* до свиданья, кадрик, до свиданья */
      wait_for_event(&event);       /* не продолжать, пока на это не будет
                                       получено разрешения */
  }
}

void receiver2(void)
{
  frame r, s;              /* буферы для кадров */
  event_type event;        /* frame_arrival является единственным
                              возможным событием */
  while (true) {
      wait_for_event(&event); /* единственное возможное событие —
                                 прибытие кадра (событие frame_arrival)*/
```

```
        from_physical_layer(&r);      /* получить прибывший кадр */
        to_network_layer(&r.info);    /* передать данные сетевому уровню */
        to_physical_layer(&s);        /* передать пустой кадр, чтобы разбудить
                                         отправителя */
    }
}
```

Хотя пересылка данных в этом примере осуществляется по симплексному принципу, только от отправителя получателю, кадры в действительности путешествуют в обоих направлениях. Следовательно, коммуникационный канал между двумя канальными уровнями должен поддерживать пересылку информации в обе стороны. Однако данный протокол диктует строгое чередование направлений пересылки: сначала отправитель посылает кадр, затем получатель посылает кадр, потом снова отправитель, потом получатель и т. д. Для такой реализации хватило бы полудуплексного физического канала.

Как и в протоколе 1, отправитель в начале цикла работы получает пакет от сетевого уровня, формирует из него кадр и отправляет кадр по линии связи. Однако теперь, в отличие от протокола 1, отправитель должен ждать прибытия кадра с подтверждением, прежде чем он пойдет на следующую итерацию цикла и обратится к сетевому уровню за следующим пакетом. В данной модели канальный уровень отправителя даже не должен просматривать полученный по линии кадр: его содержимое не имеет значения, поскольку сам кадр означает только одно: подтверждение.

Единственное отличие процедуры `receiver2` от `receiver1` заключается в том, что после передачи пакета сетевому уровню `receiver2` посылает кадр подтверждения обратно отправителю, после чего идет на следующую итерацию цикла. Поскольку для отправителя важно только прибытие ответного кадра, а не его содержание, то получателю не нужно заполнять кадр специальной информацией.

3.3.3. Симплексный протокол с ожиданием для зашумленных каналов

Теперь рассмотрим реальную ситуацию: канал связи, в котором могут быть ошибки. Кадры могут либо портиться, либо теряться. Однако мы будем предполагать, что если кадр будет поврежден при передаче, то приемная аппаратура определит это при подсчете контрольной суммы. Если кадр будет поврежден таким образом, что контрольная сумма совпадет, что очень маловероятно, то этот протокол (и любой другой протокол) передаст неверный пакет сетевому уровню.

На первый взгляд может показаться, что нужно лишь добавить таймер к протоколу 2. Получатель будет возвращать подтверждение только в случае получения правильных данных. Неверные пакеты будут просто игнорироваться. Через некоторое время у отправителя истечет интервал времени, и он отправит кадр еще раз. Этот процесс будет повторяться до тех пор, пока кадр, наконец, не прибудет в целости.

В приведенной выше схеме имеется один критический недостаток. Прежде чем читать дальше, попытайтесь понять, что же неверно в данном алгоритме.

Чтобы осознать, чем плох данный вариант протокола, вспомните, что цель канального уровня заключается в предоставлении безошибочной прозрачной связи между

двумя процессами сетевого уровня. Сетевой уровень машины *A* передает серию пакетов своему канальному уровню, который должен гарантировать доставку идентичной серии пакетов сетевому уровню машины *B* ее канальным уровнем. В частности, сетевой уровень машины *B* не может распознать недостачу пакета или дублирование пакета, поэтому канальный уровень должен гарантировать, что дублирования пакетов не произойдет ни при каких обстоятельствах.

Рассмотрим следующий сценарий.

1. Сетевой уровень машины *A* передает пакет 1 своему канальному уровню. Пакет доставляется в целости на машину *B* и передается ее сетевому уровню. Машина *B* посылает кадр подтверждения назад на машину *A*.
2. Кадр подтверждения полностью теряется в канале связи. Он просто не попадает на машину *A*. Все было бы намного проще, если бы терялись только информационные — но не управляющие — кадры, однако канал связи, к сожалению, не делает между ними большой разницы.
3. У канального уровня машины *A* внезапно истекает отведенный интервал времени. Не получив подтверждения, он предполагает, что посланный им кадр с данными был поврежден или потерян, и посылает этот кадр еще раз.
4. Дубликат кадра в целости прибывает на канальный уровень машины *B* и передается на сетевой уровень. Если машина *A* посылала на машину *B* файл, то часть этого файла продублировалась, таким образом, копия файла на машине *B* будет неверной. Другими словами, протокол допустил ошибку.

Понятно, что необходим некий механизм, с помощью которого получатель смог бы отличать новый кадр от переданного повторно. Наиболее очевидным способом решения данной проблемы является помещение отправителем порядкового номера кадра в заголовке кадра. Тогда по номеру кадра получатель сможет понять, новый это кадр или дубликат.

Поскольку протокол должен быть нейтральным и эффективным, отводить в кадре много места под заголовок нежелательно. Возникает вопрос: каково минимальное количество бит, достаточное для порядкового номера кадра? В заголовке можно выделить 1 бит, несколько бит, 1 байт или несколько байт. Важно, чтобы порядковые номера были достаточно большими для правильной работы протокола, или же от протокола будет мало толку.

Единственная неопределенность в данном протоколе может возникнуть между кадром m и следующим за ним кадром $m + 1$. Если кадр m потерян или поврежден, получатель не подтвердит его, и отправитель пошлет его еще раз. Когда он будет успешно принят, получатель пошлет отправителю подтверждение. Именно здесь находится источник потенциальной проблемы. В зависимости от того, будет получено подтверждение или нет, отправитель может послать кадр m или кадр $m + 1$.

На стороне отправителя событием, запускающим передачу кадра $m + 1$, является прибытие подтверждения получения кадра m. Но это означает, что кадр $m - 1$ уже был отправлен и подтверждение его получения было отправлено и получено. В противном случае протокол не стал бы посылать новый кадр. Следовательно, неопределенность может возникнуть только между двумя соседними кадрами.

Таким образом, должно быть достаточно всего одного бита информации (со значением 0 или 1). В каждый момент времени получатель будет ожидать прибытия кадра

с определенным порядковым номером. Кадр с верным номером принимается, передается сетевому уровню, затем отправляется подтверждение его получения. Номер следующего ожидаемого кадра увеличивается по модулю 2 (то есть 0 становится 1, а 1 — 0). Кадр с неверным номером отбрасывается как дубликат. Однако последнее подтверждение повторяется, чтобы сообщить отправителю, что кадр получен полностью.

Пример подобного протокола приведен в листинге 3.4. Протоколы, в которых отправитель ожидает положительного подтверждения, прежде чем перейти к пересылке следующего кадра, часто называются **PAR (Positive Acknowledgement with Retransmission — положительное подтверждение с повторной передачей)** или **ARQ (Automatic Repeat reQuest — автоматический запрос повторной передачи)**. Подобно протоколу 2, он также передает данные только в одном направлении.

Листинг 3.4. Протокол с положительным подтверждением и повторной передачей

```
/* Протокол 3 (PAR) обеспечивает симплексную передачу данных по ненадежному каналу. */

#define MAX_SEQ 1                   /* в протоколе 3 должно быть равно 1 */
typedef enum  {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;        /* порядковый номер следующего исходящего
                                       кадра */
  frame s;                          /* временная переменная */
  packet buffer;                    /* буфер для исходящего пакета */
  event_type event;

  next_frame_to_send = 0;           /* инициализация исходящих последовательных
                                       номеров */
  from_network_layer(&buffer);      /* получить первый пакет у сетевого уровня */
  while (true) {
      s.info = buffer;              /* сформировать кадр для передачи */
      s.seq = next_frame_to_send;   /* вставить порядковый номер в кадр */
      to_physical_layer(&s);        /* послать кадр по каналу */
      start_timer(s.seq);           /* запустить таймер ожидания подтверждения */
      wait_for_event(&event);       /* ждать события frame_arrival, cksum_err или
                                       timeout */
      if (event == frame_arrival) {
          from_physical_layer(&s); /* получить подтверждение */
          if (s.ack == next_frame_to_send) {
              from_network_layer(&buffer); /* получить следующий пакет */
                                           /* у сетевого уровня */
              inc(next_frame_to_send);     /* инвертировать значение переменной */
                                           /* next_frame_to_send */
          }
      }
  }
}
void receiver3(void)
```

продолжение ⇩

Листинг 3.4 (*продолжение*)

```
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
      wait_for_event(&event); /* ожидание возможных событий: frame_arrival, cksum_err */
      if (event == frame_arrival) {
          /* прибыл неповрежденный кадр */
          from_physical_layer(&r);         /* получить прибывший кадр */
          if (r.seq == frame_expected) {   /* именно этот кадр и ожидался */
              to_network_layer(&r.info); /* передать данные сетевому уровню */
              inc(frame_expected);   /* в следующий раз ожидать кадр с другим */
                                     /* порядковым номером */
          }
          s.ack = 1 - frame_expected; /* номер кадра, для которого посылается */
                                      /* подтверждение */
          to_physical_layer(&s);      /* ни одно из полей не используется */
      }
  }
}
```

Протокол 3 отличается от своих предшественников тем, что и отправитель, и получатель запоминают номера кадров. Отправитель запоминает номер следующего кадра в переменной `next_frame_to_send`, а получатель запоминает порядковый номер следующего ожидаемого кадра в переменной `frame_expected`. Перед бесконечным циклом в каждой процедуре размещена короткая фаза инициализации.

Передав кадр, отправитель запускает таймер. Если он уже был запущен, он настраивается на отсчет нового полного интервала времени. Период выбирается достаточно большим, чтобы даже в худшей ситуации кадр успел дойти до получателя, получатель успел его обработать и подтверждение успело вернуться к отправителю. Только по истечении отведенного времени можно утверждать, что потерялся кадр или его подтверждение, а значит, необходимо послать дубликат. Если время, после которого наступает тайм-аут, сделать слишком коротким, то передающая машина будет повторно посылать слишком много кадров, в которых нет необходимости. Хотя лишние кадры в данном случае не повлияют на правильность приема данных, они повлияют на производительность системы.

После передачи кадра отправитель запускает таймер и ждет какого-либо события. Возможны три ситуации: либо придет неповрежденный кадр подтверждения, либо будет получен поврежденный кадр подтверждения, либо просто истечет интервал времени. В первом случае отправитель возьмет у сетевого уровня следующий пакет и положит его в буфер, поверх старого пакета. Кроме того, он увеличит порядковый номер кадра. Если же прибудет поврежденный кадр подтверждения или время истечет, то ни буфер, ни номер не будут изменены, и будет послан дубликат кадра. В любом случае затем отправляется содержимое буфера (либо следующий пакет, либо дубликат предыдущего).

Когда неповрежденный кадр прибывает к получателю, проверяется его номер. Если это не дубликат, то кадр принимается и передается сетевому уровню, после чего

формируется подтверждение. Дубликаты и поврежденные кадры на сетевой уровень не передаются, но при их получении подтверждается прибытие последнего правильного кадра, благодаря чему отправитель понимает, что нужно перейти к следующему кадру или повторить пересылку поврежденного.

3.4. Протоколы скользящего окна

В предыдущих протоколах информационные кадры передавались только в одну сторону. В большинстве практических ситуаций требуется передача данных в обоих направлениях. Один из способов получения дуплексной передачи — использование двух экземпляров описанных выше протоколов, каждый из которых передает данные по отдельной симплексной связи (в противоположных направлениях). При этом физический канал имеет прямой канал для данных и обратный канал для подтверждений. В обоих случаях пропускная способность обратных каналов почти не используется.

Более прогрессивной идеей представляется использование одного канала для передачи данных в обоих направлениях. В конце концов, ведь в протоколах 2 и 3 кадры уже передавались по каналу в двух направлениях, а обратный канал обладает той же пропускной способностью, что и прямой. В такой модели кадры с данными от машины *A* для машины *B* перемешиваются с кадрами подтверждений от *A* к *B*. Получатель может отличить кадр с данными от кадра с подтверждением по специальному полю `kind` заголовка кадра.

Помимо чередования кадров с подтверждениями и информационных кадров, возможно и другое улучшение протокола. Приняв кадр с данными, получатель может не посылать сразу кадр с подтверждением, а подождать, пока сетевой уровень даст ему следующий пакет. Подтверждение добавляется к исходящему информационному кадру с помощью поля `ack` заголовка кадра. В результате для передачи подтверждения почти не будет затрачено ресурсов. Подобная техника называется **piggybacking** (комбинированная или ярусная перевозка).

Основное преимущество совмещения передачи прямых и обратных пакетов заключается в улучшенном использовании пропускной способности канала. Поле `ack` в заголовке кадра занимает всего несколько бит, тогда как отдельный кадр потребует заголовка и контрольной суммы. Кроме того, чем меньше количество прибывающих кадров, тем меньше нагрузка на получателя. В следующем рассматриваемом нами протоколе расходы на совмещение передачи прямых и обратных пакетов составляют всего 1 бит заголовка кадра. Эти расходы редко превышают несколько бит.

Однако при совмещении передачи прямых и обратных пакетов в протоколе появляются новые проблемы. Как долго должен канальный уровень ждать пакета, с которым следует переслать подтверждение? Если канальный уровень будет ждать дольше, чем отправитель, то последний пошлет кадр повторно, что неприемлемо. Если бы канальный уровень мог предсказывать будущее, он бы знал, ждать ему пакета или отправлять подтверждение отдельным кадром. Это, конечно, невозможно, поэтому следует установить еще один интервал ожидания (меньший, чем интервал ожидания отправителя), по истечении которого подтверждение отправляется отдельным кадром. Если же сетевой уровень успеет передать уровню передачи данных пакет, то подтверждение будет отослано вместе с этим пакетом в одном кадре.

Следующие три протокола являются двунаправленными и принадлежат к классу протоколов **скользящего окна (sliding window)**. Как будет показано ниже, они отличаются друг от друга эффективностью, сложностью и требованиями к размерам буфера. Во всех протоколах скользящего окна каждый исходящий кадр содержит порядковый номер (варьирующийся от 0 до некоего максимума). Поскольку на этот номер обычно отводится поле размером n бит, максимальное значение номера составляет $2^n - 1$. В протоколах скользящего окна с ожиданием обычно на это поле отводится всего один бит, что ограничивает порядковый номер значениями 0 и 1, однако в более сложных версиях может использоваться произвольное значение n.

Сущность всех протоколов скользящего окна заключается в том, что в любой момент времени отправитель работает с определенным набором порядковых номеров, соответствующих кадрам, которые ему разрешено посылать. Про такие кадры говорят, что они попадают в **посылающее окно**. Аналогично получатель работает с **принимающим окном**, соответствующим набору кадров, которые ему позволяется принять. Окно получателя и окно отправителя не обязаны иметь одинаковые нижний и верхний пределы и даже не обязаны быть одного размера. В одних протоколах размеры фиксируются, а в других размеры могут увеличиваться или уменьшаться по мере передачи или приема кадров.

Хотя данные протоколы предоставляют канальному уровню большую свободу в вопросе, касающемся порядка передачи и приема кадров, требование доставки пакетов сетевому уровню принимающей машины в том же порядке, в котором они были получены от сетевого уровня передающей машины, сохраняется. Также сохраняется аналогичное требование к физическому уровню, касающееся сохранения порядка доставки кадров. То есть физический уровень должен функционировать подобно проводу, доставляя все кадры в том порядке, в котором они были посланы.

Порядковые номера в окне отправителя соответствуют кадрам, которые уже отправлены, но на которые еще не получены подтверждения. Получаемому от сетевого уровня пакету дается наибольший порядковый номер, и верхняя граница окна увеличивается на единицу. Когда поступает подтверждение, на единицу возрастает нижняя граница окна. Таким образом, окно постоянно содержит список неподтвержденных кадров.

Так как кадры, находящиеся в окне отправителя, могут быть потеряны или повреждены во время передачи, отправитель должен хранить их в памяти на случай возможной повторной передачи. Таким образом, если максимальный размер окна равен n, то отправителю потребуется n буферов для хранения неподтвержденных кадров. Если окно достигает максимального размера, канальный уровень должен отключить сетевой уровень до тех пор, пока не освободится буфер.

Окно принимающего канального уровня соответствует кадрам, которые он может принять. Любой кадр, попадающий в окно, помещается в буфер получателя. Когда прибывает кадр с порядковым номером, соответствующим нижнему краю окна, он передается на сетевой уровень, и окно сдвигается на одну позицию. Любой кадр, не попадающий в окно, удаляется. В любом случае формируется подтверждение, для того чтобы отправитель мог понять, как ему действовать дальше. Обратите внимание, что окно единичного размера говорит о том, что канальный уровень может принимать кадры только в установленном порядке, однако при больших размерах окна это не так. Сетевому уровню, напротив, данные всегда предоставляются в строгом порядке, независимо от размера окна уровня передачи данных.

На рис. 3.11 показан пример для окна с максимальным размером 1. Вначале кадров в окне нет, поэтому само окно пустое и его верхний и нижний края совпадают, однако с течением времени ситуация меняется. В отличие от окна отправителя, окно получателя всегда сохраняет первоначальный размер, поворачиваясь по мере приема и передачи на сетевой уровень очередного кадра.

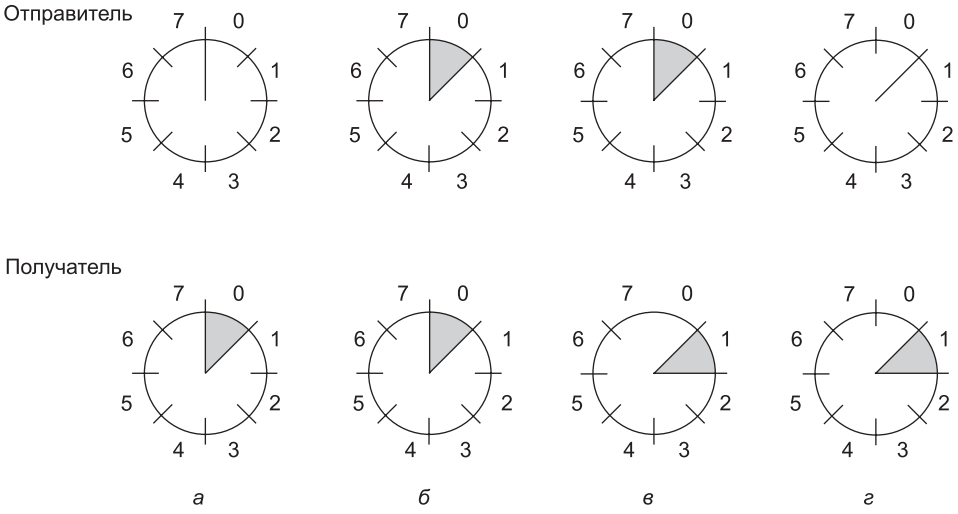


Рис. 3.11. Скользящее окно размера 1 с 3-битовым порядковым номером: *а* — начальная ситуация; *б* — после отправки первого кадра; *в* — после приема первого кадра; *г* — после приема первого подтверждения

3.4.1. Протокол однобитового скользящего окна

Прежде чем рассматривать общий случай, изучим протокол скользящего окна с максимальным размером окна, равным 1. Такой протокол использует метод ожидания, поскольку, отослав кадр, отправитель, прежде чем послать следующий кадр, должен дождаться подтверждения.

Данный протокол приведен в листинге 3.5. Как и другие протоколы, он начинается с определения некоторых переменных. Переменная `next_frame_to_send` содержит номер кадра, который отправитель пытается послать. Аналогично переменная `frame_expected` хранит номер кадра, ожидаемого получателем. В обоих случаях, возможными значениями могут быть только 0 и 1.

Листинг 3.5. 1-битовый протокол скользящего окна

```
/* Протокол 4 (скользящее окно) является дуплексным. */

#define MAX_SEQ 1                    /* в протоколе 4 должно быть равно 1 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void protocol4 (void)
```

продолжение ⇩

Листинг 3.5 (*продолжение*)

```
{
  seq_nr next_frame_to_send;       /* только 0 или 1 */
  seq_nr frame_expected;           /* только 0 или 1 */
  frame r, s;                      /* временная переменная */
  packet buffer;                   /* текущий посланный пакет */
  event_type event;

  next_frame_to_send = 0;          /* номер следующего кадра в исходящем потоке */
  frame_expected = 0;              /* номер ожидаемого кадра */
  from_network_layer(&buffer);     /* получить первый пакет у сетевого уровня */
  s.info = buffer;                 /* подготовить первый кадр для передачи */
  s.seq = next_frame_to_send;      /* вставить порядковый номер в кадр */
  s.ack = 1 - frame_expected;      /* подтверждение, посылаемое "верхом" на кадре данных
*/
  to_physical_layer(&s);           /* послать кадр по каналу */
  start_timer(s.seq);              /* запустить таймер ожидания подтверждения */

  while (true) {
      wait_for_event(&event);       /* ждать возможного события: frame_arrival, */
                                    /* cksum_err или timeout */
      if (event == frame_arrival) { /* кадр прибыл в целости */
          from_physical_layer(&r);  /* получить кадр */

          if (r.seq == frame_expected) {
              /* обработать входящий поток кадров */
              to_network_layer(&r.info);  /* передать пакет сетевому уровню */
              inc(frame_expected);  /* инвертировать порядковый номер кадра, */
                                    /* ожидаемого в следующий раз */
          }

          if (r.ack == next_frame_to_send) { /* обработать исходящий поток кадров */
              from_network_layer(&buffer);   /* получить следующий пакет */
                                             /* у сетевого уровня */
              inc(next_frame_to_send);   /* инвертировать порядковый номер */
                                         /* посылаемого кадра */
          }
      }

      s.info = buffer;                /* подготовить кадр для передачи */
      s.seq = next_frame_to_send;     /* вставить порядковый номер в кадр */
      s.ack = 1 - frame_expected;     /* порядковый номер последнего полученного кадра */
      to_physical_layer(&s);          /* передать кадр */
      start_timer(s.seq);             /* запустить таймер ожидания подтверждения */
  }
}
```

В нормальной ситуации только один канальный уровень может начинать передачу. Другими словами, только одна из программ должна содержать обращения к процедурам `to_physical_layer` и `start_timer` вне основного цикла. Начинающая машина получает первый пакет от своего сетевого уровня, создает из него кадр и посылает его.

Когда этот (или другой) кадр прибывает, получающий канальный уровень проверяет, не является ли этот кадр дубликатом, аналогично протоколу 3. Если это тот кадр, который ожидался, он передается сетевому уровню, и окно получателя сдвигается вверх.

Поле подтверждения содержит номер последнего полученного без ошибок кадра. Если этот номер совпадает с номером кадра, который пытается передать отправитель, последний понимает, что этот кадр успешно принят получателем и что он может пересылать следующий кадр. В противном случае он должен продолжать попытки переслать тот же кадр.

Теперь давайте изучим протокол 4 и посмотрим, насколько он устойчив к нестандартным ситуациям. Предположим, что машина *A* пытается послать кадр 0 машине *B*, а машина *B* пытается послать кадр 0 машине *A*. Предположим также, что на машине *A* установлен слишком короткий период ожидания подтверждения. Соответственно, машина *A* посылает серию одинаковых кадров со значениями полей `seq`=0 и `ack`=1.

Когда первый неповрежденный кадр прибудет на машину *B*, он будет принят, и значение переменной `frame_expected` будет установлено равным 1. Все последующие входящие кадры будут проигнорированы, поскольку машина *B* будет теперь ожидать кадр с порядковым номером 1, а не 0. Более того, поскольку у всех кадров дубликатов значение поля `ack`=1, а машина *B* продолжает ожидать подтверждения для кадра 0, то *B* и не станет запрашивать новый пакет у своего сетевого уровня.

В ответ на каждый отвергнутый дубликат, присылаемый машиной *A*, машина *B* посылает кадр, содержащий поля `seq`=0 и `ack`=0. Наконец, один из этих кадров принимается машиной *A*, в результате чего машина *A* переходит к передаче следующего пакета. Никакая комбинация потерянных кадров или преждевременно истекших интервалов ожидания не может заставить этот протокол ни выдать сетевому уровню дубликат пакета, ни пропустить пакет, ни зависнуть. Протокол работает корректно.

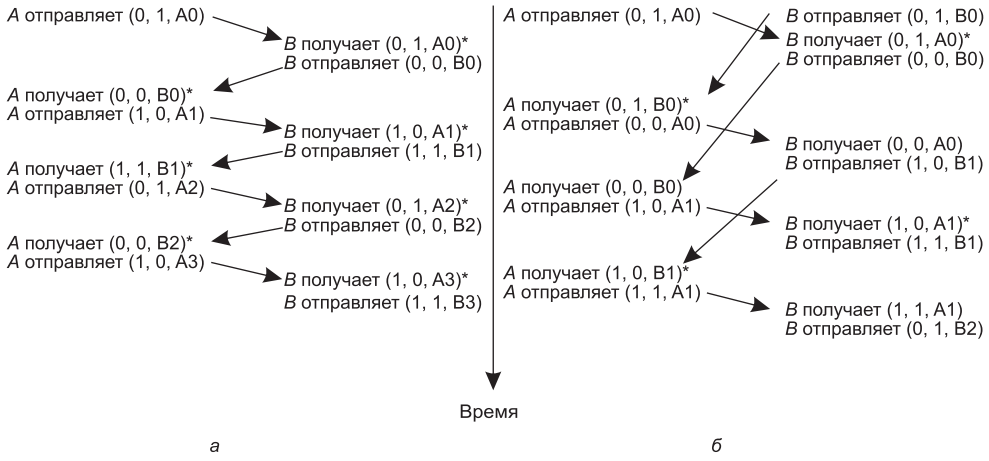


Рис. 3.12. Два сценария для протокола 4: *а* — нормальная ситуация; *б* — нештатная ситуация. Нотация: (seq, ack, номер пакета). Звездочка означает, что сетевой уровень принял пакет

Однако отношения между протоколами могут быть довольно непростыми. Если обе стороны одновременно вышлют друг другу начальный пакет, возникает запутанная ситуация, проиллюстрированная на рис. 3.12. В левой части рисунка показано нормальное функционирование протокола. Правая часть рисунка демонстрирует аномальную ситуацию. Если машина *B* ожидает первого кадра от машины *A*, прежде чем послать свой первый кадр, то последовательность будет такой, как показана в левой части рисунка. При этом принимается каждый кадр.

Однако если машины *A* и *B* одновременно начнут передачу, их первые кадры пересекутся, и канальные уровни попадут в ситуацию *б*. В ситуации *а* с каждым кадром прибывал новый пакет, и дубликатов нет. В ситуации *б* половина кадров содержит дубликаты, несмотря на то что ошибок в канале связи не было. Подобные ситуации могут возникнуть в результате преждевременного истечения периода ожидания, даже если одна сторона начнет диалог первой. В самом деле, если время ожидания истечет слишком быстро, кадр может быть послан три и более раз, приводя к ненужной трате ценных ресурсов.

3.4.2. Протокол с возвратом на n

До сих пор мы по умолчанию подразумевали, что время, необходимое на передачу кадра от отправителя к получателю, и время, необходимое на передачу подтверждения от получателя к отправителю, пренебрежимо мало. Иногда это предположение является совершенно неверным. В таких ситуациях большое время прохождения кадров по сети может значительно снижать эффективность использования пропускной способности канала. В качестве примера рассмотрим спутниковый канал связи с пропускной способностью 50 Кбит/с и временем, требуемым для прохождения сигнала в оба конца, равным 500 мс. Попытаемся использовать протокол 4 для пересылки кадров размером в 1000 бит через спутник. В момент времени $t = 0$ отправитель начинает посылать первый кадр. В момент времени $t = 20$ мс кадр полностью послан. В момент времени $t = 270$ мс получатель принял кадр полностью и отправил обратно подтверждение. В итоге в лучшем случае только через 520 мс после начала передачи кадра подтверждение будет получено отправителем. В данном случае еще предполагается, что приемник не тратит времени на обработку принятого кадра и подтверждение такое короткое, что временем его передачи и приема можно пренебречь. Это означает, что передающая машина была заблокирована в течение 500/520, или 96 % времени. Другими словами, использовалось только 4 % доступной пропускной способности. Очевидно, что сочетание большого времени прохождения сигнала, высокой пропускной способности и коротких кадров совершенно неприемлемо с точки зрения эффективности.

Описанная выше проблема является следствием правила, заставлявшего отправителя дожидаться подтверждения, прежде чем посылать следующий кадр. Смягчив это требование, можно значительно повысить эффективность. Решение проблемы заключается в разрешении отправителю послать не один кадр, а несколько, например w, прежде чем остановиться и перейти в режим ожидания подтверждений. Если подобрать достаточно большое число w, то отправитель сможет безостановочно посылать кадры, так как подтверждения для предыдущих кадров будут прибывать до того, как окно заполнится, и блокировка отправителя производиться не будет.

Для того чтобы найти подходящее значение w, необходимо понять, сколько кадров «вмещается» в канал, в то время как они путешествуют от отправителя к получателю. Емкость определяется путем умножения полосы пропускания в битах в секунду на время пересылки в одну сторону. Это значение можно разделить на число бит в кадре, чтобы выразить количество кадров. Назовем это количество BD (**bandwidth-delay product**, то есть полоса пропускания, умноженная на задержку). Следовательно, значение w нужно выбрать как $2BD + 1$. $2BD$ — это число кадров, которое может находиться в пути (неподтвержденные кадры), если отправитель отправляет кадры непрерывно (двойка обозначает, что мы также учитываем время, необходимое на получение подтверждения). Единица прибавляется, так как кадр подтверждения отправляется только после получения полного кадра данных.

Возьмем в качестве примера канал с полосой пропускания 50 Кбит/с, в котором на пересылку кадра в одну сторону тратится 250 мс. Значение BD равно 12,5 Кбит/с или 12,5 кадра, каждый из которых включает 1000 бит. $2BD + 1$ равно 26 кадрам. Отправитель начинает, как и ранее, с передачи кадра 0 и отправляет очередной кадр каждые 20 мс. К тому моменту, когда он закончит отправку 26 кадров (в момент времени $t = 520$ мс), как раз прибудет подтверждение кадра 0. Затем подтверждения станут прибывать каждые 20 мс. Таким образом, отправитель будет получать разрешения на передачу следующего кадра как раз вовремя. Начиная с этого момента у отправителя будет 25 или 26 неподтвержденных кадров и, следовательно, достаточно будет окна размером 26.

Если размер окна невелик, то канал будет загружен не на 100 %, так как иногда отправитель будет блокироваться. Загрузку можно выразить как долю времени, когда отправитель не заблокирован:

$$\textit{загрузка канала} \leq \frac{w}{1 + 2BD}.$$

Это значение выражает верхнюю границу, так как не учитывает время на обработку кадра. Также считается, что длина кадра подтверждения равна нулю — обычно они действительно короткие. Из этого неравенства понятно, что для больших значений BD необходимо выбирать большое значение размера окна w. Если задержка большая, то отправитель быстро опустошит свое окно даже при средней полосе пропускания, как в примере со спутником. Если полоса пропускания широкая, то даже при средней задержке отправитель быстро опустошит окно, если только оно не отличается особо крупным размером (например, канал с пропусной способностью 1 Гбит/с и задержкой в 1 мс удерживает 1 Мбит). Если у протокола с ожиданием значение $w = 1$, то даже если задержка распространения равна всего одному кадру, его эффективность уже падает ниже 50 %.

Такая техника, когда в пути находится сразу несколько кадров, называется **конвейерной обработкой (pipelining)**. При конвейерном режиме передачи кадров по ненадежному каналу возникает ряд серьезных проблем. Во-первых, что произойдет, если повредится или потеряется кадр в середине длинного потока? Большое количество последующих кадров прибудет к получателю прежде, чем отправитель обнаружит, что произошла ошибка. Когда поврежденный кадр приходит к получателю, он, конечно, должен быть отвергнут, однако что должен делать получатель со всеми правильными

последующими кадрами? Как уже говорилось, получающий канальный уровень обязан передавать пакеты сетевому уровню, соблюдая строгий порядок.

Существует два базовых подхода к исправлению ошибок при конвейерной обработке. Они проиллюстрированы на рис. 3.13.

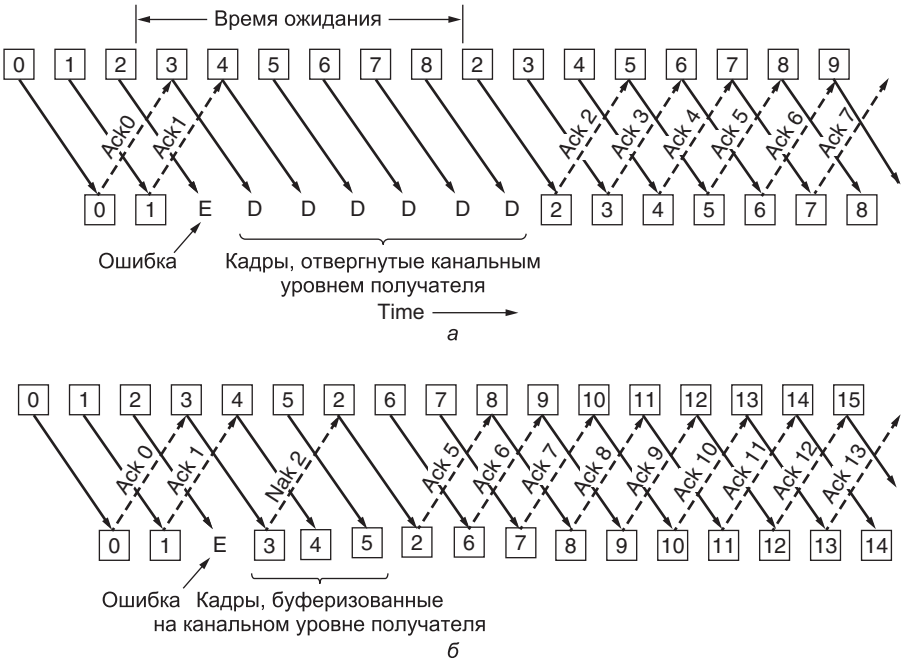


Рис. 3.13. Конвейеризация и коррекция ошибок: *а* — эффект при размере окна 1; *б* — эффект при размере окна больше 1

Первый способ называется **возвратом на n (go-back-n)** и заключается просто в игнорировании всех кадров, следующих за ошибочным. Для таких кадров подтверждения не посылаются. Эта стратегия соответствует приемному окну размера 1. Другими словами, канальный уровень отказывается принимать какой-либо кадр, кроме кадра со следующим номером, который он должен передать сетевому уровню. Если окно отправителя заполнится раньше, чем истечет период времени ожидания, конвейер начнет простаивать. Наконец, лимит времени у отправителя истечет, и он начнет передавать повторно сразу все кадры, не получившие подтверждения, начиная с поврежденного или потерянного кадра. Такой подход при высоком уровне ошибок может привести к потере большой доли пропускной способности канала.

На рис. 3.13, *б* изображен возврат на *n* для случая большого окна приемника. Кадры 0 и 1 корректно принимаются, и высылается подтверждение этого факта. Однако кадр 2 потерялся или был испорчен. Ничего не подозревающий отправитель продолжает посылать кадры, пока не выйдет время ожидания кадра 2. Только после этого он возвращается к месту сбоя и заново передает все кадры, начиная с кадра 2 (отправляя 2, 3, 4 и т. д.).

Другая общая стратегия обработки ошибок при конвейерной передаче кадров, называемая **выборочным повтором (selective repeat)**, заключается в том, что полу-

чатель хранит в буфере все правильные кадры, принятые им после неверного или потерянного кадра. При этом неверный кадр отбрасывается. Когда заканчивается время ожидания подтверждения, отправитель отправляет еще раз только самый старый кадр, для которого не пришло подтверждение. Если вторая попытка будет успешной, получатель сможет последовательно передать накопившиеся пакеты сетевому уровню. Выборочный повтор используется, когда размер окна получателя больше 1. Он может потребовать, чтобы канальному уровню получателя было доступно большое количество памяти.

Выборочный повтор часто комбинируется с отправкой получателем «отрицательного подтверждения» **(NAK — Negative Acknowledgement)** при обнаружении ошибки, например, при неверной контрольной сумме или при измененном порядке следования кадров. NAK стимулируют повторную отправку еще до того, как закончится время ожидания подтверждения от отправителя. Таким образом, эффективность работы несколько повышается.

На рис. 3.13, *б* кадры 0 и 1 снова принимаются корректно, а кадр 2 теряется. После получения кадра 3 канальный уровень приемника замечает, что один кадр выпал из последовательности. Для кадра 2 отправителю посылается NAK, однако кадр 3 сохраняется в специальном буфере. Далее прибывают кадры 4 и 5, они также буферизируются канальным уровнем вместо передачи на сетевой уровень. NAK 2 приходит к отправителю, заставляя его переслать кадр 2. Когда последний оказывается у получателя, у уровня передачи данных уже имеются кадры 2, 3, 4 и 5, которые сразу же в нужном порядке отдаются сетевому уровню. Теперь уже можно выслать подтверждение получения всех кадров, включая пятый, что и показано на рисунке. Если NAK вдруг потеряется, то отправитель по окончании времени ожидания 2 сам отправит кадр 2 (и только его!), однако это может произойти значительно позже, чем при помощи NAK.

Листинг 3.6. Протокол скользящего окна с возвратом на n

```
/* Протокол 5 (конвейерный) допускает наличие нескольких неподтвержденных кадров.
Отправитель может передать до MAX_SEQ кадров, не ожидая подтверждения. Кроме того,
в отличие от предыдущих протоколов, не предполагается, что у сетевого уровня всегда есть
новые пакеты. При появлении нового пакета сетевой уровень инициирует событие network_
layer_ready */

#define MAX_SEQ 7                        /* должно быть 2^n-1 */
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* возвращает true, если (a <=b < c циклично; иначе false */
  if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
      return(true);
  else
      return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
```

продолжение ⇩

Листинг 3.6 (*продолжение*)

```
{
/* подготовить и послать информационный кадр */
  frame s;                                /* временная переменная */

  s.info = buffer[frame_nr];              /* вставить пакет в кадр */
  s.seq = frame_nr;                       /* вставить порядковый номер в кадр */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);  /* подтверждение,
                                          посылаемое "верхом" на кадре данных */
  to_physical_layer(&s);                  /* послать кадр по каналу */
  start_timer(frame_nr);                  /* запустить таймер ожидания
                                          подтверждения */
}

void protocol5(void)
{
  seq_nr next_frame_to_send;       /* MAX_SEQ > 1; используется для исходящего потока */
  seq_nr ack_expected;             /* самый старый неподтвержденный кадр */
  seq_nr frame_expected;           /* следующий кадр, ожидаемый во входящем потоке */
  frame r;                         /* временная переменная */
  packet buffer[MAX_SEQ+1];        /* буферы для исходящего потока */
  seq_nr nbuffered;     /* количество использующихся в данный момент выходных буферов */
  seq_nr i;                        /* индекс массива буферов */
  event_type event;

  enable_network_layer();      /* разрешить события network_layer_ready */
  ack_expected = 0;            /* номер следующего ожидаемого входящего подтверждения */
  next_frame_to_send = 0;      /* номер следующего посылаемого кадра */
  frame_expected = 0;          /* номер следующего ожидаемого входящего кадра */
  nbuffered = 0;               /* вначале буфер пуст */

  while (true) {
      wait_for_event(&event);  /* четыре возможных события: см. event_type выше */

      switch(event) {
          case network_layer_ready:  /* у сетевого уровня есть пакет
                                        для передачи */
              /* получить, сохранить и передать новый кадр */
              from_network_layer(&buffer[next_frame_to_send]); /* получить новый
                                                        пакет у сетевого уровня */
              nbuffered = nbuffered + 1;  /* увеличить окно отправителя */
              send_data(next_frame_to_send, frame_expected, buffer);  /* передать
                                                                    кадр */
              inc(next_frame_to_send);  /* увеличить верхний край окна
                                           отправителя */
              break;

          case frame_arrival:      /* прибыл кадр с данными или с подтверждением */
              from_physical_layer(&r);  /* получить пришедший кадр у физического
                                           уровня */

              if (r.seq == frame_expected) {
                  /* кадры принимаются только по порядку номеров */
```

```
                to_network_layer(&r.info);  /* передать пакет сетевому уровню */
                inc(frame_expected);        /* передвинуть нижний край окна
                получателя */
            }

            /* подтверждение для кадра n подразумевает также кадры
               n - 1, n - 2 и т. д. */
            while (between(ack_expected, r.ack, next_frame_to_send)) {
                /* отправить подтверждение вместе с информационным кадром */
                nbuffered = nbuffered - 1; /* в буфере на один кадр меньше */
                stop_timer(ack_expected);   /* кадр прибыл в целости;
                                               остановить таймер */
                inc(ack_expected);          /* уменьшить окно отправителя */
            }
            break;
        case cksum_err: break;              /* плохие кадры просто игнорируются */
        case timeout:  /* время истекло; передать повторно все
                          неподтвержденные кадры */
            next_frame_to_send = ack_expected;  /* номер первого посылаемого
                                                   повторно кадра */
            for (i = 1; i <= nbuffered; i++) {
                send_data(next_frame_to_send, frame_expected, buffer);
                                                /* переслать повторно 1 кадр */
                inc(next_frame_to_send);  /* приготовиться к пересылке
                                             следующего кадра */
            }
    }

    if (nbuffered < MAX_SEQ)
        enable_network_layer();
    else
        disable_network_layer();
  }
}
```

Выбор одной из двух приведенных выше стратегий является вопросом компромисса между эффективным использованием пропускной способности и размером буфера канального уровня. В зависимости от того, что в конкретной ситуации является более критичным, может использоваться тот или иной метод. В листинге 3.6 показан конвейерный протокол, в котором принимающий уровень передачи данных принимает кадры по порядку. Все кадры, следующие за ошибочным, игнорируются. В данном протоколе мы впервые отказались от предположения, что у сетевого уровня всегда есть неограниченное количество пакетов для отсылки. Когда у сетевого уровня появляется пакет, который он хочет отправить, уровень инициирует событие `network_layer_ready`. Однако чтобы управлять потоком на основе размера окна отправителя и числа неподтвержденных кадров в любой момент времени, канальный уровень должен иметь возможность отключать на время сетевой уровень. Для этой цели служит пара библиотечных процедур — `enable_network_layer` и `disable_network_layer`.

Максимальное число неподтвержденных кадров в любой момент времени не совпадает с размером пространства порядковых номеров. Для протокола с возвратом на

n неподтвержденных кадров в любой момент времени может быть MAX_SEQ, хотя различаются MAX_SEQ + 1 порядковых номеров: от 0 до MAX_SEQ. В следующем протоколе, с выборочным повтором, мы увидим еще более жесткое ограничение. Чтобы понять, почему необходимо такое ограничение, рассмотрим сценарий с MAX_SEQ = 7.

1. Отправитель посылает кадры с 0 по 7.
2. Подтверждение для кадра 7 прибывает к отправителю.
3. Отправитель посылает следующие восемь кадров, снова с номерами с 0 по 7.
4. Еще одно подтверждение для кадра 7 прибывает к отправителю.

Но вот вопрос: все восемь кадров, входящих во второй набор, благополучно дошли до адресата, или все они потерялись (включая игнорированные кадры после ошибочного)? В обоих случаях получатель отправит кадр 7 в качестве подтверждения. У отправителя нет способа отличить один случай от другого. По этой причине максимальное количество неподтвержденных кадров должно быть ограничено числом MAX_SEQ.

Хотя в протоколе 5 кадры, поступившие после ошибки, не буферизируются получателем, тем не менее отправитель должен хранить отправленные кадры в своем буфере, пока не получит на них подтверждение. Если поступает подтверждение на кадр n, кадры $n - 1$, $n - 2$ (то есть все предыдущие кадры) автоматически считаются подтвержденными. Такой тип подтверждения называется **кумулятивным (cumulative acknowledgement)**. Эта особенность наиболее важна в случае потери или повреждения какого-либо кадра с подтверждением. Получив подтверждение, канальный уровень проверяет, не освободился ли у него какой-нибудь буфер. Если буфер освобождается, то заблокированному ранее сетевому уровню можно снова разрешить инициировать события network_layer_ready.

Для этого протокола предполагается, что всегда есть обратный трафик, по которому можно отправлять подтверждение. Протокол 4 не нуждается в подобном допущении, поскольку за каждым принятым кадром следует обратный, даже если уже был отправлен какой-то кадр в сторону отправителя. В следующем протоколе проблема отсутствия обратного трафика будет решена гораздо элегантней.

Поскольку протокол 5 хранит несколько неподтвержденных кадров, ему требуется несколько таймеров, по одному на кадр. Для каждого кадра время считается независимо от других. Однако все таймеры могут симулироваться программно, используя единственные аппаратные часы, вызывающие периодические прерывания. Данные таймеров могут храниться в программе в виде связанного списка, каждый узел которого будет хранить количество временных интервалов системных часов, оставшихся до истечения срока ожидания, номер кадра и указатель на следующий узел списка.

В качестве иллюстрации приводится рис. 3.14, на котором показана программная реализация нескольких таймеров. Предположим, что часы изменяют свое состояние каждые 1 мс. Пусть начальное значение реального времени будет 10:00:00.000 и имеются три таймера тайм-аутов, установленные на время 10:00:00.005, 10:00:00.013 и 10:00:00.019. Каждый раз, когда аппаратные часы изменяют свое значение, реальное время обновляется и счетчик этих изменений в голове списка уменьшается на единицу. Когда значение счетчика становится равным нулю, инициируется тайм-аут, а узел удаляется из списка, как показано на рис. 3.14, *б*. Такая организация таймеров не требует выполнения большой работы при каждом прерывании от системных часов,

хотя при работе процедур start_timer и stop_timer требуется сканирование списка. В протоколе у данных процедур имеется входной параметр, означающий номер кадра, таймер которого нужно запустить или остановить.

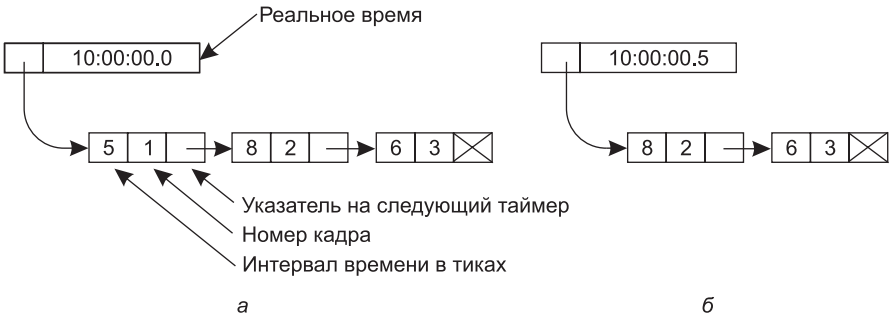


Рис. 3.14. Программная симуляция работы нескольких таймеров

3.4.3. Протокол с выборочным повтором

Протокол с возвратом на *n* хорошо работает, если ошибки встречаются нечасто, однако при плохой линии он тратит впустую много времени и ресурсов, передавая кадры по два раза. В качестве альтернативы можно использовать протокол с выборочным повтором, который позволяет получателю принимать и буферизировать кадры, переданные после поврежденного или утерянного кадра.

В этом протоколе и отправитель, и получатель работают с окнами неподтвержденных и допустимых номеров кадров соответственно. Размер окна отправителя начинается с нуля и растет до некоторого определенного уровня. Размер окна получателя, напротив, всегда фиксированного размера. Получатель должен иметь буфер для каждого кадра, номер которого находится в пределах окна. С каждым буфером связан бит, показывающий, занят буфер или свободен. Когда прибывает кадр, функция between проверяет, попал ли его порядковый номер в окно. Если да, то кадр принимается и хранится в буфере. Это действие производится независимо от того, является ли данный кадр следующим кадром, ожидаемым сетевым уровнем. Он должен храниться на канальном уровне до тех пор, пока все предыдущие кадры не будут переданы сетевому уровню в правильном порядке. Алгоритм протокола показан в листинге 3.7.

Листинг 3.7. Протокол скользящего окна с выборочным повтором

```
/* Протокол 6 (выборочный повтор) принимает кадры в любом порядке, но передает их сетевому
уровню, соблюдая порядок. С каждым неподтвержденным кадром связан таймер. При срабатывании
таймера передается повторно только этот кадр, а не все неподтвержденные кадры, как
в протоколе 5. */

#define MAX_SEQ 7                          /* должно быть 2^n-1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_
type;
```

продолжение ⇛

Листинг 3.7 (*продолжение*)

```
#include "protocol.h"
boolean no_nak = true;          /* отрицательное подтверждение (nak) еще не посылалось */
seq_nr oldest_frame = MAX_SEQ+1;          /* начальное значение для симулятора */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* то же, что и в протоколе 5, но короче и понятнее */
  return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet
buffer[])
{
/* сформировать и послать данные, а также положительное или отрицательное подтверждение */
  frame s;                                /* временная переменная */

  s.kind = fk;                            /* kind == data, ack, или nak */
  if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
  s.seq = frame_nr;                /* имеет значение только для информационных кадров */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
  if (fk == nak) no_nak = false;   /* один nak на кадр, пожалуйста */
  to_physical_layer(&s);           /* переслать кадр */
  if (fk == data) start_timer(frame_nr % NR_BUFS);
  stop_ack_timer();                /* отдельный кадр с подтверждением не нужен */
}

void protocol6(void)
{
  seq_nr ack_expected;                    /* нижний край окна отправителя */
  seq_nr next_frame_to_send;              /* верхний край окна отправителя + 1 */
  seq_nr frame_expected;                  /* нижний край окна получателя */
  seq_nr too_far;                         /* верхний край окна получателя + 1 */
  int i;                                  /* индекс массива буферов */
  frame r;                                /* временная переменная */
  packet out_buf[NR_BUFS];                /* буферы для исходящего потока */
  packet in_buf[NR_BUFS];                 /* буферы для входящего потока */
  boolean arrived[NR_BUFS];               /* входящая битовая карта */
  seq_nr nbuffered;     /* количество использующихся в данный момент выходных буферов */
  event_type event;

  enable_network_layer();          /* инициализация */
  ack_expected = 0;                /* следующий номер подтверждения во входном потоке */
  next_frame_to_send = 0;          /* номер следующего посылаемого кадра */
  frame_expected = 0;              /* номер следующего ожидаемого входящего кадра */
  too_far = NR_BUFS;               /* верхний край окна получателя + 1 */
  nbuffered = 0;                   /* вначале буфер пуст */

  for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
  while (true) {
      wait_for_event(&event);      /* пять возможных событий:
                                      см. event_type выше */
```

```
switch(event) {
    case network_layer_ready:       /* получить, сохранить и передать
                                       новый кадр */
        nbuffered = nbuffered + 1;  /* увеличить окно отправителя */
        from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]);
                                /* получить новый пакет у сетевого уровня */
        send_frame(data, next_frame_to_send, frame_expected, out_buf);
                                  /* передать кадр */
        inc(next_frame_to_send);  /* увеличить верхний край
                                     окна отправителя */
        break;

    case frame_arrival:  /* прибыл кадр с данными или с подтверждением */
        from_physical_layer(&r);  /* получить пришедший кадр
                                     у физического уровня */
        if (r.kind == data) {
            /* кадр прибыл в целости */
            if ((r.seq != frame_expected) && no_nak)
                send_frame(nak, 0, frame_expected, out_buf);
                else start_ack_timer();
            if (between(frame_expected, r.seq, too_far) &&
               (arrived[r.seq%NR_BUFS] == false)) {
                /* кадры могут приниматься в любом порядке */
                arrived[r.seq % NR_BUFS] = true;   /* пометить буфер
                                                      как занятый */
                in_buf[r.seq % NR_BUFS] = r.info;  /* поместить данные
                                                      в буфер */
                while (arrived[frame_expected % NR_BUFS]) {
                    /* передать пакет сетевому уровню и сдвинуть окно */
                    to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                    no_nak = true;
                    arrived[frame_expected % NR_BUFS] = false;
                    inc(frame_expected);  /* передвинуть нижний край
                                             окна получателя */
                    inc(too_far);         /* передвинуть верхний край
                                             окна получателя */
                    start_ack_timer();  /* запустить вспомогательный таймер
                                           на случай, если потребуется
                                           пересылка подтверждения
                                           отдельным кадром */
                }
            }
        }
        if((r.kind==nak) &&
           between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next_frame_to_send))
            send_frame(data, (r.ack+1) % (MAX_SEQ + 1),
                       frame_expected, out_buf);

        while (between(ack_expected, r.ack, next_frame_to_send)) {
            nbuffered = nbuffered - 1;  /* отправить подтверждение вместе
                                           с информационным кадром */
            stop_timer(ack_expected % NR_BUFS);  /* кадр прибыл в целости */
```

продолжение ⇩

Листинг 3.7 (*продолжение*)

```
                inc(ack_expected);                  /* передвинуть нижний край
                                                       окна отправителя */
            }
            break;

        case cksum_err: if (no_nak) send_frame(nak, 0, frame_expected, out_buf);
                        break;  /* поврежденный кадр */
        case timeout: send_frame(data, oldest_frame, frame_expected, out_buf);
                        break;  /* время истекло */
        case ack_timeout: send_frame(ack,0,frame_expected, out_buf);
/* истек период ожидания "попутки" для подтверждения; послать подтверждение */
      }

      if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
  }
}
```

Способность протокола принимать кадры в произвольном порядке накладывает дополнительные ограничения на порядковые номера кадров по сравнению с протоколами, в которых все пакеты принимались строго по порядку номеров. Проще всего проиллюстрировать это на примере. Предположим, что порядковый номер кадра состоит из 3 бит, так что отправитель может посылать до семи кадров, прежде чем перейти в режим ожидания подтверждения. Начальное состояние окон отправителя и получателя изображено на рис. 3.15, *а*. Отправитель передает кадры с 0 по 6. Окно получателя позволяет ему принимать любые кадры с номерами от 0 по 6 включительно. Все семь кадров прибывают успешно, поэтому получатель подтверждает их прием и передвигает окно для приема кадров с номерами 7, 0, 1, 2, 3, 4 и 5, как показано на рис. 3.15, *б*. Все семь буферов помечаются как свободные.

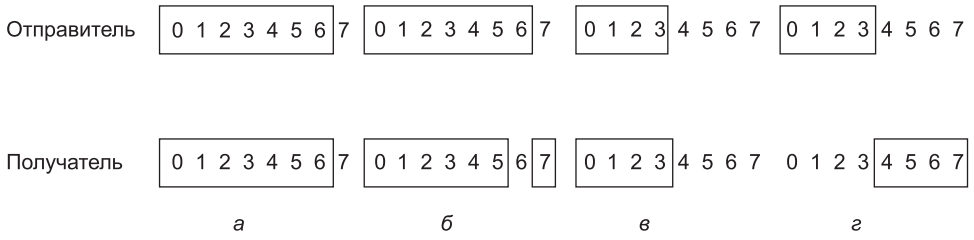


Рис. 3.15. Пример работы протокола: *а* — начальная ситуация при размере окна 7; *б* — 7 кадров были посланы и приняты, но не подтверждены; *в* — начальная ситуация при размере окна 4; *г* — ситуация после того, как 4 кадра были отправлены и получены, но не подтверждены

Именно в этот момент происходит какое-нибудь бедствие, например молния ударяет в телефонный столб и стирает все подтверждения. Протокол обязан отработать правильно, несмотря ни на какие чрезвычайные ситуации. Отправитель, не дождавшись подтверждений, посылает повторно кадр 0. К сожалению, кадр 0 попадает в новое окно и поэтому принимается получателем (рис. 3.15, *б*). Получатель снова отправляет подтверждение для кадра 6, поскольку были приняты все кадры с 0 по 6.

Отправитель с радостью узнает, что все переданные им кадры успешно достигли адресата, поэтому он тут же передает кадры 7, 0, 1, 2, 3, 4 и 5. Кадр 7 принимается получателем, и содержащийся в нем пакет передается сетевому уровню. Сразу после этого принимающий канальный уровень проверяет наличие кадра 0, обнаруживает его и передает старый буферизированный пакет сетевому уровню как новый. Таким образом, сетевой уровень получает неверный пакет; это означает, что протокол со своей задачей не справился.

Причина неудачи в том, что при сдвиге приемного окна новый интервал допустимых номеров кадров перекрыл старый интервал. Соответственно, присылаемый набор кадров может содержать как новые кадры (если все подтверждения были получены), так и повторно высланные старые кадры (если подтверждения были потеряны). У принимающей стороны нет возможности отличить одну ситуацию от другой.

Решением данной проблемы является предоставление гарантии того, что в сдвинутом положении окно не перекроет исходное окно. Для этого размер окна не должен превышать половины от количества порядковых номеров (это показано на рис. 3.15, *в* и 3.15, *г*.) Например, если для порядковых номеров используются 3 бита, они должны изменяться в пределах от 0 до 7. В таком случае, в любой момент времени только четыре кадра могут быть неподтвержденными. Таким образом, если будут получены кадры с 0 по 3 и будет передвинуто окно для приема кадров с 4 по 7, получатель сможет безошибочно отличить повторную передачу (кадры с 0 по 3) от новых кадров (с 4 по 7). Поэтому в протоколе 6 применяется окно размером (`MAX_SEQ` + 1)/2.

Возникает новый вопрос: сколько буферов должно быть у получателя? Ни при каких условиях он не должен принимать кадры, номера которых не попадают в окно. Поэтому количество необходимых буферов равно размеру окна, а не диапазону порядковых номеров. В приведенном выше примере 3-битовых порядковых номеров требуется четыре буфера с номерами от 0 до 3. Когда прибывает кадр i, он помещается в буфер $i \bmod 4$. Обратите внимание на то, что хотя i и $(i + 4)$, взятые по модулю 4, «соревнуются» за один и тот же буфер, они никогда не оказываются в одном окне одновременно, потому что это привело бы к увеличению размера окна, по крайней мере, до 5.

По этой же причине количество необходимых таймеров также равно числу буферов, а не диапазону порядковых номеров. Таким образом, удобно связать каждый таймер со своим буфером. Когда интервал времени истекает, содержимое буфера высылается повторно.

Протокол 6 также ослабляет неявное предположение о том, что загрузка канала довольно высока. Мы сделали это предположение в протоколе 5, в котором подтверждение «ехало верхом» на встречном информационном кадре. Если обратный поток информации невелик, подтверждения могут задерживаться на довольно большой период времени, создавая проблемы. В экстремальной ситуации, если в одном направлении посылается много информации, а во встречном — вообще ничего, протокол останавливается, когда окно отправителя достигает максимума.

В протоколе 6 эта проблема решена. По прибытии последовательного кадра с данными процедура `start_ack_timer` запускает вспомогательный таймер. Если таймер сработает раньше, чем появится кадр с данными для передачи, то будет послан отдельный кадр с подтверждением. Прерывание от вспомогательного таймера называется собы-

тием ack_timeout. При такой организации возможен однонаправленный поток данных, так как отсутствие встречных информационных кадров, на которых можно было бы отправлять подтверждения, больше не является препятствием. Требуется всего один таймер. При вызове процедуры start_ack_timer, если таймер уже запущен, ничего не происходит. Таймер не сбрасывается и не продляется, так как он нужен лишь для обеспечения некоторого минимального количества подтверждений.

Важно, что период времени вспомогательного таймера должен быть существенно короче интервала ожидания подтверждения. При этом условии подтверждение в ответ на полученный правильный кадр должно приходить прежде, чем у отправителя истечет период ожидания, и он пошлет этот кадр второй раз.

Протокол 6 использует более эффективную стратегию обработки ошибок, чем протокол 5. Как только у получателя появляются подозрения, что произошла ошибка, он высылает отправителю отрицательное подтверждение (NAK). Получатель может делать это в двух случаях: если он получил поврежденный кадр или если прибыл кадр с номером, отличным от ожидаемого (возможность потери кадра). Чтобы избежать передачи нескольких запросов на повторную передачу одного и того же кадра, получатель должен запоминать, был ли уже послан NAK для данного кадра. В протоколе 6 для этой цели применяется переменная no_nak, принимающая значение true, если NAK для ожидаемого кадра (с номером frame_expected) еще не был послан. Если NAK повреждается или теряется по дороге, в этом нет большой беды, так как у отправителя, в конце концов, истечет период ожидания положительного подтверждения, и он, так или иначе, вышлет недостающий кадр еще раз. Если после того, как NAK будет выслан и потерян, прибудет не тот кадр, переменной no_nak опять будет присвоено true и будет запущен вспомогательный таймер. Когда время истечет, будет послано положительное подтверждение (ACK) для восстановления синхронизации текущих состояний отправителя и получателя.

В некоторых ситуациях время, необходимое для прохождения кадра по каналу, его обработки и отсылки обратно подтверждения, практически остается неизменным. В таких ситуациях отправитель может поставить время ожидания лишь немного больше ожидаемого интервала между отправкой кадра и получением подтверждения. NAK в таких случаях применять неэффективно.

Однако в других случаях время может сильно варьироваться. Если встречный поток данных нерегулярен, то время прихода подтверждений также будет непостоянным, уменьшаясь при наличии встречного потока и увеличиваясь при его отсутствии. Перед отправителем возникает непростой выбор значения времени ожидания. Если выбрать слишком короткий интервал, то увеличится риск ненужных повторных передач. При выборе слишком большого значения протокол будет тратить много времени на ожидания после ошибки. В обоих случаях пропускная способность на что-то тратится. В целом, если среднеквадратичное отклонение интервала ожидания подтверждения велико по сравнению с самим интервалом, то таймер может быть установлен довольно «свободно», и отрицательные подтверждения могут существенно ускорить повторную передачу потерянных или поврежденных кадров.

С вопросом тайм-аутов и отрицательных подтверждений тесно связана проблема определения кадра, вызвавшего тайм-аут. В протоколе 5 это всегда кадр с номером ack_expected, поскольку он является старшим. В протоколе 6 нет столь простого способа определить кадр, интервал ожидания которого истек. Предположим, были переданы

кадры с 0 по 4, то есть список неподтвержденных кадров выглядит так: 01234 (от первого к последнему). Теперь допустим, что у кадра 0 истекает интервал ожидания и он передается повторно, затем посылается кадр 5 (новый), потом интервал ожидания истекает у кадров 1 и 2 и посылается кадр 6 (также новый). В результате список неподтвержденных кадров принимает вид: 3405126, начиная с самого старого и заканчивая самым новым. Если весь встречный поток данных потеряется, интервалы ожидания этих семи кадров истекут именно в таком порядке.

Чтобы не усложнять и без того непростой пример протокола, мы не показываем подробностей управления таймером. Вместо этого предполагается, что переменной `oldest_frame` при наступлении тайм-аута присваивается номер кадра, интервал времени которого истек.

3.5. Примеры протоколов передачи данных

В пределах одного здания для связи компьютеров широко применяются локальные сети, однако большинство глобальных сетей построено на двухточечных линиях. С локальными сетями мы познакомимся в главе 4. Здесь мы рассмотрим протоколы канального уровня, которые применяются на двухточечных каналах в Интернете в двух наиболее распространенных ситуациях. Первая — это передача пакетов по оптоволокну SONET. Например, такие каналы соединяют маршрутизаторы, установленные в разных концах сети поставщика услуг Интернета.

Вторая ситуация описывает каналы ADSL в пределах локального контура телефонной сети. Такие связи соединяют с Интернетом миллионы отдельных пользователей и компаний.

Пользователям необходимы для подключения к Интернету такие двухточечные связи, а также телефонные модемы, арендованные линии, кабельные модемы и т. д. Для пересылки пакетов по таким каналам используется стандартный протокол под названием **PPP** (**Point-to-Point Protocol**, **протокол двухточечного соединения**). Протокол PPP описан в стандарте RFC 1661 и доработан в более поздних документах RFC 1662 и др. (Simpson, 1994a, 1994b). PPP применяется в каналах SONET и ADSL, но по-разному.

3.5.1. Передача пакетов по протоколу SONET

SONET, с которым мы познакомились в главе 2, — это протокол физического уровня, который наиболее часто используется в оптоволоконных каналах, составляющих магистраль различных коммуникационных сетей, включая телефонную. Этот протокол обеспечивает хорошую, строго определенную скорость передачи данных (например, 2,4 Гбит/с в канале OC-48). Поток бит организован в виде пакетов фиксированного размера, которые посылаются каждые 125 мкс, независимо от того, содержат ли они пользовательские данные.

Для передачи пакетов по таким каналам необходим некоторый механизм формирования кадров, способный отличать иногда возникающие пакеты от непрерывного потока бит, в котором они передаются. Для обеспечения такого механизма на IP-маршрутизаторах работает протокол PPP, как показано на рис. 3.16.

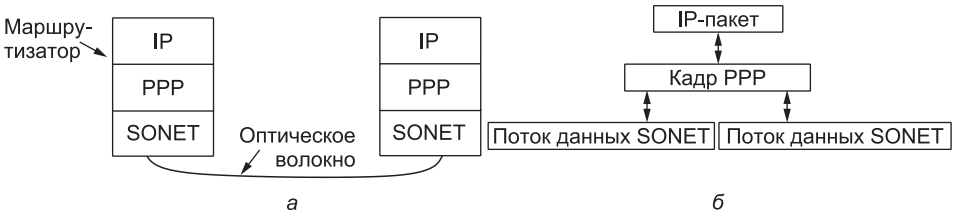


Рис. 3.16. Пакеты передаются по протоколу SONET: *а* — стек протоколов; *б* — взаимоотношение между кадрами

PPP — это улучшенный вариант более простого протокола под названием **SLIP** (**Serial Line Internet Protocol**, **интернет-протокол для последовательной линии**), выполняющий обнаружение ошибок, поддерживающий несколько протоколов, разрешающий аутентификацию и имеющий ряд других свойств. Благодаря широкому набору настроек PPP обеспечивает три основных набора методов.

1. Метод формирования кадров, однозначно обозначающий конец одного кадра и начало следующего. Формат кадров также обеспечивает обнаружение ошибок.
2. Протокол управления каналом, позволяющий устанавливать каналы связи, тестировать их, договариваться о параметрах их использования и снова отключать их, когда они не нужны. Этот протокол называется **LCP** (**Link Control Protocol**).
3. Способ договориться о параметрах сетевого уровня, который не зависит от используемого протокола сетевого уровня. Для каждого поддерживаемого сетевого уровня этот метод должен иметь свой сетевой протокол управления (**NCP**, **Network Control Protocol**).

Чтобы не изобретать велосипед, был выбран формат кадра PPP, близкий к формату кадра **HDLC** (**High-level Data Link Control**, **высокоуровневый протокол управления каналом**) — некогда популярного представителя раннего семейства протоколов.

В отличие от бит-ориентированного протокола HDLC, PPP является байт-ориентированным. В частности, в PPP применяется символьное заполнение, поэтому все кадры состоят из целого числа байт. С помощью протокола PPP невозможно послать кадр, состоящий из 30,25 байт, как это можно было сделать в протоколе HDLC.

Однако практическое значение имеет второе очень важное отличие. HDLC обеспечивает надежную передачу за счет метода скользящего окна, подтверждений и таймаутов — как мы видели выше. PPP также обеспечивает надежную передачу в шумных средах, таких как беспроводные сети; детали протокола определены в стандарте RFC 1663. Однако на практике это применяется редко. Вместо этого в Интернете для обеспечения сервиса без установки соединения и без подтверждений применяется «ненумерованный режим».

Формат кадра PPP показан на рис. 3.17. Все PPP-кадры начинаются со стандартного флагового байта протокола HDLC 0x7E (01111110). Если этот байт встречается в поле *Данные* (*Payload*), он предваряется управляющим байтом 0x7D, а следующий за ним байт представляет собой предваряемый байт, сложенный по модулю 2 со значением 0x20 (при этом переключается пятый бит). Например, 0x7D 0x5E — это управляющая последовательность для флагового байта 0x7E. Это означает, что нача-

ло и конец кадра можно найти, просто просканировав содержимое на наличие байта 0x7E. Больше он нигде встречаться не будет. Правило удаления заполняющих битов при получении — найти значение 0x7D, удалить его, а следующий байт сложить по модулю 2 со значение 0x20. Кроме того, между кадрами необходим только один флаговый байт. Несколько флаговых байтов могут применяться для заполнения канала, когда кадров с данными для отправки получателю нет.

После стартового кадра идет поле *Адрес* (*Address*), которому всегда присваивается двоичное значение 11111111, это означает, что все станции должны принимать этот кадр. Использование такого адреса позволяет избежать необходимости назначения адресов передачи данных.

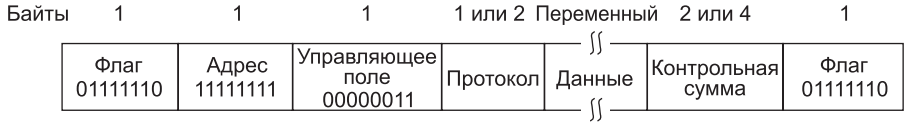


Рис. 3.17. Полный формат кадра PPP для работы в ненумерованном режиме

За полем адреса следует *Управляющее поле* (*Control*), его значение по умолчанию равно 00000011. Это число означает ненумерованный кадр.

Так как в конфигурации по умолчанию поля *Адрес* и *Управляющее поле* являются константами, протокол LCP предоставляет возможность двум сторонам договориться о возможности пропускать оба поля и сэкономить, таким образом, по 2 байта на кадр.

Четвертое поле кадра PPP — *Протокол* (*Protocol*). Оно определяет тип пакета, содержащегося в поле *Данные*. Номера, начинающиеся с бита 0, отведены для протокола IP версий 4 и 6 и других протоколов сетевого уровня, таких как IPX и AppleTalk. С бита 1 начинаются коды, используемые для конфигурационных протоколов PPP, включая LCP и различные протоколы NCP для каждого поддерживаемого протокола сетевого уровня. Размер поля *Протокол* по умолчанию составляет 2 байта, однако путем переговоров с помощью LCP этот размер может быть уменьшен до одного байта. Разработчики, вероятно, перестраховались на случай, если когда-либо будет использоваться более 256 протоколов.

Поле *Данные* может быть переменной длины, вплоть до некоего оговоренного максимального значения. Если размер не оговорен во время установки соединения при помощи LCP, то по умолчанию используется длина 1500 байт. При необходимости данные пользователя могут дополняться специальными символами.

Следом за полем *Данные* располагается поле *Контрольная сумма* (*Checksum*), которое в обычном состоянии занимает 2 байта, но в случае необходимости по договоренности может занимать 4. 4-байтовая контрольная сумма фактически представляет собой 32-битный код CRC, порождающий многочлен которого показан в соответствующем разделе выше. 2-байтовая контрольная сумма также является стандартным кодом CRC.

Итак, PPP является механизмом формирования кадров, поддерживающим различные протоколы, которым можно пользоваться в различных физических средах. Различные варианты использования PPP в сетях SONET описаны в стандарте RFC 2615 (Malis, Simpson, 1999). Применяется 4-байтовая контрольная сумма, так как она счи-

тается основным способом распознавания ошибок передачи на физическом уровне, уровне передачи данных и сетевом уровне. Рекомендуется не сжимать поля *Адрес*, *Управляющее поле* и *Протокол*, так как каналы SONET и так работают на относительно высокой скорости.

Есть и еще одна интересная особенность. Перед тем как попасть в поток данных SONET, полезная информация протокола PPP шифруется (подробнее об этом выше). При шифровании данные складываются по модулю 2 с длинной псевдослучайной последовательностью. Только после этого они пересылаются получателю. Проблема в том, что потоку данных SONET для успешной синхронизации требуется частая смена значений бит. В колебаниях голосового сигнала это происходит естественным образом, но при пересылке данных только пользователь выбирает, какую информацию отправлять, и это может быть, например, длинная последовательность нулей. Благодаря шифрованию вероятность того, что пользователь сам вызовет проблемы, передав длинную последовательность нулей, сводится почти к нулю.

Перед тем как передавать кадры PPP по каналу SONET, необходимо установить и сконфигурировать соединение PPP. Состояния, через которые проходит линия связи при ее установлении, использовании и разъединении, показаны на рис. 3.18.

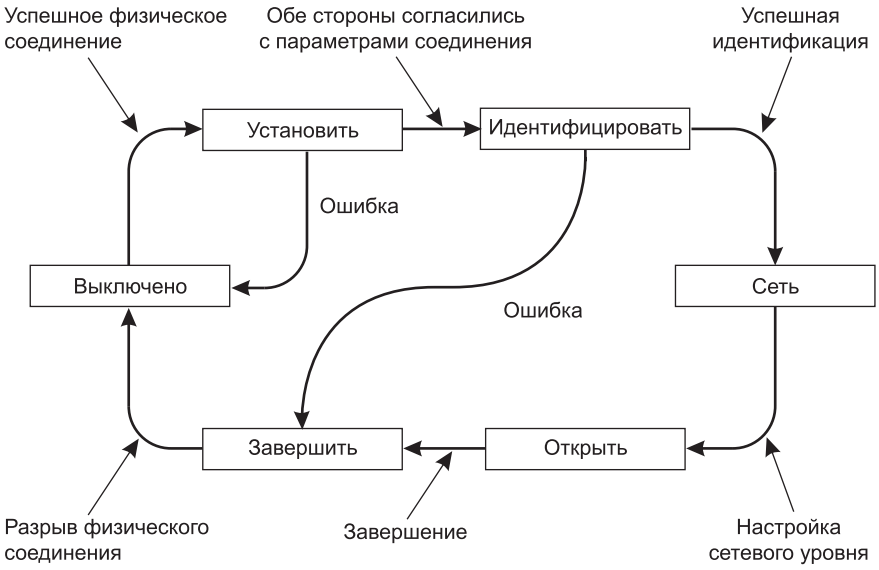


Рис. 3.18. Диаграмма состояний установки и разрыва соединения PPP

Начальное состояние протокола таково: линия отключена (*DEAD*), то есть соединения на физическом уровне не существует. После того как физическое соединение установлено, линия переходит в состояние *ESTABLISH* (установка). В этот момент начинаются переговоры о параметрах с помощью протокола LCP. Узлы PPP обмениваются пакетами LCP, каждый из которых содержится в поле *Данные* кадра PPP. Это необходимо для выбора параметров PPP из тех, что перечислены выше. Инициирующий узел предлагает варианты, а отвечающие узлы либо соглашаются с ними,

либо отвергают частично или полностью. Отвечающие узлы также могут делать свои предложения.

При успешном результате переговоров линия переходит в фазу *AUTHENTICATE* (идентифицировать). Теперь обе стороны по желанию могут проверить, кем является собеседник. После успешной аутентификации в фазе *NETWORK* (сеть) происходит обмен пакетами NCP для настройки сетевого уровня. Протоколы NCP сложно описать общими словами, так как каждый из них обладает специфическими свойствами, зависящими от соответствующего протокола сетевого уровня, и поддерживает конфигурационные запросы, характерные только для этого протокола. Например, для протокола IP наиболее важной задачей является назначение IP-адресов собеседникам на обоих концах линии.

Когда линия переходит в фазу *OPEN* (открытая), можно начинать передачу данных. Именно в этой фазе IP-пакеты пересылаются в кадрах PPP по линии SONET. Когда передача данных закончена, линия переходит к фазе *TERMINATE* (завершение), а затем снова в состояние *DEAD* (отключена), когда физическое соединение разрывается.

3.5.2. ADSL

ADSL (Asymmetric Digital Subscriber Loop, асимметричный цифровой абонентский контур) соединяет миллионы домашних пользователей с Интернетом на скоростях, равных нескольким мегабит в секунду. Для этого используется тот же локальный телефонный контур, по которому предоставляются услуги обычной телефонии. Выше мы рассматривали, как на домашней стороне устанавливается устройство под названием DSL-модем. Он отправляет биты по локальному контуру, адресуя их устройству **DSLAM (DSL Access Multiplexer, мультиплексор доступа DSL)**, установленному в местном офисе телефонной компании. Теперь мы более подробно рассмотрим процесс передачи пакетов по каналам ADSL.

Общая схема работы протоколов и устройств показана на рис. 3.19. В разных сетях применяются разные протоколы, поэтому мы выбрали для демонстрации наиболее популярный сценарий. Внутри дома компьютер посылает IP-пакеты DSL-модему. Они путешествуют по канальному уровню, такому как Ethernet. Затем DSL-модем отправляет IP-пакеты по локальному контуру устройству DSLAM, применяя для этого протоколы, которые мы рассмотрим далее. На стороне DSLAM (или, в зависимости от реализации подключенного к нему маршрутизатора) IP-пакеты извлекаются и поступают в сеть поставщика услуг Интернета, по которой и достигают назначенной точки в сети.

Показанные на рис. 3.19 протоколы, работающие в канале ADSL, начинаются с низшего, физического уровня. Они основаны на схеме цифровой модуляции под названием мультиплексирование с ортогональным делением частот (также известное как цифровая многоканальная тональная модуляция), с которым мы познакомились ранее. Ближе к вершине стека, под сетевым уровнем IP, находится PPP. Это тот же самый протокол PPP, который мы изучили при рассмотрении пакетов, путешествующих по сетям SONET. Он точно так же устанавливает и настраивает связь для передачи IP-пакетов.

Между ADSL и PPP находятся ATM и AAL5. Это новые протоколы, с которыми мы ранее не встречались. Протокол **ATM (Asynchronous Transfer Mode, режим асинхронной передачи)** был разработан в начале 1990-х годов и широко рекламировался

при первом запуске. Он обещал сетевую технологию, которая решит все мировые телекоммуникационные проблемы, объединив голос, текстовые данные, кабельное телевидение, телеграф, почтовых голубей, связанные нитью консервные банки, тамтамы и все остальные способы передачи информации в интегрированную систему, способную удовлетворить любые требования каждого пользователя. Этого не случилось. В целом, ATM столкнулся с теми же проблемами, о которых мы упомянули в разговоре о протоколах OSI: плохая синхронизация, технология, реализация и политические тонкости. Тем не менее ATM все же добился большего успеха, чем OSI. Хотя он и не завоевал мир, его все же широко применяют в таких сферах, как линии широкополосного доступа, такие как DSL, и каналы WAN в телефонных сетях.

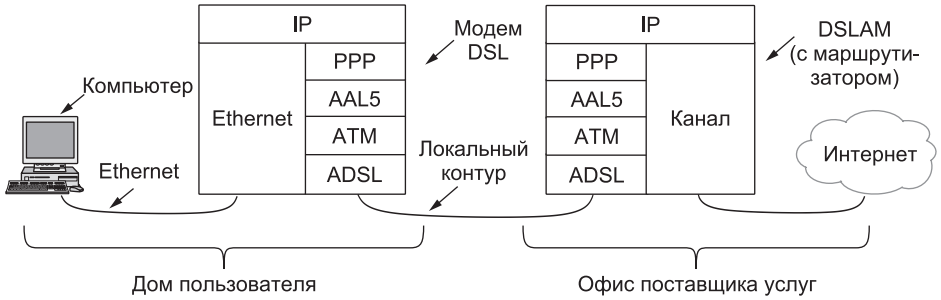


Рис. 3.19. Стек протоколов ADSL

ATM представляет канальный уровень, основанный на пересылке **ячеек (cells)** информации фиксированной длины. Асинхронная передача означает, что нет необходимости постоянно отправлять ячейки, как, например, биты по синхронным линиям (таким как SONET). Ячейки пересылаются только тогда, когда имеется какая-то информация, готовая к передаче. **ATM — это технология, ориентированная на соединение.** В заголовок каждой ячейки встраивается идентификатор **виртуального контура (virtual circuit)**, и устройства используют этот идентификатор для пересылки ячеек по различным путям внутри установленных соединений.

Длина каждой ячейки составляет 53 байта: 48 байт полезной нагрузки плюс 5 байт заголовка. Применяя ячейки небольшого размера, ATM гибко разделяет полосу пропускания физического канала между разными пользователями. Эта возможность полезна, когда, например, по одному каналу пересылаются голосовые данные и текстовая информация. Большие пакеты текстовых данных не будут приводить к длинным задержкам при пересылке фрагментов голосовой информации. Нестандартный выбор длины ячейки (сравните 53 байта с более естественным выбором значения, представляющего степень двойки) иллюстрирует политические вопросы, имевшие немалое значение при разработке протокола. 48 байт под полезную информацию — это компромисс между 32-байтовыми ячейками, которые хотела использовать Европа, и 64-байтовыми, за которые голосовала Америка. Краткое описание протокола представили Сиу и Джайн (Siu, Jain, 1995).

Для пересылки данных по сети ATM необходимо отобразить их в последовательность ячеек. Отображение выполняется на уровне адаптации протокола ATM про-

цессом, который называется сегментацией и обратной сборкой (segmentation and reassembly). Для различных служб, пересылающих, например, периодические образцы голосовых данных или пакетную информацию, были определены несколько уровней адаптации. Основной, используемый для пакетных данных — это **AAL5** (**ATM Adaptation Layer 5**, **уровень адаптации ATM 5**).

Кадр AAL5 показан на рис. **3.20. Роль заголовка у него исполняет концевик,** содержащий сведения о длине, а также 4-байтовый код CRC для обнаружения ошибок. Разумеется, это тот же самый CRC, который используется протоколом PPP и сетями стандарта IEEE 802, такими как Ethernet. Вонг и Кроукрофт (Wang, Crowcroft, 1992) продемонстрировали, что это достаточно сильная конфигурация, чтобы обнаруживать нетрадиционные ошибки, такие как сбой в порядке следования ячеек. Помимо полезной нагрузки, в кадре AAL5 есть биты заполнения (*Pad*). Они дополняют общую длину, чтобы она была кратной 48 байтам. Таким образом, кадр можно будет поделить на целое число ячеек. Хранить адреса внутри кадра не нужно, так как идентификатор виртуального контура, имеющийся в каждой ячейке, не даст ей заблудиться и приведет к нужному получателю.

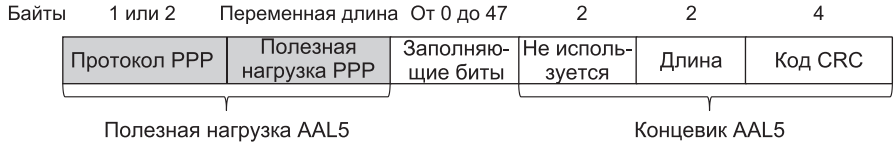


Рис. 3.20. Кадр AAL5, содержащий данные PPP

Итак, мы познакомились с протоколом ATM. Осталось только рассказать, как его задействует протокол PPP в случае подключения по каналам ADSL. Это делается с помощью еще одного стандарта, который называется **PPPoA** (**PPP over ATM**, то есть **PPP с использованием ATM**). В действительности данный стандарт нельзя назвать протоколом (поэтому на рис. 3.19 его нет). Скорее, это спецификация, описывающая, как одновременно применять протокол PPP и кадры AAL5. Подробнее об этом рассказывается в стандарте RFC 2364 (Gross и др., 1998).

Полезная нагрузка AAL5 включает только поля *Протокол* (*Protocol*) и *Данные* (*Payload*) протокола PPP, как показано на рис. 3.20. Поле протокола сообщает устройству DSLAM, является полезная нагрузка IP-пакетом или пакетом другого протокола, например LCP. Принимающая сторона знает, что ячейки содержат информацию PPP, так как виртуальный контур ATM настраивается соответствующим образом.

В кадре AAL5 механизмы формирования кадра PPP не требуются, всю работу выполняют ATM и AAL5. Дополнительно создавать кадры было бы попросту бессмысленно. Код CRC протокола PPP также не нужен, поскольку AAL5 включает тот же самый код CRC. Механизм выявления ошибок дополняет кодирование физического уровня, применяемое в каналах ADSL (код Рида—Соломона для исправления ошибок и 1-байтовый CRC для распознавания оставшихся ошибок, не пойманных другими способами). Это намного более сложный механизм устранения ошибок, чем тот, что применяется при пересылке данных в сетях SONET. Причина проста — линии ADSL куда более зашумленные.

3.6. Резюме

Задачей канального уровня является преобразование необработанного потока бит, поступающего с физического уровня, в поток кадров, которые может использовать сетевой уровень. Канальный уровень может представлять такой поток с различной степенью надежности, начиная от сервисов без установки соединения и без подтверждения и заканчивая надежными ориентированными на соединение сервисами.

Используются различные методы формирования кадров, включая подсчет байтов, символьное и битовое заполнение. Протоколы канального уровня могут обладать возможностями контроля ошибок для обнаружения и исправления поврежденных кадров и повторной передачи потерянных. Во избежание опережения медленного приемника быстрым отправителем применяется управление потоком. Механизм скользящих окон широко используется для удобного объединения контроля ошибок и управления потоком. Для окна размером в один пакет применяется протокол с остановкой и ожиданием.

Коды для обнаружения и исправления ошибок добавляют к сообщениям избыточную информацию, применяя ряд математических техник. Для исправления ошибок широко применяются сверточные коды и коды Рида—Соломона, и все большую популярность завоевывают коды с малой плотностью проверок на четность. Применяемые на практике коды для обнаружения ошибок включают циклический контроль избыточности и контрольные суммы. Все эти коды можно применять на канальном уровне, а также на физическом и более высоких уровнях.

Мы рассмотрели ряд протоколов, обеспечивающих надежную работу канального уровня за счет подтверждений и повторной передачи или, если взять более приближенный к жизни пример, за счет запросов ARQ (Automatic Repeat reQuest). Начиная с идеальной среды передачи, в которой отсутствуют ошибки, и идеального приемника, который может обработать входящий поток любого размера, мы познакомились с управлением потоком, затем с контролем ошибок при помощи порядковых номеров и, наконец, с алгоритмом с остановкой и ожиданием. Затем мы перешли к алгоритму скользящего окна, который разрешает обмен данными в двух направлениях, и узнали о концепции комбинированных пакетов. Последние два протокола используют конвейерную передачу множества кадров, чтобы отправитель не блокировался, увеличивая задержку передачи. Получатель может либо отбрасывать все кадры, за исключением очередного в последовательности, либо помещать неупорядоченные кадры в буфер и отправлять отрицательные подтверждения для более эффективного использования полосы пропускания. Первая стратегия называется протоколом с возвратом на n, а вторая — протоколом с выборочным повтором.

В Интернете в качестве основного протокола линий «точка — точка» используется PPP. Он предоставляет сервис без установки соединения и без подтверждения. Для разделения кадров применяются флаговые байты, а для распознавания ошибок — коды CRC. С помощью этого протокола пакеты передаются по множеству типов соединений, включая каналы SONET в глобальных сетях и ADSL для домашних подключений.

Вопросы

1. Сообщение верхнего уровня разбито на 10 кадров, у каждого из которых шанс дойти до назначения без повреждений составляет 80 %. Если канальный уровень не обеспечивает проверки ошибок, сколько раз в среднем потребуется пересылать все сообщение?
2. В протоколе канального уровня используется следующее кодирование символов:

   A: 01000111; B: 11100011; FLAG: 01111110; ESC: 11100000

   Как в двоичных кодах будет выглядеть кадр, состоящий из четырех символов — A B ESC FLAG, при использовании каждого из следующих методов кадрирования:

   1) подсчет байтов;

   2) флаговые байты с символьным заполнением;

   3) начальные и конечные флаговые байты с битовым заполнением.
3. В потоке данных, для которого применяется алгоритм символьного заполнения, встречается следующий фрагмент данных: A B ESC C ESC FLAG FLAG D. Каким будет выходной поток после заполнения символами?
4. Каковы максимальные накладные расходы для алгоритма символьного заполнения?
5. Один из ваших однокурсников, большой скряга, предположил, что использовать после концевого флагового байта кадра начальный флаговый байт следующего кадра — слишком расточительно, вполне можно обойтись флаговым байтом. Таким образом, можно сэкономить передачу одного байта. Вы согласитесь с ним?
6. Каким будет на выходе следующий поток бит после применения битового заполнения на уровне передачи данных: 0111101111101111110?
7. При каких обстоятельствах протокол без обратной связи (например, с кодом Хэмминга) может быть предпочтительнее протоколов с обратной связью, обсуждаемых в данной главе?
8. Для обеспечения большей надежности, нежели та, которую предоставляет единственный бит четности, в некотором методе обнаружения ошибок один бит четности суммирует все четные биты, а другой — все нечетные. Каково будет в этом случае расстояние кода по Хэммингу?
9. При помощи кода Хэмминга передаются 16-битные сообщения. Сколько контрольных бит потребуется для того, чтобы приемник гарантированно мог обнаруживать и исправлять одиночные битовые ошибки? Как будет выглядеть код для передачи следующего сообщения: 1101001100110101? Предполагается, что код Хэмминга использует проверку четных бит.
10. Приемник получает 12-битную последовательность в коде Хэмминга, ее шестнадцатеричное значение равно 0xE4F. Как (в шестнадцатеричном виде) выглядела исходная последовательность? Предполагается, что ошибочным может быть только 1 бит.
11. Один из способов обнаружения ошибок заключается в передаче данных в виде блока из n рядов по k бит с добавлением битов четности к каждому ряду и каждой строке. Бит в нижнем правом углу — это бит четности, проверяющий свою строку и столбец. Будет ли такая схема обнаруживать все одиночные ошибки? Двойные ошибки? Тройные ошибки? Докажите на примере, что эта схема не в состоянии обнаруживать некоторые четырехбитные ошибки.
12. Предположим, что данные передаются в блоках размером 1000 бит. Каков максимальный коэффициент ошибок, при котором механизм с обнаружением ошибок и повторной передачей (1 бит четности на блок) покажет себя лучше, чем код Хэмминга? Предполагается, что ошибки в битах не зависят друг от друга, а во время повторной передачи ошибок в битах не бывает.

13. В блоке битов из n рядов и k строк используются горизонтальные и вертикальные биты четности для обнаружения ошибок. Какова вероятность того, что инверсия 4 бит не будет обнаружена?
14. Используя сверточный кодировщик, показанный на рис. 3.7, покажите выходную последовательность для входной последовательности 10101010 (слева направо) и нулевого внутреннего состояния.
15. Предположим, что сообщение 1001 1100 1010 0011 передается с использованием контрольной суммы для Интернета (4-битное слово). Какова будет контрольная сумма?
16. Чему равен остаток от деления $x^7 + x^5 + 1$ на образующий многочлен $x^3 + 1$?
17. Поток бит 10011101 передается с использованием стандартного метода циклического избыточного кода (CRC), описанного в тексте. Образующий многочлен равен $x^3 + 1$. Какая битовая последовательность будет реально передаваться? Предполагается, что третий бит слева при передаче инвертировался. Докажите, что эта ошибка будет обнаружена приемником. Приведите пример ошибок в битах передаваемой строки, которые приемник обнаружить не сможет.
18. Отправляется 1024-битное сообщение, содержащее 992 бита данных и 32 бита CRC. Код CRC вычисляется с использованием стандартизированного в IEEE 802 многочлена 32 степени. Для каждого из следующих случаев объясните, распознает ли получатель ошибки передачи сообщения:

    1) произошла 1-битная ошибка;

    2) произошли 2 изолированные 1-битные ошибки;

    3) произошло 18 изолированных 1-битных ошибок;

    4) произошло 47 изолированных 1-битных ошибок;

    5) произошла последовательность ошибок длиной 24 бита;

    6) произошла последовательность ошибок длиной 35 битов.
19. При обсуждении протокола ARQ приводился пример сценария, в котором получатель принимает две копии одного и того же кадра из-за утери кадра подтверждения. Возможно ли, что получатель примет несколько копий одного кадра, если ни один из кадров (данных или подтверждения) утерян не будет?
20. Скорость передачи данных в канале составляет 4 Кбит/с, а время распространения сигнала — 20 мс. При каком размере кадров эффективность протокола с ожиданиями составит, по меньшей мере, 50 % ?
21. Возможно ли, что в протоколе 3 отправитель запустит таймер, когда тот уже работает? Если да, то в какой ситуации? Если нет, то почему?
22. Кабель T1 длиной 3000 км используется для передачи 64-байтовых кадров при помощи протокола 5. Если задержка распространения сигнала составляет 6 мкс/км, сколько бит следует отвести на порядковые номера кадров?
23. Представьте себе протокол скользящего окна, в котором используется так много бит на порядковые номера кадров, что номера никогда не используются дважды. Какое соотношение должно связывать четыре границы окна и размер окна (постоянный и одинаковый для отправителя и получателя)?
24. Предположим, что в процедуре `between` протокола 5 вместо условия $a \leqslant b < c$ проверяется условие $a \leqslant b \leqslant c$. Как это повлияет на правильность протокола и его эффективность? Поясните свой ответ.
25. Когда прибывает информационный кадр, протокол 6 проверяет, отличается ли номер кадра от ожидаемого и равна ли переменная `no_nak` значению `true`. При выполнении обоих условий

посылается NAK. В противном случае запускается вспомогательный таймер. Предположим, что в тексте программы пропущен оператор else. Повлияет ли это на правильность работы протокола?

26. Предположим, что из конца текста программы протокола 6 удалены три строки цикла while. Повлияет ли это на правильность работы протокола или же только на его быстродействие? Поясните свой ответ.

27. Расстояние от Земли до далекой планеты равно приблизительно 9×10^{10} м. Если в канале «точка-точка» со скоростью передачи данных 64 Мбит/с для пересылки кадров применяется протокол с остановкой и ожиданием, то каков коэффициент загруженности канала? Предполагается, что размер кадра равен 32 Кбайт, а скорость света равна 3×10^8 м/с.

28. В условиях предыдущей задачи используется другой протокол — протокол скользящего окна. В случае какого размера окна отправителя коэффициент загруженности канала будет равен 100 %? Время на обработку протокола на отправителе и получателе можно не учитывать.

29. В протоколе 6 в программе, обрабатывающей событие прихода кадра frame_arrival, есть раздел, используемый для отрицательных подтверждений (NAK). Этому участку программы передается управление, когда получаемый кадр является NAK, а также при выполнении другого условия. Приведите пример сценария, в котором наличие этого условия является важным.

30. Протокол 6 применяется на безошибочной линии со скоростью 1 Мбит/с. Максимальный размер кадра 1000 бит. Новые пакеты формируются примерно раз в секунду. Интервал тайм-аута установлен на период 10 мс. Если отключить специальный таймер подтверждений, то будут происходить лишние тайм-ауты. Сколько раз в среднем будет передаваться одно сообщение?

31. В протоколе 6 значение $MAX\_SEQ = 2^n - 1$. Хотя это условие, очевидно, желательно для эффективного использования битов заголовка, важность его не была показана. Будет ли протокол корректно работать, например, при $MAX\_SEQ = 4$?

32. Кадры длиной 1000 бит посылаются по спутниковому каналу с пропускной способностью 1 Мбит/с и временем прохождения 270 мс. Подтверждения всегда посылаются в информационных кадрах. Заголовки кадров очень короткие. Используются 3-битовые порядковые номера. Какой будет максимальная эффективность использования канала при применении:
    - протокола с остановкой и ожиданием;
    - протокола 5;
    - протокола 6.

33. Рассчитайте, какая часть пропускной способности канала теряется на заголовки и повторные передачи при использовании протокола 6 на сильно загруженном спутниковом канале с пропускной способностью 50 Кбит/с. Кадры данных состоят из 40-битовых заголовков и 3960 бит данных. Время распространения сигнала от Земли до спутника составляет 270 мс. Кадры ACK никогда не посылаются. Размер кадров NAK равен 40 бит. Вероятность ошибки для кадра данных составляет 1 %, а для кадра NAK она пренебрежимо мала. Порядковые номера занимают 8 бит.

34. Предположим, что безошибочный спутниковый канал с пропускной способностью 64 Кбит/с используется для пересылки 512-байтных кадров данных в одном направлении, с очень короткими подтверждениями, идущими в обратном направлении. Какова будет максимальная скорость передачи данных при размере окна, равном 1, 7, 15 и 127? Время распространения сигнала от Земли до спутника — 270 мс.

35. Кабель длиной в 100 км работает на скорости T1. Скорость распространения сигнала равна 2/3 от скорости света в вакууме. Сколько бит помещается в кабеле?
36. Назовите хотя бы одну причину, по которой в протоколе PPP применяется символьное заполнение вместо битового (для того чтобы случайно встретившийся в поле данных флаговый байт не вызвал ошибки синхронизации кадров).
37. Каковы минимальные накладные расходы при пересылке IP-пакета по протоколу PPP? Учитывайте только накладные расходы самого протокола PPP, а не заголовки протокола IP. Каковы максимальные накладные расходы?
38. IP-пакет длиной 100 байт передается по локальному контуру с использованием стека протоколов ADSL. Сколько ячеек ATM будет передано? Кратко опишите их содержимое.
39. Целью данного упражнения является реализация механизма обнаружения ошибок с помощью стандартного алгоритма циклического избыточного кода (CRC), описанного в тексте. Напишите две программы: генератор (`generator`) и верификатор (`verifier`). Программа-генератор считывает со стандартного устройства ввода n-битное сообщение из нулей и единиц, представленных в виде строки ASCII-текста. Вторая строка является k-битным многочленом (также в ASCII). На устройстве вывода печатается текст из $n + k$ нулей и единиц, представляющий собой сообщение, подлежащее пересылке. Затем печатается многочлен в том же виде, в каком он был считан. Программа-верификатор считывает результат работы генератора и выводит сообщение, в котором сообщается, корректен ли данный результат. Наконец, напишите программу (`alter`), вносящую сбой, а именно инвертирующую только один бит первой строки, в зависимости от аргумента (например, порядкового номера бита, предполагая, что слева располагается бит с номером 1). Все остальные данные передаются без изменений. Набрав в командной строке `generator <file | verifier`, пользователь должен увидеть сообщение о том, что данные переданы корректно. Набрав `generator <file | alter arg | verifier`, пользователь должен получить сообщение об ошибке при передаче.

Глава 4

Подуровень управления доступом к среде

Все сетевые технологии могут быть разделены на две категории: использующие соединения от узла к узлу и сети с применением широковещания. Двухточечные связи мы рассматривали в главе 2; эта глава посвящена широковещательным каналам и их протоколам.

Главной проблемой любых широковещательных сетей является вопрос о том, как определить, кому предоставить канал, если пользоваться им одновременно хотят несколько компьютеров. Для примера представьте себе конференцию, в которой принимают участие шесть человек, причем каждый использует свой телефон. Все они соединены таким образом, что каждый может слышать всех остальных. Весьма вероятно, что когда один из них закончит свою речь, сразу двое или трое начнут говорить одновременно, тем самым создав неловкую ситуацию. При личной встрече подобные проблемы предотвращаются внешними средствами, например поднятием руки для получения разрешения говорить. Когда доступен лишь один канал, определить, кто может говорить следующим, значительно труднее. Для решения этой проблемы разработано множество протоколов, которые и будут обсуждаться в данной главе. В литературе широковещательные каналы иногда называют **каналами с множественным доступом (multiaccess channels)** или **каналами с произвольным доступом (random access channels)**.

Протоколы, применяющиеся для определения того, кто будет говорить следующим, относятся к подуровню канального уровня, называемому **MAC (Medium Access Control — управление доступом к среде)**. Подуровень MAC особенно важен в локальных сетях, в частности в беспроводных, так как они по своей природе являются широковещательными каналами. В глобальных сетях, напротив, применяются двухточечные соединения. Исключением являются только спутниковые сети. Поскольку каналы множественного доступа тесно связаны с локальными сетями, в данной главе в основном будут обсуждаться локальные сети, включая некоторые вопросы, напрямую не связанные с темой подуровня MAC. Главной темой будет управление каналом.

Технически подуровень управления доступом к среде является нижней частью канального уровня, поэтому логичнее было бы изучить сначала его, а затем протоколы «точка-точка», рассмотренные в главе 3. Тем не менее большинству людей понять

протоколы, включающие многих участников, легче после того, как хорошо изучены протоколы с двумя участниками. По этой причине при рассмотрении уровней мы слегка отклонились от строгого следования снизу вверх по иерархической лестнице.

4.1. Проблема распределения канала

Центральной проблемой, обсуждаемой в этой главе, является распределение одного широковещательного канала между многочисленными пользователями, претендующими на него. Канал может представлять собой часть беспроводного спектра в некотором географическом регионе или один проводной или оптический канал, к которому присоединено несколько узлов. Это не имеет значения. В обоих случаях канал соединяет каждого пользователя со всеми остальными пользователями, и любой пользователь, полностью нагружающий канал, мешает другим, которые также хотели бы передавать данные.

Сначала мы в общих чертах рассмотрим недостатки статических схем распределения канала в случае неравномерного трафика. Затем изложим ключевые предположения, применяемые для моделирования динамических схем. После этого обсудим несколько примеров таких схем.

4.1.1. Статическое распределение канала

Традиционный способ разделения одного канала, например телефонного кабеля, между многочисленными конкурирующими пользователями — в разделении емкости с помощью одной из схем мультиплексирования или уплотнения каналов, таких как **FDM (Frequency Division Multiplexing — частотное уплотнение)**. При наличии N пользователей полоса пропускания делится на N диапазонов одинаковой ширины, и каждому пользователю предоставляется один из них. Поскольку при такой схеме у каждого оказывается свой личный частотный диапазон, то конфликта между пользователями не возникает. При постоянном небольшом количестве абонентов, каждый из которых отправляет стабильный поток или большие партии трафика, частотное уплотнение предоставляет простой и эффективный механизм распределения. Аналогичный беспроводной пример — радиостанции FM-диапазона. Каждая станция получает часть FM-полосы и использует ее почти постоянно, передавая свой сигнал.

Однако при большом и постоянно меняющемся количестве отправителей данных, или пульсирующем трафике, частотное уплотнение не может обеспечить достаточно эффективное распределение канала. Если количество пользователей в какой-либо момент времени меньше числа диапазонов, на которые разделен спектр частот, то большая часть спектра не используется и тратится попусту. Если, наоборот, количество пользователей окажется больше числа доступных диапазонов, то некоторым придется отказать в доступе к каналу, даже если абоненты, уже захватившие его, почти не будут использовать пропускную способность.

Даже если предположить, что количество пользователей можно каким-то способом удерживать на постоянном уровне, то разделение канала на статические подканалы все равно является неэффективным. Основная проблема здесь состоит в том, что

если какая-то часть пользователей не пользуется каналом, то эта часть спектра просто пропадает. Они сами при этом занимают линию, не передавая ничего, и другим не дают передать данные. Статическое разделение плохо подходит для большинства компьютерных систем, в которых трафик является чрезвычайно неравномерным, с частыми пиками (вполне обычным является отношение пикового трафика к среднему как 1000:1). Следовательно, большую часть времени большая часть каналов не будет использоваться.

То, что характеристики статического частотного уплотнения оказываются неудачными, можно легко продемонстрировать на примере простых вычислений теории массового обслуживания. Для начала сосчитаем среднее время задержки T для отправки кадра по каналу емкостью C бит/с. Предполагается, что кадры прибывают в случайном порядке со средней скоростью λ кадров в секунду. Длина кадров является случайной величиной, среднее значение которой равно $1/\mu$ бита. При таких параметрах скорость обслуживания канала равна μC кадров в секунду. Теория массового обслуживания говорит о том, что

$$T = \frac{1}{\mu C - \lambda}.$$

(Для любознательных: это результат для очереди M/M/1. Требуется, чтобы случайность длительности промежутков между кадрами и длины кадров соответствовали экспоненциальному распределению или, что эквивалентно, являлись результатом пуассоновского процесса.)

В нашем примере C равно 100 Мбит/с, средняя длина кадра $1/\mu$ = 10 000 бит, скорость прибытия кадров λ = 5000 кадров в секунду. Тогда T = 200 мкс. Обратите внимание: если бы мы не учли задержки при формировании очереди и просто посчитали, сколько времени нужно на передачу кадра длиной 10 000 бит по сети с пропускной способностью 100 Мбит/с, то получили бы неправильный ответ: 100 мкс. Это число приемлемо лишь при отсутствии борьбы за канал.

Теперь давайте разделим канал на N независимых подканалов, у каждого из которых будет пропускная способность C/N бит/с. Средняя входная скорость в каждом подканале теперь будет равна λ/N кадров в секунду. Сосчитав новое значение средней задержки T, получим:

$$T_N = \frac{1}{\mu(C/N) - (\lambda/N)} = \frac{N}{\mu C - \lambda} = NT. \qquad (4.1)$$

Это означает, что для поделенного на несколько частей канала значение средней задержки стало в N раз хуже значения, которое было бы в канале, если бы все кадры были каким-то волшебным образом организованы в одну общую очередь. Этот результат также демонстрирует, что в холле банка, где установлены банкоматы, лучше организовать людей в общую очередь, из которой они будут подходить к освободившимся машинам, чем сохранять отдельную очередь к каждому банкомату.

Аргументы, применимые к FDM, можно отнести и к другим способам статического распределения канала. Если использовать временное уплотнение (**TDM, Time Division**

Multiplexing — мультиплексная передача с временным разделением) и выделять каждому пользователю N-й интервал времени, то если интервал не используется абонентом, он просто пропадает. С тем же успехом можно разделить сети физически. Если взять 100-Мбитную сеть и сделать из нее десять 10-Мбитных, статически распределив по ним пользователей, то в результате средняя задержка возрастет с 200 мкс до 2 мс.

Таким образом, ни один статический метод распределения каналов не годится для пульсирующего трафика, поэтому далее мы рассмотрим динамические методы.

4.1.2. Допущения, связанные с динамическим распределением каналов

Прежде чем приступить к рассмотрению многочисленных методов распределения каналов, следует тщательно сформулировать решаемую проблему. В основе всех разработок в данной области лежат следующие пять допущений.

1. **Независимый трафик.** Модель состоит из N независимых **станций** (компьютеров, телефонов, персональных средств связи и т. д.), в каждой из которых программа или пользователь формирует кадры для передачи. Ожидаемое число кадров в интервале времени Δt равно $\lambda\Delta t$, где λ является константой (скорость прибытия новых кадров). Как только кадр сформирован, станция блокируется и ничего не делает, пока кадр не будет успешно передан.
2. **Предположение о едином канале.** Единый канал доступен для всех. Все станции могут передавать и принимать данные по нему. Все станции считаются равными, хотя программно протокол может устанавливать для них различные роли (например, приоритеты).
3. **Наблюдаемые коллизии.** Если два кадра передаются одновременно, они перекрываются по времени, в результате сигнал искажается. Такое событие называется конфликтом, или коллизией. Все станции могут обнаруживать конфликты. Искаженный вследствие конфликта кадр должен быть передан повторно. Других ошибок, кроме тех, которые вызваны конфликтами, нет.
4. **Непрерывное или дискретное время.** Время может считаться непрерывным, и тогда передача кадров может начаться в любой момент. В противном случае время может быть разделено на дискретные интервалы (такты, иногда называемые слотами). Передача кадра может начаться только с началом такта. Один временной интервал может содержать 0, 1 или более кадров, что соответствует свободному интервалу, успешной передаче кадра или коллизии соответственно.
5. **Контроль несущей или отсутствие контроля.** Если контроль несущей выполняется, станции могут определить, свободна или занята линия, до ее использования. Если канал занят, станции не будут пытаться передавать кадры по нему, пока он не освободится. Если контроля несущей нет, то станции не могут определить, свободна или занята линия, пока не попытаются ее использовать. Они просто начинают передачу. Только потом они могут определить, была ли передача успешной.

О приведенных выше допущениях следует сказать несколько слов. Первое допущение утверждает, что кадры прибывают независимо друг от друга, как на разные

станции, так и в пределах одной станции, а также, что кадры формируются непредсказуемо, но с постоянной скоростью. В действительности, это не очень хорошая модель сетевого трафика, поскольку хорошо известно, что пакеты прибывают целыми последовательностями в определенные диапазоны временной шкалы (Paxson, Floyd, 1995; Leland и др., 1994). Тем не менее пуассоновские модели, как их часто называют, полезны, так как легко описываются математически. Они помогают анализировать протоколы, составляя общее представление об изменении производительности с течением времени и о разнице между различными реализациями.

Допущение о едином канале является, на самом деле, центральным в данной модели. Никаких внешних каналов связи нет. Станции не могут тянуть руки, привлекая к себе внимание и убеждая учителя спросить их. Поэтому приходится искать лучшее решение.

Оставшиеся три допущения зависят от инженерной реализации системы, поэтому при изучении конкретных протоколов мы будем указывать, какие допущения следует считать верными.

Допущение о коллизиях является основным. Станциям необходим способ обнаружения коллизий, если они собираются повторно пересылать кадры, а не мириться с их потерей. Для проводных каналов можно использовать оборудование, умеющее определять коллизии. В этом случае станции заранее обрывают передачу, чтобы не засорять канал. В беспроводных каналах распознавать коллизии намного сложнее; об их возникновении приходится узнавать по факту, когда не прибывает ожидаемый кадр подтверждения. Также возможно успешное получение некоторых кадров, попавших в коллизию, — это зависит от типа сигнала и оборудования на получающей стороне. Подобные ситуации встречаются нечасто, поэтому будем предполагать, что все кадры, участвующие в коллизии, попросту теряются. Кроме того, мы познакомимся с протоколами, специально предназначенными для предотвращения коллизий, а не решения создаваемых ими проблем.

Причина, почему для времени существует два альтернативных допущения, заключается в том, что дискретное время помогает иногда повышать производительность. Однако использующие его станции должны синхронизироваться с главными часами или друг с другом. Это не всегда возможно. Мы рассмотрим оба варианта. В каждой конкретной системе работает только одно из возможных допущений.

Аналогично этому, контроль несущей также реализован не во всех системах. Проводные сети обычно знают, когда линия занята, однако в беспроводных сетях контроля несущей чаще всего нет, потому что отдельно взятая станция не может «слышать» все остальные из-за разницы частотных диапазонов. Аналогично, в некоторых условиях, когда станция не может напрямую общаться с другими станциями (например, им приходится пересылать информацию через кабельный модем, играющий роль центрального узла), контроль несущей бывает недоступен. Обратите внимание на слово «несущая». В данном случае оно означает электрический сигнал, распространяющийся по каналу.

Для того чтобы избежать недопонимания, стоит заметить, что ни один протокол коллективного доступа не гарантирует надежную доставку. Даже в случае отсутствия коллизий получатель может по каким-то причинам неправильно скопировать часть кадра. Надежность обеспечивают другие составляющие канального или более высоких уровней.

4.2. Протоколы коллективного доступа

Известно множество алгоритмов коллективного доступа. В следующих разделах будут рассмотрены наиболее интересные алгоритмы и даны примеры их применения на практике.

4.2.1. Система ALOHA

История нашего первого MAC начинается на нетронутых цивилизацией Гавайях в 1970-х годах. В данном случае «нетронутые цивилизацией» означает «не имеющие рабочей телефонной системы». Это не упрощало жизнь исследователя Нормана Абрамсона (Norman Abramson) и его коллег из Гавайского университета, которые пытались подключить пользователей на удаленных островах к главному компьютеру в Гонолулу. Идея протянуть кабели по дну Тихого океана даже не рассматривалась, так что исследователи искали другое решение.

Найденное решение основывалось на использовании радиосистемы ближнего радиуса действия. Терминал каждого пользователя передавал кадры на центральный компьютер в пределах общей полосы частот. Также присутствовал простой и элегантный метод решения проблемы распределения каналов. Их труды впоследствии стали основой многих исследований (Schwartz, Abramson, 2009). Хотя в работе Абрамсона, получившей название системы ALOHA, использовалась широковещательная радиосвязь со стационарными передатчиками, основная идея применима к любой системе, в которой независимые пользователи соревнуются за право использования одного общего канала.

В данном разделе мы рассмотрим две версии системы ALOHA: чистую и дискретную. Они отличаются тем, непрерывно ли время (чистая версия) или делится на дискретные интервалы, в которые должны помещаться все кадры.

Чистая система ALOHA

В основе системы ALOHA лежит простая идея: разрешить пользователям передачу, как только у них появляются данные для отсылки. Конечно, при этом будут столкновения, и столкнувшиеся кадры будут разрушены. Отправителям необходимо уметь обнаруживать такие ситуации. В системе ALOHA, после того как каждая станция отправляет свой кадр центральному компьютеру, этот компьютер рассылает полученный кадр на все остальные станции. Отправитель прослушивает широковещательную передачу, чтобы понять, насколько успешной была передача. В других системах, таких как проводные локальные сети, у отправителя может быть возможность распознавать коллизии во время передачи.

Если кадр был уничтожен, отправитель просто выжидает некоторое случайное время и пытается переслать этот кадр снова. Время ожидания должно быть случайным. В противном случае, при равных фиксированных интервалах времени ожидания коллизии будут повторяться снова и снова. Системы, в которых несколько пользователей использует один общий канал таким способом, что время от времени возникают конфликты, называются **системами с конкуренцией**.

На рис. 4.1 показан пример формирования кадров в системе ALOHA. Все кадры на нашем рисунке имеют один размер, так как при этом пропускная способность системы становится максимальной, именно за счет единого фиксированного размера кадров.

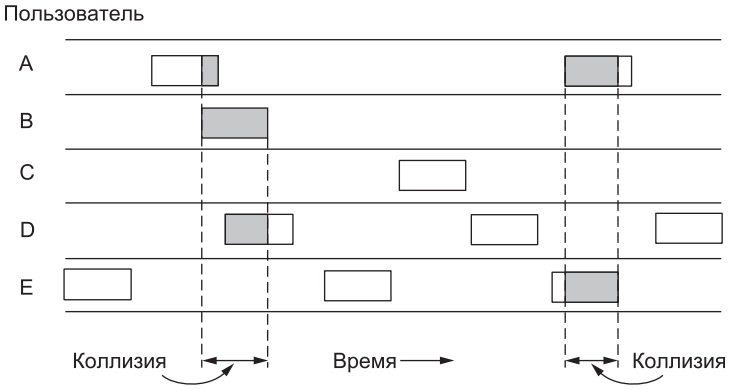


Рис. 4.1. В чистой системе ALOHA кадры передаются в абсолютно произвольное время

Когда два кадра одновременно пытаются занять канал, они сталкиваются, и происходит коллизия (как видно на рис. 4.1). Оба кадра искажаются. Даже если только один первый бит второго кадра перекрывается с последним битом первого кадра, оба кадра уничтожаются полностью (их контрольные суммы не совпадут с правильными значениями). При этом оба кадра должны быть переданы позднее повторно. Контрольная сумма не может (и не должна) отличать полную потерю информации от частичной. Потеря есть потеря.

Самым интересным в данной ситуации является вопрос об эффективности канала системы ALOHA. Другими словами, какая часть всех передаваемых кадров способна избежать коллизий при любых обстоятельствах? Сначала рассмотрим бесконечное множество пользователей, сидящих за своими компьютерами (станциями). Пользователь всегда находится в одном из двух состояний: ввод с клавиатуры и ожидание. Вначале все пользователи находятся в состоянии ввода. Закончив набор строки, пользователь перестает вводить текст, ожидая ответа. В это время станция передает кадр, содержащий набранную строку, по общему каналу на центральный компьютер и опрашивает канал, проверяя успешность передачи кадра. Если кадр передан успешно, пользователь видит ответ и продолжает набор. В противном случае пользователь ждет, пока кадр не будет передан повторно, и это может происходить несколько раз.

Пусть «время кадра» означает интервал времени, требуемый для передачи стандартного кадра фиксированной длины (то есть длину кадра, деленную на скорость передачи данных). На данный момент мы предполагаем, что новые кадры, порождаемые станциями, хорошо распределены по Пуассону со средним значением N кадров за время кадра. (Допущение о бесконечном количестве пользователей необходимо для того, чтобы гарантировать, что величина N не станет уменьшаться по мере блокирования пользователей.) Если $N > 1$, это означает, что сообщество пользователей формирует кадры с большей скоростью, чем может быть передано по каналу, и почти каждый кадр будет страдать от столкновений. Мы будем предполагать, что $0 < N < 1$.

Помимо новых кадров, станции формируют повторные передачи кадров, пострадавших от столкновений. Допустим также, что старые и новые кадры хорошо распределены по Пуассону со средним значением G кадров за время кадра. Очевидно, что $G \geqslant N$. При малой загрузке канала (то есть при $N \approx 0$) столкновений будет мало, поэтому мало будет и повторных передач, то есть $G \approx N$. При большой загрузке канала столкновений будет много, а следовательно, $G > N$. Какая бы ни была нагрузка, производительность канала S будет равна предлагаемой загрузке G, умноженной на вероятность успешной передачи, то есть $S = GP_0$, где P_0 — вероятность того, что кадр не пострадает в результате коллизии.

Кадр не пострадает от коллизии в том случае, если в течение интервала времени его передачи не будет послано больше ни одного кадра, как показано на рис. 4.2. При каких условиях затененный кадр будет передан без повреждений? Пусть t — это время, требуемое для передачи кадра. Если какой-либо пользователь сформирует кадр в интервале времени между t_0 и $t_0 + t$, то конец этого кадра столкнется с началом затененного кадра. При этом судьба затененного кадра предрешена еще до того, как будет послан его первый бит, однако, поскольку в чистой системе ALOHA станции не прослушивают канал до начала передачи, у них нет способа узнать, что канал занят и по нему уже передается кадр. Аналогичным образом, любой другой кадр, передача которого начнется в интервале от $t_0 + t$ до $t_0 + 2t$, столкнется с концом затененного кадра.

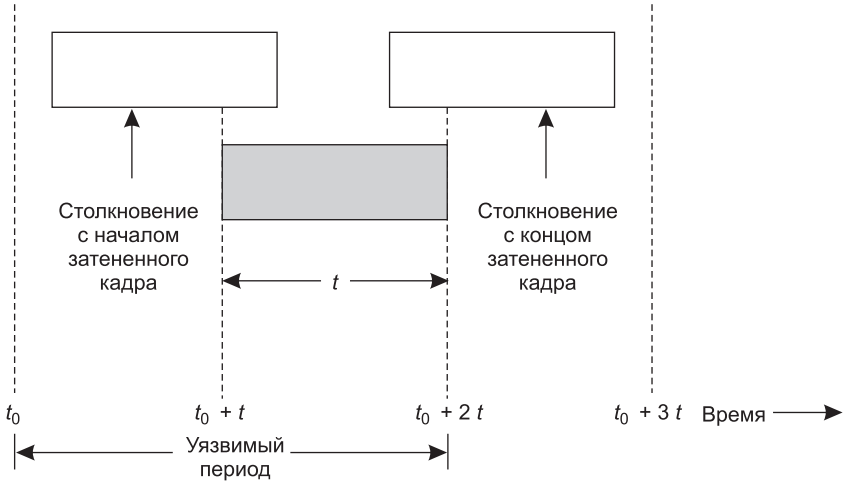


Рис. 4.2. Уязвимый период времени для затененного кадра

Вероятность того, что в течение времени кадра, когда ожидается G кадров, будет сформировано k кадров, можно вычислить по формуле распределения Пуассона:

$$\Pr[k] = \frac{G^k e^{-G}}{k!} \tag{4.2}$$

Таким образом, вероятность формирования нуля кадров в течение этого интервала времени равна e^{-G}. Среднее количество кадров, сформированных за интервал времени

длиной в два кадра, равно $2G$. Вероятность того, что никто не начнет передачу в течение всего уязвимого периода, равна $P_0 = e^{-2G}$. Учитывая, что $S = GP_0$, получаем:

$$S = Ge^{-2G}.$$

Зависимость производительности канала от предлагаемого трафика показана на рис. 4.3. Максимальная производительность достигает значения $S = 1/(2e)$, что приблизительно равно 0,184, при $G = 0{,}5$. Другими словами, лучшее, на что мы можем надеяться, — это использовать канал на 18 %. Этот результат несколько разочаровывает, однако в случае, когда каждый передает, когда хочет, трудно ожидать стопроцентного успеха.

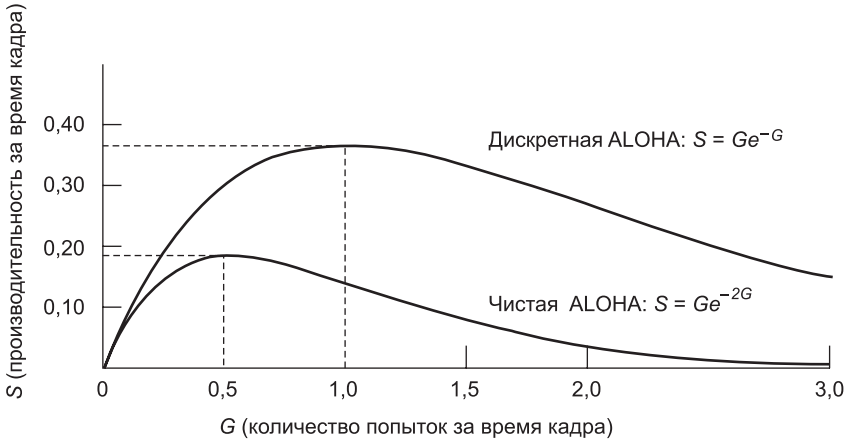


Рис. 4.3. Зависимость производительности канала от предлагаемого трафика для систем ALOHA

Дискретная система ALOHA

Вскоре после появления на сцене системы ALOHA Робертс (Roberts, 1972) опубликовал описание метода, позволяющего удвоить производительность систем ALOHA. Его предложение заключалось в разделении времени на дискретные интервалы, называемые **слотами** (или **тактами**), соответствующие времени одного кадра. При таком подходе пользователи должны согласиться с определенными временными ограничениями. Одним из способов достижения синхронизации является установка специальной станции, испускающей синхронизирующий сигнал в начале каждого интервала.

В системе Робертса, известной под названием **дискретная ALOHA**, в отличие от **чистой системы ALOHA** Абрамсона, станция не может начинать передачу сразу после ввода пользователем строки. Вместо этого она должна дождаться начала нового такта. Таким образом, система ALOHA с непрерывным временем превращается в дискретную. Уязвимый временной интервал теперь становится в два раза короче. Чтобы понять это, взгляните на рис. 4.3 и представьте, какие теперь возможны коллизии. Вероятность отсутствия передачи по каналу за тот же интервал времени, в течение которого передается тестовый кадр, равна e^{-G}. В результате получаем:

$$S = Ge^{-G}. \tag{4.3}$$

Как видно из рис. 4.3, дискретная система ALOHA имеет пик при $G = 1$. При этом производительность канала составляет $S = 1/e$, что приблизительно равно 0,368, то есть в два раза больше, чем в чистой системе ALOHA. Если система работает при условии $G = 1$, то вероятность появления пустого слота равна 0,368 (из выражения 4.2). Для дискретной системы ALOHA в оптимальной ситуации 37 % интервалов будут пустыми, 37 % — с успешно переданными кадрами и 26 % — со столкнувшимися кадрами. При увеличении количества попыток передачи в единицу времени G количество пустых интервалов уменьшается, но увеличивается количество конфликтных интервалов. Чтобы увидеть, насколько быстро растет количество конфликтных интервалов, рассмотрим передачу тестового кадра. Вероятность того, что он избежит столкновения, равна e^{-G}. Фактически это вероятность того, что все остальные станции будут молчать в течение данного тактового интервала. Таким образом, вероятность столкновения равна $1 - e^{-G}$. Вероятность передачи кадра ровно за k попыток (то есть после $k - 1$ столкновения, за которыми последует успешная передача) равна:

$$P_k = e^{-G}(1 - e^{-G})^{k-1}.$$

Ожидаемое число попыток передачи для одной строки, введенной на терминале, равно:

$$E = \sum_{k-1}^{\infty} kP_k = \sum_{k-1}^{\infty} k_k e^{-G}(1 - e^{-G})^{k-1} = e^{G}.$$

Поскольку число попыток передачи для одного кадра E экспоненциально зависит от количества попыток передачи в единицу времени G, небольшое увеличение нагрузки в канале может сильно снизить его производительность.

Дискретная система ALOHA чрезвычайно важна по одной причине, которая на первый взгляд не кажется очевидной. Она появилась в 1970-х годах, применялась в некоторых экспериментальных системах, затем была почти забыта. Когда был изобретен метод доступа в Интернет по кабельным сетям, вновь возникла проблема распределения единственного канала между большим числом конкурирующих абонентов. Тогда с полок достали запыленные описания дискретной системы ALOHA. Позднее в ситуации, когда несколько тегов RFID пытались общаться с одним считывателем RFID, возник еще один вариант старой проблемы. Дискретная система ALOHA, приправленная несколькими другими идеями, снова сумела спасти ситуацию. Не раз уже было так, что вполне работоспособные протоколы и методы оказывались невостребованными по политическим причинам (например, когда какая-нибудь крупная компания выражала желание, чтобы все на свете использовали исключительно ее продукцию) или из-за постоянно меняющихся технологий. Однако по прошествии многих лет какой-нибудь мудрый человек вспоминал о существовании одного древнего метода, способного решить современную проблему. По этой причине мы изучим в данной главе ряд элегантных протоколов, которые сейчас широко не используются, но запросто могут оказаться востребованными в будущем, — если, конечно, об их существовании будет знать достаточное количество разработчиков сетей. Разумеется, мы изучим и многие протоколы, используемые в настоящее время.

4.2.2. Протоколы множественного доступа с контролем несущей

В дискретной системе ALOHA максимальный коэффициент использования канала, который может быть достигнут, равен $1/e$. Такой скромный результат неудивителен, поскольку станции передают данные, когда хотят, не считаясь с тем, что делают остальные станции. В такой системе неизбежно возникает большое количество коллизий. Однако в локальных сетях можно организовать процесс таким образом, что станции будут учитывать поведение друг друга. За счет этого можно достичь значения коэффициента использования канала значительно большего, чем $1/e$. В данном разделе мы рассмотрим некоторые протоколы, позволяющие улучшить производительность канала.

Протоколы, в которых станции прослушивают среду передачи данных и действуют в соответствии с этим, называются **протоколами с контролем несущей**. Было разработано много таких протоколов и их давным-давно подробно проанализировали. Например, см. работу Кляйнрок и Тобаги (Kleinrock, Tobagi, 1975). Ниже мы рассмотрим несколько версий протоколов с контролем несущей.

Настойчивый и ненастойчивый CSMA

Первый протокол с опросом несущей, который мы рассмотрим, называется **1-настойчивый протокол CSMA (Carrier-Sense Multiple Access — множественный доступ с контролем несущей)**. Длинноватое название для простейшей схемы CSMA. Когда у станции появляются данные для передачи, она сначала прослушивает канал, проверяя, свободен он или занят. Если канал бездействует, то станция отправляет данные. В противном случае, когда канал занят, станция ждет, пока он освободится. Затем станция передает кадр. Если происходит столкновение, станция ждет в течение случайного интервала времени, затем снова прослушивает канал и, если он свободен, пытается передать кадр еще раз. Такой протокол называется протоколом CSMA с настойчивостью 1, так как станция передает кадр с вероятностью 1, как только обнаружит, что канал свободен.

От этой схемы можно было бы ожидать, что коллизий вообще происходить не будет, за исключением редких случаев одновременной отправки, но это не так. Если две станции придут в состояние готовности в то время, когда передает какая-то третья станция, обе будут ждать, пока она не закончит передачу, после чего сами одновременно станут передавать, и в результате произойдет столкновение. Если бы они не были столь нетерпеливы, количество столкновений было бы меньшим.

Если копнуть чуть глубже, то на количество коллизий сильное влияние оказывает задержка распространения сигнала. Существует небольшая вероятность того, что как только станция начнет передачу, другая станция также окажется готовой к передаче и опросит канал. Если сигнал от первой станции еще не успел достичь второй станции, вторая станция решит, что канал свободен, и также начнет передачу, результатом чего будет коллизия. Вероятность зависит от числа кадров, умещающихся в канал, или от показателя «полоса пропускания, умноженная на задержку» для данного канала. Если в канал умещается лишь небольшая часть кадра, как бывает в большинстве локальных сетей, где задержка распространения невелика, то и шанс коллизии мал. Чем больше

время распространения сигнала, тем выше вероятность столкновений и ниже производительность протокола.

Однако даже такая система значительно лучше чистой системы ALOHA, так как обе станции воздерживаются от передачи, пока передает третья станция. То же самое можно сказать о дискретной системе ALOHA.

Вторым протоколом с опросом несущей является **ненастойчивый протокол CSMA**. В данном протоколе предпринята попытка сдержать стремление станций начинать передачу, как только освобождается канал. Как и выше, прежде чем начать передачу, станция опрашивает канал. Если никто не передает в данный момент по каналу, станция начинает передачу сама. Однако если канал занят, станция не ждет освобождения канала, постоянно прослушивая его и пытаясь захватить сразу, как только он освободится, как в предыдущем протоколе. Вместо этого станция ждет в течение случайного интервала времени, а затем снова прослушивает линию. Очевидно, данный алгоритм должен привести к лучшему использованию канала и к большим интервалам ожидания, чем протокол CSMA с настойчивостью 1.

Наконец, третий протокол, который мы рассмотрим, это **протокол CSMA с настойчивостью p**. Он применяется в дискретных каналах и работает следующим образом. Когда станция готова передавать, она опрашивает канал. Если канал свободен, она с вероятностью p начинает передачу. С вероятностью $q = 1 - p$ она отказывается от передачи и ждет начала следующего такта. Этот процесс повторяется до тех пор, пока кадр не будет передан или какая-либо другая станция не начнет передачу. В последнем случае станция ведет себя так же, как в случае столкновения. Она ждет в течение случайного интервала времени, после чего начинает все снова. Если при первом прослушивании канала он оказывается занят, станция ждет следующего интервала времени, после чего применяется тот же алгоритм. В IEEE 802.11 применяется улучшенная версия протокола CSMA с настойчивостью p, которую мы рассмотрим далее.

На рис. 4.4 показана расчетная зависимость производительности канала от предлагаемого потока кадров для всех трех протоколов, а также для чистой и дискретной систем ALOHA.

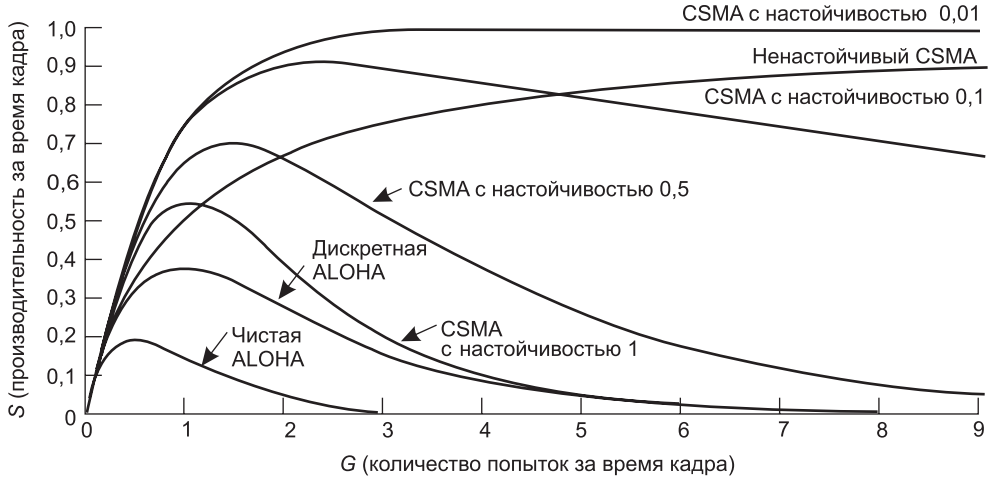


Рис. 4.4. Сравнение использования канала в зависимости от его загрузки для различных протоколов коллективного доступа

Протокол CSMA с обнаружением коллизий

Настойчивый и ненастойчивый протоколы CSMA, несомненно, являются улучшениями системы ALOHA, поскольку они гарантируют, что никакая станция не начнет передачу, если она определит, что канал уже занят. Однако если две станции, обнаружив, что канал свободен, одновременно начали передачу, столкновение все равно произойдет. Еще одно улучшение — способность станций быстро распознавать коллизию и немедленно прекращать передачу (а не завершать ее), так как данные все равно искажены. Эта стратегия экономит время, и улучшается производительность канала.

Такой протокол, называемый **CSMA/CD (Carrier-Sense Multiple Access with Collision Detection — множественный доступ с контролем несущей и обнаружением коллизий)**, является основой чрезвычайно популярных ЛВС Ethernet, поэтому мы уделим некоторое время более или менее подробному рассмотрению CSMA/CD. Важно понимать, что распознавание коллизий представляет собой аналоговый процесс. Оборудование станции должно «прослушивать» канал во время передачи. Если оно считывает сигнал и понимает, что он отличается от пересылаемого, то сразу понятно — произошла коллизия. Следствие таково, что полученный сигнал не обязательно должен идеально совпадать с отправленным (что может быть сложно в беспроводных сетях — принимаемый сигнал нередко в 1 000 000 раз слабее передаваемого) и что необходимо выбирать такой способ уплотнения, который позволит распознавать коллизии (например, коллизию двух 0-вольтовых сигналов распознать практически невозможно).

В протоколе CSMA/CD, так же как и во многих других протоколах локальных сетей, применяется концептуальная модель, показанная на рис. 4.5. В момент времени t_0 одна из станций закончила передачу кадра. Все остальные станции, готовые к передаче, теперь могут попытаться передать свои кадры. Если две станции или более одновременно начнут передачу, то произойдет столкновение. Обнаружив коллизию, станция прекращает передачу, ждет случайный период времени, после чего пытается снова, при условии, что к этому моменту не начала передачу другая станция. Таким образом, наша модель протокола CSMA/CD будет состоять из чередования периодов конкуренции и передачи, а также периодов простоя канала (когда все станции молчат).

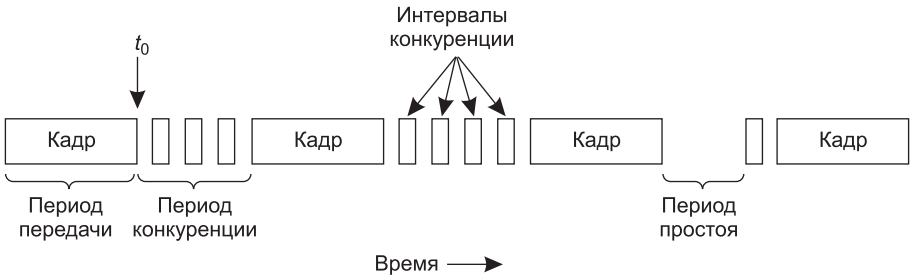


Рис. 4.5. Протокол CSMA/CD может находиться в одном из трех состояний: конкуренции, передачи и простоя

Рассмотрим более подробно алгоритм борьбы за право передачи по каналу. Предположим, две станции одновременно начали передачу в момент времени t_0. Сколько

понадобится времени на то, чтобы они поняли, что произошло столкновение? От ответа на этот вопрос зависит длина периода конкуренции, а следовательно, величина задержки и производительность канала.

Минимальное время обнаружения конфликта равно времени распространения сигнала от одной станции до другой. Исходя из этих рассуждений, можно предположить, что станция, которая не слышит столкновения в течение времени, требуемого для прохождения сигнала по всему кабелю, может быть уверена, что ей удалось захватить кабель. Под термином «захватить» имеется в виду, что все остальные станции знают, что эта станция передает, и не будут сами пытаться передавать. Однако такое заключение неверно.

Рассмотрим следующий сценарий наихудшей ситуации. Пусть время, необходимое для прохождения сигнала между двумя самыми дальними станциями, равно τ. В момент времени t_0 одна из станций начинает передачу. Через интервал времени $\tau - \varepsilon$, за мгновение до того, как сигнал достигнет самой дальней станции, та станция также начинает передавать. Конечно, почти мгновенно она обнаруживает столкновение и останавливается, но всплеск шума, вызванный столкновением, достигает передающей станции только через интервал времени $2\tau - \varepsilon$ с момента начала передачи. Другими словами, станция не может быть уверена в том, что захватила канал, до тех пор пока не пройдет интервал времени 2τ с момента начала передачи.

Понимая это, конкурирование CSMA/CD можно рассматривать как дискретную систему ALOHA с шириной интервала 2τ. В коаксиальном кабеле длиной 1 км $\tau \approx 5$ мкс. Различие между CSMA/CD и дискретной системой ALOHA состоит в том, что в первом случае за слотом, в течение которого передачу осуществляет только одна станция (то есть когда канал захвачен), следует передача оставшейся части кадра. Это позволит значительно улучшить производительность, если время кадра будет намного больше времени распространения сигнала по каналу.

4.2.3. Протоколы без столкновений

Хотя в протоколе CSMA/CD столкновения не могут происходить после того, как станция захватывает канал, они могут случаться в период конкуренции. Эти столкновения снижают производительность системы, особенно когда произведение полосы пропускания на значение задержки велико, то есть при большой длине кабеля (и больших τ) и коротких кадрах. Коллизии не только уменьшают пропускную способность, они делают время пересылки кадра непостоянным, что очень плохо для трафика, передаваемого в режиме реального времени, такого как голосовые данные по протоколу IP. Метод CSMA/CD оказывается не универсальным.

В данном разделе мы рассмотрим протоколы, которые решают проблему борьбы за право занять канал, причем делают это даже без периода конкуренции. Большинство из них в крупных системах сегодня не используются, но в такой изменчивой отрасли всегда хорошо иметь про запас несколько протоколов с великолепными свойствами, которые можно будет применить в будущем.

В описываемых ниже протоколах предполагается наличие N станций, у каждой из которых запрограммирован постоянный уникальный адрес в пределах от 0 до $N - 1$. То что некоторые станции могут часть времени оставаться пассивными, роли не играет.

Также предполагается, что задержка распространения сигнала пренебрежимо мала. Главный вопрос остается неизменным: какой станции будет предоставлен канал после передачи данного кадра? Мы будем по-прежнему использовать модель, изображенную на рис. 4.5, с ее дискретными интервалами конкуренции.

Протокол битовой карты

В первом протоколе без столкновений, который мы рассмотрим, называющемся **основным методом битовой карты (basic bit-map method)**, каждый период конкуренции состоит ровно из N временных интервалов. Если у станции 0 есть кадр для передачи, она передает единичный бит во время 0-го интервала. Другим станциям не разрешается передача в это время. Во время интервала 1 станция 1 также сообщает, есть ли у нее кадр для передачи, передавая бит 1 или 0. В результате к окончанию интервала N все N станций знают, кто хочет передавать. В этот момент они начинают передачу в соответствии со своим порядком номеров (на рис. 4.6 приведен пример для $N = 8$).

Рис. 4.6. Базовый протокол битовой карты

Поскольку все знают, чья очередь передавать, столкновений нет. После того как последняя станция передает свой кадр, что все станции отслеживают, прослушивая линию, начинается новый период подачи заявок из N интервалов. Если станция переходит в состояние готовности (получает кадр для передачи) сразу после того, как она отказалась от передачи, это значит, что ей не повезло и она должна ждать следующего цикла.

Протоколы, в которых намерение передавать объявляется всем перед самой передачей, называются **протоколами с резервированием (reservation protocols)**, так как они заранее резервируют канал для определенной станции, предотвращая коллизии. Оценим производительность такого протокола. Для удобства будем измерять время в однобитовых интервалах периода подачи заявок, при этом кадр данных состоит из d единиц времени.

При слабой загрузке канала битовая карта просто будет повторяться снова и снова, изредка перемежаясь кадрами. Рассмотрим эту ситуацию с точки зрения станции с небольшим номером, например 0 или 1. Обычно в тот момент, когда у нее возникает потребность в передаче, текущий интервал времени уже находится где-то в середине битовой карты. В среднем станция будет ждать $N/2$ интервалов до окончания текущего периода резервирования и еще N интервалов следующего (своего) периода резервирования, не считая кадров, передаваемых между двумя этими периодами, прежде чем она сможет начать передачу.

Перспективы станций с большими номерами более радужны. В среднем время ожидания передачи составит половину цикла ($N/2$ однобитовых интервалов). Станциям

с большими номерами редко приходится ждать следующего цикла. Поскольку станциям с небольшими номерами приходится ждать в среднем $1{,}5N$ интервала, а станциям с большими номерами — $N/2$ интервалов, среднее время ожидания для всех станций составляет N интервалов.

При низкой загрузке канала его производительность легко сосчитать. Накладные расходы на кадр составляют N бит, и при длине кадра в d бит эффективность равна $d/(N + d)$.

При сильной загруженности канала, когда все станции хотят что-то передать, период подачи заявок из N бит чередуется с N кадрами. При этом накладные расходы на передачу одного кадра составляют всего один бит, а эффективность равна $d/(d + 1)$. Среднее время задержки для кадра будет равно сумме времени ожидания в очереди внутри своей станции и дополнительных $(N – 1)d + N$ однобитовых интервалов, когда он попадет в начало своей внутренней очереди. Этот интервал указывает, как долго станции приходится ожидать завершения отправки кадра всеми остальными станциями и очередного получения битовой карты.

Передача маркера

Смысл протокола битовой карты в том, что он позволяет каждой станции передавать данные по очереди в заранее определенном порядке. Другой способ, аналогичный этому, основан на передаче небольшого сообщения, называемого **маркером (token)**, от одной станции к следующей в том же самом заранее определенном порядке. Маркер представляет собой разрешение на отправку. Если на станции в очереди находится кадр, готовый к пересылке, и станция получает маркер, она имеет право отправить кадр, прежде чем передавать маркер следующей станции. Если кадров для отправки нет, то она просто передает маркер.

В протоколе **маркерного кольца (token ring)** для определения порядка, в котором станции отправляют данные, используется топология сети. Станции подключены одна к другой, образуя простое кольцо. Таким образом, передача маркера заключается в получении его с одного направления и пересылке в противоположном, как видно на рис. 4.7. Кадры передаются в том же направлении, что и маркер. Они путешествуют по кольцу, проходя по всем станциям, которые оказываются на их пути. Однако для того чтобы кадр не циркулировал вечно (как маркер), какая-то станция должна извлечь его из кольца. Это может быть либо первоначальный отправитель (если кадр прошел полный цикл), либо станция-получатель.

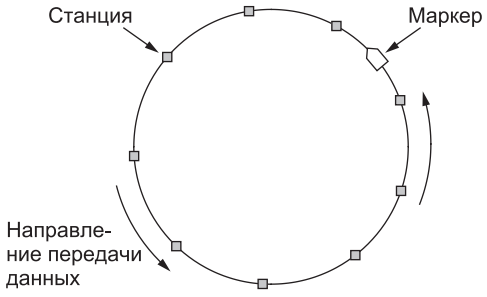


Рис. 4.7. Маркерное кольцо

Обратите внимание, что для реализации передачи маркера физическое кольцо не требуется. Канал, соединяющий станции, может иметь форму одной длинной шины. Станции просто пересылают маркер по шине соседям в предопределенном порядке. Наличие маркера позволяет станции использовать шину для отправки одного кадра, как и раньше. Такой протокол называется **маркерной шиной (token bus)**.

Производительность протокола с передачей маркера схожа с производительностью протокола с битовой картой, хотя периоды конкуренции и кадры одного цикла здесь перемешаны. После отправки кадра каждая станция должна подождать, пока все N станций (включая ее саму) передадут маркер своим соседям, и, кроме этого, $N - 1$ станция отправит кадры (если у них имеются данные для отправки). Тонкая разница заключается в том, что так как все позиции в цикле эквивалентны, никаких отклонений для сильно или слабо загруженных станций нет. В маркерном кольце, прежде чем протокол перейдет на следующий шаг, каждая станция также отправляет маркер только к соседней станции. Маркеру не нужно посещать все станции, для того чтобы протокол продвинулся на шаг вперед.

Протоколы MAC на базе маркерных колец появляются с определенной периодичностью. Один из ранних протоколов (который назывался Token Ring, то есть «Маркерное кольцо» и стандартизирован в IEEE 802.5) в 1980-е годы был популярен в качестве альтернативы классическому Ethernet. В 1990-е годы намного более быстрое маркерное кольцо под названием **FDDI (Fiber Distributed Data Interface, волоконно-оптический распределенный интерфейс данных)** потерпело поражение от коммутируемого Ethernet. В 2000-е маркерное кольцо **RPR (Resilient Packet Ring, отказоустойчивое пакетное кольцо)**, определенное в стандарте IEEE 802.17, стандартизирует множество вариантов кольцевых сетей, применяемых в городских условиях поставщиками услуг Интернета. Интересно, что появится после 2010 года.

Двоичный обратный отсчет

Недостатком базового протокола битовой карты, а также протокола с передачей маркера являются накладные расходы в один бит на станцию, из-за чего они плохо масштабируются на большие сети с тысячами станций. Используя двоичный адрес станции, можно улучшить эффективность канала. Станция, желающая занять канал, объявляет свой адрес в виде битовой строки, начиная со старшего бита. Предполагается, что все адреса станций имеют одинаковую длину. Будучи отправленными одновременно, биты адреса в каждой позиции логически складываются (логическое ИЛИ) средствами канала. Мы будем называть этот протокол **протоколом с двоичным обратным отсчетом (binary countdown)**. Он использовался в сети Datakit (Fraser, 1987). Неявно предполагается, что задержки распространения сигнала пренебрежимо малы, поэтому станции слышат утверждаемые номера практически мгновенно.

Во избежание конфликтов следует применить правило арбитража: как только станция с 0 в старшем бите адреса видит, что в суммарном адресе этот 0 заменился единицей, она сдается и ждет следующего цикла. Например, если станции 0010, 0100, 1001 и 1010 конкурируют за канал, то в первом битовом интервале они передают биты 0, 0, 1 и 1 соответственно. В этом случае суммарный первый бит адреса будет равен 1. Следовательно, станции с номерами 0010 и 0100 считаются проигравшими, а станции 1001 и 1010 продолжают борьбу.

Следующий бит у обеих оставшихся станций равен 0 — таким образом, обе продолжают. Третий бит равен 1, поэтому станция 1001 сдается. Победителем оказывается станция 1010, так как ее адрес наибольший. Выиграв торги, она может начать передачу кадра, после чего начнется новый цикл торгов. Схема протокола показана на рис. 4.8. Данный метод предполагает, что приоритет станции напрямую зависит от ее номера. В некоторых случаях такое жесткое правило может играть положительную, в некоторых — отрицательную роль.

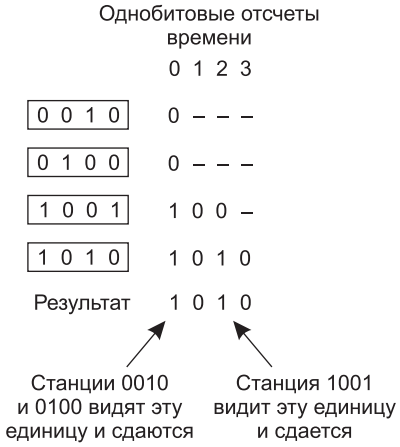


Рис. 4.8. Протокол с двоичным обратным отсчетом. Прочерк означает молчание

Эффективность использования канала при этом методе составляет $d/(d + \log_2 N)$. Однако можно так хитро выбрать формат кадра, что его первое поле будет содержать адрес отправителя, тогда даже эти $\log_2 N$ бит не пропадут зря и эффективность составит 100 %.

Двоичный обратный отсчет является примером простого, элегантного и эффективного протокола, который еще предстоит открыть заново разработчикам будущих сетей. Хочется надеяться, что когда-нибудь он займет свою нишу в сетевых технологиях.

4.2.4. Протоколы с ограниченной конкуренцией

Итак, мы рассмотрели две основные стратегии предоставления доступа к каналу в широковещательных сетях: соревнование, как в CSMA, и бесконфликтные протоколы. Каждую стратегию можно оценить по двум важным параметрам: времени задержки при низкой загрузке канала и эффективность канала при большой загрузке. В условиях низкой загрузки конфликты (то есть чистая или дискретная системы ALOHA) предпочтительнее, так как время задержки в таких системах меньше (коллизий меньше). По мере роста загруженности канала системы со столкновениями становятся все менее привлекательными, поскольку возрастают накладные расходы, связанные с конфликтами. Для бесконфликтных протоколов справедливо обратное. При низкой нагрузке у них относительно высокое время задержки, но по мере увеличения нагрузки эффективность использования канала не уменьшается, как у конфликтных протоколов, а наоборот, возрастает (накладные расходы фиксированные).

Очевидно, было бы неплохо объединить лучшие свойства обеих стратегий и получить протокол, использующий разные стратегии при разной загруженности канала. Такие протоколы мы будем называть **протоколами с ограниченной конкуренцией (limited-contention protocols)**. Они в самом деле существуют, и их рассмотрением мы завершим изучение сетей с опросом носителя.

До сих пор мы рассматривали только симметричные протоколы коллективного доступа, в которых каждая станция пытается получить доступ к каналу с равной вероятностью p. Интересно, что производительность всей системы может быть улучшена при использовании асимметричного протокола, в котором станциям назначаются различные вероятности.

Прежде чем приступить к рассмотрению асимметричных протоколов, давайте кратко рассмотрим производительность в симметричном случае. Предположим, что k станций борются за доступ к каналу. Вероятность передачи каждой станции в каждый интервал времени равна p. Вероятность того, что какая-то станция успешно получит доступ к каналу на данный интервал времени, составляется из вероятности того, что любая станция передает данные с вероятностью p, а все остальные $k - 1$ станций воздерживаются от передачи, каждая с вероятностью $1 - p$. Итоговое значение равно $kp(1 - p)^{k-1}$. Чтобы найти оптимальное значение вероятности p, продифференцируем данное выражение по p, приравняем результат нулю и решим полученное уравнение относительно p. В результате мы получим, что наилучшее значение p равно $1/k$. Заменяя в формуле p на $1/k$, получаем вероятность успеха при оптимальном значении p:

$$P\,[\textit{успех при оптимальной вероятности } p] = \left(\frac{k-1}{k}\right)^{k-1}. \qquad (4.4)$$

Зависимость этой вероятности от количества готовых станций графически показана на рис. 4.9. Для небольшого числа станций значение вероятности успеха является неплохим, однако как только количество станций достигает хотя бы пяти, вероятность снижается почти до асимптотической величины, равной $1/e$.

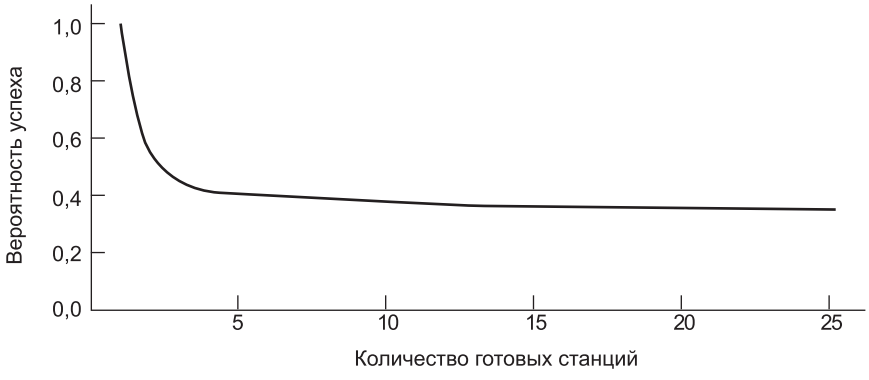


Рис. 4.9. Вероятность получения доступа к каналу в симметричном протоколе

Из рисунка очевидно, что вероятность получения доступа к каналу для какой-либо станции можно увеличить, только снизив конкуренцию за канал. Этим занимаются протоколы с ограниченной конкуренцией. Они сначала делят все станции на группы (необязательно непересекающиеся). Состязаться за интервал 0 разрешается только членам группы 0. Если кто-то из них выигрывает, он получает канал и передает по нему кадр. Если никто из них не хочет передавать или происходит столкновение, члены группы 1 состязаются за интервал 1 и т. д. При соответствующем разбиении на группы конкуренция за каждый интервал времени уменьшается, что увеличивает вероятность его успешного использования (см. левую часть графика).

Вопрос в том, как разбивать станции на группы. Прежде чем обсуждать общий случай, рассмотрим несколько частных случаев. В одном из крайних случаев в каждой группе будет по одной станции. Такое разбиение гарантирует полное отсутствие конфликтов, так как на каждый интервал времени будет претендовать только одна станция. Подобные протоколы уже рассматривались ранее (например, протокол с двоичным обратным отсчетом). Еще одним особым случаем является разбиение на группы, состоящие из двух станций. Вероятность того, что обе станции одновременно начнут передачу в течение одного интервала, равна p^2, и при малых значениях p этим значением можно пренебречь. По мере увеличения количества станций в группах, вероятность столкновений будет возрастать, однако длина битовой карты, необходимой, чтобы перенумеровать все группы, будет уменьшаться. Другим предельным случаем будет одна группа, в которую войдут все станции (то есть дискретная система ALOHA). Нам требуется механизм динамического разбиения станций на группы, с небольшим количеством крупных групп при слабой загруженности канала и большом количестве мелких групп (может быть, даже состоящих из одной станции каждая), когда загруженность канала высока.

Протокол адаптивного прохода по дереву

Одним из простых способов динамического разбиения на группы является алгоритм, разработанный во время Второй мировой войны в армии США для проверки солдат на сифилис (Dorfman, 1943). Брался анализ крови у N солдат. Часть каждого образца помещалась в одну общую пробирку. Этот смешанный образец проверялся на наличие антител. Если антитела не обнаруживались, все солдаты в данной группе объявлялись здоровыми. В противном же случае группа делилась пополам, и каждая половина группы проверялась отдельно. Подобный процесс продолжался до тех пор, пока размер группы не уменьшался до одного солдата.

В компьютерной версии данного алгоритма (Capetanakis, 1979) станции рассматриваются в виде листьев двоичного дерева, как показано на рис. 4.10. В первом временно́м интервале состязания за право передачи участвуют все станции. Если кому-нибудь это удается, то на этом работа алгоритма заканчивается. Если же происходит столкновение, то ко второму этапу состязаний допускается только половина станций, а именно станции, относящиеся к узлу 2 дерева. Если одна из станций успешно захватывает канал, то следующее состязание устраивается для второй половины станций (относящихся к узлу 3 дерева). Если снова происходит конфликт, то к следующему интервалу времени среди состязающихся остается уже четверть станций, относящихся к узлу 4.

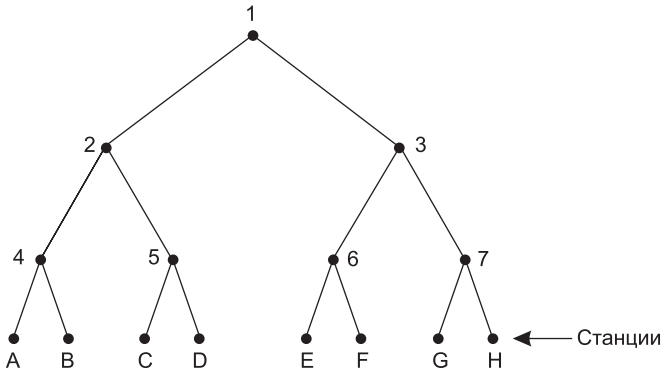


Рис. 4.10. Дерево из восьми станций

Таким образом, если столкновение происходит во время интервала 0, то все дерево сканируется на единичную глубину для поиска готовых станций. Каждый однобитовый слот ассоциируется с определенным узлом дерева. Если происходит столкновение, поиск продолжается для левого и правого дочерних узлов. Если количество станций, претендующих на передачу, равно нулю или единице, поиск в данном узле дерева прекращается, поскольку все готовые станции уже обнаружены.

При сильной загруженности канала вряд ли стоит начинать поиск готовой станции с узла 1, поскольку шансов, что всего одна станция из всех будет претендовать на канал, мало. По той же причине могут быть пропущены также узлы 2 и 3. А на каком уровне дерева следует начинать опрос в общем случае? Очевидно, что чем сильнее загруженность канала, тем на более низком уровне дерева должен начинаться поиск готовых станций. Будем предполагать, что каждая станция довольно точно может оценить q (количество готовых на данный момент станций), например, отслеживая недавний трафик.

Пронумеруем уровни дерева на рис. 4.10 — узел 1 на уровне 0, узлы 2 и 3 на уровне 1 и т. д. Обратите внимание на то, что каждый узел на уровне i включает 2^{-i} часть от всех станций. Если q готовых станций распределены равномерно, то ожидаемое их число ниже узла на уровне i равно $2^{-i}q$. Интуитивно ясно, что оптимальным уровнем для начала поиска будет тот, на котором среднее число конкурирующих в интервале станций равно 1, то есть уровень, на котором $2^{-i}q = 1$. Отсюда $i = \log_2 q$.

Были разработаны многочисленные усовершенствования базового алгоритма — в частности, некоторые детали обсуждаются у Бертсекаса (Bertsekas) и Галлагера (Gallager) в издании 1992 года. Например, рассмотрим случай, при котором передавать хотят только станции G и H. На узле 1 произойдет конфликт, поэтому будет проверен узел 2. Он окажется пустым. Узел 3 проверять нет смысла, так как там гарантированно будет столкновение. (Нам известно, что под узлом 1 находятся 2 или более станций, а так как под узлом 2 нет ни одной станции, то все они должны быть под узлом 3.) Поэтому проверку узла 3 можно пропустить и сразу проверить узел 6. Поскольку под узлом 6 ничего не оказалось, то проверку узла 7 также можно пропустить и проверить узел G.

4.2.5. Протоколы беспроводных локальных сетей

Систему, состоящую из портативных компьютеров, общающихся по радио, можно рассматривать как беспроводную локальную сеть — мы уже обсуждали это выше. Такая локальная сеть — пример сети на базе широковещательного канала. Ее свойства отличаются от свойств проводных локальных сетей, поэтому здесь требуются специальные протоколы управления доступом к среде (MAC). В данном разделе мы познакомимся с некоторыми из этих протоколов. Далее мы также подробнее поговорим о стандарте 802.11 (WiFi).

Распространенная конфигурация беспроводных локальных сетей подразумевает наличие офисного здания с заранее размещенными в нем точками доступа. Все точки доступа соединены друг с другом медным проводом или оптоволоконным кабелем; они рассылают данные на пользовательские станции. Если мощность передатчиков точек доступа и переносных компьютеров настроена так, что диапазон приема составляет около десятка метров, то соседние комнаты становятся единой сотой, а все здание превращается в большую сотовую систему, подобную традиционной сотовой телефонной системе, описанной в главе 2. В отличие от обычной сотовой системы, у каждой соты в данном случае всего один канал, работающий со всеми станциями, находящимися в нем, включая точку доступа. Обычно пропускная способность такого канала составляет от несколько мегабит в секунду до 600 Мбит/с.

Мы уже говорили выше, что обычно беспроводные системы не имеют возможности распознавать коллизии в тот момент, когда они происходят. Принимаемый сигнал на станции может быть очень слабым, возможно, в миллион раз слабее излучаемого. Искать его — то же самое, что искать иголку в стоге сена. Для обнаружения уже случившихся коллизий и других ошибок применяются подтверждения.

Есть и еще одно очень важное различие между проводными локальными сетями и беспроводными. В беспроводной сети у станций иногда нет возможности передавать или получать кадры с других станций из-за ограниченного диапазона радиопередачи. В проводных сетях, если одна станция отправляет кадры, все остальные его получают. Такая особенность приводит к различным сложностям.

Для простоты мы допустим, что каждый передатчик работает в некой фиксированной области, которую можно представить как регион покрытия, имеющий форму круга. Внутри него другая станция может слышать и принимать данные с этой станции. Важно понимать, что на практике регион покрытия будет неправильной формы, так как распространение радиосигналов зависит от среды. Стены и другие препятствия, ослабляющие и отражающие сигналы, приводят к тому, что сила сигнала в разных направлениях меняется. Однако модель с окружностью для наших целей вполне подходит.

Можно наивно попытаться применить в локальных беспроводных сетях протокол **CSMA (Carrier-Sense Multiple Access — множественный доступ с опросом несущей)** — просто прослушивать эфир и осуществлять передачу только тогда, когда он никем не занят. Однако проблема заключается в том, что в действительности имеет значение интерференция на приемнике, а не на передатчике, поэтому этот протокол для беспроводных сетей подходит не очень хорошо. Чтобы наглядно увидеть суть проблемы, рассмотрим рис. 4.11, где показаны четыре беспроводные станции. Для нашей

проблемы не имеет значения, какая из них является точкой доступа, а какая — переносной. Мощность передатчиков такова, что взаимодействовать могут только соседние станции, то есть *A* с *B*, *C* с *B* и *D*, но не с *A*.

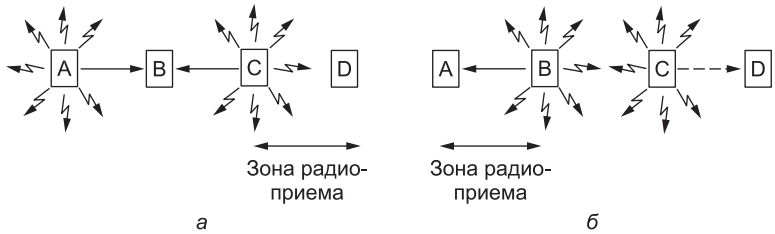


Рис. 4.11. Беспроводная локальная сеть: *а* — *A* и *C* — скрытые станции во время пересылки данных на B; *б* — *B* и *C* — засвеченные станции во время пересылки данных на *A* и *D*

Сначала рассмотрим, что происходит, когда станции *A* и *C* передают данные станции *B*, как изображено на рис. 4.11, *а*. Если станция *A* отправляет данные, а станция *C* сразу же опрашивает канал, то она не будет слышать станцию *A*, поскольку та расположена слишком далеко, и может прийти к неверному выводу о том, что канал свободен и что можно посылать данные станции *B*. Если станция *C* начнет передавать, она будет конфликтовать со станцией *B* и исказит кадр, передаваемый станцией *A*. (Мы предполагаем, что никакая схема по типу CDMA не используется для предоставления нескольких каналов, поэтому из-за коллизий сигналы искажаются и оба кадра разрушаются.) Нам необходим MAC-протокол, который предотвратит такой тип коллизий, ведь это лишняя трата полосы пропускания. Проблема, заключающаяся в том, что одна станция не может слышать возможного конкурента, поскольку конкурент расположен слишком далеко от нее, иногда называется **проблемой скрытой станции (hidden terminal problem)**.

Теперь рассмотрим другую ситуацию: станция *B* передает данные станции *A* в то же время, когда станция *C* хочет начать передачу станции *D*, как показано на рис. 4.11, *б*. Станция *C* при опросе канала слышит выполняемую передачу и может ошибочно предположить, что она не может передавать данные станции *D* (пунктирная стрелка на рисунке). В действительности такая передача создала бы помехи только в зоне от станции *B* до станции *C*, где в данный момент не ведется прием. Нам необходим MAC-протокол, который предотвратит такой тип задержек, ведь это лишняя трата полосы пропускания. Такая ситуация иногда называется **проблемой засвеченной станции (exposed terminal problem)**.

Сложность заключается в том, что перед тем как начать передачу, станции необходимо знать, есть ли какая-нибудь активность в радиодиапазоне вблизи приемника. Протокол CSMA же всего лишь может сообщить об активности вокруг передатчика путем опрашивания несущей. В случае передачи по проводу все сигналы достигают всех станций, поэтому такого различия не наблюдается. Однако во всей системе одновременно только одна станция может вести передачу. В системе с использованием радиосвязи, радиус передачи и приема которой ограничен небольшими зонами, одновременно могут передавать данные несколько станций, если только они передают различным принимающим станциям, находящимся достаточно далеко друг от друга. Нам

нужно, чтобы даже в растущей ячейке одновременная передача не прекращалась — точно так же гости на вечеринке не ждут, пока все замолчат, чтобы начать говорить. Одновременно в большом помещении может происходить несколько разговоров, если только не все пытаются пообщаться с одним и тем же собеседником.

Одним из первых значительных протоколов, разработанных для беспроводных локальных сетей и умеющих справляться с этими проблемами, является **MACA (Multiple Access with Collision Avoidance — множественный доступ с предотвращением коллизий)** (Karn, 1990). Идея, лежащая в основе этого протокола, заключается в том, что отправитель заставляет получателя передать короткий кадр, чтобы окружающие станции могли услышать эту передачу и воздержаться от действий на время, требуемое для приема большого информационного кадра. Эта техника заменяет технику прослушивания несущей.

Протокол MACA проиллюстрирован на рис. 4.12. Рассмотрим ситуацию, в которой станция *A* передает станции *B*. Станция *A* начинает с того, что посылает станции *B* кадр **RTS (Request To Send — запрос на передачу)**, как показано на рис. 4.12, *а*. Этот короткий кадр (30 байт) содержит длину кадра данных, который последует за ним. Затем станция *B* отвечает кадром **CTS (Clear To Send — разрешение передачи)**, как показано на рис. 4.12, *б*. Кадр CTS также содержит длину информационного кадра (скопированную из кадра RTS). Приняв кадр CTS, станция *A* начинает передачу.

Теперь посмотрим, как реагируют станции, которые слышат передачу одного из этих кадров. Любая станция, которая слышит кадр RTS, находится близко к станции *A* и поэтому должна хранить молчание, пока кадр CTS не будет принят станцией *A*. Станции, слышащие кадр CTS, находятся вблизи от станции *B*, следовательно, должны воздержаться от передачи, пока станция *B* не получит кадр данных, длину которого они могут узнать из кадра CTS.

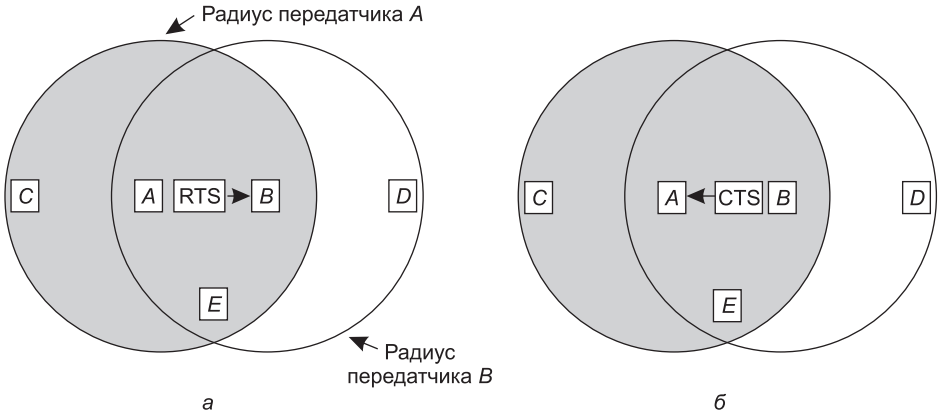


Рис. 4.12. Протокол MACA: *а* — станция *A* посылает кадр RTS станции *B*; *б* — станция *B* отвечает кадром CTS станции *A*

На рис. 4.12 станция *C* находится в зоне станции *A*, но не входит в зону станции *B*. Поэтому она слышит кадр RTS, передаваемый станцией *A*, но не слышит кадр CTS, которым отвечает станция *B*. Поскольку она не интерферирует с кадром CTS, она не

обязана воздерживаться от передачи в то время, пока пересылается информационный кадр. Станция *D*, напротив, находится близко от станции *B*, но далеко от станции *A*. Она не слышит кадра RTS, но слышит кадр CTS, а это означает, что она находится вблизи станции, собирающейся принять кадр с данными. Поэтому ей нельзя вести передачу, пока этот кадр не будет передан. Станция *E* слышит оба управляющих сообщения и так же, как и станция *D*, должна хранить молчание, пока не будет завершена передача информационного кадра.

Несмотря на все меры предосторожности, конфликты все равно могут произойти. Например, станции *B* и *C* могут одновременно послать кадры RTS станции *A*. При этом кадры столкнутся и не будут приняты. В этом случае передатчики, не услышав кадр CTS в установленный срок, ждут случайное время и после этого повторяют попытку.

4.3. Сеть Ethernet

Итак, мы в целом закончили обсуждение общих вопросов, касающихся протоколов распределения канала. Пришло время перейти к практическим приложениям. Большое число технологий для персональных (PAN), локальных (LAN) и общегородских (MAN) сетей стандартизировано в серии стандартов IEEE 802. Некоторые стандарты выжили, некоторые — нет (см. табл. 1.4). Люди, верящие в реинкарнацию, считают, что одним из членов Ассоциации стандартов IEEE является вновь родившийся Чарльз Дарвин, отбраковывающий слабые технологии. В общем-то, действительно выжили сильнейшие. Наиболее важны стандарты 802.3 (Ethernet) и 802.11 (беспроводные ЛВС). Bluetooth (беспроводные персональные сети) развернуты сегодня очень широко, но их описывают другие стандарты, помимо 802.15. О 802.16 (беспроводные региональные сети) говорить всерьез пока не приходится. Вероятно, им будет посвящен раздел в 6-й редакции этой книги.

Мы начнем исследование реальных систем с Ethernet, вероятно, наиболее распространенном типе компьютерных сетей в мире. Существует два типа Ethernet: **классический Ethernet (classic Ethernet)**, который решает проблему множественного доступа с помощью техник, представленных в этой главе; и **коммутируемый Ethernet (switched Ethernet)**, в котором для соединения компьютеров используются устройства под названием коммутаторы. Важно понимать, что хотя в обоих названиях присутствует слово Ethernet, между этими сетями много различий. Классический Ethernet — это воплощение оригинальной задумки; эти сети работали на скоростях от 3 до 10 Мбит/с. Коммутируемый Ethernet — это более высокий уровень; эти сети работают на скоростях 100, 1000 или 10 000 Мбит/с и носят названия Fast Ethernet («быстрый Ethernet»), Gigabit Ethernet («гигабитный Ethernet») и 10-Gigabit Ethernet («10-гигабитный Ethernet»). Сегодня на практике используется только коммутируемый Ethernet.

Мы обсудим эти исторические формы Ethernet в хронологическом порядке, продемонстрировав развитие сети. Так как Ethernet и IEEE 802.3 — это одно и то же (за исключением двух небольших деталей, которые мы вкратце обсудим), то многие используют оба названия. Мы тоже будем говорить то «Ethernet», то «IEEE 802.3». Дополнительную информацию, касающуюся Ethernet, можно найти в книге (Spurgeon, 2000).

4.3.1. Физический уровень классической сети Ethernet

История Ethernet начинается приблизительно во времена ALOHA, когда студент Боб Меткальф получил магистерскую степень в Массачусетском технологическом университете, а потом сменил место жительство, чтобы защитить докторскую в Гарварде. Во время учебы он познакомился с работой Абрамсона. Она так заинтересовала его, что после выпуска из Гарварда он решил провести лето на Гавайях, работая с Абрамсоном, и только после этого перебираться в исследовательский центр Xerox. Оказавшись в исследовательском центре, он увидел то, что впоследствии должно было стать персональным компьютером. Однако машины были изолированы. Используя свои знания о работе Абрамсона, он, вместе с коллегой Дэвидом Боггсом, создал и реализовал первую локальную сеть (Metcalfe, Boggs, 1976). Для нее использовался длинный толстый коаксиальный кабель, а скорость сети составляла 3 Мбит/с.

Они назвали систему **Ethernet** в честь *люминофорного эфира*, через который, как когда-то считалось, распространяются электромагнитные лучи. (Когда в XIX веке британский физик Джеймс Клерк Максвелл обнаружил, что электромагнитное излучение можно описать волновым уравнением, ученые предположили, что пространство должно быть заполнено некой эфирной средой, с помощью которой излучение распространяется. Только после знаменитого эксперимента Майкельсона—Морли, проведенного в 1887 году, физики поняли, что электромагнитное излучение способно распространяться в вакууме.)

Система Xerox Ethernet оказалась настолько успешной, что в 1978 году DEC Intel и Xerox разработали стандарт 10-мегабитного Ethernet, который называется **стандартом DIX (DIX standard)**. С небольшими изменениями в 1983 году стандарт DIX превратился в стандарт IEEE 802.3. К несчастью для Xerox, у этой компании уже к тому моменту была длинная история значительных изобретений (таких как персональный компьютер), которые они не сумели успешно выпустить на рынок (прочитайте об этом в книге «Fumbling the Future», Smith, Alexander, 1988). Когда стало понятно, что у Xerox нет никакой заинтересованности в Ethernet и предложить эта компания может разве что помощь в стандартизации, Меткальф основал собственную компанию, 3Com, и стал продавать адаптеры Ethernet для персональных компьютеров. Были проданы миллионы устройств.

Классический Ethernet — это один длинный кабель, обвивающий здание, к которому подключаются компьютеры. Такая архитектура показана на рис. 4.13. Первый вариант, называемый в народе **толстым Ethernet (thick Ethernet)**, напоминал желтый садовый шланг с маркировкой каждые 2,5 метра — в этих местах подключались компьютеры. (По стандарту 802.3 не *требовалось*, чтобы кабель был желтым, но это *подразумевалось*.) Ему на смену пришел **тонкий Ethernet (thin Ethernet)**; эти кабели лучше гнулись, а соединения выполнялись с помощью стандартных разъемов BNC. Тонкий Ethernet был намного дешевле и проще в установке, но длина сегмента не превышала 185 метров (вместо 500 метров для толстого Ethernet), и каждый сегмент поддерживал не более 30 машин (вместо 100).

Все версии Ethernet имеют ограничения по длине кабелей в сегменте, то есть участкам кабелей без использования усилителя. Для построения сетей больших раз-

меров несколько кабелей соединяются **повторителями (repeaters)**. Повторитель — это устройство физического уровня. Он принимает, усиливает (регенерирует) и передает сигналы в обоих направлениях. С точки зрения программного обеспечения, ряд кабелей, соединенных повторителями, не отличается от сплошного кабеля (отличие заключается только во временной задержке, связанной с повторителями).

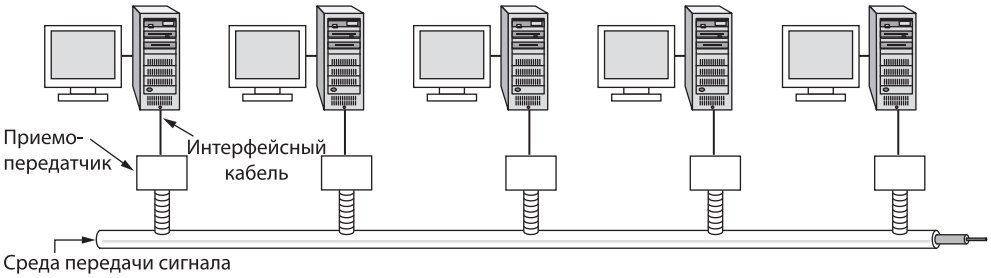


Рис. 4.13. Архитектура классической сети Ethernet

Информация по этим кабелям передается с использованием манчестерского кода. Сеть Ethernet может состоять из большого количества сегментов кабеля и повторителей, однако два приемопередатчика должны располагаться на расстоянии не более 2,5 км и между ними должно быть не более четырех повторителей. Причина такого ограничения лежит в протоколе MAC, о котором мы поговорим далее.

4.3.2. Протокол подуровня управления доступом к среде в классическом Ethernet

Формат кадра, применяемый для отправки данных, показан на рис. 4.14. Сначала идет поле *Preamble* (преамбула, заголовок) длиной 8 байт, которое содержит последовательность 10101010 (за исключением последнего байта, в котором значения последних двух битов равны 11). Последний байт в стандарте 802.3 называется разделителем *Start of Frame* (Начало кадра). Манчестерское кодирование такой последовательности битов дает в результате меандр с частотой 10 МГц и длительностью 6,4 мкс, что позволяет получателю синхронизировать свои часы с часами отправителя. Два последних бита, равных единице, говорят получателю, что сейчас начнется новый кадр.

Затем следуют два адреса: получателя и отправителя. Каждый занимает по 6 байт. Первый передаваемый бит адреса получателя содержит 0 для обычных адресов и 1 для групповых получателей. Групповые адреса позволяют нескольким станциям принимать информацию от одного отправителя. Кадр, отправляемый групповому адресату, может быть получен всеми станциями, входящими в эту группу. Такой механизм называется **групповой рассылкой (multicasting)**. Если адрес состоит только из единиц, то кадр могут принять абсолютно все станции сети. Таким способом осуществляется **широковещание (broadcasting)**. Групповая рассылка более избирательна, но требует некоторых усилий при управлении группами. Широковещание — это более грубая технология, но зато не требует никакой настройки групп.

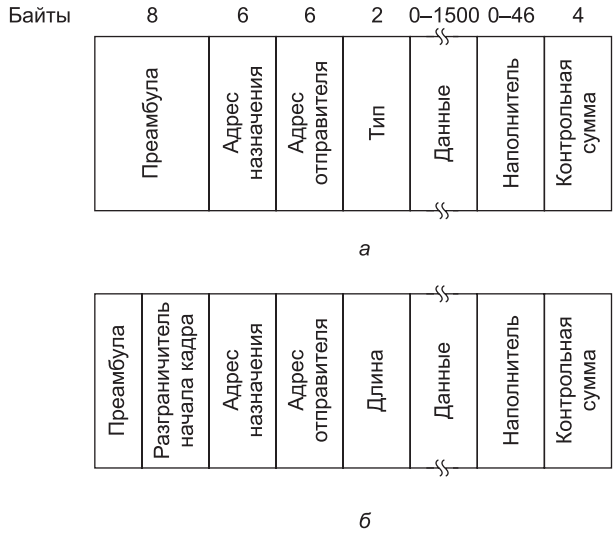


Рис. 4.14. Форматы кадров: *а* — DIX Ethernet; *б* — IEEE 802.3

Интересной особенностью исходных адресов станций является глобальная уникальность. Они централизованно назначаются IEEE, и это гарантирует, что один и тот же глобальный адрес не используется двумя станциями нигде в мире. Идея заключается в том, что каждая станция может быть однозначно идентифицирована по ее 48-битному номеру. Для этого первые 3 байта поля адреса используются для **OUI (Organizationally Unique Identifier**, **организационно уникальный идентификатор)**. Значения этого поля назначаются IEEE и однозначно определяют производителя. Производителям выделяются блоки по 2^{24} адресов. Производитель назначает последние 3 байта адреса и программирует весь адрес в сетевой карте перед тем, как она поступает в продажу.

Затем следует поле *Type* или *Length*, в зависимости от того, относится кадр к стандарту Ethernet или IEEE 802.3. В сетях Ethernet поле *Type* показывает приемнику, что делать с кадром. Дело в том, что одновременно на одной и той же машине может работать несколько протоколов сетевого уровня, поэтому, когда приходит кадр Ethernet, операционная система должна понимать, какому протоколу его передать. Поле *Type* определяет процесс, который должен взять себе кадр. Например, код типа 0x0800 означает, что данные содержат пакет IPv4.

Создатели IEEE 802.3 в своей безграничной мудрости решили, что в этом поле должна передаваться длина кадра. Но поскольку для определения длины необходимо заглянуть внутрь данных, то мы наблюдаем нарушение правил использования сетевых уровней. Разумеется, это означало, что получатель никак не мог выяснить, что же ему делать с входящим кадром. Эту проблему решили путем добавления в данные еще одного заголовка для протокола **LLC (Logical Link Control**, **управление логическим каналом)**. Он занимает 8 байт и передает 2 байта информации о типе протокола.

К сожалению, к моменту опубликования стандарта 802.3 использовалось так много оборудования и программного обеспечения для DIX Ethernet, что лишь немногие про-

изводители и пользователи проявили энтузиазм в отношении переопределения полей *Type* и *Length*. В 1997 году IEEE признал поражение и согласился с обоими способами. К счастью, значения всех полей *Type*, использовавшиеся до 1997 года, были больше 1500 (тогда это значение было установлено в качестве максимального размера поля данных). Теперь правило таково, что любое значение, не превышающее 0x600 (1536), можно интерпретировать как *Length*, а любое число больше 0x600 — как *Type*. Теперь IEEE спокойна, что все используют ее стандарт, причем можно продолжать делать то, что программисты делали и до этого, не утруждая себя мыслями о LLC и не чувствуя вины за нарушение стандарта.

Наконец, за полем *Type* следует поле данных, размер которого ограничен 1500 байт. Такое ограничение было выбрано, в общем-то, произвольно в те времена, когда официально был закреплен стандарт Ethernet. При выборе ссылались на то, что приемопередатчику нужно довольно много оперативной памяти для того, чтобы хранить весь кадр. А память в том далеком 1978 году была еще очень дорогой. Соответственно, увеличение верхней границы размера поля данных привело бы к необходимости установки большего объема памяти, а значит, к удорожанию всего приемопередатчика.

Между тем, кроме верхней границы размера поля данных очень важна и нижняя граница. Поле данных, содержащее 0 байт, вызывает определенные проблемы. Дело в том, что когда приемопередатчик обнаруживает столкновение, он обрезает текущий кадр, а это означает, что отдельные куски кадров постоянно блуждают по кабелю. Чтобы было легче отличить нормальные кадры от мусора, сети Ethernet требуется кадр размером не менее 64 байт (от поля адреса получателя до поля контрольной суммы включительно). Если в кадре содержится меньше 46 байт данных, в него вставляется специальное поле *Pad* (наполнитель), с помощью которого размер кадра доводится до необходимого минимума.

Другой (и даже более важной) целью установки ограничения размера кадра снизу является предотвращение ситуации, когда станция успевает передать короткий кадр раньше, чем его первый бит дойдет до самого дальнего конца кабеля, где он может столкнуться с другим кадром. Эта ситуация показана на рис. 4.15. В момент времени 0 станция *A* на одном конце сети посылает кадр. Пусть время прохождения кадра по кабелю равно τ. За мгновение до того, как кадр достигнет конца кабеля (то есть в момент времени $\tau - \varepsilon$), самая дальняя станция *B* начинает передачу. Когда станция *B* замечает, что получает большую мощность, нежели передает сама, она понимает, что произошло столкновение. Тогда она прекращает передачу и выдает 48-битный шумовой сигнал, предупреждающий остальные станции. Примерно в момент времени 2τ отправитель замечает шумовой сигнал и также прекращает передачу. Затем он выжидает случайное время и пытается возобновить передачу.

Если размер кадра будет слишком маленьким, отправитель закончит передачу прежде, чем получит шумовой сигнал. В этом случае он не сможет понять, произошло это столкновение с его кадром или с каким-то другим, и, следовательно, может предположить, что его кадр был успешно принят. Для предотвращения такой ситуации все кадры должны иметь такую длину, чтобы время их передачи было больше 2τ. Для локальной сети со скоростью передачи 10 Мбит/с при максимальной длине кабеля в 2500 м и наличии четырех повторителей (требование спецификации 802.3) минимальное время передачи одного кадра должно составлять в худшем случае примерно 50 мкс.

Следовательно, длина кадра должна быть такой, чтобы время передачи было по крайней мере не меньше этого минимума. При скорости 10 Мбит/с на передачу одного бита тратится 100 нс, значит, минимальный размер кадра должен быть равен 500 бит. Из соображений большей надежности это число было увеличено до 512 бит или 64 байт.

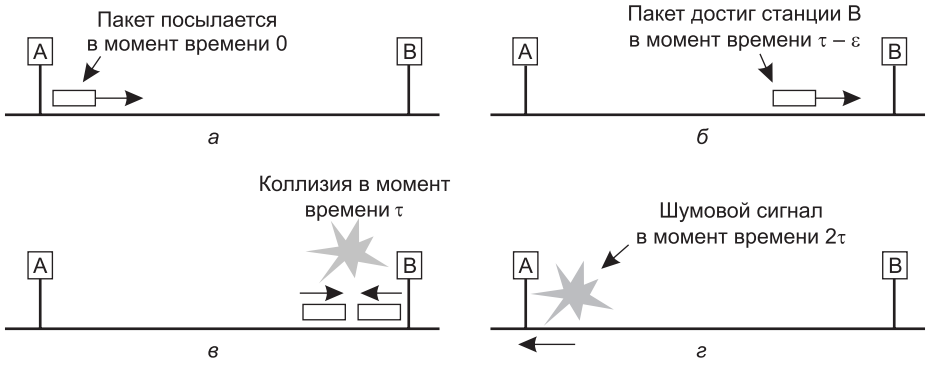


Рис. 4.15. Обнаружение столкновения может занять 2τ

Последнее поле кадра стандарта Ethernet — *Checksum*, которое содержит контрольную сумму. По сути дела, это 32-битный код CRC того же типа, как мы обсуждали выше. Он определен как раз при помощи того же порождающего многочлена, что используется для PPP, ADSL и других типов каналов. Этот CRC позволяет обнаруживать ошибки: он проверяет, правильно ли приняты биты кадра. Исправления ошибок нет, и если ошибка обнаруживается, кадр удаляется.

Экспоненциальный двоичный алгоритм выдержки

В классическом Ethernet используется алгоритм CSMA/CD с настойчивостью 1, который мы рассматривали выше. Это означает, что станции прослушивают среду передачи, когда у них появляется кадр на отправку, и отправляют данные, когда среда передачи освобождается. Отправляя кадр, они проверяют, не произошло ли в канале коллизий. Если столкновение случилось, они прерывают передачу, отправляя короткий сигнал о наличии конфликта, и повторяют отправку данных через случайный интервал времени.

Рассмотрим, как определяется случайная длина интервала ожидания после столкновения, так как это новый метод. Модель остается та же, что представлена на рис. 4.5. После возникновения коллизии время делится на дискретные интервалы, длительность которых равна максимальному времени кругового обращения сигнала (то есть его прохождения по кабелю в прямом и обратном направлении), 2τ. Для удовлетворения потребностей Ethernet при максимальном размере сети необходимо, чтобы один интервал составлял 512 битовых интервалов или 51,2 мкс.

После первого столкновения каждая станция ждет или 0 или 1 интервал, прежде чем попытаться передавать опять. Если две станции столкнутся и выберут одно и то же псевдослучайное число, то они столкнутся снова. После второго столкновения каждая станция выбирает случайным образом 0, 1, 2 или 3 интервала из набора и ждет опять.

При третьем столкновении (вероятность такого события после двойного столкновения равна 1/4) интервалы будут выбираться в диапазоне от 0 до $2^3 - 1$.

В общем случае, после i столкновений случайный номер выбирается в диапазоне от 0 до $2^i - 1$, и это количество интервалов станция пропускает. Однако после 10 столкновений подряд интервал рандомизации фиксируется на отметке 1023. После 16 столкновений подряд контроллер признает свое поражение и возвращает компьютеру ошибку. Дальнейшим восстановлением занимаются более высокие уровни.

Этот алгоритм, называемый **экспоненциальным двоичным алгоритмом выдержки (binary exponential backoff)**, был выбран для динамического учета количества станций, пытающихся осуществить передачу. Если выбрать интервал рандомизации равным 1023, то вероятность повторного столкновения будет пренебрежимо мала, однако среднее время ожидания составит сотни тактов, в результате чего среднее время задержки будет слишком велико. С другой стороны, если каждая станция будет выбирать время ожидания всего из двух вариантов, 0 и 1, то в случае столкновения сотни станций они будут продолжать сталкиваться снова и снова до тех пор, пока 99 из них не выберут 1, а одна станция — 0. Такого события можно будет ждать годами. Экспоненциально увеличивая интервал рандомизации по мере возникновения повторных столкновений, алгоритм обеспечивает небольшое время задержки при столкновении небольшого количества станций и одновременно гарантирует, что при столкновении большого числа станций конфликт будет разрешен за разумное время.

Если коллизии не произошло, отправитель предполагает, что кадр, вероятно, был успешно доставлен. Таким образом, ни в CSMA/CD, ни в Ethernet подтверждения не применяются. Такой вариант подходит для кабельных и оптоволоконных каналов с низким числом ошибок. Любые возникающие ошибки распознаются с помощью кода CRC и исправляются более высокими уровнями. Как мы увидим далее, в зашумленных беспроводных каналах подтверждения используются.

4.3.3. Производительность сети Ethernet

Оценим производительность классической сети Ethernet в условиях большой постоянной загрузки, то есть когда k станций постоянно готовы к передаче. Строгий анализ экспоненциального двоичного алгоритма выдержки довольно сложен. Вместо этого мы последуем за рассуждениями Меткальфа (Metcalfe) и Боггса (Boggs) (1976) и предположим, что вероятность повторной передачи в каждом интервале времени постоянна. Если каждая станция передает в течение одного интервала времени с вероятностью p, то вероятность того, что какой-либо станции удастся завладеть каналом, равна

$$A = kp(1 - p)^{k-1}. \tag{4.5}$$

Значение A будет максимальным, когда $p = 1/k$. При k, стремящемся к бесконечности, A будет стремиться к $1/e$. Вероятность того, что период соревнования за канал будет состоять ровно из j интервалов, будет равна $A(1 - A)^{j-1}$, следовательно, среднее число интервалов борьбы за канал будет равно

$$\sum_{j=0}^{\infty} jA(1 - A)^{j-1} = \frac{1}{A}.$$

Так как длительность каждого интервала времени равна 2τ, средняя продолжительность периода борьбы будет составлять $w = 2\tau/A$. При оптимальном значении вероятности p среднее количество интервалов за период борьбы никогда не будет превосходить e, таким образом, средняя продолжительность периода борьбы будет равна $2\tau e \approx 5{,}4\tau$.

Если среднее время передачи кадра составляет P с, то эффективность канала при его сильной загруженности будет равна:

$$\text{Эффективность канала} = \frac{P}{P + 2\pi/A}. \tag{4.6}$$

В этой формуле мы видим, как максимальная длина кабеля влияет на производительность. Чем длиннее кабель, тем более долгим становится период борьбы за канал. Из этих рассуждений становится понятно, почему стандарт Ethernet накладывает ограничение на максимальное расстояние между станциями.

Полезно переформулировать выражение (4.6) в терминах длины кадра F, пропускной способности сети B, длины кабеля L и скорости распространения сигнала c для оптимального случая: e интервалов столкновений на кадр. При $P = F/B$ выражение (4.6) примет вид:

$$\text{Эффективность канала} = \frac{1}{1 + 2BLe/cF} \tag{4.7}$$

Если второе слагаемое делителя велико, эффективность сети будет низкой. В частности, увеличение пропускной способности или размеров сети (произведение BL) уменьшит эффективность при заданном размере кадра. К сожалению, основные исследования в области сетевого оборудования нацелены именно на увеличение этого произведения. Пользователи хотят большой скорости при больших расстояниях (что обеспечивают, например, оптоволоконные региональные сети), следовательно, для данных приложений стандарт Ethernet будет не лучшим решением. Другие реализации Ethernet мы увидим в следующем разделе.

На рис. 4.16 показана зависимость эффективности канала от числа готовых станций для $2\tau = 51{,}2$ мкс и скорости передачи данных, равной 10 Мбит/с. Для расчетов используется выражение (4.7). При 64-байтном временном интервале 64-байтные кадры оказываются неэффективными, и это неудивительно. С другой стороны, если использовать кадры длиной 1024 байта, то при асимптотическом значении e периода состязания за канал, равном 64-байтовому интервалу, то есть 174 байтам, эффективность канала составит 85 %. Это намного лучший результат, чем 37 % в системе ALOHA с дискретными интервалами.

Следует отметить, что теоретическому анализу производительности сетей Ethernet (и других сетей) было посвящено много работ. Большинство результатов следует воспринимать с долей (или, лучше, с тонной) скептицизма по двум причинам. Во-первых, практически во всех этих теоретических исследованиях предполагается, что трафик подчиняется пуассоновскому распределению. Когда же исследователи рассмотрели реальные потоки данных, то обнаружилось, что сетевой трафик редко распределен по Пуассону, а чаще включает множество пиков (Paxson and Floyd, 1994; Leland и др., 1994). Это означает, что при увеличении периода усреднения трафик не сглаживается.

Помимо использования сомнительных моделей, многие из этих работ фокусируются на «интересных» случаях, то есть считают, что канал всегда очень сильно загружен. Боггс и др. (1988 год) на практике доказали, что Ethernet хорошо работает в реальных условиях, даже когда загрузка относительно высока.

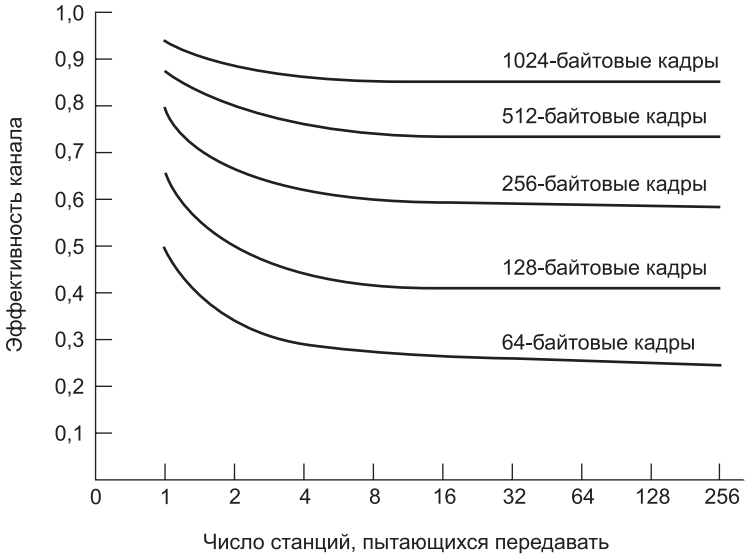


Рис. 4.16. Эффективность сетей стандарта 802.3 на скорости 10 Мбит/с с 512-битовыми интервалами времени

4.3.4. Коммутируемые сети Ethernet

Очень скоро Ethernet стал отходить от архитектуры с одним длинным кабелем, которая использовалась в классическом варианте. Проблема поиска обрывов или ведущих в пустоту соединений привела к новому способу подключения, в котором каждая станция соединяется с центральным **концентратором (hub)** отдельным кабелем. Концентратор просто соединяет все провода в электрическую схему, как если бы они были спаяны вместе. Такая конфигурация показана на рис. 4.17, *а*.

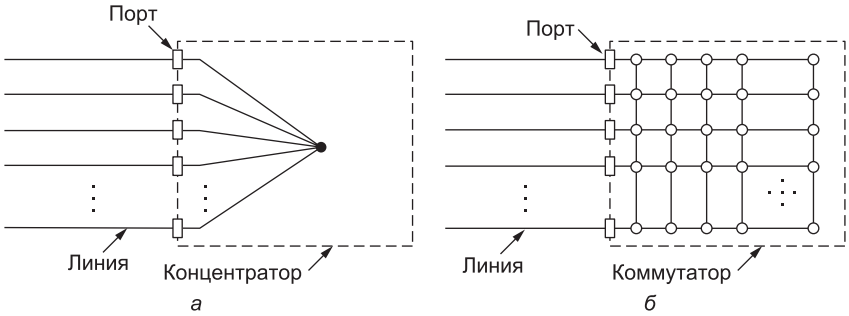


Рис. 4.17. Конфигурация Ethernet: *а* — концентратор; *б* — коммутатор

Для соединения использовалась витая пара, проложенная телефонной компанией, так как большинство офисных зданий и так было хорошо охвачено кабелем, а пустых пар было достаточно. Такой вариант использования был весьма выгодным, но он ограничивал максимальную длину кабеля между компьютером и концентратором до 100 м (или 200 при условии качественной витой пары категории 5). В подобной конфигурации было легко удалять и добавлять станции, также несложно было находить разрывы кабеля. Благодаря преимуществам использования существующей кабельной разводки и простоте обслуживания концентраторы на витой паре вскоре стали ведущей формой реализации сетей Ethernet.

Однако концентраторы не увеличивают емкость, так как логически они эквивалентны одному длинному кабелю классической сети Ethernet. При добавлении станций доля каждой из них в общей фиксированной емкости канала уменьшается. Наконец, локальная сеть насытится. Одним из решений в данном случае является увеличение скорости передачи данных — например, переход с 10 на 100 Мбит/с, 1 Гбит/с или даже больше. Однако доля мультимедийных данных мощных серверов в общем потоке становится все заметнее, и даже гигабитные версии Ethernet могут перестать справляться со своей задачей.

К счастью, возможно не столь радикальное решение, а именно коммутируемая локальная сеть Ethernet. Сердцем системы является **коммутатор (switch)**, содержащий высокоскоростную плату, объединяющие все порты (см. рис. 4.17, *б*). Снаружи коммутатор ничем не отличается от концентратора. Это обычные коробки, оборудованные несколькими (от 4 до 48) стандартными разъемами RJ-45 для подключения витой пары. Каждый кабель соединяет коммутатор или концентратор с одним компьютером, как показано на рис. 4.18. У коммутатора есть все преимущества концентратора. Новую станцию легко добавить или удалить, подключив или отключив один провод. Большинство сбоев кабеля или портов легко обнаруживаются по неправильной работе всего лишь одной станции. Общий компонент все же может подвести систему — речь идет о самом коммутаторе, — но если сеть пропадет на всех станциях, инженеры сразу поймут, в чем дело, и заменят устройство.

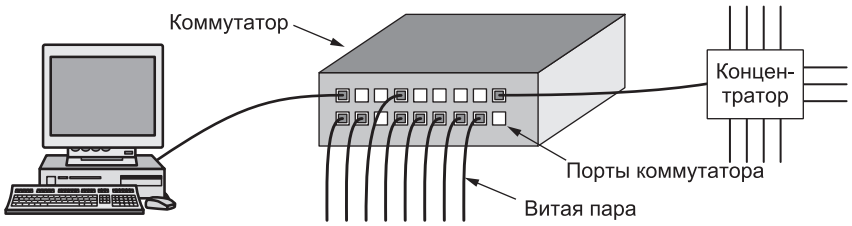


Рис. 4.18. Коммутатор Ethernet

Внутри же коммутатора происходит нечто совсем иное. Коммутаторы отдают кадры только на те порты, для которых эти кадры предназначены. Когда на порт коммутатора приходит кадр Ethernet со станции, коммутатор проверяет адреса Ethernet и узнает, на какой порт этот кадр нужно отдать. Для данного шага требуется, чтобы коммутатор умел сопоставлять номера портов и адреса; этот процесс мы обсудим далее, когда будем рассматривать подключение нескольких коммутаторов друг к другу. Пока что

предположим, что коммутатор знает порт получателя кадра. Он пересылает кадр по своей высокоскоростной плате на порт получателя. Скорость платы составляет несколько гигабит в секунду; а используемый протокол стандартизировать не требуется, так как никуда за пределы коммутатора он не выходит. Порт получателя затем отправляет кадр станции назначения по соединяющему их проводу. Другие порты об этом кадре даже не подразумевают.

Что произойдет, если две машины или два порта одновременно станут передавать кадры? И снова поведение коммутаторов отличается от концентраторов. Внутри концентратора все станции находятся в одном и том же **пространстве столкновений (collision domain)**. Для планирования пересылки кадров им необходимо применять алгоритм CSMA/CD. У коммутатора каждый порт находится в своем пространстве столкновений. В распространенном случае, когда передача по кабелю осуществляется в дуплексном режиме, и станция, и порт могут одновременно посылать кадры, не беспокоясь о других станциях и портах. Столкновения невозможны, и CSMA/CD не требуется. Однако если кабель полудуплексный, то станция и порт должны договариваться о передачи, применяя CSMA/CD обычным способом.

Что касается производительности, у коммутатора два преимущества перед концентратором. Во-первых, поскольку коллизии отсутствуют, емкость расходуется более эффективно. Во-вторых, и это очень важно, благодаря коммутатору разные станции могут посылать свои кадры одновременно. Эти кадры достигнут портов коммутатора и перейдут по внутренней плате устройства на правильные выходные порты. Однако так как на один выходной порт может быть одновременно отправлено два кадра, внутри коммутатора должен быть реализован буфер для временного хранения входных кадров, если моментальная доставка на выходной порт невозможна. В целом, эти усовершенствования дают большой выигрыш в производительности, который с концентратором невозможен. Общую производительность системы можно увеличить на порядок, в зависимости от числа портов и схем пересылки трафика.

Изменения в технологии портов, на которые пересылаются кадры, также дают преимущества, связанные с безопасностью. Большинство интерфейсов локальных сетей (сетевых адаптеров) работают в «**неразборчивом режиме**» (**promiscuous mode**), когда *все* кадры передаются на все компьютеры, а не только адресату. Если применяется концентратор, то каждый подключенный к нему компьютер может видеть трафик, пересылаемый между всеми остальными компьютерами. Шпионы и сплетники очень любят эту особенность. Коммутатор передает трафик только на порты адресатов. Такое ограничение обеспечивает лучшую изоляцию: трафик не «сбегает» и не попадает в нечистые руки. Однако если вопрос безопасности стоит в вашей организации очень серьезно, в дополнение к этому лучше шифровать трафик.

Так как коммутатор ожидает на каждом входном порту кадры Ethernet, можно использовать некоторые из этих портов в качестве концентраторов. На рис. 4.18 порт в правом верхнем углу соединен не с одной станцией, а с 12-портовым концентратором. Прибывая в концентратор, кадры состязаются самым обычным образом, включая столкновения и двоичную выдержку. Удачливые кадры попадают в коммутатор через концентратор и подвергаются там той же процедуре, что и все остальные входящие кадры. Коммутатор не знает о том, что им пришлось с боем прорываться к нему. Оказавшись в коммутаторе, они перенаправляются на нужные выходные линии через

высокоскоростную объединяющую плату. Также возможно, что адресатом была одна из линий, подключенных к концентратору; это означает, что кадр уже был доставлен, так что коммутатор просто удаляет его. Концентраторы проще и дешевле коммутаторов, но из-за стремительного удешевления последних они находятся под угрозой исчезновения. В современных сетях в основном применяется коммутируемый Ethernet. Тем не менее все еще существуют действующие концентраторы.

4.3.5. Fast Ethernet

Примерно тогда же, когда коммутаторы набирали популярность, скорость Ethernet 10 Мбит/с перестала казаться чем-то из ряда вон выходящим. Когда-то казалось, что 10 Мбит/с — это просто фантастически высокая скорость. Примерно так же воспринимали пользователи телефонных модемов появление кабельных модемов. Однако мир меняется очень быстро. В качестве одного из следствий закона Паркинсона («Работа занимает все отведенное на нее время») можно привести следующее правило: «Данные занимают всю предоставленную пропускную способность канала».

Многим приложениям требовалась высокая пропускная способность, и поэтому появились 10-Мбитные ЛВС, связанные лабиринтами повторителей, концентраторов и коммутаторов. Сетевым администраторам иногда казалось, что система держится еле-еле и может развалиться от любого прикосновения. Но даже с коммутаторами Ethernet максимальная полоса пропускания одного компьютера ограничивалась кабелем, которым тот соединялся с портом коммутатора.

Вот при таких обстоятельствах институт IEEE начал в 1992 году пересмотр стандартов и дал заказ комитету 802.3 выработать спецификацию более быстрых сетей. Одно из предложений состояло в том, чтобы сохранить 802.3 без изменений и просто увеличить скорость работы. Другое заключалось в том, чтобы полностью его переделать, снабдить новым набором функций — например, обеспечить возможность передачи данных реального времени, оцифрованной речи. При этом предлагалось сохранить старое название стандарта (такой коммерческий прием). После некоторых колебаний комитет решил все-таки изменить лишь скорость работы 802.3, а все остальные параметры оставить прежними. Такая стратегия позволила бы добиться желаемого прежде, чем технология изменится, и избежать непредвиденных проблем с совершенно новыми разработками. Также обеспечивалась бы обратная совместимость с существующими локальными сетями Ethernet. Сторонники отвергнутого предложения поступили так, как в этой ситуации поступил бы любой человек, связанный с компьютерной индустрией: они хлопнули дверью, организовали собственный комитет и разработали свой стандарт (собственно, 802.12), который, впрочем, с треском провалился.

Работа шла довольно быстро (по меркам комитета стандартизации), и уже в июне 1995 года официально объявили о создании стандарта **802.3u**. С технической точки зрения, в нем нет ничего нового по сравнению с предыдущей версией. Честнее было бы назвать это не новым стандартом, а расширением 802.3 (чтобы еще больше подчеркнуть обратную совместимость с ним). Такая стратегия часто используется. Поскольку жаргонное название «**Fast Ethernet**» (быстрый Ethernet) используется уже практически всеми, то и мы будем следовать этой моде.

Основная идея Fast Ethernet довольно проста: оставить без изменений все старые форматы кадров, интерфейсы, процедуры и лишь уменьшить время передачи одного бита с 100 до 10 нс. Как это технически осуществить? Можно скопировать принцип, применяемый в классическом 10-мегабитном Ethernet, но в 10 раз уменьшить максимальную длину сегмента. Однако преимущества витой пары были столь неоспоримы, что практически все системы типа «Fast Ethernet» в результате были построены именно на этом типе кабеля. Таким образом, в Fast Ethernet используются исключительно концентраторы (хабы) и коммутаторы; никаких моноканалов с ответвителями типа «зуб вампира» или с BNC-коннекторами здесь нет.

Однако некоторые технические решения все же необходимо было принять. Самый важный вопрос заключался в том, какие типы кабелей поддерживать. Одним из претендентов была витая пара категории 3. Основным аргументом в его пользу было то, что практически все западные офисы уже были оборудованы по крайней мере четырьмя витыми парами категории 3 (а то и лучше): они использовались в телефонных линиях и их длина (до ближайшего телефонного щита) составляла не более 100 м. Иногда можно было встретить два таких кабеля. Таким образом, можно было установить в организациях Fast Ethernet, и для этого не требовалось перекладывать кабель во всем здании. Это было очень существенно для многих.

Было во всем этом лишь одно неудобство: витые пары третьей категории не способны передавать сигналы 100-мегабитной сети на расстояние 100 метров (именно таково максимальное расстояние между компьютером и концентратором, установленное стандартом для 10-мегабитных концентраторов). Витые пары категории 5 с такой задачей справились бы без всяких проблем, а для оптоволокна это и вовсе смешная цифра. Надо было найти какой-то компромисс. Не мудрствуя лукаво, комитет 802.3 разрешил применять все три типа кабелей, как показано в табл. 4.2, с условием, что решения на основе витой пары третьей категории будут чуть живее и смогут обеспечить необходимую емкость канала.

Таблица 4.2. Основные типы кабелей для сетей Fast Ethernet

Название	Тип	Длина сегмента, м	Преимущества
100Base-T4	Витая пара	100	Использование неэкранированной витой пары категории 3
100Base-TX	Витая пара	100	Полный дуплекс при 100 Мбит/с (витая пара 5 категории)
100Base-FX	Оптоволокно	2000	Полный дуплекс при 100 Мбит/с; большая длина сегмента

В схеме **100Base-4T**, использующей витую пару категории 3, сигнальная скорость составляет 25 МГц, что лишь на 25 % больше, чем 20 МГц стандарта Ethernet (помните, что в Манчестерском кодировании, которое обсуждается выше, требуется удвоенная частота). Чтобы достичь требуемой пропускной способности, в схеме 100Base-4T применяются четыре витые пары. Из четырех витых пар одна всегда направляется на концентратор, одна — от концентратора, а две оставшиеся переключаются в зависимости от текущего направления передачи данных. Для достижения скорости

100 Мбит/с в направлении передачи на каждой из витых пар применяется довольно сложная схема с отправкой троичных цифровых сигналов с тремя разными уровнями напряжения. Вряд ли эта схема выиграет приз за элегантность, так что детали мы опустим. Однако так как в стандартной телефонной проводке десятилетиями использовались четыре витые пары в каждом кабеле, в большинстве офисов можно применять уже проложенные кабели. Разумеется, для этого придется отказаться от офисного телефона, но не такая уж это большая цена за быструю электронную почту.

Система 100Base-T4 не дошла до финала, поскольку во многих офисных зданиях проложили витую пару пятой категории для сетей **100Base-TX**, которые в итоге завоевали рынок. Такая схема проще, так как кабели этого типа могут работать с сигналами на частоте 125 МГц. Поэтому для каждой станции используются только две витые пары: одна к концентратору, другая от него. Не применяется ни прямое битовое кодирование (то есть NRZ), ни Манчестерский код. Вместо них имеется специальная система кодирования, называемая **4B/5B** (подробнее об этом выше). Четыре бита данных кодируются в форме пяти сигнальных бит и отправляются на частоте 125 МГц, обеспечивая скорость 100 Мбит/с. Это простая схема, но в ней выполняется достаточное число переходов для обеспечения синхронизации, и полоса пропускания расходуется довольно эффективно. Система 100Base-TX является полнодуплексной, станции могут передавать на скорости 100 Мбит/с по одной витой паре и одновременно принимать на той же скорости по другой.

Последний вариант, **100Base-FX**, использует два оптических многомодовых кабеля, по одному для передачи в каждом направлении, то есть также полный дуплекс на скорости 100 Мбит/с в каждом направлении. В таком варианте расстояние между станцией и коммутатором может достигать 2 км.

Fast Ethernet поддерживает соединение с помощью либо концентраторов, либо коммутаторов. Для того чтобы алгоритм CSMA/CD работал, необходимо соблюдать соотношение между минимальным размером кадра и максимальной длиной кабеля, учитывая возрастание скорости от 10 до 100 Мбит/с. Таким образом, либо нужно увеличить минимальный размер кадра (сделать его больше 64 байт), либо пропорционально уменьшить максимальную длину кабеля (менее 2500 м). Самый простой способ оказался следующим: уменьшить максимальное расстояние между двумя станциями в 10 раз, так как концентратор со стометровыми кабелями точно не будет выходить за новые границы. Однако кабели 100Base-FX в 2 км слишком длинны для 100-мегабитного концентратора с обычным алгоритмом управления коллизиями в сетях Ethernet. Эти кабели нужно подключать к коммутатору, чтобы они могли работать в полнодуплексном режиме без коллизий.

Пользователям очень полюбился Fast Ethernet, но они не собирались так просто выбрасывать 10-Мбитные платы Ethernet со старых компьютеров. В результате практически все коммутаторы могут поддерживать и 10-Мбитные, и 100-Мбитные станции. Для упрощения обновлений сам стандарт предусматривает механизм под названием **автоматическое согласование** (**autonegotiation**), который позволяет двум станциям автоматически договориться об оптимальной скорости (10 или 100 Мбит) и дуплексном режиме (полный дуплекс или полудуплекс). Обычно он работает без проблем, однако известны случаи возникновения проблем несовпадения дуплексного режима.

Одна станция применяет автоматическое согласование, а на другой оно не работает, и сразу же устанавливается полнодуплексный режим (Shalunov, Carlson, 2005). Большая часть аппаратуры Ethernet использует эту функцию для самонастройки.

4.3.6. Gigabit Ethernet

Не успели еще, как говорится, высохнуть чернила на только что созданном стандарте Fast Ethernet, как комитет 802 приступил к работе над новой версией. Ее почти сразу окрестили **Gigabit Ethernet (гигабитной сетью Ethernet)**. IEEE ратифицировал наиболее популярную форму сети в 1999 году под названием 802.3ab. Ниже мы обсудим некоторые ключевые свойства Gigabit Ethernet. Более подробную информацию можно найти в (Spurgeon, 2000).

Главные предпосылки комитета в отношении Gigabit Ethernet были те же самые, что и для Fast Ethernet: увеличить производительность в 10 раз, сохранив обратную совместимость со старыми сетями Ethernet. В частности, Gigabit Ethernet должен был обеспечить дейтаграммный сервис без подтверждений, как при одноадресной, так и при широковещательной передаче. При этом необходимо было сохранить неизменными 48-битную схему адресации и формат кадра, включая нижние и верхние ограничения его размера. Новый стандарт удовлетворил всем этим требованиям.

Как и Fast Ethernet, все конфигурации Gigabit Ethernet строятся по принципу «точка-точка». Простейшая гигабитная сеть, показанная на рис. 4.19, *а*, состоит из двух компьютеров, напрямую соединенных друг с другом. В более общем случае, однако, имеется коммутатор или концентратор, к которому подсоединяется множество компьютеров, возможна также установка дополнительных коммутаторов или концентраторов (рис. 4.19, *б*). Но в любом случае, к одному кабелю Gigabit Ethernet всегда присоединяются два устройства, ни больше, ни меньше.

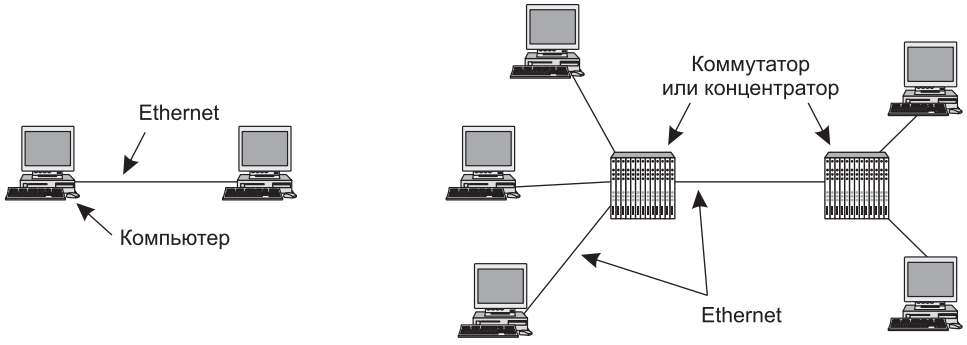


Рис. 4.19. Сеть Ethernet, состоящая: *а* — из двух станций; *б* — из множества станций

Аналогично, как и Fast Ethernet, Gigabit Ethernet может работать в двух режимах: полнодуплексном и полудуплексном. «Нормальным» считается полнодуплексный, при этом трафик может идти одновременно в обоих направлениях. Этот режим используется, когда имеется центральный коммутатор, соединенный с периферийными

компьютерами или коммутаторами. В такой конфигурации сигналы всех линий буферизируются, поэтому абоненты могут отправлять данные, когда им вздумается. Отправитель не прослушивает канал, потому что ему не с кем конкурировать. На линии между компьютером и коммутатором компьютер — это единственный потенциальный отправитель; передача произойдет успешно даже в том случае, если одновременно с ней ведется передача со стороны коммутатора (линия полнодуплексная). Так как конкуренции в данном случае нет, протокол CSMA/CD не применяется, поэтому максимальная длина кабеля определяется исключительно мощностью сигнала, а вопросы времени распространения шумового всплеска здесь не встают. Коммутаторы могут работать на смешанных скоростях; более того, они автоматически выбирают оптимальную скорость. Самонастройка поддерживается так же, как и в Fast Ethernet, но теперь можно выбирать скорость 10, 100 или 1000 Мбит/с.

Полудуплексный режим работы используется тогда, когда компьютеры соединены не с коммутатором, а с концентратором. Концентратор не буферизирует входящие кадры. Вместо этого он электрически соединяет все линии, симулируя моноканал обычного Ethernet. В этом режиме возможны коллизии, поэтому применяется CSMA/CD. Поскольку кадр минимального размера (то есть из 64 байт) может передаваться в 100 раз быстрее, чем в классической сети Ethernet, максимальная длина кабеля должна быть, соответственно, уменьшена в 100 раз. Она составляет 25 м — именно при таком расстоянии между станциями шумовой всплеск гарантированно достигнет отправителя до окончания его передачи. Если бы кабель имел длину 2500 м, то отправитель кадра из 64 байт при 1 Гбит/с успел бы много чего наделать даже за то время, пока его кадр прошел бы только десятую часть пути в одну сторону, не говоря уж о том, что сигнал должен еще и вернуться обратно.

Ограничение на длину было таким жестким, что комитет решил добавить в стандарт два новых свойства, позволивших увеличить максимальную длину кабеля до 200 м, что, вероятно, должно удовлетворить большинство организаций. Первое называется **расширением носителя (carrier extension)**. Заключается это расширение всего-навсего в том, что аппаратура вставляет собственное поле заполнения, растягивающее нормальный кадр до 512 байт. Поскольку это поле добавляется отправителем и изымается получателем, то программному обеспечению нет до него никакого дела. Конечно, тратить 512 байт на передачу 64 байт — это несколько расточительно с точки зрения эффективности использования пропускной способности. Эффективность такой передачи составляет всего 9 %.

Второе свойство, позволяющее увеличить допустимую длину сегмента, — это **пакетная передача кадров (frame bursting)**. Это означает, что отправитель может посылать не единичный кадр, а пакет, объединяющий в себе сразу много кадров. Если полная длина пакета оказывается менее 512 байт, то, как в предыдущем случае, производится аппаратное заполнение фиктивными данными. Если же кадров, ждущих передачу, хватает на то, чтобы заполнить такой большой пакет, то работа системы оказывается очень эффективной. Такая схема, разумеется, предпочтительнее расширения носителя.

Трудно представить себе организацию, которая потратила бы немало средств на установку современных компьютеров с платами для гигабитной сети Ethernet, а потом соединила бы компьютеры древними концентраторами, симулирующими работу классического Ethernet со всеми его коллизиями и прочими проблемами. Интерфейсы

и коммутаторы Gigabit Ethernet когда-то были довольно дороги, но когда спрос на них возрос, цены существенно упали. Однако обратная совместимость — это нечто священное в компьютерной индустрии, поэтому, несмотря ни на что, комитету необходимо было это учесть. Сегодня большинство компьютеров поставляются с интерфейсом Ethernet, поддерживающим работу на скоростях 10, 100 и 1000 Мбит/с и совместимым с любыми из них.

Gigabit Ethernet поддерживает как медные, так и волоконно-оптические кабели, что отражено в табл. 4.3. Работа на скорости около 1 Гбит/с означает необходимость кодирования и отправки бита каждую наносекунду. Первоначально этого достигали за счет коротких экранированных медных кабелей (версия 1000Base-CX) и оптоволокна. Волоконная оптика допускает две длины волны, и, следовательно, существуют две разные версии: 0,85 мкм (короткие волны, для 1000Base-SX) и 1,3 мкм (длинные, для 1000Base-LX).

Таблица 4.3. Кабели Gigabit Ethernet

Название	Тип	Длина сегмента, м	Преимущества
1000Base-SX	Оптоволокно	550	Многомодовое волокно (50, 62,5 мкм)
1000Base-LX	Оптоволокно	5000	Одномодовое (10 мкм) или многомодовое (50, 62,5 мкм) волокно
1000Base-CX	2 экранированные витые пары	25	Экранированная витая пара
1000Base-T	4 неэкранированные витые пары	100	Стандартная витая пара 5-й категории

Передача сигналов с помощью коротких волн возможна с дешевыми светодиодами. Такой вариант применяется с многомодовым волокном для соединения станций внутри здания, так как для 50-мкм волокна допустимая длина — не более 500 м. Передача сигналов на длинных волнах требует более дорогих лазеров. С другой стороны, в сочетании с одномодовым (10 мкм) волокном разрешается длина кабеля до 5 км. Это означает возможность подключения друг к другу зданий, например, в студенческом городке, аналогично связям «точка-точка». Более поздние вариации стандарта допускали даже более длинные связи на одномодовом волокне.

Для отправки бит по этим версиям соединений Gigabit Ethernet из другой сетевой технологии под названием Fibre Channel (оптоволоконный канал) была заимствована система кодирования 8B/10B, о которой говорилось выше. В этой системе 8 бит данных кодируются в кодовые слова из 10 бит, которые отправляются по проводу или оптическому волокну — отсюда и название 8B/10B. Кодовые слова выбираются так, чтобы они могли быть сбалансированы (например, имеющие равное число нулей и единиц) и чтобы переход осуществлялся достаточное число раз для восстановления синхронизации. Отправка NRZ закодированных бит требует лишь на 25 % большей полосы пропускания, чем для незакодированных бит, — значительное улучшение по сравнению со стопроцентным увеличением для Манчестерского кода.

Однако все это требовало новых медных или оптоволоконных кабелей, поддерживающих более быструю передачу сигналов. Ни один из них не опирается на огромное

количество витой пары пятой категории, которая была проложена для сетей Fast Ethernet. В течение года потребность была заполнена благодаря 1000Base-T, и с тех пор это остается наиболее популярной формой Gigabit Ethernet. Очевидно, людям не очень нравится заново прокладывать кабели в своих зданиях.

Для того чтобы сеть Ethernet могла работать на проводах пятой категории со скоростью 1000 Мбит/с, требуется более сложная схема передачи сигналов. Во-первых, используются все четыре витые пары в кабеле; каждая пересылает данные одновременно в обоих направлениях, применяя цифровую обработку сигналов для их разделения. Для обеспечения скорости 125 Мсимволов/с в каждом проводе применяется пять уровней напряжения, которые переносят по 2 бита. Схема создания символов из бит не так проста. Она включает шифрование (для безопасной передачи) и код исправления ошибок, в котором четыре значения внедряются в пять сигнальных уровней.

1 Гбит/с — это довольно много. Например, если приемник отвлечется на какое-то дело в течение 1 мс и при этом забудет или не успеет освободить буфер, это означает, что он проспит примерно 1953 кадра. Может быть и другая ситуация: один компьютер выдает данные по гигабитной сети, а другой принимает их по классическому Ethernet. Вероятно, первый быстро завалит данными второго. В первую очередь, переполнится буфер обмена. Исходя из этого, было принято решение о внедрении в систему Gigabit Ethernet контроля потока. Для реализации контроля потока одна из сторон посылает служебный кадр, сообщающий о том, что второй стороне необходимо приостановиться на некоторое время. Служебные кадры PAUSE — это, на самом деле, обычные кадры Ethernet, в поле *Type* которых записано 0x8808. Продолжительность паузы определяется в единицах времени передачи минимального кадра. Для Gigabit Ethernet такая единица равна 512 нс, а паузы могут длиться до 33,6 мс.

Вместе с Gigabit Ethernet было добавлено и еще одно расширение. **Джамбо-пакеты (Jumbo frames)** допускают кадры длиной более 1500 байт, обычно до 9 Кбайт. Это расширение защищено патентом. Оно не определяется в стандарте, потому что в противном случае Ethernet уже не будет совместим с предыдущими версиями. Тем не менее большинство производителей его все же поддерживают. Обоснование таково, что 1500 байт — это слишком маленькая единица информации на гигабитных скоростях. Манипулируя большими блоками информации, можно уменьшить частоту пересылки кадров и снизить нагрузку из-за необходимой обработки (например, не придется прерывать процессор, чтобы сообщить о прибытии кадра, или разбивать и заново соединять сообщения, не поместившиеся в одном кадре Ethernet).

4.3.7. 10-гигабитный Ethernet

Gigbit Ethernet был стандартизован, и комитет 802 заскучал. Тогда IEEE предложил ему начать работу над 10-Gigbit Ethernet (10-гигабитным Ethernet). Работа шла по тому же принципу, что и раньше, при стандартизации предыдущих версий Ethernet. Первые стандарты для оптоволоконного и экранированного медного кабеля появились в 2002 и 2004 годах, а стандарт для медной витой пары последовал в 2006 году.

10 Гбит/с — это поистине колоссальная скорость. В 1000 раз быстрее первоначального стандарта Ethernet! Где она может понадобиться? Ответ скрывается в дата-центрах и точках обмена трафиком с высококлассными маршрутизаторами,

коммутаторами и серверами, а также в сильно загруженных магистральных каналах, соединяющих офисы компаний в разных городах. Весь город можно охватить единой сетью на базе оптоволокна и Ethernet. Такие длинные связи используют оптическое волокно, тогда как более короткие связи можно выполнять с помощью медных кабелей.

Все версии 10-Gigbit Ethernet поддерживают только полнодуплексную передачу данных. CSMA/CD больше не является частью архитектуры, и стандарты фокусируются на деталях физического уровня, которые обеспечивают такую высокую скорость. Однако совместимость не потеряла своего значения, поэтому интерфейсы 10-Gigabit Ethernet выполняют автоматическое согласование скорости и выбирают максимально возможное значение для обоих концов линии.

Основные типы 10-Gigbit Ethernet перечислены в табл. 4.4. На средних расстояниях применяется многомодовое волокно с длиной волны 0,85 мкм, а на больших расстояниях — одномодовое волокно с длиной волны 1,3 и 1,5 мкм. Сеть 10GBase-ER может охватывать до 40 км, что хорошо подходит для глобальных приложений. Все эти версии отправляют последовательный поток информации, которая получается путем смешивания бит данных и кодирования 64B/66B. Такое кодирование требует меньше накладных расходов, чем 8B/10B.

Таблица 4.4. Кабели 10-Gigbit Ethernet

Название	Тип	Длина сегмента	Преимущества
10GBase-SR	Оптоволокно	До 300 м	Многомодовое волокно (0,85 мкм)
10GBase-LR	Оптоволокно	10 км	Одномодовое (1,3 мкм) волокно
10GBase-ER	Оптоволокно	40 км	Одномодовое (1,5 мкм) волокно
10GBase-CX4	4 пары биаксиального кабеля	15 м	Биаксиальный медный кабель
10GBase-T	4 пары неэкранированной витой пары	100 м	Неэкранированная витая пара категории 6a

Версия, определенная первой, 10GBase-CX4, работает на базе кабелей с четырьмя парами биаксиального медного провода. В каждой паре используется кодирование 8B/10B, они работают на скорости 3,125 Гсимволов/с, обеспечивая скорость передачи данных 10 Гбит/с. Эта версия дешевле волоконной и первой вышла на рынок, однако еще непонятно, сумеет ли она вытеснить с рынка 10-Gigbit Ethernet на базе витой пары.

10GBase-T — это версия, работающая на неэкранированной витой паре. Несмотря на то что официально она требует прокладки кабеля категории 6a, пока что можно использовать и более старые категории (включая пятую), то есть уже проложенные во множестве зданий по всему миру кабели. Неудивительно, что для достижения скорости 10 Гбит/с на витой паре огромные старания приходится приложить физическому уровню. Мы взглянем только на самые общие особенности. Каждая из четырех витых пар используется для пересылки данных в обоих направлениях на скорости 2500 Мбит/с. Это достигается за счет скорости пересылки сигналов 800 Мсимволов/с на 16 уровнях напряжения. Символы создаются путем перемешивания данных, при-

менения кода **LDPC (Low Density Parity Check)** и последующего кодирования для исправления ошибок.

Различные варианты 10-Gigbit Ethernet еще не поделили рынок, а комитет 802.3 уже идет дальше. В конце 2007 года IEEE создала группу по стандартизации сетей Ethernet, работающих на скоростях 40 и 100 Гбит/с. Такой рывок позволит Ethernet стать серьезным соперником альтернативным технологиям в таких областях, требующих высокой производительности, как междугородные соединения в магистральных сетях и короткие соединения по системным платам устройств. Описание стандарта еще не завершено, однако определенные патентованные продукты уже доступны.

4.3.8. Ретроспектива Ethernet

Ethernet существует вот уже 30 лет, и никаких серьезных конкурентов за это время не появилось. Похоже, и в ближайшее время не появятся. Очень немногие микропроцессорные архитектуры, операционные системы и языки программирования могут похвастаться таким долгим и уверенным лидерством. Вероятно, Ethernet чем-то очень выгодно отличается от всех остальных систем. Чем же?

Возможно, основной причиной столь длительного успеха является простота и гибкость системы. Простота в данном случае означает, прежде всего, надежность, невысокую цену и легкость обслуживания. С тех пор как признание получили архитектуры на базе концентраторов и коммутаторов, чисто технические поломки стали чрезвычайно редки. Человек так устроен, что он с трудом может отказаться от чего-либо, что хорошо работает, в пользу чего-то другого. Нужно принять во внимание и тот факт, что огромное количество кое-как собранной компьютерной аппаратуры работает не слишком надежно. Именно по этой причине так называемые «апгрейды» часто дают результат, ровно противоположный ожидаемому. Бывает так, что системы после них работают не лучше, а даже хуже.

Вторая причина популярности Ethernet — это низкая цена. Витая пара сравнительно недорога, так же как аппаратные компоненты. Затрат может потребовать, например. переход на новые платы Gigabit Ethernet или коммутаторы, но это всего лишь дополнения к существующей сети (а не замена всего имеющегося оборудования), к тому же оптовые цены значительно выгоднее розничных.

Сети Ethernet не доставляют большой головной боли системным администраторам — они обслуживаются без особых проблем. Не нужно устанавливать никакое программное обеспечение (кроме драйверов), и очень мало конфигурационных таблиц (в которых так просто ошибиться). Новые узлы добавляются очень просто.

Еще одно достоинство Ethernet заключается в хорошем взаимодействии с TCP/IP — доминирующим протоколом сети Интернет. IP — это протокол без установления соединения, поэтому он без проблем внедряется в локальных сетях Ethernet, которые также используют протоколы без установления соединения. IP имеет довольно плохую совместимость с сетями ATM, ориентированными на установку соединения. Этот факт крайне негативно сказывается на популярности ATM.

И что важнее всего, разработчикам Ethernet удалось добиться хороших показателей по самым главным направлениям. Скорости выросли на несколько порядков, в систему были внедрены коммутаторы и концентраторы, но эти изменения никак не коснулись

программного обеспечения. Помимо этого, они допускают использование существующей кабельной разводки в течение довольно длительного времени. Если продавец скажет: «Вот отличная новая сетевая система! Она работает просто фантастически быстро и надежно! Вам необходимо только выкинуть весь ваш старый железный хлам и стереть все старые программы», — у него возникнут проблемы с объемами продаж.

Многие альтернативные технологии, о которых вы, вероятно, даже не слышали, в моменты своего появления были даже быстрее тогдашнего Ethernet. Помимо ATM, этот список включает FDDI и волоконно-оптический канал (**FC — Fibre Channel**) — две оптические локальные сети на базе кольца. Обе были несовместимы с Ethernet. Ни одна не выжила. Они были слишком сложны, что приводило к усложнению микросхем и повышению цен. Урок очень прост: БПД — будь проще, дурачок (**KISS — Keep It Simple, Stupid**). Потом оказалось, что Ethernet догнал и перегнал их по скорости работы, по пути заимствуя составляющие технологий конкурентов (например, кодирование 4B/5B у FDDI и 8B/10B у FC). У соперников не осталось никаких преимуществ, и они либо тихо скончались, либо сбежали в узкоспециализированные сферы применения.

Создается впечатление, что области применения Ethernet продолжат расширяться. 10-Gigabit Ethernet освободился от ограничений максимального расстояния, накладываемых CSMA/CD. Много внимания уделяется **Ethernet операторского класса (carrier-grade Ethernet), который позволит сетевым провайдерам предлагать основанные** на Ethernet услуги своим клиентам в городских и глобальных сетях (Fouli, Maler, 2009). Это приложение способно передавать кадры Ethernet на большие расстояния по оптоволоконному кабелю и требует усовершенствования возможностей управления, для того чтобы операторы смогли предлагать пользователям надежные высококачественные услуги. Скоростные сети также находят применение в системных платах, соединяющих компоненты в больших маршрутизаторах или серверах. Оба этих варианта использования представляют собой лишь дополнения к задаче пересылки кадров между компьютерами в офисах.

4.4. Беспроводные локальные сети

Беспроводные локальные вычислительные сети становятся все более популярными, все больше и больше домов, офисных зданий, кафе, библиотек, аэровокзалов, зоопарков и других общественных мест оборудуются соответствующей аппаратурой для подключения компьютеров, КПК и смартфонов к Интернету. В беспроводной сети два или несколько соседних компьютеров могут обмениваться данными и без подключения к Интернету.

Основной стандарт беспроводных локальных сетей — это 802.11. Некоторую общую информацию мы уже приводили в разделе 1.5.3. Теперь более пристальный взгляд обратим на технологическую сторону стандарта 802.11. В последующих разделах речь пойдет о стеке протоколов, методах радиопередачи (на физическом уровне), протоколе подуровня MAC, структуре кадра и сервисах. Дополнительную информацию о стандарте 802.11 можно найти в издании (Gast, 2005). Чтобы получить информацию из первых рук, обратитесь к официальному техническому описанию стандарта IEEE 802.11-2007.

4.4.1. Стандарт 802.11: архитектура и стек протоколов

Сети 802.11 можно использовать в двух режимах. Самый популярный режим — это подключение клиентов, таких как ноутбуки и смартфоны, к другой сети, например внутренний сети компании или Интернету. Такая схема показана на рис. 4.20, *а*. В **инфраструктурном режиме (infrastructure mode)** каждый режим связывается с **точкой доступа (Access Point, AP)**, которая, в свою очередь, подключена к сети. Клиент отправляет и получает пакеты через точку доступа. Несколько точек доступа можно соединить вместе, обычно в кабельную сеть под названием **распределительная система (distribution system)**. Так формируется расширенная сеть 802.11. В данном случае клиенты могут отправлять кадры другим клиентам через их точки доступа.

Второй режим, показанный на рис. 4.20, *б*, называется **произвольной сетью** (ad hoc network). Это набор компьютеров, которые связаны таким образом, чтобы они могли напрямую отправлять кадры друг другу. Точка доступа не используется. Поскольку доступ в Интернет — революционная технология в беспроводных соединениях, произвольные сети не очень популярны.

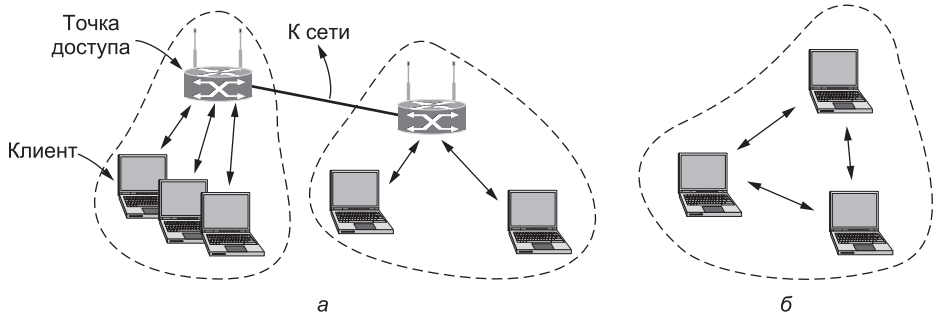


Рис. 4.20. Архитектура сети стандарта 802.11: *а* — инфраструктурный режим; *б* — произвольный режим

Теперь взглянем на протоколы. Все протоколы, используемые семейством стандартов 802.x, включая 802.11 и Ethernet, схожи по структуре. Часть стека протоколов изображена на рис. 4.21. Стек одинаков для клиентов и для точек доступа. Физический уровень практически соответствует физическому уровню в модели OSI, а вот канальный уровень во всех протоколах 802.x разбит на два или более подуровня. Что касается 802.11, то подуровень MAC (подуровень управления доступом к среде) отвечает за распределение канала, то есть за то, какая станция будет передавать следующей. Над MAC в иерархии находится подуровень LLC (управления логическим соединением), задача которого состоит в том, чтобы сделать различия стандартов 802.x невидимыми для сетевого уровня. Это могло бы стать очень ответственной задачей, но в настоящее время ключевым слоем считается LLC; именно он отвечает за идентификацию протокола (например, IP), информация о котором передается в кадре 802.11.

После первого появления в 1997 году физический уровень обзавелся несколькими новыми методами передачи. Два первоначальных метода, инфракрасная передача (как

в пульте дистанционного управления телевизором) и режим скачкообразного изменения частоты в диапазоне 2,4 ГГц, сегодня не используются. Третий из первоначальных методов, широкополосный сигнал с прямой последовательностью на скорости 1 или 2 Мбит/с в диапазоне 2,4 ГГц, был расширен и завоевал популярность со скоростями до 11 Мбит/с. Этот стандарт известен под названием 802.11b.

Рис. 4.21. Часть стека протоколов 802.11

Для того чтобы дать фанатам беспроводных сетей такое желанное увеличение скорости, в 1999 и 2003 годах были представлены новые методы передачи на основе схемы OFDM (Orthogonal Frequency Division Multiplexing), описанной ранее в разделе 2.5.3. Первый метод называется 802.11a и работает в другом диапазоне частот, 5 ГГц. Второй остался в диапазоне 2,4 ГГц для обеспечения совместимости. Он называется 802.11g. Оба работают на скоростях до 54 Мбит/с.

Совсем недавно, в октябре 2009 года, в рамках стандарта 802.11n была завершена работа над методами передачи данных, которые одновременно используют несколько антенн на приемнике и передатчике, что дает очередной выигрыш в скорости. Благодаря четырем антеннам и более широким каналам стандарт 802.11 теперь определяет скорости до поразительных 600 Мбит/с.

Сейчас мы изучим вкратце все эти методы передачи. Мы рассмотрим только используемые методы, отбросив устаревшие стандарты 802.11. Технически, они относятся к физическому уровню, которому была посвящена, вообще говоря, вся глава 2. Но из-за того, что они так тесно связаны с ЛВС вообще и с ЛВС стандарта 802.11 в частности, мы обращаемся к ним здесь.

4.4.2. Стандарт 802.11: физический уровень

Все рассматриваемые ниже методы передачи данных позволяют передать кадр подуровня MAC с одной станции на другую по радиоканалу. Отличаются они используемыми технологиями и достижимыми скоростями. Детальное рассмотрение этих методов выходит за рамки данной книги, мы лишь дадим краткое описание, которое, возможно, заинтересует читателей и снабдит их необходимыми терминами для поиска более подробной информации где-либо еще (см. также раздел 2.5).

Все методы стандарта 802.11 используют радиосигналы ближнего радиуса действия в диапазоне частот ISM 2,4 ГГц или 5 ГГц (подробнее об этом см. раздел 2.3.3). Преимущество этих диапазонов в том, что они не требуют лицензирования, то есть доступны для любого передатчика, отвечающего небольшому числу ограничений (излучаемая мощность до 1 Вт, хотя для большинства передатчиков в беспроводных сетях характерна модность 50 мВт). К сожалению, этот факт также известен производителям автоматических гаражных дверей, беспроводных телефонов, микроволновых печей и множества других устройств, конкурирующих за один и тот же спектр частот с ноутбуками. Диапазон 2,4 ГГц более населен, чем диапазон 5 ГГц, поэтому для некоторых приложений предпочтительнее использовать последний, несмотря на меньший радиус действия (из-за более высокой частоты).

Все методы передачи также определяют разные скорости. Идея заключается в том, чтобы использовать разные показатели скорости в зависимости от текущих условий. Если беспроводной сигнал слабый, выбирается низкая скорость. Если сигнал сильный, то скорость можно повысить. Такая корректировка называется **адаптацией скорости (rate adaptation)**. Поскольку скорости могут различаться в десятки раз, хорошая адаптация скорости важнее хорошей производительности соединения. Разумеется, поскольку для возможности взаимодействия это не обязательно, в стандартах не говорится, каким именно способом нужно корректировать скорость.

Первый метод передачи, на который мы взглянем, — это **802.11b**. Это технология расширенного спектра, поддерживающая скорости 1, 2, 5,5 и 11 Мбит/с, хотя на практике почти всегда удается удерживать ее на самом высоком уровне. Она похожа на систему CDMA, с которой мы познакомились ранее, за исключением того, что здесь только один код расширения спектра, который используется всеми пользователями. Расширение применяется для удовлетворения требованию FCC о том, что мощность должна быть распределена по диапазону ISM. Для стандарта 201.11b используется **последовательность Баркера (Barker sequence)**. Ее отличительная особенность заключается в том, что автокорреляция низка, за исключением случаев, когда последовательности выровнены. Благодаря этому получатель может захватить начало передачи. Для того чтобы пересылать данные на скорости 1 Мбит/с, последовательность Баркера комбинируется с модуляцией BPSK, и с каждыми 11 чипами отправляется 1 бит. Чипы пересылаются со скоростью 11 Мчипов/с. Чтобы достичь скорости 2 Мбит/с, последовательность комбинируется с модуляцией QPSK, и с каждыми 11 чипами отправляются 2 бита. На более высоких скоростях дело обстоит по-другому. Вместо последовательности Баркера для конструирования кодов применяется техника под названием **CCK (Complementary Code Keying, схема ключей дополнительного кода)**. На скорости 5,5 Мбит/с в каждом 8-чиповом коде отправляется 4 бита, а на скорости 11 Мбит/с — 8 бит.

Переходим к 802.11a, который поддерживает скорости до 54 Мбит/с в 5-гигагерцовом диапазоне ISM. Можно было бы подумать, что 802.11a появился раньше 802.11b, но это не так. Хотя группа 802.11a была основана раньше, стандарт 802.11b первым получил одобрение. Соответствующие продукты вышли на рынок раньше продуктов 802.11a частично еще и из-за сложностей работы в более высоком диапазоне 5 ГГц.

Метод 802.11a основан на технологии **OFDM (Orthogonal Frequency Division Multiplexing)**, так как она эффективно использует спектр и устойчива к искажению

беспроводного сигнала, например, из-за многолучевого распространения. Биты параллельно отправляются по 52 поднесущим, из которых 48 содержат данные и 4 служат для синхронизации. Каждый символ длится 4 мкс и отправляет 1, 2, 4 или 6 бит. Биты кодируются для исправления ошибок, для этого применяется сверточный код. Поэтому только 1/2, 2/3 или 3/4 бит не являются избыточными. В разных комбинациях 802.11a может обеспечивать восемь разных показателей скорости, от 6 до 54 Мбит/с. Это значительно выше, чем у 802.11b, к тому же в диапазоне 5 ГГц намного меньше помех. Однако радиус действия 802.11b примерно в семь раз дальше, чем у 802.11a, что во многих ситуациях бывает крайне важно.

Несмотря на бóльшую дальность действия, разработчики 802.11b не собирались давать этому молодому да раннему стандарту шансов на победу в соревновании скоростей. К счастью, в мае 2002 года FCC отменила давнее правило, требующее, чтобы все беспроводное коммуникационное оборудование, работающее в США в диапазонах ISM, применяло расширение спектра, поэтому стало возможным запустить работу над 802.11g — этот стандарт был одобрен комитетом IEEE в 2003 году. Он копирует методы модуляции OFDM стандарта 802.11a, но вместе с 802.11b используется в ограниченном диапазоне ISM 2,4 ГГц. Он предлагает те же скорости, что и 802.11a (от 6 до 54 Мбит/с) плюс, разумеется, совместимость с любыми устройствами 802.11b, которые могут оказаться рядом. Все эти различия зачастую сбивают простых пользователей с толку, поэтому продукты обычно поддерживают 802.11a/b/g в одной общей плате.

Не сбираясь останавливаться на достигнутом, комитет IEEE начал работу над физическим уровнем 802.11n, характеризующимся очень высокой пропускной способностью. Он был одобрен в 2009 году. Цель 802.11n — обеспечить пропускную способность не менее 100 Мбит/с, устранив все накладные расходы беспроводной связи. Для этого требуется увеличение базовой скорости как минимум в четыре раза. Комитет удвоил ширину каналов с 20 до 40 МГц и снизил накладные расходы на пересылку кадров, разрешив совместную отправку целой группы кадров. Что еще важнее, в стандарте 802.11n предусмотрено использование до четырех антенн для пересылки до четырех потоков информации одновременно. Сигналы потоков смешиваются на стороне получателя, но их можно разделить с помощью коммуникационных техник **MIMO (Multiple Input Multiple Output**, **несколько входов — несколько выходов)**. Наличие нескольких антенн дает огромный выигрыш в скорости либо бóльший радиус действия и повышение надежности. MIMO, как и OFDM, — это одна из тех хитрых коммуникационных идей, которые в корне меняют дизайн беспроводных сетей и о которых мы наверняка нередко будем слышать и в будущем. Краткое описание техники использования нескольких антенн в стандарте 802.11 см в (Halperin и др., 2010).

4.4.3. Стандарт 802.11: протокол подуровня управления доступом к среде

Однако вернемся из области электротехники в область компьютерных наук. Протокол подуровня MAC (напомним, MAC расшифровывается как Medium Access Control — Управление доступом к среде) в стандарте 802.11 довольно сильно отличается от аналогичного протокола Ethernet вследствие двух фундаментальных факторов, характерных для беспроводного обмена данными.

Во-первых, радиопередатчики почти всегда работают в полудуплексном режиме. Это означает, что они не могут на одной и той же частоте одновременно передавать сигналы и прослушивать всплески шума. Получаемый сигнал может быть в миллион раз слабее передаваемого и его может быть просто не слышно. В Ethernet станция ожидает, пока в канале настанет тишина, и тогда начинает передачу. Если шумовой всплеск не приходит обратно в течение времени, необходимого на пересылку 64 байт, то можно утверждать, что кадр почти наверняка доставлен корректно. В беспроводных сетях такой механизм распознавания коллизий не работает.

Вместо этого 802.11 пытается избегать коллизий за счет протокола **CSMA/CA** (**CSMA with Collision Avoidance**, **CSMA с предотвращением коллизий**). Концепция данного протокола схожа с концепцией CSMA/CD для Ethernet, где канал прослушивается перед началом отправки, а период молчания после коллизии вычисляется экспоненциально. Однако если у станции есть кадр для пересылки, то она начинает цикл с периода молчания случайной длины (за исключением случаев, когда она давно не использовала канал, и он бездействует). Станция не ожидает коллизий. Число слотов, в течение которых она молчит, выбирается в диапазоне от 0 до, скажем, 15 в случае физического уровня OFDM. Станция дожидается бездействия канала в течение короткого периода времени (называемого DIFS; подробнее о нем ниже) и отсчитывает слоты бездействия, приостанавливая отсчет на время отправки кадров. Свой кадр она отправляет, когда счетчик достигает нуля. Если кадр проходит успешно, то адресат немедленно отправляет обратно короткое подтверждение. Если подтверждение отсутствует, делается вывод, что произошла ошибка — коллизия или иная. В таком случае отправитель удваивает период молчания и повторяет попытку, продолжая экспоненциально наращивать длину паузы (как с Ethernet), пока кадр не будет успешно передан или пока не будет достигнуто максимальное число повторов.

Пример некоторой временной шкалы приводится на рис. 4.22. Станция *A* отправляет кадр первой. Пока станция *A* занята отправкой, станции *B* и *C* переходят в режим готовности к отправке. Они видят, что канал занят, и дожидаются бездействия канала. Вскоре после получения станцией *A* подтверждения канал переходит в режим бездействия. Однако вместо того чтобы сразу же отправлять кадры (что привело бы к коллизии), станции *B* и *C* начинают свои периоды молчания. Станция *C* выбирает короткий период молчания, поэтому ей удается отправить данные первой. Станция *B* приостанавливает обратный отсчет, когда видит, что канал занят станцией *C*, и возобновляет только после получения станцией *C* подтверждения. Вскоре период молчания станции *B* завершается, и она также отправляет кадр.

По сравнению с Ethernet, здесь два основных отличия. Во-первых, раннее начало периодов молчания помогает избегать коллизий. Это важное преимущество, так как коллизии обходятся дорого, ведь даже если столкновение происходит, кадр все равно отправляется целиком. Во-вторых, для того чтобы станции могли «догадываться» о коллизиях, которые распознать невозможно, применяется схема с подтверждениями.

Такой режим работы называется **DCF** (**Distributed Coordination Function**, **распределенная координация**). Все станции действуют независимо, централизованный контроль не осуществляется. Стандарт также включает необязательный режим **PCF** (**Point Coordination Function**, **сосредоточенная координация**), в котором всей деятельностью в ячейке управляет точка доступа — как базовая станция сотовой сети.

Однако PCF на практике не применяется, потому что невозможно запретить станциям из соседней сети передавать конкурирующий трафик.

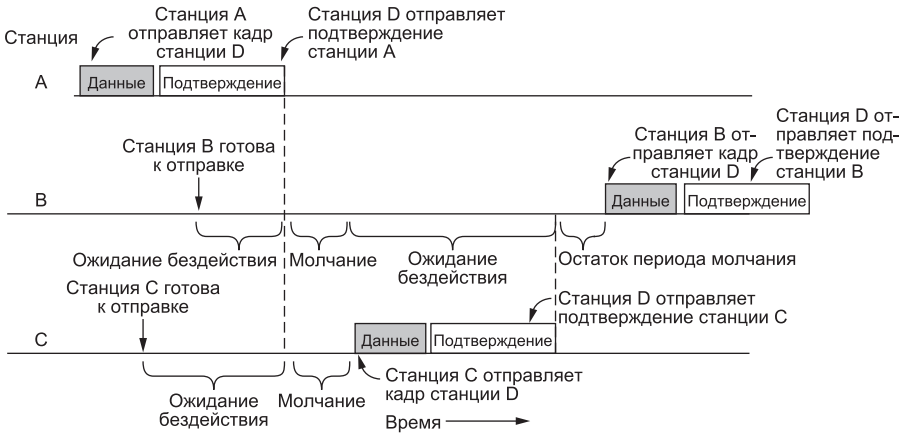


Рис. 4.22. Отправка кадра с протоколом CSMA/CA

Вторая проблема заключается в том, что области передачи разных станций не обязательно совпадают. В кабельной сети система спроектирована таким образом, чтобы все станции могли слышать друг друга. Сложности передачи радиосигналов не позволяют обеспечить такое постоянство для беспроводных станций. Следовательно, возможно возникновение ситуаций, таких как проблема скрытой станции — мы уже упоминали о ней ранее, а сейчас приводим еще и иллюстрацию на рис. 4.23, *а*. Поскольку не все станции могут слышать друг друга, передача, идущая в одной части ячейки, может быть просто не воспринята станцией, находящейся в другой ее части. В приведенном на рисунке примере станция *C* передает данные станции *B*. Если станция *A* прослушает канал, она не обнаружит ничего подозрительного и сделает ложный вывод о том, что она имеет право начать передачу станции *B*. Это решение приводит к коллизии.

Кроме того, есть и обратная проблема, показанная на рис. 4.23, *б*. Здесь *B* хочет отправить данные для станции *C* и прослушивает канал. Услышав, что в нем уже осуществляется какая-то передача, станция *B* делает опять-таки ложный вывод о том, что передача для *C* сейчас невозможна. Между тем, станция *A* — источник сигнала, который смутил станцию *B*, — может, на самом деле, осуществлять передачу для станции *D* (на рисунке не показана). Таким образом, теряется возможность передать информацию.

Для того чтобы разрешить непонимание относительно того, какая станция будет отправлять данные, в стандарте 802.11 прослушивание канала определяется на физическом и виртуальном уровнях. При физическом прослушивании среда просто проверяется на наличие сигнала. Виртуальное прослушивание заключается в том, что каждая станция ведет логический журнал использования канала, отслеживая **NAV (Network Allocation Vector**, **вектор распределения сети)**. Каждый кадр содержит поле NAV, которое сообщает, как долго последовательность, включающая данный кадр, будет передаваться. Станции, услышавшие этот кадр, понимают, что канал будет

занят в течение периода, указанного в NAV, даже если физический сигнал в канале отсутствует. Например, NAV для кадров данных включает также время, необходимое для отправки подтверждения. Все станции, услышавшие этот кадр данных, воздерживаются от пересылки данных в течение периода отправки подтверждения, независимо от того, слышали ли они его в канале.

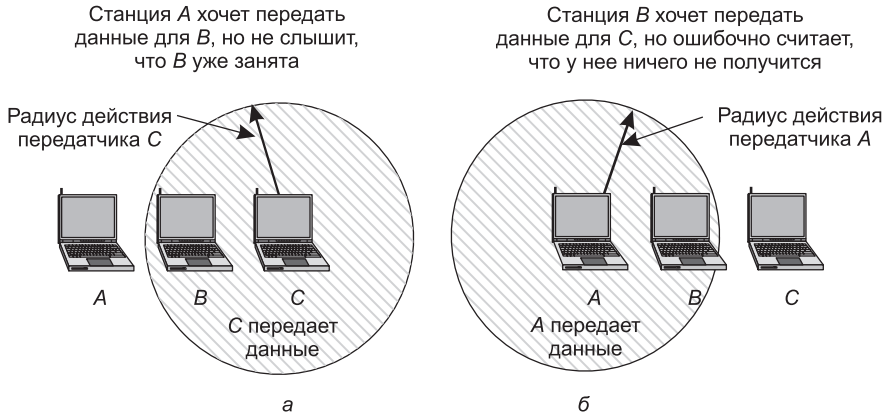


Рис. 4.23. Проблема: *а* — скрытой станции; *б* — засвеченной станции

Необязательный механизм RTS/CTS с помощью NAV запрещает станциям отправлять кадры одновременно со скрытыми станциями. Пример показан на рис. 4.24. В этом примере станция *A* хочет передать данные станции *B*. Станция *C* находится в зоне действия (то есть слышит) *A*, а также, возможно, в зоне действия *B*, но это не имеет значения. Станция *D* входит в зону действия *B*, но не входит в зону действия *A*.

Протокол начинает работать тогда, когда *A* решает, что ей необходимо послать данные *B*. *A* посылает станции *B* кадр RTS, запрашивая разрешение на передачу. Если *B* может принять данные, она отсылает обратно подтверждение о том, что канал чист — кадр CTS. После приема CTS *A* отправляет кадр и запускает таймер ACK. В случае корректного приема *B* генерирует кадр ACK, завершающий передачу. Если интервал времени таймера на станции *A* истекает прежде, чем получен ACK, то считается, что произошла коллизия, и весь алгоритм работы протокола повторяется с самого начала после периода молчания.

Теперь рассмотрим этот же процесс с точки зрения станций *C* и *D*. *C* находится в зоне действия *A*, поэтому она также принимает кадр RTS и понимает, что скоро по каналу будут передаваться какие-то данные. Исходя из информации, содержащейся в RTS, станция *C* может предположить, сколько времени займет передача последовательности, включая конечный ACK. Поэтому, чтобы не мешать другим, она воздерживается от передачи данных, пока обмен не будет завершен. Для этого она обновляет свою запись NAV, указывая, что канал занят, как показано на рис. 4.24. Станция *D* не слышит RTS, зато слышит CTS и также выставляет NAV. Обратите внимание: сигналы NAV не передаются, а являются лишь внутренними напоминаниями станций о том, что нужно хранить молчание в течение определенного промежутка времени.

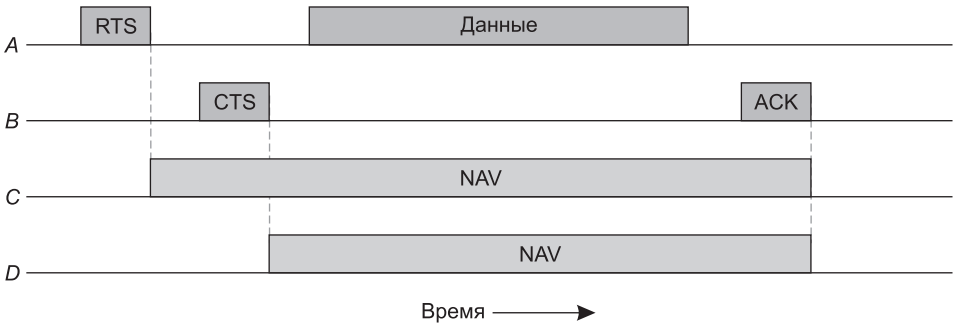


Рис. 4.24. Использование прослушивания виртуального канала в протоколе CSMA/CA

Однако, несмотря на теоретическую привлекательность модели RTS/CTS, это один из тех методов, практическая реализация которых провалилась. Есть несколько причин, почему она используется так редко. Она не рассчитана на короткие кадры (которые отправляются вместо RTS) и на присутствие точек доступа (которые, по определению, должны быть слышны всем). В других ситуациях она также замедляет работу. RTS/CTS в стандарте 802.11 немного отличается от протокола MACA, с которым мы познакомились в разделе 4.2, потому что каждый, кто получает RTS или CTS, сохраняет молчание в течение какого-то промежутка, для того чтобы подтверждение ACK сумело пройти по каналу без коллизий. По причине этого проблема засвеченной станции не решается, как при использовании протокола MACA, устраняется только проблема скрытых станций. Чаще всего скрытых станций совсем немного, и CSMA/CA и так помогает им. Эта технология замедляет станции, которым по какой-либо причине не удается успешно передать данные, для того чтобы повысить вероятность удачной пересылки.

CSMA/CA с физическим и виртуальным прослушиванием составляет суть протокола 802.11. Однако есть несколько других механизмов, разработанных для того же стандарта. Каждый из этих механизмов вызван определенными потребностями, связанными с фактическими условиями.

Первая потребность — это надежность. В противоположность проводным каналам, беспроводные шумны и ненадежны, в какой-то степени из-за влияния других устройств, таких как СВЧ-печи, работающих в том же диапазоне ISM. Использование подтверждений и повторных передач мало помогает, если вероятность передачи кадра мала.

Основная стратегия, используемая для увеличения числа успешных передач, состоит в том, чтобы понизить скорость передачи. Более медленные скорости используют более сильные методы модуляции сигнала, который с большей вероятностью будет правильно получен для данного отношения сигнал/шум. Если потеряно слишком много кадров, станция может понизить скорость. Если кадры доставляются с небольшой потерей, станция может иногда пробовать более высокую скорость, чтобы увидеть, может ли она использоваться.

Другая стратегия улучшить шанс кадра дойти неповрежденным состоит в том, чтобы посылать более короткие кадры. Если вероятность ошибки в одном бите

равна p, то вероятность того, что n-битный кадр будет принят корректно, равна $(1 - p)^n$. Например, для $p = 10^{-4}$ вероятность корректной передачи полного Ethernet-кадра длиной 12 144 бит составляет менее 30 %. Большая часть кадров будет потеряна. Но если длина кадров будет составлять только одну треть (4048 бит), то две трети их будут получены правильно. Теперь большинство кадров пройдет, и будет необходимо меньше повторных передач.

Использование более коротких кадров может быть реализовано сокращением максимального размера сообщения, которое принимается от сетевого уровня. С другой стороны, 802.11 позволяет разделять кадры на мелкие кусочки, названные **фрагментами (fragments)**, каждый со своей контрольной суммой. Размер фрагмента не фиксирован, а является параметром, который может быть скорректирован точкой доступа. Фрагменты нумеруются и подтверждаются индивидуально с использованием протокола с ожиданием (то есть отправитель не может передать фрагмент с номером $k + 1$, пока не получит подтверждения о доставке фрагмента с номером k). Они идут один за другим с подтверждением (и возможно повторной передачей) между ними, пока или весь кадр не будет передан успешно, или время передачи не достигнет позволенного максимума. Механизм NAV удерживает станции от передачи только до прихода первого подтверждения о доставке. Но есть и другой механизм (он описан ниже), позволяющий получателю принять всю пачку фрагментов, без кадров от других станций между ними.

Вторая потребность, которую мы обсудим, — экономия энергии. Время работы от аккумулятора для мобильных беспроводных устройств всегда представляет проблему. Стандарт 802.11 обращает внимание на проблему управления электропитанием так, чтобы клиенты не тратили энергию впустую, когда у них нет посылаемой или получаемой информации.

Основной механизм для экономии энергии основывается на **кадрах «маяках» (beacon frames)**. Это периодические широковещательные сообщения точки доступа (например, каждые 100 мс). Кадры сообщают клиентам о присутствии точки доступа и несут системные параметры, такие как идентификатор, время, интервал до следующего маяка и настройки безопасности.

Клиенты могут установить бит управления электропитанием в кадрах, которые они посылают в точку доступа, чтобы сообщить ей, что они входят в **энергосберегающий режим (power-save mode)**. В этом режиме клиент может дремать, и точка доступа будет буферизовать предназначенный для него трафик. Чтобы проверить наличие входящего трафика, клиент просыпается для каждого маяка и проверяет карту трафика, которую ему посылают как часть маяка. Эта карта говорит клиенту о наличие буферизованного трафика. Если он есть, клиент посылает сообщение опроса в точку доступа, и она посылает буферизованный трафик. Затем клиент может вернуться в спящий режим до следующего маяка.

В 2005 году к 802.11был добавлен другой энергосберегающий механизм, названный **APSD (Automatic Power Save Delivery — автоматический переход в режим сохранения энергии)**. С этим новым механизмом точка доступа буферизирует кадры и посылает их клиенту сразу после того, как клиент посылает кадры в точку доступа. Клиент может заснуть, пока у него нет большего количества трафика для отправки (и получения). Этот механизм хорошо работает на таких приложениях, как

IP-телефония, у которых часто есть трафик в обоих направлениях. Например, беспроводной IP-телефон мог бы использовать этот механизм, чтобы посылать и получать кадры каждые 20 мс, что намного чаще, чем интервал маяка (100 мс), и находится в спящем режиме в промежутках.

Третья и последняя потребность, которую мы исследуем, — это качество обслуживания. Когда трафик IP-телефонии в предыдущем примере конкурирует с трафиком соединения равноправных узлов ЛВС, пострадает трафик IP-телефонии. Он будет отложен в соревновании с трафиком соединения равноправных узлов ЛВС высокой пропускной способности, даже при том, что пропускная способность IP-телефонии низка. Эти задержки, вероятно, ухудшат голосовые вызовы. Чтобы предотвратить это ухудшение, мы хотели бы позволить трафику IP-телефонии идти перед трафиком соединения равноправных узлов ЛВС, как имеющего более высокий приоритет.

В IEEE 802.11 есть умный механизм, обеспечивающий этот вид качества обслуживания, который был введен в 2005 году как набор расширений под именем 802.11e. Он работает, расширяя CSMA/CA с тщательно определенными интервалами между кадрами. После того как кадр послан, прежде чем любая станция может послать кадр, требуется определенное количество времени простоя, чтобы проверить, что канал больше не занят. Эта уловка должна определить различные временные интервалы для различных видов кадров.

На рис. 4.25 изображено пять интервалов. Интервал между регулярными кадрами данных называется **DIFS (DCF InterFrame Spacing — межкадровый интервал DCF)**. Любая станция может попытаться захватить канал, чтобы послать новый кадр после того, как среда была неактивна для DIFS. Применяются при этом обычные правила борьбы, включая двоичную экспоненциальную выдержку в случае коллизии.

Самый короткий интервал — это **SIFS (Short InterFrame Interval — короткий межкадровый интервал)**. Он используется для того, чтобы одна из сторон в диалоге могла получить шанс начать первой. Примеры включают разрешение получателю послать ACK, другие последовательности кадров управления, такие как RTS и CTS, или разрешение отправителю передать пакет фрагментов. Отправка следующего фрагмента после ожидания только SIFS препятствует тому, чтобы другая станция вмешалась с кадром в середине обмена.

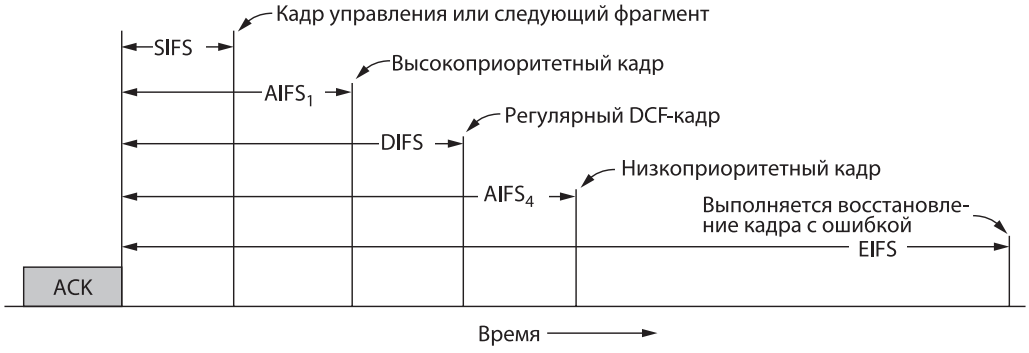


Рис. 4.25. Межкадровые интервалы в стандарте 802.11

Два интервала **AIFS (Arbitration InterFrame Space — межкадровый арбитражный интервал)** показывают примеры двух различных уровней приоритета. Короткий интервал, $AIFS_1$, короче чем DIFS, но длиннее, чем SIFS. Он может использоваться точкой доступа, чтобы переместить голос или другой приоритетный трафик в начало очереди. Точка доступа будет ждать более короткого интервала, прежде чем пошлет голосовой трафик, и, таким образом, пошлет его раньше регулярного трафика. Длинный интервал, $AIFS_4$, больше чем DIFS. Он используется для фонового трафика, который может быть задержан до окончания регулярного трафика. Прежде чем послать этот трафик, точка доступа будет ждать в течение более длинного интервала, давая возможность сначала передать регулярный трафик. Полный механизм качества обслуживания определяет четыре различных приоритетных уровня, у которых есть различные параметры выдержки, а также различные параметры времени ожидания.

Последний временной интервал называется **EIFS (Extended InterFrame Spacing — расширенный межкадровый интервал)**. Он используется только той станцией, которая только что получила испорченный или неопознанный кадр и хочет сообщить о проблеме. Идея в том, что приемник может сразу не сообразить, что происходит, и ему нужно выждать в течение какого-то интервала, чтобы не прервать своим возмущенным возгласом идущий в это время диалог между станциями.

Дальнейшая часть расширений, обеспечивающих качество обслуживания, — понятие **возможности передачи** (transmission opportunity) или **TXOP**. Первоначальный механизм CSMA/CA позволял станциям посылать один кадр за один раз. Эта схема была прекрасна, пока диапазон скоростей не увеличился. В 802.11a/g одна станция могла бы посылать кадры со скоростью 6 Мбит/с, а другая — 54 Мбит/с. Каждой из них надо послать один кадр, но первой станции нужно для отправки ее кадра в 9 раз больше времени (не считая фиксированных накладных расходов), чем второй. У этого неравенства есть неприятный побочный эффект замедления быстрого отправителя, который конкурирует с медленным отправителем, примерно до скорости медленного отправителя. Например, снова игнорируя фиксированные накладные расходы, работая по отдельности отправители реализовывали бы свои собственные скорости 6 Мбит/с и 54 Мбит/с, а работая вместе они оба получат в среднем скорость 5,4 Мбит/с. Что большая неприятность для быстрого отправителя. Эта проблема известна как **аномалия скорости (rate anomaly)** (Heusse и др., 2003).

При использовании TXOP каждая станция получает одинаковое количество эфирного времени, а не одинаковое количество кадров. Станции, которые посылают на более высокой скорости, получат в течение своего эфирного времени более высокую пропускную способность. В нашем примере отправители, совместно работающие со скоростями 6 и 54 Мбит/с, теперь достигнут скоростей 3 и 27 Мбит/с, соответственно.

4.4.4. Стандарт 802.11: структура кадра

Стандарт 802.11 определяет три класса кадров, передаваемых по радиоканалу: информационные, служебные и управляющие. Все они имеют заголовки с множеством полей, используемых подуровнем MAC. Кроме того, есть поля, используемые физическим уровнем, но они в основном относятся к методам модуляции, поэтому здесь мы их рассматривать не будем.

В качестве примера мы рассмотрим формат информационного кадра. Он показан на рис. 4.26. Вначале идет поле *Управление кадром* (Frame Control). Оно содержит 11 вложенных полей. Первое из них — *Версия протокола*, установлено в 00 (2 бита). Именно оно позволит будущим версиям 802.11 работать одновременно в одной ячейке сети. Затем следуют поля *Тип* (информационный, служебный или управляющий) и *Подтип* (например, RTS или CTS). Для обычного кадра данных (без указания качества обслуживания) они установлены как бинарные 10 и 0000. Биты *К DS* и *От DS* говорят о направлении движения кадра: в сеть или из сети, соединенной с точкой доступа, которая называется распределительной системой (distribution system). Бит *Дополнительные фрагменты* говорит о том, что далее следует еще один фрагмент. Бит *Повтор* маркирует повторно посылаемый кадр. Бит *Управление питанием* используется станцией-отправителем для указания на свое переключение в режим пониженного энергопотребления или на выход из этого режима. Бит *Продолжение* говорит о том, что у отправителя имеются еще кадры для пересылки. Бит *Шифрование* является индикатором использования шифрования в теле кадра. Наконец, установленный бит *Порядок* говорит приемнику о том, что кадры с этим битом должны обрабатываться строго по порядку.

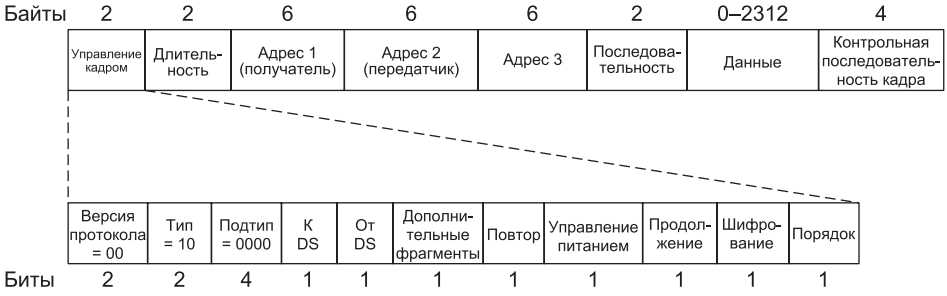


Рис. 4.26. Информационный кадр стандарта 802.11

Второе основное поле информационного кадра — это поле *Длительность*. В нем задается время в микросекундах, которое будет потрачено на передачу кадра и подтверждения. Это поле присутствует во всех типах кадров, в том числе в служебных кадрах, и именно в соответствии с ним станции выставляют признаки NAV.

Далее следуют адреса. Кадры данных содержат три адреса в формате, соответствующем стандарту IEEE 802. Понятно, что нужны адреса отправителя и получателя, но что же содержится в третьем?

Помните, что точка доступа — это просто пункт ретрансляции кадров, когда они движутся между клиентом и другой точкой сети, возможно удаленным клиентом или интернет-порталом. Третий адрес — адрес этой удаленной конечной точки.

Поле *Последовательность* позволяет нумеровать фрагменты, чтобы было возможно определить дубликаты. Из 16 доступных бит 4 идентифицируют фрагмент, а 12 содержат число, которое растет с каждой новой передачей.

Поле *Данные* содержит передаваемую по каналу информацию, его длина может достигать 2312 байт. Первые байты этой полезной нагрузки находятся в формате, известном как **LLC (подуровень управления логическим соединением)**. Этот уровень —

связующий элемент, который идентифицирует протокол более высокого уровня (например, IP), к которому нужно передать полезную нагрузку.

В конце, как обычно, расположено поле *Контрольная последовательность кадра*, который является тем же самым 32-битовым CRC, который мы видели в разделе 3.2.2 и в других местах.

Управляющие кадры имеют такой же формат, как формат информационных кадров, плюс формат для части данных, которая меняется в зависимости от подтипа (например, параметры в кадрах «маяках»).

Служебные кадры короткие. Как и во всех кадрах, в них содержится *Управление кадром*, *Длительность* и *Контрольная последовательность кадра*. При этом они могут иметь только один адрес и не иметь поля *Данные*. Ключевой здесь является информация, содержащаяся в поле *Подтип* (RTS, CTS или ACK).

4.4.5. Сервисы

Стандарт 802.11 определяет сервисы, чтобы клиенты, точки доступа и соединяющие их сети могли быть согласованными беспроводными ЛВС.Их можно разделить на несколько категорий.

Ассоциация (association). Этот сервис используется мобильными станциями для подключения к точкам доступа. Обычно он применяется сразу же после вхождения в зону действия точки доступа. По прибытии станция узнает идентификационную информацию и возможности точки доступа или от кадров-маяков, или прямо запросив точку доступа. Возможности точки доступа включают поддерживаемую скорость передачи данных, меры безопасности, возможности энергосбережения, поддержку качества обслуживания и т. д. Мобильная станция посылает запрос на ассоциацию с точкой доступа, которая может принять либо отвергнуть этот запрос.

Реассоциация (reassociation) позволяет станции сменить точку доступа. Эта возможность полезна при перемещении станции от одной точки доступа к другой в той же расширенной 802.11 ЛВС, по аналогии с передачей в сотовой сети. Если она проходит корректно, то при переходе никакие данные не теряются. (Однако, как и в сети Ethernet, в стандарте 802.11 все услуги предоставляются лишь с обязательством приложения максимальных усилий к их исполнению, но не с гарантией.) По инициативе мобильной станции или точки доступа может быть произведена **дизассоциация (disassociate)**, то есть разрыв отношений. Она требуется при выключении станции или ее уходе из зоны действия точки доступа. Точка доступа также может быть инициатором дизассоциации, если, например, она временно выключается для проведения технического обслуживания.

Прежде чем станции смогут посылать кадры через точку доступа, они должны пройти **аутентификацию (authenticate)**. В зависимости от выбора схемы безопасности аутентификация поддерживается по-разному. Если сети 802.11 «открыты», их разрешают использовать любому. Если нет — для аутентификации нужны параметры учетной записи. Рекомендуемая схема, названная **WPA2 (WiFi Protected Access 2 — WiFi Защищенный Доступ 2)**, обеспечивает безопасность как определено стандартом 802.11i. (Просто WPA — временная схема, которая обеспечивает подмножество 802.11i. Мы пропустим ее и перейдем прямо к полной схеме.) С WPA2 точка доступа может

взаимодействовать с сервером аутентификации, у которого есть имя пользователя и база данных паролей, чтобы определить, разрешено ли станции получить доступ к сети. Либо может быть сконфигурирован предустановленный ключ (pre-shared key), который является необычным названием сетевого пароля. Несколько кадров с запросом и ответом пересылаются между станцией и точкой доступа, что позволяет станции доказать, что у нее есть правильные учетные данные. Этот обмен происходит после ассоциации.

Схема, которая использовалась до WPA, называется **WEP (Wired Equivalent Privacy — приватность на уровне проводной связи)**. Для этой схемы аутентификация с предустановленным ключом выполнялась перед ассоциацией. Однако ее польза не велика из-за недостатков конструкции, которые делают WEP легко взламываемым. Первая практическая демонстрация взлома WEP произошла, когда Адам Стабблефилд был летним стажером в AT&T (Stubblefield и др., 2002). Он смог написать код и проверить атаку за одну неделю, большая часть которой была потрачена на получение разрешения администрации на покупку карт WiFi, необходимых для эксперимента. Программное обеспечение для взлома паролей WEP теперь есть в свободном доступе.

Когда кадры достигают точки доступа, **служба распределения (distribution service)** определяет их маршрутизацию. Если адрес назначения является локальным для данной точки доступа, то кадры следуют напрямую по радиоканалу. В противном случае, их необходимо пересылать по проводной сети.

Служба интеграции (integration service) поддерживает трансляцию, необходимую, если кадр нужно выслать за пределы сети стандарта 802.11 или если он получен из сети не этого стандарта. Типичный случай здесь — соединение между беспроводной ЛВС и Интернетом.

Доставка данных (data delivery). Собственно говоря, именно этот сервис является ключевым во всей работе сети. Ведь сеть 802.11 существует для обмена данными. Эта служба позволяет станциям передавать и получать данные по протоколам, которые мы описали ранее в этой главе. Поскольку стандарт 802.11 основан на стандарте Ethernet, а в последнем доставка данных не является гарантированной на 100 %, то для беспроводных сетей это тем более верно. Верхние уровни должны заниматься обнаружением и исправлением ошибок.

Беспроводная передача — это широковещательный сигнал. Для сохранения конфиденциальности информации, посланной по беспроводной ЛВС, она должна быть зашифрована. Эта цель достигается **службой конфиденциальности (privacy service)**, которая управляет деталями шифрования и дешифрования. Алгоритм шифрования для WPA2 основан на **AES (Advanced Encryption Standard — улучшенный стандарт шифрования)**, американском правительственном стандарте, одобренном в 2002 году. Ключи, которые используются для шифрования, определяются во время процедуры аутентификации.

Для обработки трафика с различными приоритетами имеется служба **планирования трафика QOS (QOS trffic scheduling)**. Она использует протоколы, которые мы описали, чтобы дать голосовому и видео трафику преимущество перед трафиком «с максимальными усилиями» и фоновым трафиком. Сопутствующая служба также обеспечивает синхронизацию более высокого уровня. Это позволяет станци-

ям координировать свои действия, что может быть полезным для обработки мультимедиа.

Наконец, есть две службы, которые помогают станциям управлять использованием спектра. **Регулирование мощности передатчика** (transmit power control) дает станциям информацию, которая нужна им, чтобы соответствовать установленным нормативным пределам мощности передачи, которые варьируются в зависимости от региона. Служба **динамического выбора частоты (dynamic frequency selection)** дает станциям информацию, необходимую, чтобы избежать передачи в частотном диапазоне 5 ГГц, который используется радарами.

С помощью этих сервисов стандарт 802.11 обеспечивает богатый набор возможностей для того, чтобы соединить близко расположенных мобильных клиентов с Интернетом. Это был огромный успех, и стандарт неоднократно исправлялся, чтобы добавить еще больше возможностей. Увидеть, откуда и куда движется этот стандарт, можно в работе (Hiertz и др., 2010).

4.5. Широкополосные беспроводные сети

Что-то мы засиделись в помещении. Выйдем и посмотрим на улицу, где тоже есть интересные сети, то, что называется «последней милей». Во многих странах телефонная система в какой-то момент перестала быть жестко управляемой, и появилось множество фирм, предлагающих локальные услуги голосовой связи и интернет-услуги.

Предложений действительно много. Проблема только в том, что прокладка волоконно-оптического или коаксиального кабеля к миллионам абонентов обходится очень дорого. Что же делать?

Ответ одновременно и прост, и сложен. Нужны широкополосные беспроводные сети. Установить одну большую антенну на горке где-нибудь рядом с населенным пунктом и расставить на крышах домов абонентов приемные антенны гораздо проще и дешевле, чем рыть множество траншей и протягивать кабель. Таким образом, конкурирующие операторы связи начали экспериментировать в развитии многомегабитных беспроводных систем связи, реализующих услуги голосовой коммуникации, доступа к Интернету, видео по заказу и т. д.

Для стимулирования рынка, IEEE сформировал группу, чтобы стандартизировать широкополосную городскую беспроводную сеть. Следующее число в нумерации 802 было **802.16**, таким образом, стандарт получил этот номер. Неофициально технологию называют **WiMAX (Worldwide Interoperability for Microwave Access)**. Мы будем использовать термины 802.16 и WiMAX попеременно.

Впервые стандарт 802.16 был апробирован в декабре 2001 года. Ранние версии обеспечили беспроводную местную линию связи между фиксированными точками с прямой видимостью друг друга. Эта схема вскоре была изменена, чтобы сделать WiMAX более конкурентоспособной альтернативой кабелю и DSL для доступа к Интернету. К январю 2003 года стандарт 802.16 был пересмотрен, чтобы поддержать связи вне прямой видимости с использованием технологии OFDM в частотах между 2 и 10 ГГц. Это изменение сделало установку намного легче, хотя станции все еще были с фиксированными местоположениями. Угрозу представлял рост сотовых сетей

поколения 3G, обещающий высокие скорости передачи данных и подвижность. В ответ к декабрю 2005 года 802.16 снова был улучшен, чтобы позволить подвижность на скоростях транспорта. Мобильный широкополосный доступ к Интернету — цель текущего стандарта, IEEE 802.16-2009.

Подобно некоторым другим стандартам из серии 802, стандарт 802.16 построен с использованием идей модели OSI. Здесь можно найти и уровни, и подуровни, используется схожая терминология, сервисные примитивы и т. п. К сожалению, как и OSI, стандарт 802.16 страдает громоздкостью. Фактически, **форум WiMAX (WiMAX Forum)** создавался для того, чтобы определить имеющие возможность взаимодействовать подмножества стандарта для коммерческих предложений. В последующих разделах мы приведем краткое описание основных свойств 802.16, но такое исследование является далеко не полным, в нем опущены многие детали. Дополнительную информацию о WiMAX и широкополосных беспроводных сетях в целом вы найдете в (Andrews и др., 2007).

4.5.1. Сравнение стандарта 802.16 с 802.11 и 3G

Казалось бы, зачем провозглашать новый стандарт? Почему бы не использовать 802.11 или 3G? Фактически, WiMAX комбинирует подходы и 802.11 и 3G, становясь больше похожа на технологию 4G.

Как и стандарт 802.11, WiMAX относится к беспроводным технологиям присоединения устройств к Интернету на скоростях порядка нескольких мегабит в секунду, вместо использования кабеля или DSL. Устройства могут быть мобильными, или, по крайней мере, портативными. WiMAX не начинался с добавления данных с низкой скоростью к подобным голосовым сотовым сетям; 802.16 был разработан, чтобы перенести IP пакеты по радиоканалу и соединиться с основанной на IP сетью с минимумом суеты. Чтобы поддержать различные приложения, пакеты могут нести трафик соединения равноправных узлов ЛВС, звонки IP-телефонии или потоковые мультмедиа-трансляции. Так же как и 802.11, WiMAX основан на технологии OFDM, чтобы гарантировать хорошую работу, несмотря на ухудшения беспроводного сигнала, такие как многолучевое затухание, и на технологии MIMO, чтобы достигнуть высоких уровней пропускной способности.

Однако WiMAX больше походит на 3G (и, таким образом, отличается от 802.11) в нескольких ключевых отношениях. Ключевая техническая проблема состоит в том, чтобы достигнуть большой емкости эффективным использованием спектра, так чтобы большое количество абонентов в зоне охвата могли получить высокую пропускную способность. Типичные расстояния, по крайней мере, в 10 раз больше чем для сетей 802.11. Следовательно, базовые станции WiMAX — более мощные, чем точки доступа 802.11. Чтобы обработать более слабые сигналы на больших расстояниях, базовая станция использует большую мощность и лучшие антенны, а также выполняет больше работы по обработке ошибок. Чтобы максимизировать пропускную способность, передачи для каждого абонента тщательно распланированы базовой станцией; использование спектра не позволяет использовать CSMA/CA, который с коллизиями может впустую потратить пропускную способность.

Имеющий лицензию спектр — ожидаемый случай для WiMAX, в США это, как правило, приблизительно 2,5 ГГц. Система в целом существенно более оптимизирована, чем 802.11. Сложность стоит того, учитывая крупную сумму денег, потраченную на лицензированный спектр. В отличие стандарта 802.11 результатом является управляемый и надежный сервис с хорошей поддержкой качества обслуживания.

Со всеми этими особенностями 802.16 наиболее близко к сотовым сетям 4-го поколения (4G), которые теперь стандартизируются под именем **LTE (Long Term Evolution)**. В то время как сотовые сети 3G основаны на CDMA и поддерживают речь и данные, сети 4G будут основаны на OFDM с MIMO и ориентированы на передачу данных, а передача голоса будет только одним из приложений. Выглядит так, как будто WiMAX и 4G находятся на встречных курсах с точки зрения технологии и приложений. Возможно, эта конвергенция неудивительна, учитывая, что Интернет — революционная технология, а OFDM и MIMO — самые известные технологии для того, чтобы эффективно использовать спектр.

4.5.2. Стандарт 802.16: архитектура и стек протоколов

Архитектура стандарта 802.16 показана на рис. 4.27. Базовые станции соединяются непосредственно с базовой сетью провайдера, которая, в свою очередь, соединена с Интернетом. Базовые станции общаются со станциями по беспроводному радиоинтерфейсу. Существует два вида станций. Абонентские станции (subscriber stations) остаются в неподвижном местоположении, например, в случае широкополосного доступа к Интернету для домов. Мобильные станции могут обслуживаться в то время, как они перемещаются, например автомобиль, оборудованный WiMAX.

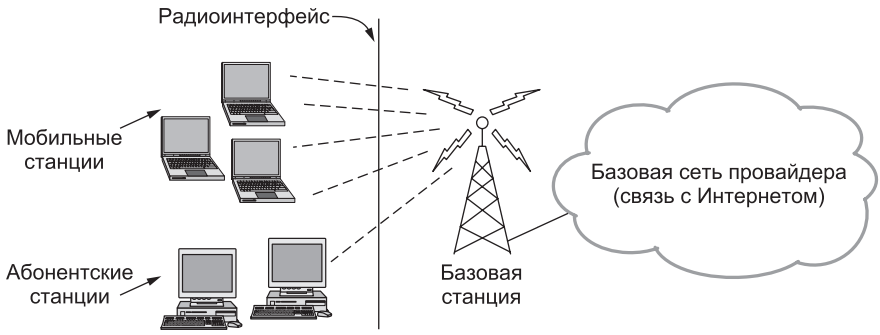


Рис. 4.27. Архитектура стандарта 802.16

Стек протокола стандарта 802.16, который используется в радиоинтерфейсе, показан на рис. 4.28. Общая структура подобна другим стандартам серии 802, однако здесь больше подуровней. Нижний уровень имеет дело с передачей, и здесь мы показали только популярные предложения 802.16, неподвижный и мобильный WiMAX. Для каждого предложения имеется свой физический уровень. Оба уровня работают в лицензированном спектре ниже 11 ГГц и используют OFDM, но по-разному.

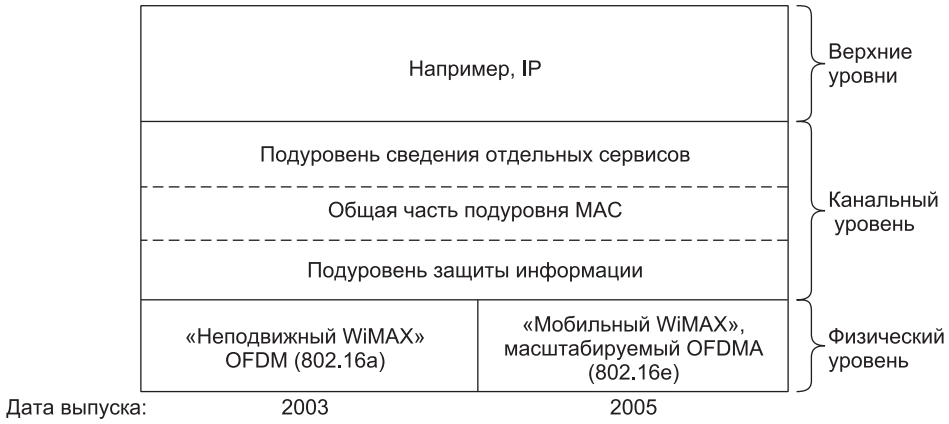


Рис. 4.28. Стек протоколов 802.16

Находящийся над физическим уровнем канальный уровень состоит из трех подуровней. Нижний из них относится к защите информации (security sublayer), что очень критично для публичных уличных сетей, в отличие от частных сетей в помещениях. На этом подуровне производится шифрование, дешифрование данных, а также управления ключами доступа.

Затем следует общая часть подуровня MAC. Именно на этом уровне иерархии располагаются основные протоколы — в частности, протоколы управления каналом. Идея состоит в том, что базовая станция полностью контролируют всю систему. Она очень эффективно распределяет очередность передачи нисходящего трафика абонентам и играет главную роль в управлении восходящим трафиком (от абонента к базовой станции). От всех остальных стандартов 802.x MAC-подуровень стандарта 802.16 отличается тем, что он полностью ориентирован на установку соединения. Таким образом, можно гарантировать определенное качество обслуживания при предоставлении услуг телефонной связи и при передаче мультимедиа.

Подуровень сведения отдельных сервисов (service specific convergence sublayer) играет роль подуровня управления логическим соединением в других протоколах 802.x. Его функция заключается в организации интерфейса для сетевого уровня. Чтобы легко объединяться с различными верхними уровнями, определены различные уровни конвергенции. Важный выбор — IP, хотя стандарт определяет отображения также и для таких протоколов, как Ethernet и ATM. Так как IP — протокол без установления соединения, а 802.16 подуровня MAC — модель на основе соединения, этот уровень должен осуществить отображение между адресами и соединениями.

4.5.3. Стандарт 802.16: физический уровень

Большая часть реализаций WiMAX использует лицензируемый спектр около 3,5 ГГц или 2,5 ГГц. Ключевая проблема, как и для 3G, — найти доступный спектр. Поэтому стандарт 802.16 разработан с гибкостью. Он допускает работу в диапазонах от 2 до 11 ГГц. Поддержаны и каналы различных размеров, например 3,5 МГц для неподвижного WiMAX и от 1,25 до 20 МГц для мобильного WiMAX.

Передачи посылаются по этим каналам с применением метода OFDM, который был описан в разделе 2.5.3. По сравнению с 802.11, схема OFDM в 802.16 оптимизирована, чтобы максимально использовать лицензированный спектр и широкую область передачи. Канал разделен на большее количество поднесущих с более длительной продолжительностью символа, чтобы выдержать большие беспроводные деградации сигнала; параметры WiMAX приблизительно в 20 раз больше, чем сопоставимые параметры в 802.11. Например, в мобильном WiMAX есть 512 поднесущих для канала на 5 МГц, и время, чтобы послать символ на каждой поднесущей составляет примерно 100 мкс.

Символы на каждой поднесущей посылаются с модуляцией по схеме QPSK, QAM-16 или QAM-64, которые были описаны в разделе 2.5.3. Если мобильный телефон или абонент расположен недалеко от БС и получаемый сигнал имеет высокий уровень соотношения сигнал/шум, то может применяться QAM-64 с шестью битами на символ. Для достижения удаленных станций с низким уровнем сигнал/шум может быть использована схема QPSK с двумя битами на символ.

Сначала данные кодируются для устранения ошибок с использованием сверточного кодирования (или еще лучшей схемы), как было описано в разделе 3.2.1. Такое кодирование распространено на шумных каналах, чтобы допускать отдельные битовые ошибки, без необходимости выполнять повторные передачи. Фактически, методы модуляции и кодирования должны казаться знакомыми к настоящему времени, так как они используются, как мы изучили, для многих сетей, включая 802.11, кабель и DSL. Конечный результат состоит в том, что базовая станция может передавать информацию со скоростью до 12,6 Мбит/с для нисходящего трафика и до 6,2 Мбит/с для восходящего трафика на канал в 5 МГц и пару антенн.

Разработчикам сетей 802.16 не нравились схемы работы стандартов GSM и DAMPS: и там, и там для нисходящего и восходящего трафика используются эквивалентные по ширине полосы частот. Таким образом, они неявно предполагают, что нисходящего трафика столько же, сколько восходящего. Голосовая связь действительно в основном симметрична, но для доступа в Интернет (и, разумеется, веб-серфинга) обычно нисходящий трафик превосходит восходящий. Соотношение составляет 2:1, 3:1 или еще больше.

Поэтому разработчики выбрали гибкую схему деления канала между станциями, называемую **OFDMA (Orthogonal Frequency Division Multiple Access — множественный доступ с ортогональным частотным разделением каналов)**. С OFDMA различные наборы поднесущих могут быть назначены на различные станции, так чтобы больше чем одна станция могла послать или получать одновременно. Если бы это был стандарт 802.11, то все поднесущие в любой данный момент использовались бы одной станцией. Дополнительная гибкость такого метода назначения полосы пропускания может увеличить производительность, потому что данная поднесущая могла быть утрачена в одном приемнике из-за многолучевых эффектов, но быть чистой в другом. Поднесущие могут быть назначены станциям, которые могут использовать их лучше всего.

Имея асимметричный трафик, станции обычно чередуют передачу и прием. Этот метод называют **TDD (Time Division Duplex — дуплекс с временным разделением)**. Альтернативный метод, при котором станция посылает и получает данные в то же самое время (на различных поднесущих частотах), называют **FDD (Frequency Division**

Duplex — дуплекс с частотным разделением). WiMAX допускает оба метода, но предпочителен TDD, потому что он более гибкий и его легче осуществить.

На рис. 4.29 показан пример структуры кадра, которая повторяется в течение долгого времени. Она начинается с преамбулы для синхронизации всех станций, затем следует нисходящая передача от базовой станции. Сначала базовая станция посылает карты, которые говорят всем станциям, как нисходящие и восходящие поднесущие назначены кадру. Базовая станция управляет картами, таким образом, она может выделять различную часть полосы пропускания станциям от кадра к кадру, в зависимости от потребностей каждой станции.

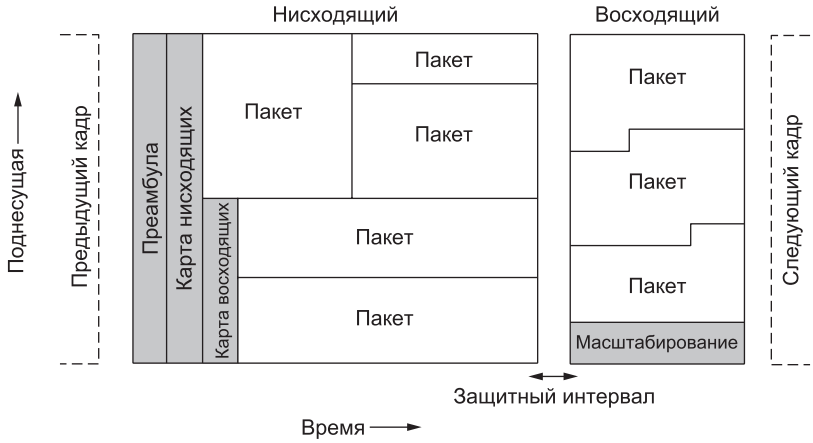


Рис. 4.29. Структура кадра для OFDMA и дуплекса с временным разделением

Затем базовая станция посылает пакет трафика абонентским и мобильным станциям на поднесущих в соответствии с временем, указанным в карте. Передачи нисходящего трафика заканчиваются защитным интервалом, позволяющим станциям переключиться с режима приема на передачу.

Наконец, абонентские и мобильные станции посылают свои пакеты трафика к базовой станции в восходящих позициях, которые были выделены для них в карте. Один из этих восходящих пакетов зарезервирован для **масштабирования (ranging)** — это процесс, при котором новые станции корректируют свою синхронизацию и запрашивают начальную полосу пропускания, чтобы соединиться с базовой станцией. Так как никакое соединение на данном этапе не установлено, новые станции только осуществляют передачу и надеются, что коллизий нет.

4.5.4. Стандарт 802.16: протокол подуровня MAC

Итак, уровень передачи данных разделен на три подуровня, как показано на рис. 2.28. Поскольку мы не будем вплоть до главы 8 касаться принципов криптографии, то сейчас нет смысла пояснять работу подуровня защиты информации. Достаточно сказать, что для сокрытия передаваемых данных применяется шифрование, причем шифруются только сами данные, а заголовки не шифруются. Это означает, что

злоумышленник может узнать, кто с кем разговаривает, но не может подслушать содержание разговора.

Если вы уже знакомы с криптографией, то ниже приводится один абзац, из которого вы поймете, какие именно принципы применяются подуровнем защиты информации. В противном случае, в следующем абзаце вы найдете мало знакомых слов. Лучше перечитать его после ознакомления с главой 8.

Когда абонент соединяется с базовой станцией, выполняется взаимная идентификация с использованием алгоритма RSA с открытым ключом (сертификат X.509). Сама передаваемая информация шифруется с помощью симметричного криптографического ключа: или AES (Rijndael), или DES со сцеплением зашифрованных блоков (cipher block chaining). Целостность данных проверяется алгоритмом SHA-1. Ну что, не очень страшный абзац получился?

Теперь перейдем к общей части подуровня MAC. Подуровень MAC ориентирован на соединение и является точка-многоточечным, это означает, что одна базовая станция общается с несколькими абонентскими станциями. Большая часть этой схемы заимствована у кабельных модемов, в которых один головной узел кабеля управляет обменом с несколькими кабельными модемами в помещениях пользователей.

Канал нисходящего трафика устроен довольно просто. Базовая станция управляет пакетами физического уровня, которые используются, чтобы послать информацию различным абонентским станциям. Подуровень MAC просто упаковывает свои кадры в эту структуру. Существуют несколько различных вариантов уменьшения служебных данных. Например, кадры MAC можно посылать индивидуально или упакованными один за другим в группу.

С восходящим каналом все несколько сложнее, поскольку имеются конкурирующие между собой станции, желающие получить доступ к нему. Его распределение тесно связано с вопросом качества обслуживания. Определены четыре класса сервисов.

1. Сервис с постоянной битовой скоростью.
2. Сервис реального времени с переменной битовой скоростью.
3. Сервис, работающий не в реальном масштабе времени, с переменной битовой скоростью.
4. Сервис с обязательством приложения максимальных усилий по предоставлению услуг.

Все предоставляемые стандартом 802.16 сервисы ориентированы на соединение, и каждое соединение получает доступ к одному из приведенных выше классов сервиса, что определяется при установке связи. Такое решение сильно отличается как от 802.11, так и от Ethernet, где отсутствовали какие-либо намеки на установление соединения на подуровне MAC.

Сервис с постоянной битовой скоростью предназначен для передачи несжатой речи, такой как передается по каналу T1. Здесь требуется передавать предопределенный объем данных в предопределенные временные интервалы, что реализуется путем назначения каждому соединению такого типа своих интервалов. После того как канал оказывается распределенным, доступ к временным интервалам осуществляется автоматически и нет необходимости запрашивать каждый из них по отдельности.

Сервис реального масштаба времени с переменной битовой скоростью применяется при передаче сжатых мультимедийных данных и других программных приложений реального времени. Необходимая в каждый момент времени полоса пропускания может меняться. Та или иная полоса выделяется базовой станцией, которая опрашивает абонента через определенные промежутки времени с целью выявления необходимой на текущий момент ширины канала.

Сервис, работающий не в реальном масштабе времени, с переменной битовой скоростью предназначен для интенсивного трафика — например, передачи файлов большого объема. Здесь базовая станция тоже опрашивает абонентов довольно часто, но не в строго установленные моменты времени. Соединения с этим сервисом могут также использовать описанный ниже сервис с обязательством приложения максимальных усилий, чтобы запросить полосу.

Наконец, сервис с обязательством приложения максимальных усилий используется для всех остальных типов передачи. Никаких опросов здесь нет, а станции, желающие захватить канал, должны соперничать с другими станциями, которым требуется тот же класс сервиса. Запрос пропускной способности осуществляется во временных интервалах, помеченных в карте распределения восходящего потока как доступные для конкуренции. Если запрос прошел удачно, это будет отмечено в следующей карте распределения нисходящего потока. В противном случае абонент-неудачник должен продолжать борьбу. Для минимизации числа коллизий используется взятый из Ethernet алгоритм двоичной экспоненциальной выдержки.

4.5.5. Стандарт 802.16: структура кадра

Все кадры подуровня управления доступом к среде (MAC) начинаются с одного и того же заголовка. За ним следует (или не следует) поле данных, и кончается кадр также не обязательным полем контрольной суммы (CRC). Структура кадра показана на рис. 4.30. Поле данных отсутствует в служебных кадрах, которые предназначены, например, для запроса временных интервалов. Контрольная сумма (как ни странно) тоже является необязательной, благодаря тому, что исправление ошибок производится на физическом уровне, и никогда не бывает попыток повторно переслать кадры информации, передающейся в реальном масштабе времени. Так если все равно нет повторных передач, зачем же беспокоить аппаратуру вычислением и проверкой контрольных сумм? Но если контрольная сумма есть, она стандартная для IEEE 802, а подтверждения и повторные передачи используются для надежности.

Давайте кратко рассмотрим поля заголовка (рис. 4.30, *а*). Бит *EC* говорит о том, шифруется ли поле данных. Поле *Тип* указывает тип кадра (в частности, сообщает о том, пакуется ли кадр и есть ли фрагментация). Поле *CI* указывает на наличие либо отсутствие поля финальной контрольной суммы. Поле *EK* сообщает, какой из ключей шифрования используется (если он вообще используется). В поле *Длина* содержится информация о полной длине кадра, включая заголовок. *Идентификатор соединения* сообщает, какому из соединений принадлежит кадр. В конце заголовка имеется поле *Контрольная сумма заголовка*, значение которого вычисляется с помощью полинома $x^8 + x^2 + x + 1$.

Рис. 4.30. Кадр: *а* — обычный; *б* — запроса канала

В протоколе 802.16 имеется много типов кадров. На рис. 4.33, *б* показан пример кадра запроса канала. Он начинается с единичного, а не нулевого бита и в целом напоминает заголовок обычного кадра, за исключением второго и третьего байтов, которые составляют 16-битное число, говорящее о требуемой полосе для передачи соответствующего числа байт. В кадре запроса канала отсутствует поле данных, нет и контрольной суммы всего кадра.

Можно долго говорить о стандарте 802.16, но все-таки не здесь. За дополнительной информацией обращайтесь, пожалуйста, к официальному описанию стандарта IEEE802.16-2009.

4.6. Bluetooth

В 1994 году компания Л. М. Эриксона (L. M. Ericsson) заинтересовалась вопросом беспроводной связи между мобильными телефонами и другими устройствами (например, портативными компьютерами). Совместно с четырьмя другими небезызвестными компаниями (IBM, Intel, Nokia и Toshiba) в 1998 году была сформирована специальная группа (SIG — Special Interest Group, то есть консорциум), которая занялась развитием стандарта беспроводного соединения вычислительных устройств и устройств связи, а также созданием аксессуаров, использующих недорогие маломощные радиоустройства небольшого радиуса действия. Проект был назван **Bluetooth** («Синий зуб») в честь великого короля викингов по имени Гаральд Синий Зуб II (940—981), который объединил (читай, завоевал) Данию и Норвегию. Ну да, он тоже сделал это без помощи проводов.

Bluetooth 1.0 появился в июле 1999 года, и с тех пор SIG никогда не оглядывалась назад. Теперь всевозможные потребительские электронные устройства используют Bluetooth — от мобильных телефонов и ноутбуков до наушников, принтеров, клавиатур, мышей, игровых приставок, часов, аудиоплееров, навигационных устройств и т. д. Протоколы Bluetooth позволяют этим устройствам находить друг друга

и соединяться с помощью действия, называемого **сопряжение** (**pairing**), и затем надежно передавать данные.

Протоколы в течение прошедшего десятилетия также развивались. После того как стабилизировались начальные протоколы, в 2004 году к Bluetooth 2.0 были добавлены более высокие скорости передачи данных. Версия Bluetooth 3.0 2009 года может использоваться для сопряжения устройств в комбинации с 802.11 для высокоскоростной передачи данных. Версия 4.0 от декабря 2009 года определила работу с низким энергопотреблением. Это будет удобно для людей, которые не хотят регулярно менять батареи во всех устройствах в доме. Ниже мы опишем основные аспекты Bluetooth.

4.6.1. Архитектура Bluetooth

Начнем изучение системы Bluetooth с краткого обзора того, из чего она состоит и для чего предназначена. Основу Bluetooth составляет **пикосеть** (**piconet**), состоящая из одного главного узла и нескольких (до семи) подчиненных узлов, расположенных в радиусе 10 метров. В одной и той же комнате, если она достаточно большая, могут располагаться несколько пикосетей. Более того, они могут даже связываться друг с другом посредством моста (специального узла), как показано на рис. 4.31. Несколько объединенных вместе пикосетей составляют **рассеянную сеть** (**scatternet**).

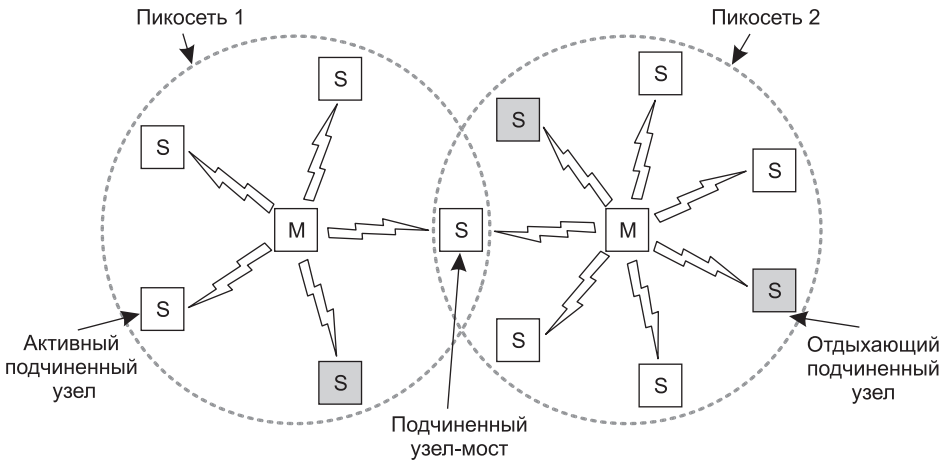


Рис. 4.31. Две пикосети могут, соединившись, сформировать рассеянную сеть

Помимо семи активных подчиненных узлов, один главный узел может поддерживать до 255 так называемых отдыхающих узлов. Это устройства, которые главный узел перевел в режим пониженного энергопотребления — за счет этого продлевается ресурс их источников питания. В таком режиме узел может только отвечать на запросы активации или на сигнальные последовательности от главного узла. Существует еще два промежуточных режима энергопотребления — приостановленный и анализирующий, но мы их сейчас рассматривать не будем.

Такое решение с главным и подчиненным узлом оказалось очень простым и дешевым в реализации (вся микросхема Bluetooth стоит менее $5). Поскольку этого и до-

бивались разработчики, такой вариант и был принят. Последствием этого является то, что подчиненные узлы получились очень неразговорчивыми — они лишь выполняют то, что им прикажет главный узел. В основе пикосетей лежит принцип централизованной системы с временным уплотнением. Главный узел контролирует временные интервалы и распределяет очередность передачи данных каждым из подчиненных узлов. Связь существует только между подчиненным и главным узлами. Прямой связи между подчиненными узлами нет.

4.6.2. Приложения Bluetooth

Большинство сетевых протоколов просто предоставляют каналы связи между коммуникационными единицами и оставляют прикладное использование этих каналов на усмотрение разработчиков. Например, в стандарте 802.11 ничего не говорится о том, что пользователи должны использовать свои ноутбуки для чтения электронной почты, работы в Интернете и т. п. В противоположность этому, Bluetooth специфицирует отдельные поддерживаемые приложения и для каждого из них предоставляет свой набор протоколов. На момент написания данного раздела было 25 таких приложений, называемых **профилями (profiles)**. К сожалению, это приводит к сильному усложнению системы. Мы опустим многие детали в нашем описании, но коротко рассмотрим профили, чтобы увидеть, что группа Bluetooth пыталась достичь.

Шесть профилей предназначены для различного использования аудио и видео. Например, профиль intercom позволяет двум телефонам соединяться друг с другом наподобие раций. Профили наушников и устройств hands-free и обеспечивают этим устройствам связь с базовой станцией. Это удобно, например, при управлении автомобилем.

Другие профили предназначены для потоковой передачи стереозвука и видео, скажем, от портативного аудиоплеера к наушникам или от цифрового фотоаппарата до телевизора.

Профиль HID предназначен для устройств взаимодействия с человеком — соединения с компьютером клавиатур и мышей. Другие профили позволяют мобильному телефону или другому компьютеру получать изображение от камеры или посылать изображения принтеру. Возможно, более интересен профиль, позволяющий использовать мобильный телефон в качестве пульта дистанционного управления для телевизора (с поддержкой Bluetooth).

Следующая группа профилей имеет отношение к сетям. Профиль доступа к ЛВС позволяет устройству Bluetooth подсоединиться к сети непосредственно или получить удаленный доступ к сети, как и в 802.11, через точку доступа. Профиль удаленного доступа (dail-up networking) был, собственно говоря, тем, ради чего изначально был задуман весь проект. Он позволяет ноутбуку соединяться с мобильным телефоном, имеющим встроенный модем, без использования проводов.

Были также определены профили для обмена информации на более высоком уровне. В частности, профиль синхронизации предназначен для загрузки данных в мобильный телефон, когда его владелец выходит из дома, и извлечения их после возвращения.

Мы пропустим остальную часть профилей, упомянем только, что некоторые профили служат основой, на которой построены профили, упомянутые выше. Профиль

группового доступа, на котором строятся все другие профили, обеспечивает установку и поддержку защищенной от несанкционированного доступа связи (канала) между главным и подчиненным узлами. Другие групповые профили определяют основы обмена объектами и передачи аудио и видео. Служебные профили широко используются для таких функций, как эмуляция последовательного канала, что особенно полезно при работе со многими устаревшими приложениями.

Неужели действительно так необходимо было подробно описывать в стандарте все приложения и предоставлять наборы протоколов для каждого из них? Может быть и нет, но было создано довольно много рабочих групп, занимавшихся различными аспектами применения системы. Каждая рабочая группа разработала свой профиль. Считайте это демонстрацией закона Конвея в действии. (В апреле 1968 года в журнале *Datamation* была опубликована статья Мелвина Конвея (Melvin Conway), в которой утверждалось, что если поручить написание компилятора n программистам, то получится n-проходный компилятор. В более общем виде эта мысль звучит так: структура программного обеспечения отражает структуру группы разработчиков.) Наверное, можно было обойтись не 25, а двумя наборами протоколов — один для передачи файлов и один для передачи данных в реальном масштабе времени.

4.6.3. Bluetooth: набор протоколов

Стандарт Bluetooth включает в себя множество протоколов, довольно свободно разбитых на уровни, как показано на рис. 4.32. Структура на первый взгляд не следует ни модели OSI, ни TCP/IP, ни 802, ни какой-либо другой известной модели.

В самом низу находится физический (радиотехнический) уровень, который вполне соответствует моделям OSI и 802. На нем описывается радиосвязь и применяемые методы модуляции. Многое здесь направлено на то, чтобы сделать систему как можно дешевле и доступнее массовому покупателю.

Уровень управления каналом связи (прямой передачи) чем-то напоминает подуровень MAC, но включает в себя и некоторые элементы физического уровня. Здесь описывается то, как главный узел управляет временными интервалами и как эти интервалы группируются в кадры.

Далее следуют два протокола, которые используют протокол управления каналом связи. Протокол управления соединением устанавливает логические каналы между устройствами, управляет режимами энергопотребления, сопряжением и шифрованием, а также качеством обслуживания. Он находится ниже линии интерфейса хост-контроллера. Этот интерфейс — удобство для реализации: как правило, протоколы ниже линии реализуются на чипе Bluetooth, а протоколы выше линии — на устройстве Bluetooth, где чип размещен.

Протокол канального уровня — это **L2CAP (Logical Link Control and Adaptation Protocol — протокол управления логическими каналами и согласования)**. Он собирает сообщения переменной длины и при необходимости обеспечивает надежность. L2CAP используется многими протоколами, в том числе и двумя описанными ранее служебными протоколами.

Протокол обнаружения сервисов используется для определения местонахождения служб в пределах сети. Протокол RFcomm эмулирует работу стандартного последо-

вательного порта ПК, к которому обычно подключаются клавиатура, мышь, модем и другие устройства.

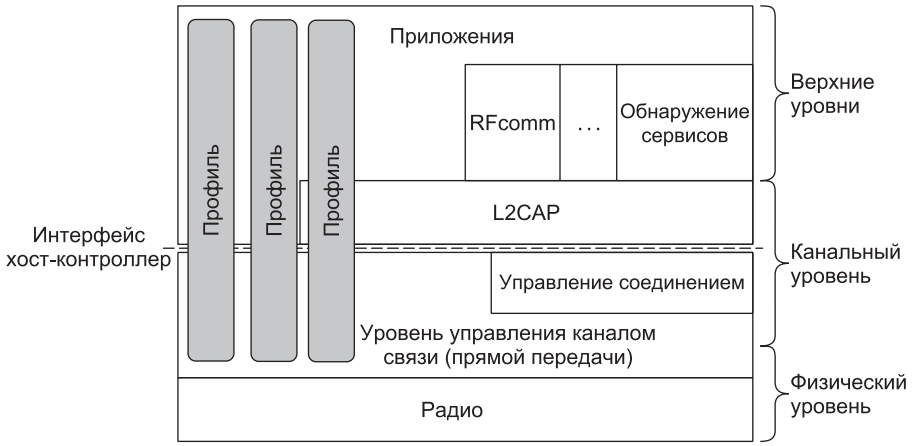


Рис. 4.32. Архитектура протоколов Bluetooth: версия 802.15

На самом верхнем уровне находятся приложения. Профили представлены вертикальными прямоугольниками, потому что каждый из них определяет часть стека протокола для конкретной цели. Специфические профили, например профили для устройств типа гарнитур, используют только те протоколы, которые необходимы для их работы. Например, профили могут включать L2CAP, если у них есть пакеты для отправки, и пропустить L2CAP в случае, если у них есть только фоновые аудиоотсчеты.

В следующих разделах мы рассмотрим уровень радиосвязи и различные протоколы канального уровня Bluetooth, поскольку они пусть грубо, но все-таки соответствуют физическому уровню и подуровню MAC в других стеках протоколов, которые мы изучили ранее.

4.6.4. Bluetooth: уровень радиосвязи

Уровень радиосвязи переносит информацию бит за битом от главного узла к подчиненным и обратно. Это маломощная приемопередающая система с радиусом действия порядка 10 метров. Она работает в ISM диапазоне 2,4 ГГц, как и 802.11. Диапазон разделен на 79 каналов по 1 МГц в каждом. Чтобы сосуществовать с другими сетями, использующими ISM диапазон, применяется расширенный спектр со скачкообразной перестройкой частоты. Возможно до 1600 скачков частоты в секунду, длительность одного временного интервала (слота или такта) — 625 мкс. Все узлы пикосетей перестраивают частоты одновременно, в соответствии с синхронизацией тактов и псевдослучайной последовательностью скачков, генерируемой главным узлом.

К сожалению, оказалось, что ранние версии Bluetooth и 802.11 интерферируют так, что нарушают передачи друг друга. Некоторые компании отреагировали на это отказом от Bluetooth в целом, но, в конечном счете, техническое решение было найдено. Оно

заключалось в том, чтобы адаптировать последовательность скачков для исключения каналов, на которых есть другие радиосигналы. Этот процесс, названный **адаптивной перестройкой рабочей частоты (adaptive frequency hopping)**, уменьшает помехи.

Для отправки бит по каналу используются три формы модуляции. Базовая схема состоит в использовании кодирования со сдвигом частоты, чтобы посылать 1-битовый символ каждую микросекунду, что дает общую скорость данных 1 Мбит/с. Большие скорости появились начиная с версии Bluetooth 2.0. Эти скорости используют кодирование со сдвигом фазы, чтобы послать или 2 или 3 бита за символ, для достижения общей скорости данных 2 или 3 Мбит/с. Такие более высокие скорости применяются только для кадров, содержащих данные.

4.6.5. Bluetooth: уровень немодулированной передачи

Уровень немодулированной передачи (управления каналом связи) — это наиболее близкий к MAC-подуровню элемент иерархии Bluetooth. Он трансформирует простой поток бит в кадры и определяет некоторые ключевые форматы. В простейшем случае главный узел каждой пикосети выдает последовательности временных интервалов по 625 мкс, причем передача данных со стороны главного узла начинается в четных тактах, а со стороны подчиненных узлов — в нечетных. Эта схема, по сути дела, традиционное временно́е уплотнение, в котором главная сторона получает одну половину временных интервалов, а подчиненные делят между собой вторую. Кадры могут быть длиной 1, 3 или 5 тактов.

В каждом кадре уходит 126 служебных бит на код доступа и заголовок, кроме того, время установки занимает 250—260 мкс на переключение частоты, чтобы позволить недорогим радиосхемам становиться устойчивыми. Полезные данные кадра могут быть для конфиденциальности зашифрованы с помощью ключа, который выбирается, когда ведущее устройство соединяется с ведомым. Переключения частоты происходят только между кадрами, но не во время передачи кадра. В результате передача 5-тактового кадра намного более эффективна чем 1-тактового, потому что при тех же служебных расходах посылается больше данных.

Протокол управления соединениями устанавливает логические каналы, называемые **соединениями (links)**, чтобы переносить кадры между главными и подчиненными устройствами, которым необходимо обнаруживать друг друга.

Прежде чем будет использоваться соединение, два устройства проходят процедуру сопряжения. Более старый метод сопряжения — оба устройства должны быть сконфигурированы с одним и тем же PIN-кодом из четырех цифр (**PIN**, **Personal Identification Number — личный идентификационный номер**). Соответствие PIN позволяет устройству знать, что оно соединилось с нужным удаленным устройством. Однако лишенные воображения пользователи и использование значений по умолчанию устройств, таких как «0000» и «1234» ведут к тому, что этот метод на практике обеспечивает не очень высокий уровень безопасности.

Новый **безопасный простой метод сопряжения (secure simple pairing)** позволяет пользователям подтвердить, что оба устройства показывают один и тот же ключ, или видеть ключ на одном устройстве и ввести его на втором. Этот метод более безопасен,

потому что пользователи не должны выбирать или устанавливать PIN. Они просто подтверждают ключ, более длинный и произведенный устройством. Конечно, этот метод не может использоваться на некоторых устройствах с ограниченным вводом/выводом, таких как беспроводные гарнитуры.

Когда сопряжение завершено, протокол устанавливает соединения. Существует два основных типа соединений. Первый вид называется **SCO (Synchronous Connection Oriented — синхронный с установлением связи)**. Он предназначен для передачи данных в реальном масштабе времени — это требуется, например, при телефонных разговорах. Такой тип канала получает фиксированный временной интервал для передачи в каждом из направлений. У подчиненного узла может быть до трех соединений типа SCO с главным узлом, каждое из которых представляет собой аудиоканал PCM с пропускной способностью 64 000 бит/с. Из-за критичной ко времени передачи природы SCO кадры, переданные по данному типу канала, никогда не пересылаются заново. Вместо этого может быть использована прямая коррекция ошибок, обеспечивающая более надежное соединение.

Другой тип соединения называется **ACL (Asynchronous Connectionless — асинхронный без установления связи)**. Этот тип связи используется для коммутации пакетов данных, которые могут появиться в произвольный момент времени. Трафик ACL доставляется по принципу максимально прилагаемых усилий для обеспечения сервиса. Никаких гарантий не дается. Кадры могут теряться и пересылаться повторно. У подчиненного узла может быть только одно ACL-соединение со своим главным узлом.

Данные, отправленные по ACL-соединению, появляются с уровня L2CAP. Этот уровень выполняет четыре основные функции. Во-первых, он принимает пакеты размером до 64 Кбайт с верхних уровней и разбивает их на кадры для передачи по физическому каналу. На противоположном конце этот же уровень используется для обратного действия — объединения кадров в пакеты.

Во-вторых, L2CAP занимается мультиплексированием и демультиплексированием множества источников пакетов. После сборки пакета он определяет, куда следует направить пакет (например, протоколу RFcomm или протоколу обнаружения сервисов).

В-третьих, L2CAP управляет контролем ошибок и повторной пересылкой кадров. Он определяет ошибки и пересылает пакеты, которые не были опознаны. Наконец, L2CAP обеспечивает качество обслуживания, требуемое несколькими соединениями.

4.6.6. Bluetooth: структура кадра

Существует несколько форматов кадров Bluetooth, наиболее важный из которых показан в двух формах на рис. 4.33. В начале кадра указывается код доступа, который обычно служит идентификатором главного узла. Это позволяет двум главным узлам, которые расположены достаточно близко, чтобы «слышать» друг друга, различать, кому из них предназначаются данные. Затем следует заголовок из 54 бит, в котором содержатся поля, характерные для кадра подуровня MAC. Если кадр отправляется с базовой скоростью, далее расположено поле данных. Его размер ограничен 2744 битами (для передачи за пять тактов). Если кадр имеет длину, соответствующую одному тактовому интервалу, то формат остается таким же, с той разницей, что поле данных в этом случае составляет 240 бит.

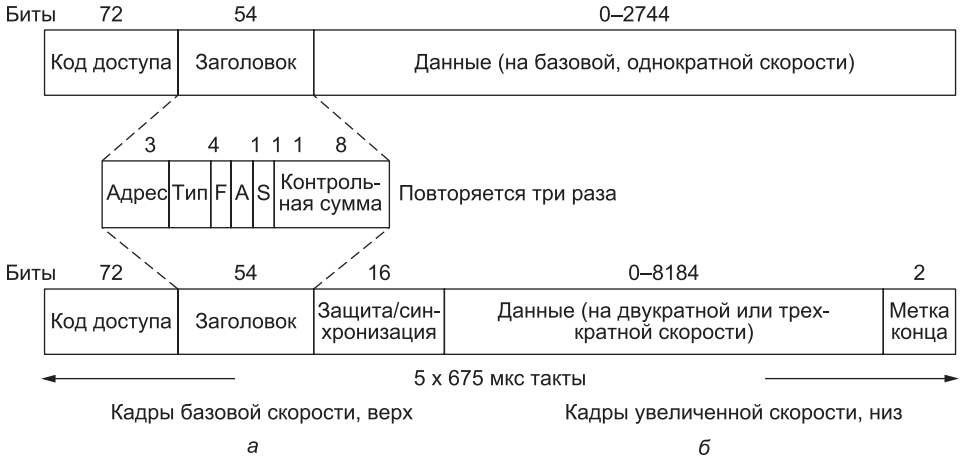


Рис. 4.33. Типичный информационный кадр Bluetooth: *а* — на базовой скорости; *б* — на увеличенной скорости

Если кадр посылается на увеличенной скорости, часть данных может быть в два или три раза больше, потому что каждый символ переносит 2 или 3 бита вместо одного бита. Этим данным предшествуют защитный интервал и образец синхронизации, который используется, чтобы переключиться на более высокую скорость передачи данных. Таким образом, код доступа и заголовок передаются на базовой скорости, и только часть данных передается на большей скорости. Кадры с большей скоростью заканчиваются короткой меткой конца.

Рассмотрим, из чего состоит обычный заголовок кадра. Поле *Адрес* идентифицирует одно из восьми устройств, которому предназначена информация. Поле *Тип* определяет тип передаваемого кадра (ACL, SCO, опрос или пустой кадр), метод коррекции ошибок и количество временных интервалов, из которых состоит кадр. Бит *F* (Flow — поток) выставляется подчиненным узлом и сообщает о том, что его буфер заполнен. Этот бит обеспечивает примитивную форму управления потоком. Бит *A* (Acknowledgement — подтверждение) представляет собой подтверждение (ACK), отсылаемое заодно с кадром. Бит *S* (Sequence — последовательность) используется для нумерации кадров, что позволяет обнаруживать повторные передачи. Это протокол с ожиданием, поэтому одного бита действительно оказывается достаточно. Далее следует 8-битная контрольная сумма заголовка. Весь 18-битный заголовок кадра повторяется трижды, что в итоге составляет 54 бита, как показано на рис. 4.33. На принимающей стороне несложная схема анализирует все три копии каждого бита. Если они совпадают, бит принимается таким, какой он есть. В противном случае все решает большинство. Как видите, на передачу 10 бит тратится в данном случае 54 бита. Причина очень проста: за все нужно платить. За обеспечение передачи данных с помощью дешевых, маломощных устройств (2,5 мВт) с невысокими вычислительными способностями приходится платить большой избыточностью.

Для ACL- и SCO-кадров применяются различные форматы поля данных. В кадрах SCO с базовой скоростью кадры устроены просто: длина поля данных всегда равна 240 бит. Возможны три варианта: 80, 160 или 240 бит полезной информации. При этом

оставшиеся биты поля данных используются для исправления ошибок. В самой надежной версии (80 бит полезной информации) одно и то же содержимое повторяется три раза (что и составляет 240 бит), как и в заголовке кадра.

Мы можем вычислить емкость следующим образом. Поскольку подчиненные узлы могут использовать только нечетные временные интервалы, им достается 800 интервалов в секунду. Столько же получает и главный узел. При 80 битах полезных данных, передающихся в одном кадре, емкость канала подчиненного узла равна 64 000 бит/с. Этому же значению равна и емкость канала главного узла. Этого как раз хватает для организации полнодуплексного PCM-канала голосовой связи (именно поэтому 1600 скачков в секунду было выбрано в качестве скорости перестройки частот). Все эти цифры говорят о том, что полнодуплексный канал со скоростью 64 000 бит/с в каждую сторону при самом надежном способе передачи информации вполне устраивает пикосеть, невзирая на то, что суммарная скорость передачи данных на физическом уровне равна 1 Мбит/с.

Эффективность 13% — результат расходов 41% емкости на время стабилизации, 20% на заголовки, и 26% на повторном кодировании. Этот недостаток выделяет значение увеличенных скоростей и кадров более чем из одного слота.

О Bluetooth можно сказать намного больше, но нет места, чтобы говорить об этом здесь. Интересующиеся могут прочитать все подробности в спецификации Bluetooth 4.0.

4.7. RFID

Мы уже рассмотрели схемы MAC от локальных до городских и персональных вычислительных сетей. В качестве последнего примера мы изучим категорию беспроводных устройств «низшего класса», которые, возможно, не признаются формирующими компьютерные сети: это **RFID (Radio Frequency Identification — радиочастотная идентификация)** метки и считыватели, которые мы описали в разделе 1.5.4.

Технология RFID существует во многих формах, используемых в смарткартах, имплантантах для домашних животных, паспортах, библиотечных книгах и пр. Форма, которую мы рассмотрим, была развита при разработках **EPC (Electronic Product Code — электронный код товара)**, которые начинались в Auto-ID Центре в Массачусетском технологическом институте в 1999 году. EPC — это замена штрихкода, которая может нести большее количество информации и считываться с помощью электроники на расстоянии до 10 м, даже не в прямой видимости. Эта технология отличается, например, от RFID-используемой в паспортах, которые должны быть помещены достаточно близко к считывателю для получения информации. Способность общаться на расстоянии делает EPC более значимым для нашего изучения.

Организация EPCglobal была создана в 2003 году, чтобы коммерциализировать развитую Auto-ID Центром технологию RFID. Их усилия получили развитие в 2005 году, когда компания Walmart потребовала от 100 своих крупнейших поставщиков маркировать все поставки метками RFID. Широкому распространению технологии препятствовала трудность конкуренции с дешевыми печатаемыми штрихкодами, но появились новые сферы использования, например водительские права. Мы опишем второе поколение этой технологии, которую неофициально называют **EPC Gen 2** (EPCglobal, 2008).

4.7.1. Архитектура EPC Gen 2

Архитектура EPC Gen 2 сети RFID показана на рис. 4.34. Она имеет два ключевых компонента: метки (теги) и считыватели. Метки RFID — маленькие, недорогие устройства, имеющие уникальный 96-битовый идентификатор EPC и небольшое количество памяти, запись и чтение которой может производиться RFID-считывателем. Память может использоваться для записи истории местоположения элемента, например, когда он движется по системе поставок.

Часто метки похожи на этикетки, которые могут быть помещены, например, на джинсы на полках в магазине. Большая часть этикетки занята антенной, которая на ней напечатана. Крошечная точка в середине — интегральная схема RFID. Другой вариант — метка RFID может быть внедрена в объект, например в водительские права. В обоих случаях метки не содержат никакой батареи и должны получать энергию для работы из радиопередачи ближайшего RFID-считывателя. Этот вид меток называют метки «Class 1», чтобы отличить их от меток с бóльшими возможностями, у которых есть батареи.

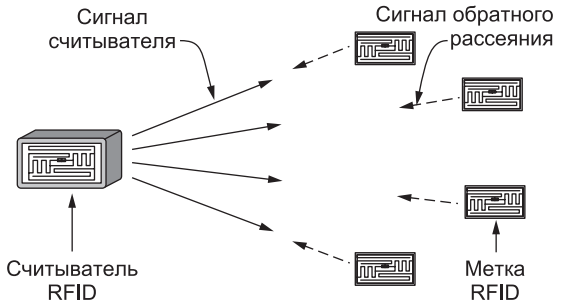


Рис. 4.34. Архитектура RFID

Считыватель — это информационный центр системы, аналогичной базовым станциям и точкам доступа в сотовых сетях и WiFi. Считыватели намного более энергоемкие, чем метки. Они имеют собственные источники энергии, часто имеют разнообразные антенны и отвечают за то, когда метки посылают и получают сообщения. Так как в диапазоне чтения обычно бывает несколько разных меток, считыватели должны решить задачу множественного доступа. Возможно и присутствие нескольких считывателей в одном пространстве, которые могут конфликтовать друг с другом.

Основное дело считывателя — инвентаризировать ближайшие метки, то есть обнаруживать их идентификаторы. Инвентаризация производится протоколом физического уровня и протоколом идентификации метки, которые в общих чертах описаны в следующих разделах.

4.7.2. Физический уровень EPC Gen 2

Физический уровень определяет, как биты пересылаются между метками и RFID-считывателем. В основном используются методы отправки беспроводных сигналов, которые мы видели ранее. В США обмен происходит в нелицензированном диапазоне ISM

полосы 902–928 МГц. Эта полоса относится к диапазону UHF (УВЧ), поэтому соответствующие метки называют UHF-метками RFID. Считыватель выполняет скачки частоты по крайней мере каждые 400 мс, чтобы распространить сигнал по каналу, ограничить помехи и выполнить нормативные требования. Чтобы закодировать биты, считыватель и метки используют формы амплитудной модуляции ASK, которую мы описали в разделе 2.5.2. Они посылают биты по очереди, таким образом, канал — полудуплексный.

Имеется два основных отличия от других физических уровней, которые мы изучили. Прежде всего, сигнал всегда передает считыватель, независимо от того, осуществляет ли коммуникацию он или метка. Естественно, считыватель передает сигнал, чтобы послать биты в метки. А для того чтобы метки послали биты считывателю, он передает сигнал фиксированной несущей, который не переносит битов. Метки собирают этот сигнал для получения энергии, которая необходима для их работы; таким образом, метка не могла бы передавать первой. Чтобы послать данные, метка меняет свое поведение — отражает ли она сигнал от считывателя, как радарный сигнал, возвращающийся от цели, или поглощает его.

Этот метод называют **обратным рассеянием (backscatter)**. Он отличается от всех других беспроводных ситуаций, которые мы видели до сих пор, когда отправитель и получатель никогда не передают оба в одно и то же время. Обратное рассеяние — низкоэнергетический способ создания меткой собственного слабого сигнала, который обнаруживается считывателем. Для того чтобы считыватель расшифровал поступающий сигнал, он должен отфильтровать исходящий сигнал, который он передает. Поскольку сигнал метки слаб, метки могут посылать биты только с низкой скоростью и не могут получать или даже обнаруживать передачи от других меток.

Второе отличие в том, что применяются очень простые формы модуляции, так чтобы они могли быть осуществлены меткой, которая работает на очень небольшой мощность и очень дешево стоит. Чтобы послать данные в метки, считыватель использует два уровня амплитуды. Биты 0 или 1 определяются в зависимости от того, сколько времени считыватель ждет перед периодом низкой мощности. Метка измеряет время между периодами низкой мощности и сравнивает это время с измеренной во время преамбулы. Как показано на рис. 4.35, 1 более длинный, чем 0.

Ответы метки состоят из изменения состояния его обратного рассеяния через фиксированные интервалы, что создает серию импульсов в сигнале. Чтобы закодировать каждый 0 или 1, может использоваться от одного до восьми периодов импульса, в зависимости от потребности в надежности. 1 имеют меньше переходов, чем 0, как показано на рис. 4.35 в примере кодирования с периодом в два импульса.

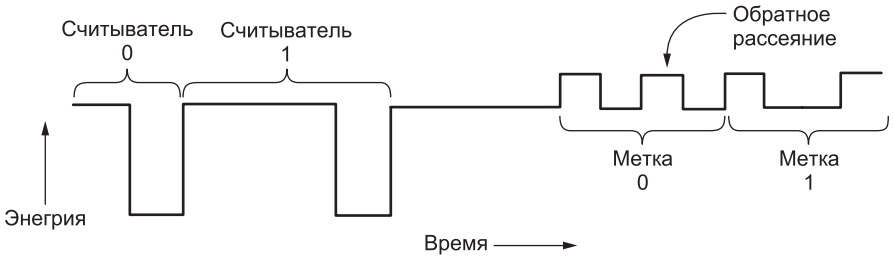


Рис. 4.35. Сигналы считывателя и сигналы обратного рассеяния от метки

4.7.3. Уровень идентификации метки EPC Gen 2

Чтобы инвентаризировать ближайшие метки, считыватель должен получить от каждой из них сообщение, содержащее идентификатор метки. Эта ситуация — задача множественного доступа, для которой в общем случае количество меток неизвестно. Считыватель мог бы передать широковещательный запрос, чтобы попросить все метки прислать свои идентификаторы. Однако немедленный ответ меток привел бы к коллизии, почти такой же, как у станций в классическом Ethernet.

В этой главе мы видели уже много способов, относящихся к задаче множественного доступа. Самый близкий к данной ситуации, когда метки не могут слышать друг друга, протокол дискретная ALOHA, один из самых первых изученных нами. Этот протокол адаптирован к использованию в Gen 2 RFID.

Последовательность сообщений, используемых, чтобы идентифицировать тег, показана на рис. 4.36. В первый слот (слот 0) считыватель посылает сообщение *Query*, чтобы запустить процесс. Каждое сообщение *QRepeat* подается в следующий слот. Считыватель сообщает меткам диапазон слотов, по которым можно рандомизировать передачи. Использовать диапазон необходимо, потому что считыватель синхронизирует метки, когда запускает процесс; в отличие от станций на Ethernet, метки не просыпаются с сообщением в выбранное время.

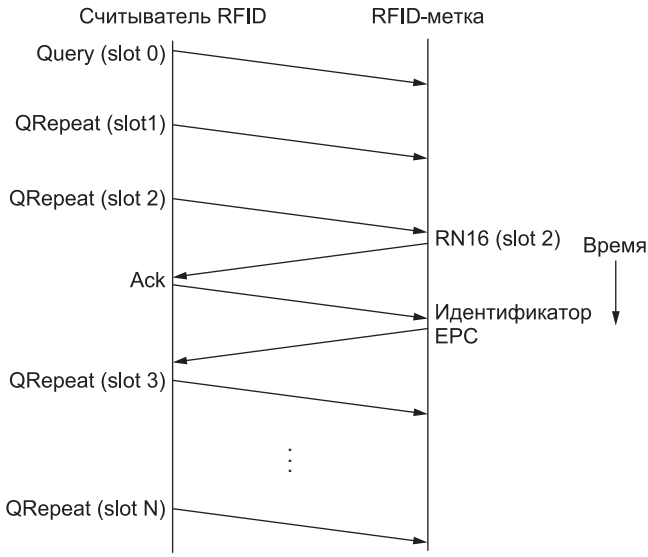


Рис. 4.36. Пример обмена сообщениями для идентификации метки

Метки выбирают случайный слот, в котором можно отвечать. На рис. 4.36 метка отвечает в слоте 2. Однако отвечая, метки не сразу посылают свои идентификаторы. Вместо этого они посылают короткое 16-битовое случайное число в сообщении *RN16*. Если нет коллизий, считыватель получает это сообщение и посылает собственное сообщение *ACK*. На этом этапе метка получает слот и посылает свой идентификатор EPC.

Обмен производится таким способом потому, что идентификаторы EPC — длинные, поэтому коллизии содержащих их сообщений были бы дороги. Вместо этого используется короткий обмен, чтобы проверить, может ли метка безопасно использовать слот, чтобы послать свой идентификатор. Как только ее идентификатор успешно передан, метка временно прекращает отвечать на новые сообщения *Query*, чтобы могли быть идентифицированы другие метки.

Ключевая проблема для считывателя — определить такое количество слотов, чтобы избежать коллизий, но не использовать слишком много слотов, отчего пострадает производительность. Это согласование похоже на двойную экспоненциальную выдержку в Ethernet. Если считыватель видит слишком много слотов без ответов или слишком много слотов с коллизиями, он может послать сообщение *QAdjust*, чтобы уменьшить или увеличить диапазон слотов, по которым отвечают метки.

Считыватель RFID может выполнять на метках и другие операции. Например, он может выбрать подмножество меток прежде, чем выполнить инвентаризацию, например собрать ответы у меток на джинсах, но не у меток на рубашках. Считыватель может также записать данные на идентифицированные метки. Эта функция может быть использована, чтобы сделать запись торговой точки или другой релевантной информации.

4.7.4. Форматы сообщения идентификации метки

Формат сообщения запроса показан на рис. 4.40 как пример сообщения от считывателя к метке. Сообщение компактно, потому что скорости передачи информации ограничены, от 27 до 128 Кбит/с. Поле команды содержит код 1000, что идентифицирует сообщение как *Query*.

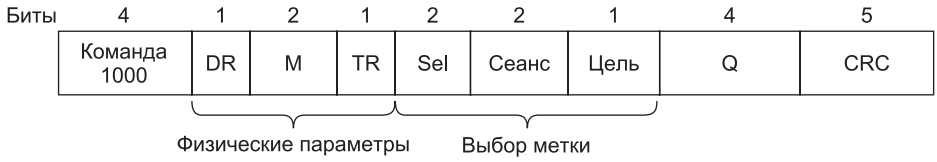


Рис. 4.37. Формат сообщения Query

Следующие флаги, *DR*, *M* и *TR*, определяют параметры физического уровня для передачи считывателя и ответов метки. Например, скорость ответа может быть установлена между 5 и 640 Кбит/с. Мы пропустим подробности этих флагов.

Затем следуют три поля, *Sel*, *Сеанс* и *Цель*, для выбора отвечающих меток. Так же как считыватели имеют возможность выбрать подмножество идентификаторов, метки отслеживают до четырех параллельных сеансов и были ли они идентифицированы в этих сеансах. Таким образом, несколько считывателей могут действовать в пересекающейся области при использовании разных сеансов.

Затем идет самый важный параметр этой команды, *Q*. Это поле определяет диапазон слотов, по которым ответят метки, от 0 до $2^Q - 1$. Наконец, имеется CRC, чтобы защитить поля сообщения. Она занимает 5 бит, что короче, чем большинство CRC, которые мы рассматривали, но и сообщение *Query* намного короче, чем большинство пакетов.

Сообщения от метки к считывателю более просты. Так как считыватель управляет ситуацией, он знает, какое сообщение ожидать в ответ на каждую из своих передач. Ответы метки просто переносят данные, такие как идентификатор EPC.

Первоначально метки применялись только в целях идентификации. Однако со временем они выросли и стали напоминать очень маленькие компьютеры. У некоторых исследовательских меток есть датчики, и они в состоянии выполнить маленькие программы, чтобы собрать и обработать данные (Sample и др., 2008). Один из взглядов на эту технологию — «Интернет вещей», который присоединяет объекты материального мира к Интернету (Welbourne и др., 2009; Gershenfeld и др., 2004).

4.8. Коммутация на канальном уровне

У многих организаций имеется по нескольку локальных сетей, которые необходимо объединять между собой. Может быть, удобно объединить эти сети в одну большую локальную сеть? Это можно сделать с помощью специальных устройств, называемых **мостами (bridges)**. Коммутаторы Ethernet, которые мы описали в разделе 4.3.4, — это современное название мостов; они обеспечивают функциональность, которая идет вне классического Ethernet и концентраторов Ethernet, чтобы облегчить соединение нескольких ЛВС в большую и более быструю сеть. Мы будем использовать термины «мост» и «коммутатор» попеременно.

Мосты работают на канальном уровне. Они анализируют адреса, содержащиеся в кадрах этого уровня, и в соответствии с ними осуществляют маршрутизацию. Поскольку мосты не исследуют сами данные, передающиеся в кадрах, то они одинаково хорошо справляются с пакетами IP, а также с другими типами пакетов, например пакетами AppleTalk. В отличие от мостов *маршрутизаторы* (*routers*) анализируют адреса в пакетах и работают, основываясь на этой информации, поэтому они могут работать только с теми протоколами, для которых предназначены.

В этом разделе мы рассмотрим работу мостов и соединение с их помощью нескольких физических локальных сетей в одну логическую локальную сеть. Кроме того, мы рассмотрим противоположную задачу — разделение одной физической локальной сети на несколько логических локальных сетей, так называемых **виртуальных ЛВС (ВЛВС — VLAN, Virtual LAN)**. Обе технологии предоставляют полезную гибкость в управлении сетями. Подробную информацию о мостах, коммутаторах и близких темах можно найти у (Seifert и Edwards, 2008) и (Perlman, 2000).

4.8.1. Применение мостов

Прежде чем перейти к обсуждению мостов, давайте рассмотрим несколько часто встречающихся ситуаций, в которых они используются. Перечислим шесть причин, по которым в организации может появиться несколько локальных сетей.

Во-первых, у многих подразделений университетов и отделов корпораций есть свои локальные сети, соединяющие персональные компьютеры, серверы и такие устройства, как принтеры. Поскольку цели различных факультетов или отделов различны,

то и объединение в локальные сети часто происходит по факультетам и отделам, которым не очень интересно, как построена сеть у соседей. Однако рано или поздно им требуется взаимодействие, поэтому появляется необходимость в мостах. В данном примере несколько отдельных локальных сетей образовалось вследствие автономности их владельцев.

Во-вторых, организации могут размещаться в нескольких зданиях, значительно удаленных друг от друга. Может оказаться дешевле создать несколько отдельных локальных сетей и затем соединить их с помощью мостов и нескольких оптоволоконных кабелей на большое расстояние, вместо того чтобы протягивать все кабели к одному центральному коммутатору.

Даже если кабели легко проложить, есть пределы их возможной длины (например, 200 м для витой пары Gigabit Ethernet). Сеть не будет работать с более длинными кабелями из-за чрезмерного ослабления сигнала или задержки туда и обратно. Единственное решение состоит в том, чтобы разделить ЛВС и установить мосты, соединяющие ее части, чтобы увеличить полную физическую дистанцию, которая может быть преодолена.

В-третьих, иногда бывает необходимо логически разделить одну локальную сеть на несколько отдельных локальных сетей, соединенных мостами, чтобы снизить нагрузку. Так, например, во многих крупных университетах в сети объединены тысячи рабочих станций, на которых работают студенты и сотрудники. В организации могут работать тысячи сотрудников. Огромные масштабы не позволяют объединить все эти рабочие станции в одну локальную сеть — компьютеров больше, чем портов в любом Ethernet концентраторе, станций больше, чем возможно в одном классическом Ethernet.

Однако емкость двух отдельных ЛВС в два раза больше, чем у одной. Мосты позволяют объединять ЛВС, сохраняя эту емкость. Идея в том, чтобы не посылать трафик на порты, где он не нужен, так чтобы каждая ЛВС могла работать на максимальной скорости. Такое поведение также увеличивает надежность, так как на одной ЛВС дефектный узел, который продолжает выводить непрерывный поток мусора, может забить всю ЛВС. Решая, что пересылать, а что нет, мост действует, как пожарные двери в здании, защищая всю систему от разрушения одним ненормальным узлом.

Для лучшего использования этих преимуществ мосты должны быть полностью прозрачны. Должна быть возможность пойти и купить мосты, включить в них кабели ЛВС, и чтобы все немедленно отлично работало.

Все это должно происходить без каких-либо изменений аппаратуры, настроек адресов коммутатора, программного обеспечения или конфигурационных таблиц и т. п. Кроме того, работа существующих ЛВС вообще не должна быть затронута мостами. Что касается затронутых станций, не должно быть никакого заметного различия, являются ли они частью ЛВС с мостами или нет. Должно быть столь же легко переместить станции в ЛВС с мостами, как и в одиночной ЛВС.

Удивительно, но создать прозрачные мосты возможно. Используется два алгоритма: алгоритм противоточного обучения, чтобы остановить трафик, посылаемый, где это не необходимо; и алгоритм связующего дерева, чтобы разрушить циклы, которые могут возникнуть при соединении коммутаторов. Теперь рассмотрим эти алгоритмы по очереди, чтобы изучить, как это волшебство достигнуто.

4.8.2. Обучаемые мосты

Топология двух ЛВС, соединенных мостом, показана на рис. 4.38 для двух вариантов. Слева, к двум многоточечным ЛВС, таким как классический Ethernet, присоединяется специальная станция — мост, который сидит на обеих ЛВС. Справа объединены ЛВС с двухточечными кабелями, включая один концентратор. Мосты — устройства, к которым присоединены станции и концентратор. Если технология ЛВС — Ethernet, мосты более известны под названием коммутаторы.

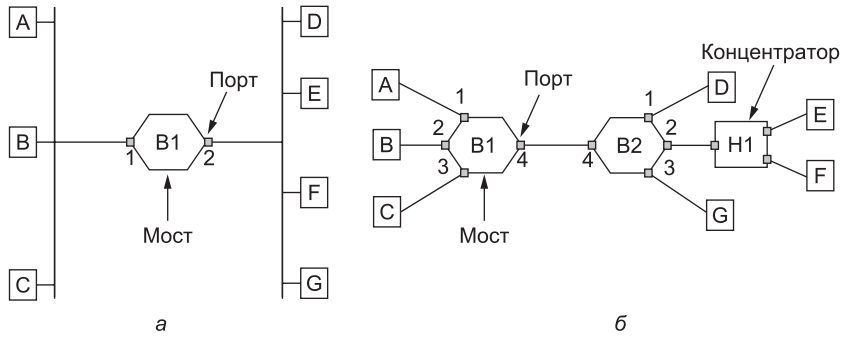


Рис. 4.38. Мосты: *а* — мост, соединяющий две многоточечные ЛВС; *б* — мосты (и концентратор), соединяющие семь станций по двухточеной схеме

Мосты были развиты, когда использовался классический Ethernet, поэтому их часто показывают в топологии с многоточечными кабелями, как на рис. 4.38, *а*. Однако все топологии, которые можно встретить сейчас, состоят из двухточечных кабелей и коммутаторов. Мосты работают одинаково в обеих ситуациях. Все станции, присоединенные к тому же самому порту на мосту, принадлежат тому же самому домену коллизий, который отличается от доменов коллизий других портов. Если есть больше, чем одна станция, как в классическом Ethernet, концентратор или полудуплексный канал, для отправки кадров используется протокол CSMA/CD.

Однако есть различие в том, как устроены соединенные ЛВС. Чтобы соединить многоточечные ЛВС, мост добавлен как новая станция в каждой из них, как показано на рис. 4.38, *а*. Чтобы соединить двухточечные ЛВС, концентраторы или соединены с мостом, или, предпочтительно, заменены мостом, чтобы увеличить производительность. На рис. 4.38, *б* мосты заменили все, кроме одного концентратора.

К одному мосту также могут быть присоединены различные виды кабелей. Например, кабель, соединяющий *B1* с мостом *B2* на рис. 4.38, *б*, мог бы быть длинным оптоволоконным каналом, в то время как кабель, соединяющий мосты со станциями, мог бы быть короткой линией на витой паре. Такое расположение полезно для соединения ЛВС в различных зданиях.

Теперь давайте рассмотрим то, что происходит в мостах. Каждый мост работает в неразборчивом режиме, то есть принимает каждый кадр, переданный станциями, присоединенными к каждому из его портов.

При появлении кадра мост должен решить, игнорировать его или переправить, и если переправить, то в какой порт. Выбор производится на основе адреса получателя.

Например, рассмотрим топологию на рис. 4.38, *а*. Если станция *A* пошлет кадр станции *B*, то мост *B1* получит кадр на порту 1. От этого кадра можно немедленно отказаться без дальнейшей суматохи, потому что он уже находится на правильном порту. Теперь предположим в топологии на рис. 4.38, *б*, что *A* посылает кадр *D*. Мост *B1*, получит кадр на порту 1 и выведет его на порт 4. Затем мост *B2* получит кадр на своем порту 4 и выведет его на своем порту 1.

Простой способ реализовать эту схему состоит в том, чтобы мост имел большую (хэш) таблицу. В таблице могут быть перечислены все возможные места назначения и к какому порту каждое относится. Например, на рис. 4.38, *б* таблица в *B1* перечисляла бы *D* как принадлежащий порту 4, так весь *B1* знал бы, в какой порт отправить кадры для *D*. Тогда, фактически, дальнейшая пересылка произойдет позже, когда достигший *B2* кадр не представляет интереса для *B1*.

Когда мосты включаются первый раз, все их хэш-таблицы пусты. Ни один мост не знает, где находятся адресаты, поэтому они используют алгоритм заливки (flooding): каждый приходящий кадр с неизвестным адресом переправляется сразу по всем направлениям, кроме того, откуда он пришел. Со временем мосты узнают расположение адресатов. Кадры, расположение получателей которых известно, направляются только в одну нужную сеть, они не заливаются.

Для обучения прозрачных мостов используется алгоритм так называемого **противоточного обучения (backward learning)**. Как уже упоминалось выше, мосты работают в неразборчивом режиме, поэтому они видят все кадры, посылаемые во всех их портах. Просматривая адреса отправителей, они могут определить, какая станция доступна по какому порту. Например, если мост *B1* на рис. 4.38, *б* видит кадр, приходящий к нему на порт 3 от станции *C*, то он понимает, что станция *C* достижима через порт 3, и делает соответствующую запись в своей таблице. Поэтому любой последующий кадр, адресованный станции *C* и приходящий в *B1* по любому другому порту, будет переправляться в порт 3.

Топология сети может меняться, по мере того как отдельные станции и мосты будут включаться, выключаться, а также перемещаться. Для поддержки динамической топологии в таблице помимо номера станции и номера сети указывается также время прибытия кадра от данной станции. При получении новых кадров это время обновляется. Таким образом, для каждой станции известно время последнего полученного от нее кадра.

Время от времени процесс сканирует хэш-таблицу и удаляет все записи, сделанные ранее нескольких минут тому назад. Таким образом, если какой-либо компьютер был выключен, перенесен в новое место и включен снова, уже через несколько минут он сможет нормально работать, и для этого не потребуется никаких специальных действий. Обратная сторона такого алгоритма заключается в том, что кадры, направляемые какой-либо станции, молчавшей в течение нескольких минут, должны будут снова посылаться во все концы методом заливки.

Процедура обработки входящего кадра зависит от того порта, через который он прибыл (порт источника) и в какой адрес направляется (адрес назначения). Процедура выглядит следующим образом:

1. Если порт источника и порт для адреса назначения совпадают, кадр игнорируется.
2. Если порт источника и порт для адреса назначения различаются, кадр переправляется в порт назначения.

3. Если порт назначения неизвестен, используется алгоритм заливки, и кадр пересылается во все порты, кроме порта источника.

Вы могли бы задаться вопросом, может ли первый случай произойти с двухточечными линиями. Ответ — это может произойти, если для соединения группы компьютеров с мостом используются концентраторы. Пример показан на рис. 4.38, *б*, где станции *E* и *F* соединены с концентратором *H1*, который, в свою очередь, соединен с мостом *B2*. Если *E* пошлет кадр *F*, то концентратор передаст его *B2*, так же как и *F*. Именно это делают концентраторы — они связывают все порты вместе так, чтобы кадр, введенный на одном порту, просто выводится на всех других портах. Кадр достигнет *B2* на порту 4, который уже является правильным выходным портом, чтобы достигнуть места назначения. Мост *B2* должен просто отказаться от кадра.

Поскольку этот алгоритм должен быть применен к каждому прибывающему кадру, обычно он осуществляется специальными чипами СБИС. Чип производит поиск и обновляет записи таблицы за несколько микросекунд. Поскольку мосты смотрят только на MAC адреса, чтобы решить, как отправить кадры, возможно начать отправку, как только появилось поле заголовка назначения, до того как дошла остальная часть кадра (конечно, если выходная линия доступна). Эта схема сокращает время ожидания прохождения через мост, а также количество кадров, которые мост должен быть в состоянии буферизовать. Такой способ называют **коммутация без буферизации пакетов (cut-through switching)** или **маршрутизация способом коммутации каналов (wormhole routing)**, и обычно он реализуется аппаратными средствами.

Мы можем посмотреть на работу моста с точки зрения стека протоколов, чтобы понять, что это означает быть устройством уровня канала. Рассмотрим кадр, посланный от станции *A* к станции *D* в конфигурации на рис. 4.38, *а*, в которой ЛВС — Ethernet. Кадр пройдет через один мост. Вид стека протокола обработки показан на рис. 4.39.

Пакет прибывает из более высокого уровня и спускается на уровень MAC Ethernet. Он приобретает заголовок Ethernet (а также метку конца, не показанную на рисунке). Этот кадр передается физическому уровню, выходит по кабелю и принимается мостом.

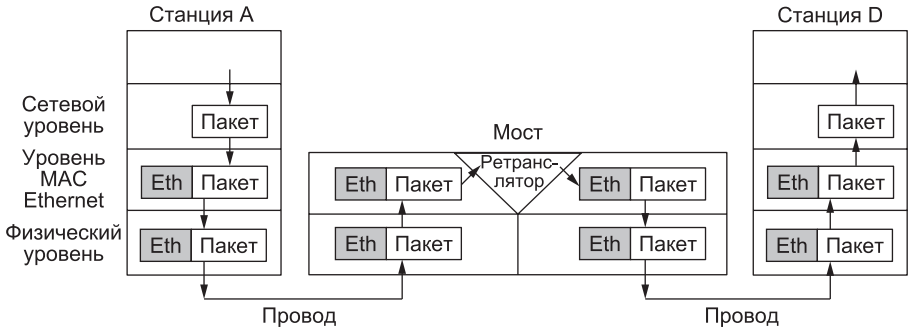


Рис. 4.39. Протоколы, обрабатываемые на мосту

В мосте кадр передается с физического уровня на уровень MAC Ethernet. Этот уровень расширяет обработку по сравнению с уровнем MAC Ethernet на станции. Он передает на ретранслятор, все еще в пределах уровня MAC. Функция ретрансляции моста использует только заголовок MAC Ethernet, чтобы определить, как обработать

кадр. В нашем случае он передает кадр тому порту уровня MAC Ethernet, который используется для достижения станции *D*, и кадр продолжает свой путь.

В общем случае, ретрансляторы на некотором уровне могут переписать заголовки для этого уровня. ВЛВС вскоре обеспечит пример. Мост ни в коем случае не должен смотреть внутрь кадра и узнавать, что он переносит IP-пакет; это не важно для обработки мостом и нарушило бы иерархическое представление протокола. Также отметьте, что мост, имеющий k портов, будет иметь k экземпляров MAC-уровней и физических уровней. В нашем простом примере $k = 2$.

4.8.3. Мосты связующего дерева

Для повышения надежности между мостами можно использовать избыточные соединения. На рис. 4.40 показаны два параллельных соединения между двумя мостами. Эта конструкция гарантирует, что, если одно соединение нарушено, сеть не будет разделена в два набора компьютеров, которые не могут говорить друг с другом.

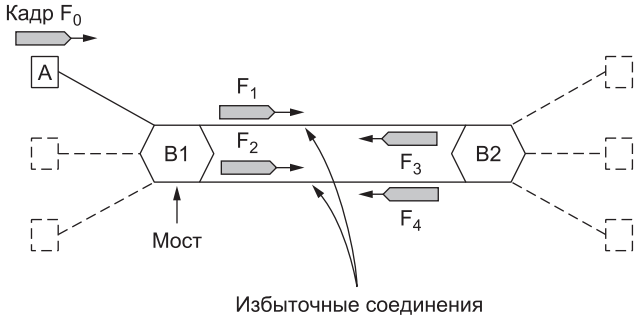


Рис. 4.40. Мосты с двумя параллельными соединениями

Такое решение, впрочем, создает некоторые дополнительные проблемы, поскольку в топологии образуются кольца. В качестве примера, иллюстрирующего указанные проблемы, рассмотрим кадр, отправленный A с ранее неизвестным адресом назначения (рис. 4.40). Каждый мост, действуя по обычным правилам обработки кадров с неизвестным получателем, использует метод заливки. В данном примере это означает, что кадр из A попадает к мосту B1, F_0. Мост посылает копии этого кадра во все остальные свои порты. Мы рассмотрим только порты, соединяющие B1 и B2 (хотя кадр будет послан и в другие). Так как из B1 в B2 есть два соединения, в B2 попадут две копии кадра. Они показаны на рис. 4.40 как F_1 и F_2.

Вскоре после этого мост B1 получает эти кадры. Разумеется, он не знает (и не может знать), что это копии одного кадра, а не два разных кадра, посланных один за другим. Поэтому мост B2 отправляет копии кадра F_1 во все свои порты. Так возникают F_3 и F_4, которые по двум соединениям отправляются обратно в B1. Мост B1 видит два новых кадра с неизвестным адресом назначения и копирует их снова. Этот цикл продолжается вечно.

Решение данной проблемы заключается в установлении связи между мостами и наложении на реальную топологию сети **связующего дерева (spanning tree)**, до-

стигающего каждого моста. В результате некоторые возможные соединения между мостами игнорируются с целью создания фиктивной бескольцевой топологии, которая является подмножеством реальной топологии.

Например, на рис. 4.41 показаны пять мостов, которые связаны и имеют соединенные с ними станции. Каждая станция соединяется только с одним мостом. Есть некоторые избыточные соединения между мостами, так что если будут использоваться все соединения, кадры будут отправлены в циклы.

Эта система может быть представлена в виде графа с мостами в качестве узлов и соединениями между ними — в качестве ребер. Такой граф можно редуцировать до связующего дерева, которое по определению не имеет циклов, удалив из него соединения, изображенные на рис. 4.41 пунктирными линиями. В получившемся связующем дереве между каждыми двумя станциями существует только один путь. После того как мосты договорятся друг с другом о топологии связующего дерева, все коммуникации осуществляются только по ветвям этого дерева. Поскольку путь от отправителя к получателю единственный, зацикливание невозможно.

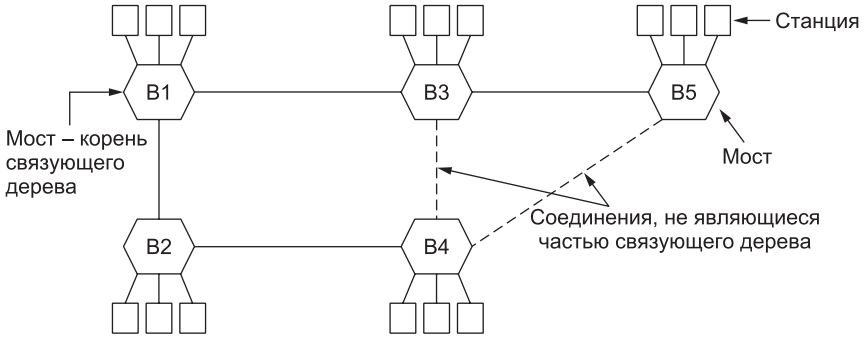


Рис. 4.41. Связующее дерево, соединяющее пять мостов. Пунктирными линиями показаны соединения, которые не входят в связующее дерево

Чтобы построить связующее дерево, мосты применяют следующий распределенный алгоритм. Каждый мост периодически рассылает по всем своим портам конфигурационное сообщение своим соседям и обрабатывает сообщения, полученные от других мостов, как описано ниже. Эти сообщения не передаются дальше, так как их цель — построение дерева, которое затем используется для пересылки.

Мосты должны сначала выбрать один мост, который будет корнем связующего дерева. Чтобы сделать этот выбор, каждый из них включает в конфигурационное сообщение идентификатор, основанный на своем MAC-адресе, так же как идентификатор моста, который он предполагает корнем. MAC-адреса установлены изготовителем и гарантировано уникальны во всем мире, что делает эти идентификаторы удобными и гарантированно разными. Мосты выбирают в качестве корня мост с наименьшим идентификатором. После обмена достаточным числом сообщений, чтобы распространить эту новость, все мосты договорятся, какой мост является корнем. На рис. 4.41 мост B1 имеет наименьший идентификатор и становится корнем.

Затем создается дерево кратчайших путей от корня до каждого моста. На рис. 4.41 мосты B2 и B3 могут быть достигнуты от моста B1 непосредственно, за один шаг, ко-

торый является кратчайшим путем. Мост B4 может быть достигнут за два шага, или через B2 или через B3. Чтобы разрубить этот узел, выбирается путь через мост с наименьшим идентификатором, таким образом, B4 будет достигнут через B2. Мост B5 может быть достигнут за два шага через B3.

Чтобы найти эти кратчайшие пути, мосты включают в конфигурационные сообщения расстояние от корня. Каждый мост помнит кратчайший путь, который он находит к корню. Затем мосты отключают порты, которые не являются частью кратчайшего пути.

Хотя дерево охватывает все мосты, но не все соединения (или даже мосты) обязательно присутствуют в дереве. Это происходит, потому что отключение портов ликвидирует некоторые соединения в сети, чтобы предотвратить появление циклов. Алгоритм построения дерева продолжает работать постоянно, обнаруживая изменения в топологии и обновляя структуру дерева.

Автором распределенного алгоритма построения связующего дерева является Радья Перлман (Radia Perlman). Ее задачей было решить проблему объединения локальных сетей без циклов. Ей была дана неделя на решение этой задачи, но она придумала идею алгоритма связующего дерева за один день, так что у нее осталось время изложить ее в виде стихотворения (Perlman, 1985).

I think that I shall never see
A graph more lovely than a tree.
A tree whose crucial property
Is loop-free connectivity.
A tree which must be sure to span.
So packets can reach every LAN.
First the Root must be selected
By ID it is elected.
Least cost paths from Root are traced
In the tree these paths are placed.
A mesh is made by folks like me
Then bridges find a spanning tree.

Алгоритм связующего дерева стандартизован как IEEE 802.1D и используется уже много лет. В 2001 году он был пересмотрен для более быстрого нахождения нового связующего дерева после изменения топологии. Для более подробного рассмотрения мостов см. Perlman (2000).

4.8.4. Повторители, концентраторы, мосты, коммутаторы, маршрутизаторы и шлюзы

Мы уже успели в нашей книге рассмотреть множество способов доставки кадров и пакетов из одного компьютера в другой. Мы упоминали повторители, концентраторы, мосты, маршрутизаторы и шлюзы. Все эти устройства используются очень широко, однако в чем-то они отличаются едва уловимо, а в чем-то весьма существенно. Число их весьма велико, поэтому лучше рассмотреть их все в совокупности, отмечая сходства и различия.

Чтобы понять, как работают эти устройства, надо осознать, что они работают на разных уровнях, как показано на рис. 4.42, *а*. Уровень имеет значение, поскольку от этого зависит, какую часть информации устройство использует для маршрутизации. Типичный сценарий таков: у пользователя появляются какие-то данные, которые необходимо отправить на удаленную машину. Они передаются на транспортный уровень, который добавляет к ним свой заголовок (например, заголовок TCP) и передает результирующую единицу информации на сетевой уровень. Тот, в свою очередь, тоже добавляет свой заголовок, в результате чего формируется *пакет* сетевого уровня (например, IP-пакет). На рис. 4.42, *б* IP-пакет выделен серым цветом. Пакет отправляется на канальный уровень, где обрастает еще одним заголовком и контрольной суммой (CRC). Наконец, формируется кадр, который спускается на физический уровень и может быть передан, например, по ЛВС.

Рис. 4.42. Соответствие устройств уровням (*а*); кадры, пакеты и заголовки (*б*)

Приступим к рассмотрению коммутирующих устройств и взглянем на то, как они соотносятся с пакетами и кадрами. На самом нижнем, физическом уровне работают повторители. Это аналоговые устройства, к которым подсоединяются концы двух сегментов кабеля. Сигнал, появляющийся на одном из них, очищается, усиливается повторителем и выдается на второй. Повторители не знают слов «пакет», «кадр» или «заголовок». Они знают символы, кодирующие биты в напряжение. В классическом Ethernet, например, допускается установка четырех повторителей, что усилит сигнал, чтобы увеличить максимальную длину кабеля с 500 до 2500 м.

Теперь обратимся к концентраторам. Концентратор имеет несколько входов, объединяемых электрически. Кадры, прибывающие на какой-либо вход, передаются на все остальные линии. Если одновременно по разным линиям придут два кадра, они столкнутся, как в коаксиальном кабеле. Все линии, подсоединяемые к нему, должны работать с одинаковыми скоростями. Концентраторы отличаются от повторителей тем, что они обычно не усиливают входные сигналы, поскольку предназначены не

для этого. Их задача — обеспечивать согласованную работу с несколькими входами, к которым подключаются линии с похожими параметрами. Впрочем, во всем остальном хабы не очень отличаются от повторителей. И те и другие являются устройствами физического уровня, они не анализируют адреса уровня каналов и не используют их.

Перейдем теперь на канальный уровень. Здесь мы обнаружим мосты и коммутаторы. Только что мы как раз более или менее подробно обсуждали мосты, поэтому знаем, что мост соединяет две или более ЛВС. Как и в концентраторах, в современных мостах имеются несколько портов, рассчитанных обычно на от 4 и до 48 входящих линий определенного типа. В отличие от концентратора каждый порт изолирован, чтобы быть собственным доменом коллизий; если у порта есть полнодуплексная двухточечная линия, в алгоритме CSMA/CD нет необходимости. Когда прибывает кадр, мост извлекает из заголовка и анализирует адрес назначения, сопоставляя его с таблицей и определяя, куда этот кадр должен быть передан. Для Ethernet этот адрес — 48-битный адрес назначения (рис. 4.14). Мост только выдает кадр в нужный порт и может передавать несколько кадров одновременно.

Мосты предлагают намного лучшую производительность, чем концентраторы, а изоляция между портами моста также означает, что входные линии могут работать на различных скоростях, возможно даже с различными типами сетей. Типичный пример — мост с портами, которые соединяются с 10-, 100- и 1000-Мбит/с Ethernet. Буферизация в мосте необходима, чтобы принять кадр на одном порту и передать его на другой порт. Если кадры приходят быстрее, чем они могут быть повторно переданы, мост может исчерпать буферное пространство и начать отказываться от кадров. Например, если Gigabit Ethernet заливает биты в 10-Мбит/с Ethernet на большой скорости, мост должен будет буферизовать их, надеясь не исчерпать память. Эта проблема существует, даже если все порты работают на одной и той же скорости, потому что в данный порт назначения кадры могут быть посланы из нескольких портов.

Мосты первоначально предназначались для того, чтобы соединять различные виды ЛВС, например Ethernet и Token Ring. Однако из-за различий между ЛВС это никогда не работало хорошо. Различные форматы кадра требуют копирования и переформатирования, которое занимает время центрального процессора, требует нового вычисления контрольной суммы и добавляет возможность необнаруженных ошибок из-за плохих битов в памяти моста. Еще одна серьезная проблема без хорошего решения — различные максимальные длины кадра. Нужно стремиться избавиться от слишком длинных кадров. В первую очередь — для совместимости.

Двумя другими областями, где ЛВС могут отличаться, является безопасность и качество обслуживания. У некоторых ЛВС есть шифрование канального уровня, например у 802.11, у некоторых, например Ethernet, его нет. У некоторых ЛВС есть возможности для обеспечения качества обслуживания, такие как приоритеты, например у 802.11, у некоторых, например Ethernet, их нет. Следовательно, когда кадр должен быть передан между этими ЛВС, безопасность или качество обслуживания, ожидаемое отправителем, возможно, не может быть обеспечено. По всем этим причинам современные мосты обычно работают с сетями одного типа, а для присоединения к сетям различных типов используются маршрутизаторы, к которым мы скоро перейдем.

Коммутаторы — это другое название современных мостов. Различия больше связаны с маркетингом, чем с техническими особенностями, но есть несколько важных

моментов. Мосты были разработаны, когда использовался классический Ethernet, таким образом, они имеют тенденцию присоединяться к относительно небольшому числу ЛВС, а значит, иметь относительно немного портов. Термин «коммутатор» более популярен в настоящее время. Кроме того, все современные системы используют двухточечные линии, такие как кабели витой пары, таким образом, отдельные компьютеры включаются непосредственно в коммутатор, и поэтому коммутатор будет стремиться иметь много портов. Наконец, «коммутатор» также используется в качестве общего термина. С мостом функциональность ясна. С другой стороны, «коммутатор» может относиться к коммутатору Ethernet или абсолютно другому виду устройства, которое принимает решения по перенаправлению, такому как телефонный коммутатор.

Итак, мы рассмотрели вкратце повторители и концентраторы, которые весьма сходны друг с другом, а также коммутаторы и мосты, которые еще более похожи. Теперь же мы перейдем к маршрутизаторам, которые резко отличаются от всего рассмотренного выше. Когда пакет прибывает на маршрутизатор, отрезаются заголовки и концевики кадров и остаются только поля данных (выделены серым на рис. 4.42), которые и передаются программному обеспечению маршрутизатора. Далее анализируется заголовок пакета и в соответствии с ним выбирается его дальнейший путь. Если это IP-пакет, то в заголовке будет содержаться 32-битный (IPv4) или 128-битный (IPv6), а не 48-битный IEEE 802 адрес. Программное обеспечение маршрутизатора не интересуется адресами кадров и даже не знает, откуда эти кадры взялись (то ли с ЛВС, то ли с двухточечной линии). Более подробно мы изучим маршрутизаторы и принципы маршрутизации в главе 5.

Поднявшись еще на уровень выше, мы обнаружим транспортные шлюзы. Они служат для соединения компьютеров, использующих различные транспортные протоколы, ориентированные на установление соединения. Например, такая ситуация возникает, когда компьютеру, использующему ориентированный на соединение протокол TCP, необходимо передать данные компьютеру, использующему другой ориентированный на соединение протокол SCTP. Транспортный шлюз может копировать пакеты между соединениями, одновременно приводя их к нужному формату.

Наконец, шлюзы приложений уже работают с форматами и содержимым пакетов, занимаясь переформатированием на более высоком уровне. Например, шлюз e-mail может переводить электронные письма в формат SMS-сообщений для мобильных телефонов. Как и «коммутатор», «шлюз» — своего рода общий термин. Он относится к процессу пересылки, который выполняется в верхнем уровне.

4.8.5. Виртуальные локальные сети

На заре развития технологий локальных сетей толстые желтые провода опутали огромное количество офисов. Можно было подключить к сети каждый компьютер, мимо которого шел такой провод. Никто не задумывался над тем, к какой ЛВС подключен конкретный компьютер. Все люди из соседних офисов были подключены к одной сети, без учета того, насколько им это было нужно. «География» управляла корпоративными организационными структурами.

С появлением в 1990-е годы витых пар и концентраторов все изменилось. Из офисных зданий стали исчезать эти желтые провода, напоминающие садовые шланги, и им

на смену пришли витые пары, которые шли к щитам, висящим по концам коридоров (или в центральных машинных залах) и напичканным проводами, как показано на рис. 4.43. Если чиновник, ответственный за прокладку кабелей в здании, был способен смотреть в будущее, то устанавливались витые пары категории 5; если же он был крохобором, то использовались существующие (телефонные) витые пары категории 3, которые были заменены только с приходом сетей типа Fast Ethernet.

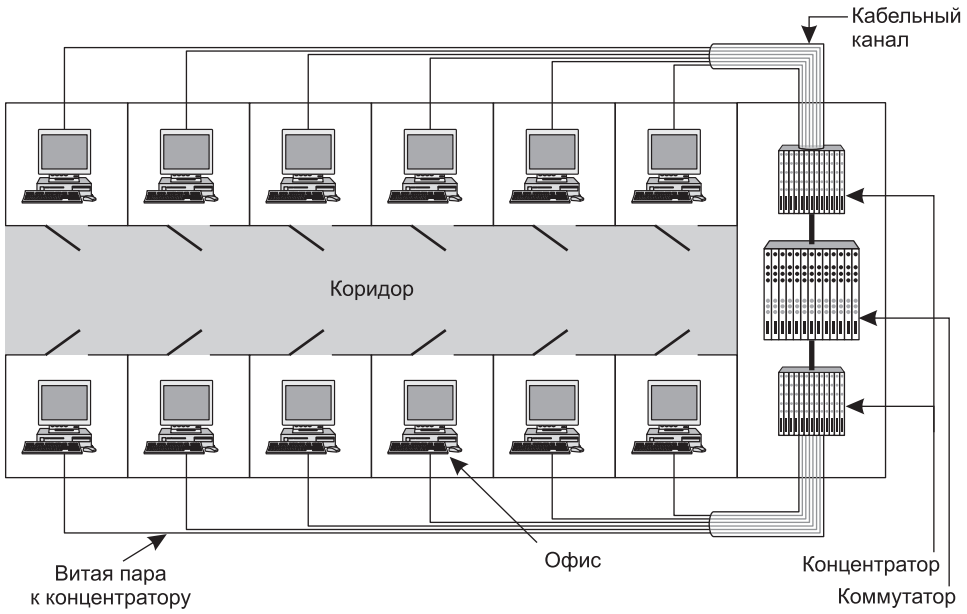


Рис. 4.43. Здание с централизованной проводкой с использованием концентратора и коммутатора

Сегодня кабели изменились, а концентраторы стали коммутаторами, но общая схема осталась прежней. Она позволяет настраивать локальные сети не физически, а логически. Если компании требовалось *k* ЛВС, она может приобрести *k* коммутаторов. Аккуратно собрав сеть (то есть вставив нужные соединители в нужные разъемы), можно было определить ее пользователей таким образом, чтобы это имело некий реальный организационный смысл и не зависело от расположения станций внутри здания.

А вообще, разве важно, кто подключен к какой ЛВС? Ведь все равно почти во всех организациях сети объединены между собой. На самом же деле, это действительно зачастую бывает важно. Сетевые администраторы по целому ряду причин любят группировать пользователей ЛВС в соответствии со структурой организации, а не по принципу физического расположения компьютеров. Одной из таких причин является безопасность. Одна ЛВС может содержать веб-серверы и другие компьютеры для общественного пользования. Другая может содержать записи отдела кадров, которые не должны выходить за пределы организации. В этой ситуации наилучшим выходом является объединение компьютеров всех работников отдела в одну ЛВС и запрет на

передачу каких-либо данных наружу. Руководство обычно не устраивает заявление о том, что такое решение невозможно реализовать.

Второй причиной является нагрузка на сеть. Сеть может оказаться настолько загруженной, что ее будет лучше разделить на несколько. Например, если народ из отдела исследований решит провести какой-нибудь хитрый эксперимент, то бухгалтерия будет не очень счастлива от сознания, что он проводится за счет емкостей их общей сети, где они проводят видеоконференцию. С другой стороны, это могло бы внушить руководству потребность установить более быструю сеть.

Еще одна причина — широковещание. Мосты поддерживают широковещание, когда местонахождение адресата неизвестно, и протоколы верхних уровней поддерживают широковещание. Например, если пользователь хочет отправить пакет на IP-адрес *x*, как ему узнать, какой MAC-адрес подставлять в кадры? Мы изучим этот вопрос более подробно в главе 5, а сейчас в двух словах обрисуем решение проблемы: данная станция должна широковещательным методом послать запрос: «Кто знает, какой MAC-адрес работает с IP-адресом *x*?» Затем она дожидается ответа. Поскольку количество компьютеров в ЛВС растет, растет и количество широковещательных сообщений. Каждая широковещательная передача потребляет больше ресурсов ЛВС, чем обычный кадр, потому что она направлена каждому компьютеру в ЛВС. Если размеры ЛВС сохраняются не больше чем необходимо, воздействие широковещательного трафика уменьшается.

С широковещанием связана еще одна проблема: время от времени сетевой интерфейс может сломаться или быть неправильно сконфигурирован, и начать генерировать бесконечный поток кадров, получаемый всеми станциями. Если сеть будет действительно неудачна, то некоторые из этих кадров вызовут ответы, которые приведут к еще большему количеству трафика. Это **широковещательный шторм (broadcast storm)**, который состоит в том, что, во-первых, вся пропускная способность сети занята этими бессмысленными кадрами, и, во-вторых, все машины объединенных сетей вынуждены заниматься исключительно обработкой и отвержением этого мусора.

На первый взгляд кажется, что широковещательные штормы можно ограничить путем установки своего рода дамб — разделяя сети мостами или коммутаторами на несколько частей. Однако если речь идет о прозрачности (то есть о предоставлении машинам возможности перенесения в другие ЛВС без каких-либо изменений конфигурации), то широковещательные кадры должны обрабатываться и пересылаться мостами.

Рассмотрев причины того, что компаниям требуются многочисленные локальные сети с ограниченными размерами, давайте вернемся к проблеме разделения логической и физической топологии. Создание физической топологии, чтобы отразить организационную структуру, может добавить работу и стоимость проекта, даже с централизованной проводкой и коммутаторами. Например, если два человека из одного отдела работают в различных зданиях, может быть легче присоединить их к различным коммутаторам, которые являются частью различных ЛВС. Даже если дело обстоит не так, пользователь мог перейти в пределах компании из одного отдела в другой, но остаться в том же офисе, или наоборот — сменить офис, не меняя отдел. Это могло бы привести к тому, что пользователь окажется в неправильной ЛВС, пока администратор не изменяет разъем пользователя от одного коммутатора на другой.

Кроме того, количество компьютеров, принадлежащих различным отделам, возможно, не очень хорошо соответствует количеству портов на коммутаторах; некоторые отделы могут быть слишком малы, а другие настолько крупные, что требуют нескольких коммутаторов. Это приводит к потраченным впустую портам коммутаторов, которые не используются.

Во многих компаниях организационные перестановки происходят постоянно, то есть сетевой администратор постоянно занимается манипуляциями с соединителями и разъемами. А в некоторых случаях такие перестановки оказываются невозможными из-за того, что провод пользовательского компьютера проходит слишком далеко и просто не дотягивается до нужного коммутатора (вследствие непродуманной прокладки кабелей) или доступные порты коммутатора относятся к другой ЛВС.

Пользователи стали требовать у производителей создания более гибких сетей, и те ответили им созданием **виртуальных ЛВС** (**ВЛВС** — **VLAN**, **Virtual LAN**), для внедрения которых необходимо было изменить только программное обеспечение. Виртуальные сети даже были в какой-то момент стандартизованы комитетом IEEE 802. Сейчас они широко применяются. Ниже мы вкратце обсудим виртуальные сети. Дополнительную информацию можно найти в работе (Seifert и Edwards, 2008).

Виртуальные сети построены на основе ВЛВС-совместимых коммутаторов. Для создания системы, построенной на виртуальных ЛВС, сетевому администратору прежде всего нужно решить, сколько всего будет виртуальных сетей, какие компьютеры будут в них входить и как они будут называться. Зачастую ВЛВС (неформально) называют в соответствии с цветами, поскольку тогда сразу становятся понятнее и нагляднее цветные диаграммы, показывающие принадлежность пользователей виртуальным сетям. Скажем, пользователи «красной» сети будут изображены красным цветом, «синей» — синим и т. д. Таким образом, на одном рисунке можно отобразить физическую и логическую структуру одновременно.

В качестве примера рассмотрим ЛВС с мостом, изображенные на рис. 4.44. Здесь девять машин входят в виртуальную сеть *С* («Серая»), а еще пять машин — в виртуальную сеть *Б* («Белая»). Машины «серой» сети распределены между двумя коммутаторами, в том числе две машины, подключенные к коммутатору через концентратор.

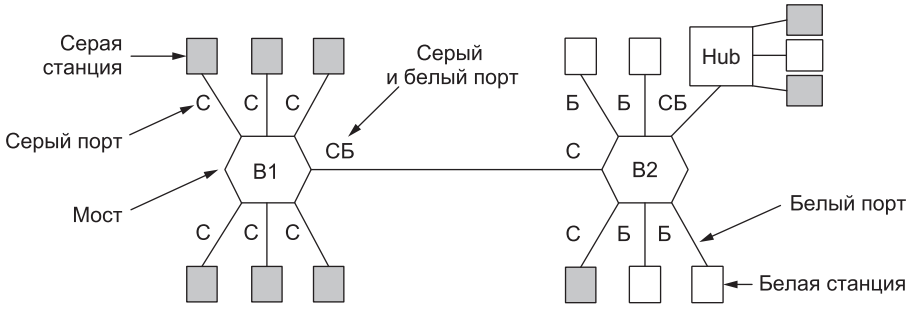


Рис. 4.44. Две виртуальные сети, серая и белая, в сети с мостом

Чтобы виртуальные сети функционировали корректно, необходимо наличие конфигурационных таблиц в мостах. Эти таблицы сообщают о том, через какие порты

производится доступ к тем или иным виртуальным сетям. Когда кадр прибывает, например, из «серой» ВЛВС, его нужно разослать на все порты, помеченные буквой С. Это правило справедливо как для ординарных (то есть однонаправленных) передач, для которых мост не изучает местоположение назначения, так и для групповых и широковещательных. Имейте в виду, что порт может быть помечен сразу несколькими цветами ВЛВС.

В качестве примера предположим, что одна из «серых» станций, подключенных к мосту В1, посылает кадр, на заранее не видимое место назначения. Мост В1 примет кадр и заметит, что он пришел с машины с пометкой С. Поэтому его необходимо залить на все порты (кроме того, с которого пришел кадр), принадлежащие «серой» виртуальной сети. Кадр будет направлен на остальные пять «серых» станций, подключенных к В1, а также по соединению В1 с мостом В2. Мост В2 аналогично перешлет кадр на все порты, помеченные С. При этом кадр будет отослан одной оставшейся станции и концентратору (который передаст кадр всем своим станциям). Концентратор имеет обе метки (С и Б), поскольку к нему подключены станции из обоих ВЛВС. Кадр не посылается через другие порты, кроме имеющих метку С, поскольку мост знает, что станции из «серой» ВЛВС через них недостижимы.

В нашем примере кадр будет передан только от моста В1 к мосту В2, так как машины «серой» ЛВС присоединены к В2. Глядя на «белую» ЛВС мы видим, что порт В2, который соединен с мостом В1, не помечен Б. Это означает, что кадр «белой» ЛВС не будет переслан от В2 к В1. Такое поведение корректно, поскольку к В1 не присоединено ни одной «белой» станции.

Стандарт IEEE 802.1Q

Чтобы реализовать эту схему, мосты должны каким-то образом узнавать, к какой ВЛВС принадлежит входящий кадр. Без этой информации, например, когда мост В2 получает кадр от моста В1 (см. рис. 4.44), он не знает, передавать его «белой» или «серой» ВЛВС. Если мы проектируем новый тип сети, вполне можно было бы просто добавить специальное поле заголовка. Но что делать с Ethernet — доминирующей сетью, у которой нет никаких «запасных» полей, которые можно было бы отдать под идентификатор виртуальной сети?

Комитет IEEE 802 озаботился этим вопросом в 1995 году. После долгих дискуссий было сделано невозможное — изменен формат заголовка кадра Ethernet! Новый формат было опубликован под именем 802.1Q в 1998 году. В заголовок кадра был вставлен флаг ВЛВС, который мы сейчас вкратце рассмотрим. Понятно, что внесение изменений в нечто уже устоявшееся, такое как Ethernet, должно быть произведено каким-то нетривиальным образом. Встают, например, следующие вопросы:

1. И что, теперь надо будет выбросить на помойку несколько миллионов уже существующих сетевых карт Ethernet?
2. Если нет, то кто будет заниматься генерированием новых полей кадров?
3. Что произойдет с кадрами, которые уже имеют максимальный размер?

Конечно, комитет 802 тоже был озабочен этими вопросами, и решение, несмотря ни на что, было найдено.

Идея состоит в том, что на самом деле поля ВЛВС реально используются только мостами да коммутаторами, а *не* машинами пользователей. Так, скажем, сеть, изображенную на рис. 4.44, не очень-то волнует их наличие в каналах, идущих от оконечных станций, до тех пор пока кадры не доходят до мостов. Кроме того, чтобы использовать ВЛВС, мосты должны быть ВЛВС-совместимыми. Этот факт делает проект выполнимым.

Что касается старых сетевых карт Ethernet, то выкидывать их не приходится. Комитет 802.3 никак не мог заставить людей изменить поле *Тип* на поле *Длина*. Вы можете себе представить, какова была бы реакция на заявление о том, что все существующие карты Ethernet можно выбросить? Тем не менее новые модели 802.1Q-совместимы и могут корректно заполнять поля идентификации виртуальных сетей.

Так как некоторые компьютеры (и коммутаторы) могут быть не совместимы с ВЛВС, то первый встретившийся на пути кадра ВЛВС-совместимый мост вставляет это поле, а последний — вырезает его. Пример смешанной топологии показан на рис. 4.45. В этом примере ВЛВС-совместимые компьютеры генерируют снабженные тегами (то есть 802.1Q) кадры непосредственно, и дальнейшие коммутаторы используют эти теги. Закрашенные значки — ВЛВС-совместимые, пустые — нет.

В соответствии с 802.1Q, кадры «окрашиваются» в зависимости от порта, на котором они получены. Для применения этого метода все машины одного порта должны принадлежать одной ВЛВС, что уменьшает гибкость. Например, на рис. 4.44 это свойство выполнено для всех портов, где отдельный компьютер соединяется с мостом, но не для порта, где концентратор соединяется с мостом B2.

Дополнительно, мост может использовать протокол более высокого уровня, чтобы выбрать цвет. Таким образом, кадры, прибывающие в порт, могли бы быть помещены в различные ВЛВС в зависимости от того, переносят ли они IP-пакеты или кадры PPP.

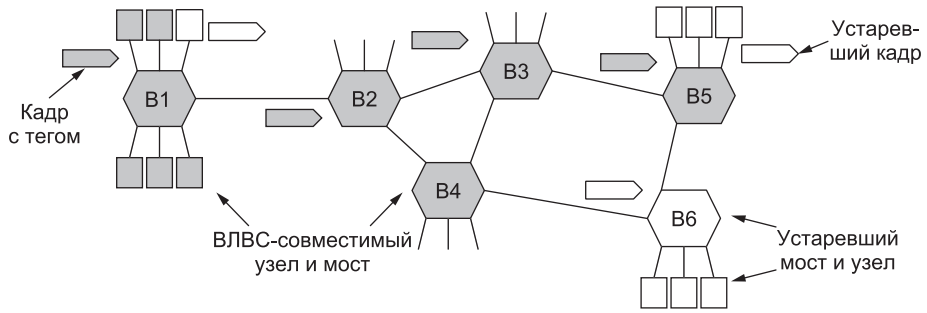


Рис. 4.45. ЛВС с мостами, частично совместимая с ВЛВС. Затененные символы — это ВЛВС-совместимые устройства. Все остальные не совместимы с виртуальными сетями

Возможны и другие методы, но они не поддерживаются в 802.1Q. Например, для выбора цвета ВЛВС может использоваться MAC-адрес. Это могло бы быть полезно для кадров из соседних 802.11 ЛВС, в которых ноутбуки посылают кадры через различные порты, в зависимости от своего перемещения. Один MAC-адрес был бы тогда отображен на одну и ту же ВЛВС, независимо от порта, через который он входит в сеть.

Что касается проблемы кадров, длина которых превышает 1518 байт, то в стандарте 802.1Q она решается путем повышения лимита до 1522 байт. К счастью, только ВЛВС-совместимые компьютеры и коммутаторы обязаны поддерживать такие длинные кадры.

Теперь рассмотрим собственно формат 802.1Q. Он показан на рис. 4.46. Единственное изменение, которое мы тут видим, — это добавление пары 2-байтовых полей. Первое называется **Идентификатором протокола ВЛВС (VLAN protocol ID)**. Оно всегда имеет значение 0x8100. Поскольку это число превышает 1500, то все сетевые карты Ethernet интерпретируют его как «тип», а не как «длину». Неизвестно, что будет делать с таким кадром карта, несовместимая с 802.1Q, поэтому такие кадры, по идее, не должны к ней никоим образом попадать.

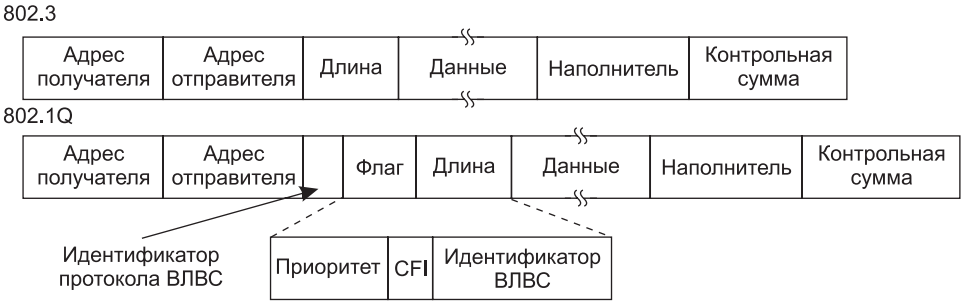


Рис. 4.46. Форматы кадров Ethernet-стандартов 802.3 и 802.1Q

Во втором двухбайтовом поле есть три вложенных поля. Главным из них является **Идентификатор ВЛВС (VLAN identifier)**, который занимает 12 младших бит. Он содержит ту информацию, из-за которой все эти преобразования форматов, собственно, и были затеяны: в нем указан «цвет» виртуальной сети, которой принадлежит кадр. Трехбитовое поле *Приоритет* не имеет вообще ничего общего с виртуальными сетями. Просто изменение формата Ethernet-кадра — это такой ежедекадный ритуал, который занимает три года и исполняется какой-то сотней людей. Почему бы не оставить память о себе в виде трех дополнительных бит, да еще и с таким привлекательным назначением? Поле *Приоритет* позволяет различать трафик «жесткого» реального времени, трафик «мягкого» реального времени и трафик, для которого время передачи не критично. Это позволяет обеспечить более высокое качество обслуживания в Ethernet. Оно используется также при передаче голоса по Ethernet (хотя вот уже четверть века в IP имеется подобное поле и никто никогда его не использовал).

Последнее поле, **CFI (Canonical Format Indicator — индикатор классического формата)**, следовало бы назвать **Индикатором эгоизма компании (CEI — Corporate ego indicator)**. Изначально оно предназначалось для того, чтобы показывать порядок бит в MAC-адресе («остроконечники против тупоконечников»), однако в пылу дискуссий об этом как-то забыли. Его присутствие сейчас означает, что поле данных содержит усохший кадр 802.5, который ищет еще одну сеть формата 802.5 и в Ethernet попал совершенно случайно. То есть, на самом деле, он просто использует Ethernet в качестве средства передвижения. Все это, конечно, практически никак не связано с обсуждаемыми в данном разделе виртуальными сетями. Но политика комитета

стандартизации не сильно отличается от обычной политики: если ты проголосуешь за введение в формат моего бита, то я проголосую за твой бит.

Как уже упоминалось выше, когда кадр с флагом виртуальной сети приходит на ВЛВС-совместимый коммутатор, последний использует идентификатор виртуальной сети в качестве индекса таблицы, в которой он ищет, на какие порты послать кадр. Но откуда берется эта таблица? Если она разрабатывается вручную, это означает возврат в исходную точку: ручное конфигурирование коммутаторов. Вся прелесть прозрачности мостов состоит в том, что они настраиваются автоматически и не требуют для этого никакого вмешательства извне. Было бы очень стыдно потерять это свойство. К счастью, мосты для виртуальных сетей также являются самонастраивающимися. Настройка производится на основе информации, содержащейся во флагах приходящих кадров. Если кадр, помеченный как ВЛВС 4, приходит на порт 3, значит, несомненно, одна из машин, подключенных к этому порту, находится в виртуальной сети 4. Стандарт 802.1Q вполне четко поясняет, как строятся динамические таблицы. При этом делаются ссылки на соответствующие части стандарта 802.1D.

Прежде чем завершить разговор о маршрутизации в виртуальных сетях, необходимо сделать еще одно замечание. Многие пользователи сетей Интернет и Ethernet фанатично привязаны к сетям без установления соединения и неистово противопоставляют их любым системам, в которых есть хотя бы намек на соединение на сетевом или канальном уровне. Однако в виртуальных сетях один технический момент как раз-таки очень сильно напоминает установку соединения. Речь идет о том, что работа виртуальной сети невозможна без того, чтобы в каждом кадре был идентификатор, использующийся в качестве индекса таблицы, встроенной в коммутатор. По этой таблице определяется дальнейший вполне определенный маршрут кадра. Именно это и происходит в сетях, ориентированных на соединение. В системах без установления соединения маршрут определяется по адресу назначения, и там отсутствуют какие-либо идентификаторы конкретных линий, через которые должен пройти кадр. Более подробно мы рассмотрим эту тенденцию в главе 5.

4.9. Резюме

В некоторых сетях для любой связи используется единственный моноканал. При их разработке основной проблемой является распределение этого канала между соревнующимися за право его использования станциями. Простейшими схемами распределения являются частотное и временное уплотнение. Они эффективны при небольшом количестве станций и постоянном трафике. Оба метода широко применяются в этих условиях, например, для разделения полосы пропускания в телефонных магистралях.

Когда количество станций велико и непостоянно или трафик является пульсирующим — типичный случай для компьютерных сетей, — частотное и временное уплотнение использовать нецелесообразно.

Было разработано несколько алгоритмов динамического распределения каналов. Протокол ALOHA (чистый или дискретный) используется в многочисленных вариантах в реальных системах, например в кабельных модемах и RFID. Его усовершен-

ствование — возможность прослушивать состояние канала, тогда станции могут отказываться от передачи, когда слышат, что канал занят другой станцией. Применение этого метода — контроля несущей — повлекло создание различных CSMA протоколов, применяемых в локальных и региональных сетях. Это основа классических сетей Ethernet и сетей 802.11.

Широко известен класс протоколов, полностью устраняющий борьбу за канал или, по меньшей мере, значительно снижающий ее напряженность. Протокол битовой карты, протокол с двоичным обратным отсчетом и такие топологии, как кольца, полностью устраняют состязание за канал. Протокол адаптивного дерева снижает его остроту, динамически деля станции на две непересекающиеся группы различного размера, и позволяет состязание только внутри группы; в идеале группа выбирается так, чтобы только одной станции, готовой к передаче, разрешалось это сделать.

Дополнительной проблемой беспроводных ЛВС является трудность обнаружения коллизий передач и возможность различий зон обслуживания станций. В доминирующей беспроводной ЛВС, IEEE 802.11, станции, чтобы смягчить первую проблему, используют CSMA/CA, оставляя маленькие промежутки, чтобы избежать коллизий. Станции могут также использовать протокол RTS/CTS, чтобы сражаться со скрытыми терминалами, которые возникают из-за второй проблемы. IEEE 802.11 обычно используется, чтобы соединить ноутбуки и другие устройства с точками доступа, но он может использоваться и между устройствами. Может использоваться любой из нескольких физических уровней, в том числе многоканальный FDM с и без нескольких антенн, и расширение спектра.

Как и 802.11, считыватели и метки RFID используют протокол произвольного доступа, чтобы сообщить идентификаторы. Другие беспроводные персональные и городские сети имеют отличающееся устройство. Система Bluetooth осуществляет беспроводное присоединение к компьютеру наушников и других видов периферийных устройств. IEEE 802.16 обеспечивает широкую область беспроводного доступа к Интернету для неподвижных и мобильных компьютеров. Обе эти сети используют централизованную схему на основе соединений, где ведущее устройство Bluetooth и базовая станция WiMAX решают, когда каждая станция может послать или получить данные. Для 802.16 эта схема поддерживает различное качество обслуживания для трафика в реальном времени, такого как телефонные звонки, и интерактивного трафика, такого как просмотр веб-страниц. Сложность ведущего устройства Bluetooth приводит к дешевизне ведомых устройств.

Ethernet является доминирующей технологией проводных локальных вычислительных сетей. Классический Ethernet производит распределение канала при помощи метода CSMA/CD с желтым кабелем толщиной с садовый шланг, который тянулся от машины к машине. Архитектура изменилась, скорости возросли от 10 Мбит/с до 1 Гбит/с и продолжают расти. Теперь линии точка-точка, такие как витая пара, присоединяются к концентраторам и коммутаторам. С современными коммутаторами и полнодуплексными каналами нет никакой конкуренции в каналах, и коммутарор может пересылать кадры между различными портами параллельно.

Когда в здании много локальных сетей, необходим какой-то способ их объединения. Для этих целей используются plug-and-play устройства — мосты. При построении мостов применяются алгоритм обратного обучения и алгоритм связующего дерева.

С добавлением в современные коммутаторы этих функций термины «мост» и «коммутатор» стали взаимозаменяемы.

Для упрощения управления ЛВС с мостами появились виртуальные локальные сети, которые позволили отделить физическую топологию от логической. Был разработан стандарт для виртуальных локальных сетей — IEEE 802.1Q, вводящий новый формат Ethernet кадров.

Вопросы

1. Для решения задачи используйте формулу, приведенную в данной главе, записав ее в общем виде. Кадры для передачи прибывают случайным образом на 100-Мбит/с канал. Если в момент прибытия канал оказывается занят, кадр ставится в очередь ожидания. Длина кадра распределяется по экспоненциальному закону с математическим ожиданием, равным 10 000 бит/кадр. Для каждой из приведенных ниже скоростей прибытия кадров вычислите задержку (включая время ожидания в очереди и время передачи) кадра средней длины.

   1) 90 кадров/с.

   2) 900 кадров/с.

   3) 9000 кадров/с.

2. Группа из N станций совместно использует канал чистой системы ALOHA, работающий со скоростью 56 Кбит/с. Каждая станция передает 1000-битный кадр в среднем каждые 100 с, даже если предыдущий кадр еще не был передан (например, станции могут буферизировать исходящие кадры). Каково максимальное значение N?

3. Сравните время задержки чистой и дискретной систем ALOHA при низкой нагрузке. У какой из систем это время будет меньшим? Поясните свой ответ.

4. Большая группа пользователей системы ALOHA формирует 50 запросов в секунду, включая первичные и повторные передачи. Время разделено на интервалы по 40 мс.

   1) Каковы шансы успеха с первой попытки?

   2) Какова вероятность того, что перед успехом произойдет ровно k столкновений?

   3) Чему равно среднее число попыток передачи?

5. В дискретной системе ALOHA с бесконечным числом пользователей средний период ожидания станции между столкновением и повторной попыткой составляет 4 временных интервала. Нарисуйте зависимость задержки от потока в канале для данной системы.

6. Какова длина слота разрешения спора в CSMA/CD для:

   1) 2-километрового двухпроводного кабеля (скорость распространения сигнала составляет 82 % скорости распространения сигнала в вакууме)?

   2) 40-километрового многомодового оптоволоконного кабеля (скорость распространения сигнала составляет 65 % скорости распространения сигнала в вакууме)?

7. Сколько времени в худшем случае придется ожидать начала передачи станции s, если в локальной сети применяется базовый протокол битовой карты?

8. В протоколе двоичного отсчета объясните, как станция с более низким номером может лишаться возможности отправки пакета

9. Шестнадцать станций, пронумерованных от 1 до 16, соревнуются за право использования общего канала, используя протокол движения по адаптивному дереву. Сколько интервалов времени потребуется для разрешения спора, если все станции, чьи номера являются простыми числами, одновременно станут готовыми к передаче?

10. Рассмотрим пять беспроводных станций, *A*, *B*, *C*, *D* и *E*. Станция *A* может общаться со всеми остальными станциями. *B* может общаться с *A*, *C* и *E*. *C* может общаться с *A*, *B* и *D*. *D* может общаться с *A*, *C* и *E*. *E* может общаться с *A*, *D* и *B*.

    1) Когда *A* посылает в *B*, какие другие коммуникации возможны?

    2) Когда *B* посылает в *A*, какие другие коммуникации возможны?

    3) Когда *B* посылает в *C*, какие другие коммуникации возможны?

11. Шесть станций, отмеченных буквами *A*—*F*, взаимодействуют друг с другом по протоколу MACA. Возможна ли ситуация двух одновременных передач данных? Ответ поясните.

12. В семиэтажном офисном здании на каждом этаже расположено по 15 офисов. В каждом офисе на стене установлен разъем для подключения терминала, так что в вертикальной плоскости эти разъемы образуют прямоугольную сетку с расстоянием по 4 м между гнездами, как по горизонтали, так и по вертикали. Предполагая, что можно проложить кабель по прямой между любой парой гнезд, по горизонтали, вертикали или диагонали, сосчитайте, сколько метров кабеля потребуется для соединения всех гнезд при помощи:

    1) конфигурации «звезда» с одним маршрутизатором посередине;

    2) классической сети 802.3.

13. Чему равна скорость в бодах стандартной локальной сети Ethernet со скоростью 10 Мбит/с?

14. Как будет выглядеть манчестерский код для классической сети Ethernet следующей двоичной последовательности: 0001110101?

15. В сети с протоколом CSMA/CD (не 802.3) длиной 1 км со скоростью передачи данных 10 Мбит/с скорость распространения сигнала составляет 200 м/мкс. Повторителей в этой системе нет. Длина кадров данных равна 256 бит, включая 32 бита заголовка, контрольную сумму и другие накладные расходы. Первый интервал времени после успешной передачи кадра резервируется для передачи получателем 32-битового кадра с подтверждением. Какова эффективная скорость передачи данных без учета накладных расходов при предположении о том, что столкновений нет?

16. Две станции в сети с протоколом CSMA/CD пытаются передавать длинные (состоящие из нескольких кадров) файлы. После передачи каждого кадра они состязаются за канал при помощи алгоритма двоичной экспоненциальной выдержки. Какова вероятность того, что борьба закончится в *k*-м раунде, и чему равно среднее число раундов в периоде состязания?

17. IP-пакет необходимо передать по сети Ethernet. Длина пакета — 60 байт, включая все служебные поля. Если не используется LLC, необходимо ли заполнение Ethernet-кадра? Если да, то сколькими байтами?

18. Кадры Ethernet должны быть не короче 64 байт для того, чтобы в случае коллизии на дальнем конце провода передатчик все еще передавал тот же самый кадр. В сетях типа Fast Ethernet минимальный размер кадра также равен 64 байтам, однако биты могут выдаваться в десять раз чаще, чем в классическом варианте Ethernet. Каким образом в системе удалось сохранить прежний минимальный размер кадра?

19. В некоторых изданиях можно прочесть, что максимальный размер кадра Ethernet равен 1522 байтам (а не 1500 байтам). Ошибаются ли авторы этих изданий? Ответ поясните.

20. Сколько кадров в секунду может обрабатывать Gigabit Ethernet? Хорошо подумайте перед тем, как отвечать. Подсказка: имеет значение тот факт, что это именно Gigabit Ethernet.

21. Назовите две сетевые технологии, позволяющие паковать кадры друг за другом. Чем выгодно такое свойство?

22. На рис. 4.24 показаны четыре станции: *A*, *B*, *C* и *D*. Как вы думаете, какая из двух последних станций находится ближе к *A* и почему вы так решили?

23. Приведите пример, показывающий, что в протоколе 802.11 RTS/CTS немного отличается от RTS/CTS в протоколе MACA.
24. В беспроводной ЛВС с одной точкой доступа есть 10 клиентских станций. У четырех станций скорость передачи данных составляет 6 Мбит/с, у других четырех — 18 Мбит/с и у еще двух — 54 Мбит/с. Какова скорость передачи данных каждой станции, когда все десять станций посылают данные одновременно и

    1) TXOP не используется?

    2) TXOP используется?
25. Пусть по 11 Мбит/с локальной сети 802.11b передаются друг за другом по радиоканалу 64-байтные кадры с вероятностью ошибки 10^{-7} на бит. Сколько кадров в секунду в среднем будет искажаться при передаче?
26. Сеть стандарта 802.16 обладает каналом с шириной полосы пропускания 20 МГц. Сколько бит в секунду может отправлять по ней абонентская станция?
27. Назовите две причины, по которым в сетях может быть предпочтительнее использовать исправление ошибок вместо обнаружения ошибок и повторной передачи.
28. Назовите два сходства и два отличия WiMAX от 802.11.
29. На рис. 4.31 видно, что устройство системы Bluetooth может находиться одновременно в двух пикосетях. Почему одно и то же устройство не может являться главным сразу в двух пикосетях?
30. Каков максимальный размер поля данных для кадра Bluetooth с тремя слотами на базовой скорости? Объясните свой ответ.
31. На рис. 4.21 показано несколько протоколов физического уровня. Какой из них больше всего напоминает протокол физического уровня Bluetooth? В чем состоит основная разница этих двух протоколов?
32. Как упомянуто в разделе 4.6.6, эффективность кадра с одним слотом с кодированием повторения составляет приблизительно 13 % на базовой скорости данных. Какова будет эффективность, если вместо этого будет использоваться 5-слотовый кадр?
33. Кадры-«маяки» в варианте 802.11 с методом частотных скачков содержат время пребывания. Как вы думаете, аналогичные кадры-«маяки» в Bluetooth также содержат время пребывания? Обсудите свой ответ.
34. Предположим, что вокруг RFID-считывателя есть 10 меток. Каково лучшее значение Q? Насколько вероятно, что одна метка отвечает в данном слоте без коллизий?
35. Перечислите некоторые из проблем безопасности системы RFID.
36. Коммутатор, предназначенный для работы с Fast Ethernet, имеет объединительную плату, которая может передавать данные со скоростью 10 Гбит/с. Сколько кадров в секунду может быть обработано таким коммутатором?
37. Опишите вкратце разницу между коммутаторами с ожиданием (store-and-forward) и сквозными (cut-through) коммутаторами.
38. Рассмотрите расширенную ЛВС, содержащую мосты B1 и B2, на рис. 4.38, *б*. Предположите, что хэш-таблицы в двух мостах пусты. Перечислите все порты, в которые будет отправлен пакет для следующей последовательности передач данных:

    1) *A* посылает пакет *C*.

    2) *E* посылает пакет *F*.

    3) *F* посылает пакет *E*.

    4) *Г* посылает пакет *E*.

5) *D* посылает пакет *A*.

6) *B* посылает пакет *F*.

39. Коммутаторы с ожиданием имеют преимущество перед сквозными коммутаторами при обработке испорченных кадров. Объясните, почему.

40. Как упомянуто в разделе 4.8.3, некоторые мосты могут не присутствовать в связующем дереве. Обрисуйте в общих чертах ситуацию, когда мост может не присутствовать в связующем дереве.

41. Чтобы виртуальная сеть заработала, мостам и коммутаторам нужны конфигурационные таблицы. А что если в виртуальных сетях, показанных на рис. 4.44, использовать концентраторы вместо коммутаторов? Понадобятся ли им конфигурационные таблицы? Ответ поясните.

42. На рис. 4.45 коммутатор обычного оконечного домена (справа) является ВЛВС-совместимым. Можно было бы поставить здесь обычный коммутатор? Ответ поясните.

43. Напишите программу, симулирующую поведение протокола CSMA/CD в системе Ethernet с *N* станциями, готовыми к передаче во время передачи по каналу кадра. Программа должна выводить временные метки тех моментов, когда каждая из станций смогла успешно начать передачу своего кадра. Пусть часы изменяют свое состояние каждый такт (51,2 мкс), и обнаружение коллизии с отправкой преднамеренной помехи, сообщающей об этом, также занимает один такт. Все кадры имеют максимально допустимую длину.

Глава 5
Сетевой уровень

Сетевой уровень занимается разработкой маршрутов доставки пакетов от отправителя до получателя. Чтобы добраться до пункта назначения, пакету может потребоваться преодолеть несколько транзитных участков между маршрутизаторами. Функции, выполняемые на сетевом уровне, резко контрастируют с деятельностью канального уровня, цель которого была более скромной — просто переместить кадры с одного конца провода на другой. Таким образом, сетевой уровень оказывается самым нижним уровнем, который имеет дело с передачей данных по всему пути от одного конца до другого.

Для достижения этих целей сетевой уровень должен обладать информацией о топологии сети (то есть о множестве всех маршрутизаторов и связей) и выбирать нужный путь по этой сети, даже если она достаточно крупная. При выборе маршрутизаторов он должен также заботиться о том, чтобы нагрузка на маршрутизаторы и линии связи была, по возможности, более равномерной. Наконец, если источник и приемник находятся в различных сетях, именно сетевой уровень должен уметь решать проблемы, связанные с различиями в сетях. В данной главе мы рассмотрим все эти аспекты и проиллюстрируем их в основном на примере Интернета и его протокола сетевого уровня — IP.

5.1. Вопросы проектирования сетевого уровня

В следующих разделах мы рассмотрим некоторые вопросы, с которыми приходится сталкиваться разработчикам сетевого уровня. К этим вопросам относятся сервисы, предоставляемые транспортному уровню, и внутреннее устройство сети.

5.1.1. Метод коммутации пакетов с ожиданием

Прежде чем начать подробное рассмотрение сетевого уровня, необходимо восстановить в памяти окружение, в котором ему приходится функционировать. Оно показано на рис. 5.1. Основными компонентами сети являются устройства интернет-провайдера (маршрутизаторы, соединенные линиями связи), показанные внутри затененного овала, а также устройства, принадлежащие клиенту и показанные вне овала. Хост H1 напрямую соединен с одним из маршрутизаторов интернет-провайдера *A*

(как, например, домашний компьютер, подключенный к DSL-модему). Хост H2, напротив, находится в ЛВС (например, офисной сети Ethernet) с маршрутизатором *F*, принадлежащим клиенту, который с ним работает. Этот маршрутизатор связывается с интернет-провайдером по выделенной линии. Мы показали *F* вне овала, потому что он не принадлежит интернет-провайдеру. Однако в контексте данной главы мы будем считать маршрутизаторы клиента частью сети интернет-провайдера, поскольку в них применяются те же самые алгоритмы, что и в маршрутизаторах интернет-провайдеров (а основным предметом рассмотрения будут именно алгоритмы).

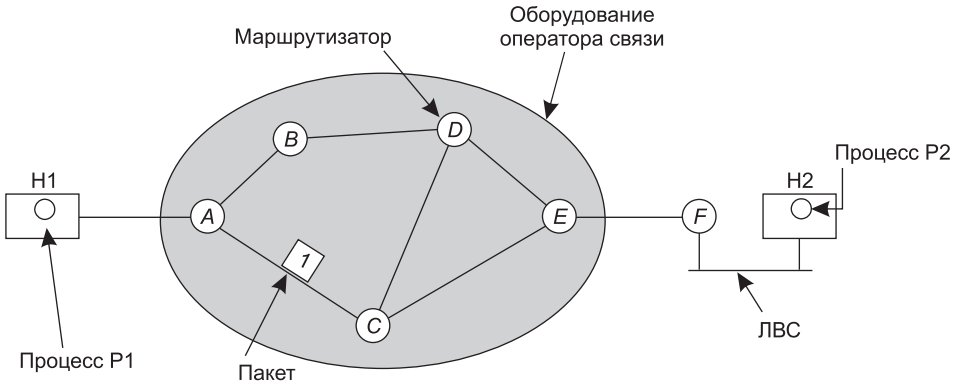


Рис. 5.1. Окружение, в котором функционируют протоколы сетевого уровня

Система работает следующим образом. Хост, у которого есть пакет для передачи, посылает его на ближайший маршрутизатор либо в своей ЛВС, либо по двухточечному соединению интернет-провайдеру. Там пакет хранится до тех пор, пока не будет принят целиком и не пройдет полную обработку, включая верификацию контрольной суммы. Затем он передается по цепочке маршрутизаторов, которая в итоге приводит к пункту назначения. Такой механизм называется коммутацией пакетов с ожиданием (store-and-forward), и мы уже рассматривали его в предыдущих главах.

5.1.2. Сервисы, предоставляемые транспортному уровню

Сетевой уровень предоставляет транспортному уровню сервисы через интерфейс между сетевым и транспортным уровнями. Важным вопросом является именно то, какой именно вид сервиса сетевой уровень предоставляет транспортному. Разработка таких сервисов требует особой аккуратности и при этом необходимо учитывать следующее:

- Сервисы сетевого уровня не должны зависеть от технологии маршрутизатора.
- Транспортный уровень должен быть независим от количества, типа и топологии присутствующих подсетей с маршрутизаторами.
- Сетевые адреса, доступные транспортному уровню, должны использовать единую систему нумерации, даже между локальными и глобальными сетями..

Находясь в рамках поставленной перед ними задачи, разработчики оказываются абсолютно свободными в написании детальной спецификации сервисов, которые должны предоставляться транспортному уровню. Эта свобода часто вырождается в яростную борьбу между двумя непримиримыми группировками. В центре дискуссии оказывается вопрос о том, какие сервисы должен предоставлять сетевой уровень — ориентированные на соединение или не требующие соединений.

Один лагерь (представленный интернет-сообществом) заявляет, что работа маршрутизаторов заключается исключительно в перемещении с места на место пакетов, и больше ни в чем. С этой точки зрения (основанной на примерно сорокалетнем опыте работы с реальными компьютерными сетями), сеть обладает врожденной ненадежностью, вне зависимости от того, как она спроектирована. Хосты должны учитывать это и защищаться от ошибок своими силами (то есть заниматься обнаружением и исправлением ошибок), а также самостоятельно управлять потоком.

Из этого следует, что сетевой сервис должен быть сервисом, не требующим установки соединения и состоящим в основном из примитивов SEND PACKET (послать пакет) и RECEIVE PACKET (принять пакет). В частности, сюда нельзя включать упорядочивание пакетов и контроль потока — все равно эти действия будет выполнять хост. От того что одна и та же работа будет выполнена дважды, качество обслуживания не повысится. Такое рассуждение — пример применения **«сквозного» принципа (end-to-end argument)**, оказавшего значительное влияние на формирование Интернета (Saltzer и др., 1984). Кроме того, каждый пакет должен содержать полный адрес получателя, так как пересылка производится независимо от предшествующих пакетов.

Другой лагерь, представленный телефонными компаниями, утверждает, что сеть должна предоставлять надежный, ориентированный на соединение сервис. Они утверждают, что 100 лет успешного управления телефонными системами по всему миру — это серьезный аргумент в их пользу. По их мнению, качество обслуживания является определяющим фактором, и без установления соединения в сети очень сложно добиться каких-либо приемлемых результатов, особенно когда дело касается трафика реального масштаба времени — например, передачи голоса и видео.

Этот спор остается актуальным даже по прошествии нескольких десятков лет. Раньше самые распространенные сети передачи данных (такие как X.25, использовавшаяся в 70-х, и популярная в 80-х Frame Relay) были ориентированы на соединение. Однако после появления ARPANET и на ранних этапах развития Интернета сетевые уровни без установления соединения стали чрезвычайно популярными. Сейчас протокол IP является вездесущим символом успеха. На его популярность не повлияло даже появление ориентированной на соединение технологии ATM, созданной в 80-х годах с целью заменить IP: в настоящее время ATM используется только в отдельных случаях, тогда как в ведении IP оказываются все телефонные сети. Однако с ростом требований к качеству обслуживания развитие Интернета предполагает появление новых возможностей, ориентированных на соединение. В качестве примеров таких технологий можно привести MPLS (MultiProtocol Label Switching, «мультипротокольная коммутация по меткам»), о которой мы поговорим в этой главе, и VLAN, которую мы рассматривали в главе 4. Обе эти технологии сейчас широко используются.

5.1.3. Реализация сервиса без установления соединения

Рассмотрев два класса сервисов, которые сетевой уровень может предоставлять своим пользователям, можно перейти к обсуждению устройства этого уровня. Возможны два варианта в зависимости от типа сервиса. Если предоставляется сервис без установления соединения, пакеты внедряются в сеть по отдельности, и их маршруты рассчитываются независимо. При этом никакой предварительной настройки не требуется. В этом случае пакеты часто называют **дейтаграммами (datagrams)**, по аналогии с телеграммами, а сети соответственно **дейтаграммными (datagram network)**. При использовании сервиса, ориентированного на соединение, весь путь от маршрутизатора-отправителя до маршрутизатора-получателя должен быть установлен до начала передачи каких-либо пакетов данных. Такое соединение называется **виртуальным каналом (VC, Virtual Circuit)**, по аналогии с физическими каналами, устанавливаемыми в телефонной системе. Сеть при этом называется **сетью виртуального канала (virtual-circuit network)**. В этом разделе мы обсудим дейтаграммные сети; в следующем разделе — сети виртуального канала.

Рассмотрим принцип работы дейтаграммных сетей. Пусть процесс P1 (рис. 5.2) хочет послать длинное сообщение для P2. Он передает свое послание транспортному уровню, сообщает ему о том, что доставить данные необходимо процессу P2, выполняющемуся на хосте H2. Код транспортного уровня исполняется на хосте H1; более того, обычно он является частью операционной системы. Заголовок транспортного уровня вставляется в начало сообщения, и в таком виде оно передается на сетевой уровень. Обычно это просто еще одна процедура операционной системы.

Предположим, что в нашем примере сообщение в четыре раза длиннее максимального размера пакета, поэтому сетевой уровень должен разбить его на четыре пакета (1, 2, 3 и 4) и послать их все поочередно на маршрутизатор *A* с использованием какого-нибудь протокола двухточечного соединения, например PPP. Здесь вступает в игру интернет-провайдер. Каждый маршрутизатор имеет свою внутреннюю таблицу, по которой он определяет дальнейший путь пакета при каждом из возможных адресов назначения. Каждая запись таблицы состоит из двух полей: пункт назначения (адресат) и выходящая линия для данного адресата. Во втором поле могут использоваться только линии, непосредственно соединенные с данным маршрутизатором. Так, например, на рис. 5.2 у маршрутизатора *A* имеются только две исходящие линии — ведущие к *B* и к *C*, — поэтому все входящие пакеты должны пересылаться на какой-то из этих двух маршрутизаторов, даже если они не являются адресатами. Изначальная таблица маршрутизации *A* показана на рисунке под соответствующей надписью.

В маршрутизаторе *A* пакеты 1, 2 и 3, поступившие на вход, кратковременно сохраняются для верификации контрольной суммы. Затем в соответствии с таблицей *A* каждый пакет пересылается по исходящему соединению на маршрутизатор *C* с использованием нового кадра. После этого пакет 1 уходит на *E*, откуда доставляется на маршрутизатор локальной сети, *F*. Когда он прибывает на *F*, он передается внутри кадра по ЛВС на хост H2. Пакеты 2 и 3 следуют по тому же маршруту.

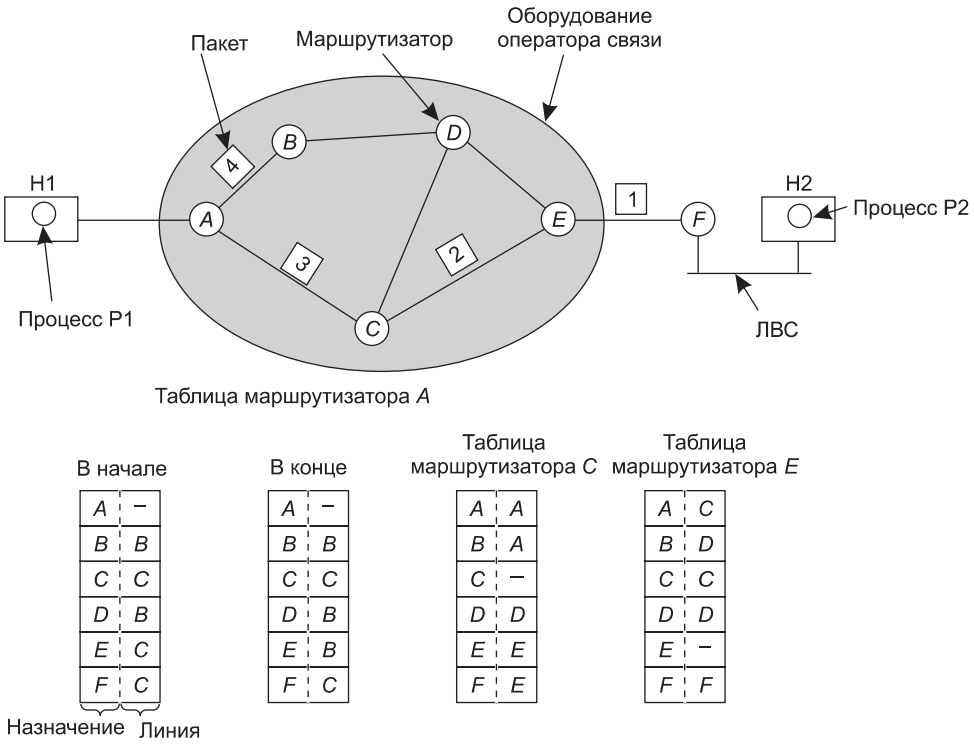


Рис. 5.2. Маршрутизация внутри дейтаграммной подсети

Однако с пакетом 4 связана несколько иная история. После прибытия на *A* он пересылается на маршрутизатор *B*, несмотря на то что адресом назначения является *F*, как и у первых трех пакетов. По каким-то своим причинам маршрутизатор *A* решил послать пакет 4 по новому маршруту. Может быть, это стало следствием затора где-то на линии *ACE*, возникшего при пересылке трех пакетов, в результате чего маршрутизатор решил обновить свою таблицу (на рисунке показана под надписью «В конце»). Алгоритм, управляющий таблицами маршрутизации и принимающий решения, называется **алгоритмом маршрутизации (routing algorithm)**. Именно изучению алгоритмов маршрутизации будет уделено основное внимание в этой главе. Как мы увидим, существует несколько типов таких алгоритмов.

IP (Internet Protocol, «межсетевой протокол»), составляющий основу всей сети Интернет, является наиболее ярким примером сетевого сервиса без установления соединения. Каждый пакет содержит IP-адрес назначения, с помощью которого маршрутизатор осуществляет индивидуальную отправку пакета. В пакетах IPv4 используются адреса длиной 32 бита, а в IPv6 — 128 бит. Более подробно о протоколах IP мы поговорим далее в этой главе.

5.1.4. Реализация сервиса с установлением соединения

Сервису с установлением соединения нужна сеть виртуального канала. Рассмотрим ее работу. Идея виртуальных каналов состоит в предотвращении выбора своего маршрута для каждого пакета, как было показано на рис. 5.2. Вместо этого маршрут от отправляющей до получающей машины выбирается в процессе установления соединения и хранится в специальных таблицах, встроенных в маршрутизаторы. Один и тот же маршрут используется для всего трафика, проходящего через данное соединение. Именно так работает телефонная система. Когда соединение разрывается, виртуальный канал также прекращает свое существование. При использовании сервиса, ориентированного на установление соединения, каждый пакет включает в себя идентификатор виртуального канала.

В качестве примера рассмотрим ситуацию, изображенную на рис. 5.3. Хост H1 установил соединение с хостом *H2*. Это соединение запоминается и становится первой записью во всех таблицах маршрутизации. Так, первая строчка таблицы маршрутизатора *A* говорит о том, что если пакет с идентификатором соединения 1 пришел с хоста H1, то его нужно направить на *C* с идентификатором соединения 1. Точно так же первая запись *C* направляет пакет на *E* все с тем же идентификатором соединения 1.

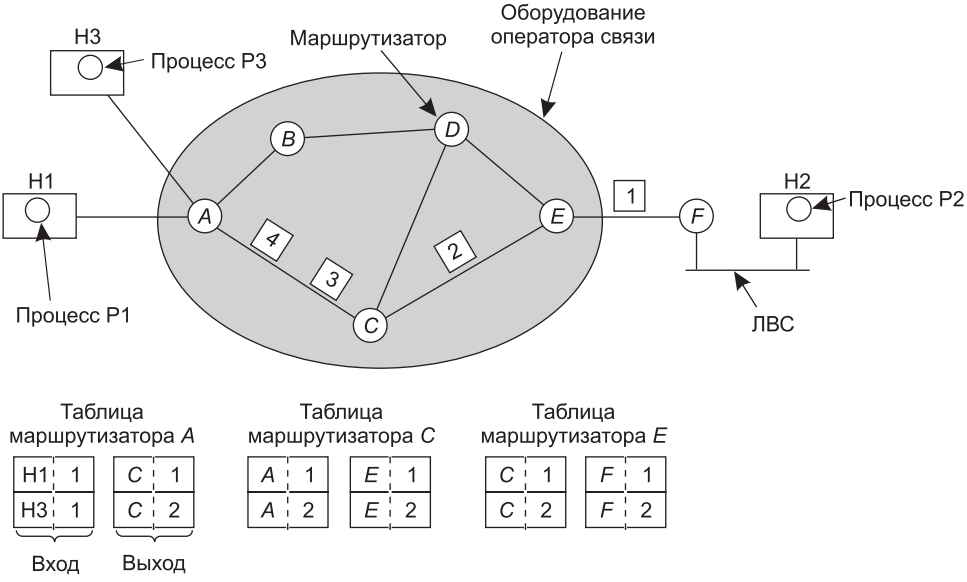


Рис. 5.3. Маршрутизация в сети виртуального канала

Теперь рассмотрим, что будет, если хост H3 захочет установить соединение с H2. Он выбирает идентификатор соединения 1 (у него просто нет выбора, поскольку это на данный момент единственное существующее соединение) и просит сеть установить виртуальный канал. Таким образом, в таблице появляется вторая запись. Обратите внимание на то, что здесь возникает, на самом деле, конфликт, потому что если *A* еще

может отличить пакеты соединения 1, пришедшие с H1, от пакетов соединения 1, пришедших с H3, то *C* такой возможности не имеет. По этой причине *A* присваивает новый идентификатор соединения исходящему трафику и тем самым создает второе соединение. Предотвращение конфликтов подобного рода является причиной того, почему маршрутизаторам нужна возможность изменения идентификаторов соединения в исходящих пакетах. Иногда этот процесс называется **коммутацией меток (label switching)**. Одним из примеров сетевого сервиса, ориентированного на соединение, является **MPLS (MultiProtocol Label Switching, «мультипротокольная коммутация по меткам»)**. Он используется в сетях интернет-провайдеров; при этом IP-пакеты получают MPLS-заголовок, содержащий 20-битный идентификатор соединения или метку. Если интернет-провайдер устанавливает длительное соединение для передачи крупных объемов данных, MPLS часто остается невидимым для клиентов. Однако сейчас он становится все более необходимым в случаях, когда на первый план выходит качество предоставляемого сервиса, а также для решения других задач, связанных с обменом данными. К обсуждению MPLS мы еще вернемся далее в этой главе.

5.1.5. Сравнение сетей виртуальных каналов и дейтаграммных сетей

Как виртуальные каналы, так и дейтаграммы имеют своих сторонников и противников. Попробуем обобщить аргументы обеих сторон. Основные аспекты сведены в табл. 5.1, хотя наверняка можно найти контраргументы для каждого пункта таблицы.

Таблица 5.1. Сравнение виртуальных каналов и дейтаграмм

Проблема	Дейтаграммы	Виртуальные каналы
Установка канала	Не требуется	Требуется
Адресация	Каждый пакет содержит полный адрес отправителя и получателя	Каждый пакет содержит короткий номер виртуального канала
Информация о состоянии	Маршрутизаторы не содержат информации о состоянии	Каждый виртуальный канал требует места в таблице маршрутизатора
Маршрутизация	Маршрут каждого пакета выбирается независимо	Маршрут выбирается при установке виртуального канала. Каждый пакет следует по этому маршруту
Эффект от выхода из строя маршрутизатора	Никакого, кроме потерянных пакетов	Все виртуальные каналы, проходившие через отказавший маршрутизатор, прекращают существование
Обеспечение качества обслуживания	Трудно реализовать	Легко реализуется при наличии достаточного количества ресурсов для каждого виртуального канала
Борьба с перегрузкой	Трудно реализовать	Легко реализуется при наличии достаточного количества ресурсов для каждого виртуального канала

Оба подхода к созданию сетей в ряде вопросов находят некие компромиссы. Во-первых, существует компромисс между временем установки соединения и временем обработки адреса. Виртуальный канал требует определенных затрат времени на его установку, однако в результате это существенно упрощает обработку пакетов данных: чтобы понять, куда должен быть отправлен пакет, маршрутизатору требуется всего лишь обратиться к таблице, зная номер канала. Дейтаграммная сеть не требует установки, однако определение адреса назначения осуществляется с помощью более сложной процедуры поиска.

Другая проблема, связанная с этим, состоит в следующем: адреса назначения в дейтаграммных сетях гораздо длиннее, чем номера каналов в сетях виртуальных каналов, поскольку они обладают глобальным значением. При сравнительно небольшом размере пакетов включение полного адреса назначения в каждый пакет может привести к существенным издержкам и фактически к снижению пропускной способности.

Еще одна проблема — количество памяти, которое маршрутизатор должен выделить для хранения таблиц. В дейтаграммной сети должно быть предусмотрено место для любого возможного адреса назначения, тогда как в сети виртуальных каналов — только для каждого канала. Однако такое преимущество на деле оказывается обманчивым, поскольку пакеты, требующиеся для установки соединения, используют адреса назначения так же, как и дейтаграммы.

Виртуальные каналы обладают некоторыми преимуществами, помогающими им предоставлять гарантированное качество обслуживания и избегать заторов в сети, так как ресурсы (такие как буфер, пропускная способность, время центрального процессора) могут быть зарезервированы заранее, во время установки соединения. Когда начинают прибывать пакеты, необходимая пропускная способность и мощность маршрутизатора будут предоставлены. В дейтаграммной сети предотвращение заторов реализовать значительно сложнее.

В системах обработки транзакций (например, при запросе магазина на верификацию кредитной карты) накладные расходы на установку соединения и удаление виртуального канала могут сильно снизить потребительские свойства сети. Если объем информации, передаваемой во время одного соединения, невелик, то использование виртуального канала не имеет смысла. Однако в случае длительных операций, таких как обмен данными через VPN внутри одной компании, постоянные виртуальные каналы (установленные вручную и не разрываемые месяцами и даже годами) могут оказаться полезными.

Недостатком виртуальных каналов является их уязвимость в случае выхода из строя или временного выключения маршрутизатора. Даже если он будет быстро починен и снова включен, все виртуальные каналы, проходившие через него, будут прерваны. Если же в дейтаграммной сети маршрутизатор выйдет из строя, то будут потеряны только те пакеты, которые находились в данный момент на маршрутизаторе (причем по всей вероятности, даже они не пострадают, поскольку отправитель, скорее всего, сразу же выполнит повторную передачу). Обрыв линии связи для виртуальных каналов является фатальным, а в дейтаграммной системе может оказаться почти незамеченным. Кроме того, дейтаграммная система позволяет соблюдать баланс между загрузкой маршрутизаторов и линий связи.

5.2. Алгоритмы маршрутизации

Основная функция сетевого уровня заключается в выборе маршрута для пакетов от начальной до конечной точки. В большинстве сетей пакетам приходится проходить через несколько маршрутизаторов. Единственным исключением являются широковещательные сети, но даже в них маршрутизация является важным вопросом, если отправитель и получатель находятся в разных сегментах сети. Алгоритмы выбора маршрутов и используемые ими структуры данных являются значительной областью при проектировании сетевого уровня.

Алгоритм маршрутизации реализуется той частью программного обеспечения сетевого уровня, которая отвечает за выбор выходной линии для отправки пришедшего пакета. Если сеть использует дейтаграммную службу, выбор маршрута для каждого пакета должен производиться заново, так как оптимальный маршрут мог измениться. Если сеть использует виртуальные каналы, маршрут выбирается только при создании нового виртуального канала. После этого все информационные пакеты следуют по установленному маршруту. Последний случай иногда называют **сеансовой маршрутизацией (session routing)**, так как маршрут остается в силе на протяжении всего сеанса связи (например, все время, пока вы подключены к сети VPN).

Полезно понимать разницу между маршрутизацией, при которой системе приходится делать выбор определенного маршрута следования, и пересылкой — действием, происходящим при получении пакета. Можно представить себе маршрутизатор как устройство, в котором функционируют два процесса. Один из них обрабатывает приходящие пакеты и выбирает для них по таблице маршрутизации исходящую линию. Такой процесс называется **пересылкой (forwarding)**. Второй процесс отвечает за заполнение и обновление таблиц маршрутизации. Именно здесь в игру вступает алгоритм маршрутизации.

Вне зависимости от того, отдельно ли выбираются маршруты для каждого отправляемого пакета или же только один раз для соединения, желательно, чтобы алгоритм выбора маршрута обладал определенными свойствами — корректностью, простотой, надежностью, устойчивостью, справедливостью и эффективностью. Корректность и простота вряд ли требуют комментариев, а вот потребность в надежности не столь очевидна с первого взгляда. Во время работы большой сети постоянно происходят какие-то отказы аппаратуры и изменения топологии. Алгоритм маршрутизации должен уметь справляться с изменениями топологии и трафика без необходимости прекращения всех задач на всех хостах. Представьте себе, что было бы, если бы сеть перезагружалась при каждой поломке маршрутизатора!

Алгоритм маршрутизации должен также обладать устойчивостью. Существуют алгоритмы выбора маршрута, никогда не сходящиеся к фиксированному набору путей, независимо от того, как долго они работают. Устойчивый алгоритм должен достигать состояния равновесия и оставаться в нем. Но он также должен быстро находить этот набор путей, так как соединение может быть прервано до того, как будет достигнуто равновесие.

Такие цели, как справедливость и эффективность, могут показаться очевидными — вряд ли кто-нибудь станет возражать против них, — однако они зачастую оказываются взаимоисключающими. Для примера рассмотрим ситуацию, показанную на рис. 5.4.

Предположим, что трафик между станциями *A* и *A′*, *B* и *B′*, а также *C* и *C′* настолько интенсивный, что горизонтальные линии связи оказываются полностью насыщенными. Чтобы максимизировать общий поток данных, трафик между станциями *X* и *X′* следовало бы совсем отключить. Однако станции *X* и *X′*, скорее всего, имеют другую точку зрения по данному вопросу. Очевидно, необходим компромисс между справедливым выделением трафика всем станциям и оптимальным использованием канала в глобальном смысле.

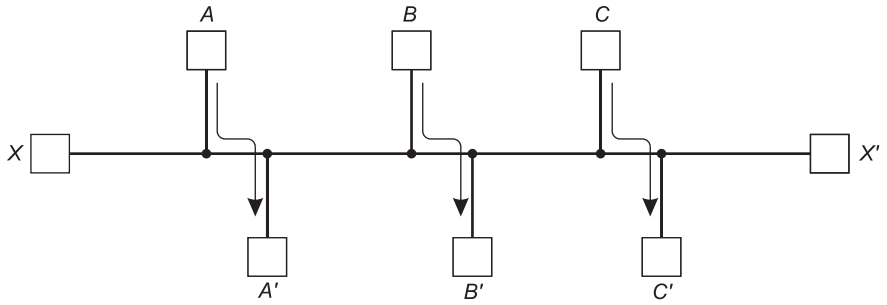


Рис. 5.4. Конфликт справедливости и оптимальности

Прежде чем пытаться искать приемлемое соотношение справедливости и эффективности, следует решить, что именно мы будем стремиться оптимизировать. Для увеличения эффективности передачи данных по сети можно попробовать минимизировать среднее время задержки или увеличить общую пропускную способность сети. Однако эти цели также противоречат друг другу, поскольку работа любой системы с очередями вблизи максимума производительности предполагает долгое стояние в очередях. В качестве компромисса многие сети стараются минимизировать расстояние, которое должен пройти пакет, или просто снизить количество пересылок для каждого пакета. В обоих случаях время прохождения каждого пакета по сети снижается, в результате чего улучшается пропускная способность всей сети.

Алгоритмы выбора маршрута можно разбить на два основных класса: неадаптивные и адаптивные. **Неадаптивные алгоритмы** не учитывают при выборе маршрута топологию и текущее состояние сети и не измеряют трафик на линиях. Вместо этого выбор маршрута для каждой пары станций производится заранее, в автономном режиме, и список маршрутов загружается в маршрутизаторы во время загрузки сети. Такая процедура иногда называется **статической маршрутизацией (static routing)**. Поскольку статическая маршрутизация не реагирует на сбои, она, как правило, используется в тех случаях, когда выбор маршрута очевиден. Например, маршрутизатор *F* на рис. 5.3 должен отправлять пакеты, передаваемые по сети, на маршрутизатор *E* независимо от конечного адреса назначения.

Адаптивные алгоритмы, напротив, изменяют решение о выборе маршрутов при изменении топологии и также иногда в зависимости от загруженности линий. Эти **динамические алгоритмы маршрутизации** (dynamic routing algorithms) отличаются источниками получения информации (такие источники могут быть, например, локальными, если это соседние маршрутизаторы, либо глобальными, если это вообще

все маршрутизаторы сети), моментами изменения маршрутов (например, при изменении топологии или через определенные равные интервалы времени при изменении нагрузки) и данными, использующимися для оптимизации (расстояние, количество транзитных участков или ожидаемое время пересылки).

В следующих разделах мы обсудим различные алгоритмы маршрутизации. Помимо отправки пакета от источника к месту назначения такие алгоритмы предусматривают модель предоставления информации. Иногда требуется отправить пакет на несколько адресов из заданного списка, на все такие адреса или на один из них. Все алгоритмы, о которых мы будем здесь говорить, принимают решения на основании топологии; вопрос о возможности учета интенсивности передачи данных мы оставим до раздела 5.3.

5.2.1. Принцип оптимальности маршрута

Прежде чем перейти к рассмотрению отдельных алгоритмов, возможно, следует привести некие общие положения, описывающие оптимальные маршруты, вне зависимости от топологии или трафика. Такой основополагающей идеей является **принцип оптимальности** (Беллман, 1957). В соответствии с этим принципом, если маршрутизатор J располагается на оптимальном маршруте от маршрутизатора I к маршрутизатору K, то оптимальный маршрут от маршрутизатора J к маршрутизатору K совпадет с частью первого маршрута. Чтобы убедиться в этом, обозначим часть маршрута от маршрутизатора I к маршрутизатору J как r_1, а остальную часть маршрута — r_2. Если бы существовал более оптимальный маршрут от маршрутизатора J к маршрутизатору K, чем r_2, то его можно было объединить с r_1, чтобы улучшить маршрут от маршрутизатора I к маршрутизатору K, что противоречит первоначальному утверждению о том, что маршрут r_1r_2 является оптимальным.

Прямым следствием принципа оптимальности является возможность рассмотрения множества оптимальных маршрутов от всех источников к конкретным приемникам в виде дерева, имеющего приемник в качестве корня. Такое дерево называется **входным деревом (sink tree)**. Оно изображено на рис. 5.5, *б*. Расстояния измеряются количеством транзитных участков. Цель всех алгоритмов выбора маршрутов заключается в вычислении и использовании входных деревьев для всех маршрутизаторов.

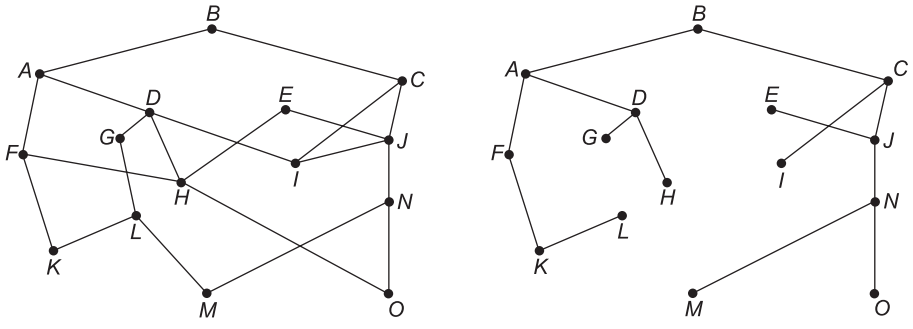


Рис. 5.5. Сеть (*а*); входное дерево для маршрутизатора *B* (*б*)

Обратите внимание на то, что входное дерево не обязательно является уникальным; у одной сети может существовать несколько деревьев с одинаковыми длинами путей. Если мы будем считать все возможные пути допустимыми, мы получим более общую структуру, и наше дерево будет подходить под определение **направленного ациклического графа (directed acyclic graph, DAG)**. В таких графах нет циклов. Мы будем использовать понятие входного дерева для обозначения обоих вариантов. Оба они также основываются на предположении, что пути не мешают друг другу (из этого следует, в частности, что появление затора на одном пути не приводит к изменению другого пути).

Поскольку входное дерево действительно является деревом, оно не содержит петель, поэтому каждый пакет будет доставлен за конечное и ограниченное число пересылок. На практике все это не так просто. Линии связи и маршрутизаторы могут выходить из строя и снова появляться в сети во время выполнения операции, поэтому у разных маршрутизаторов могут оказаться различные представления о текущей топологии сети. Кроме того, мы обошли вопрос о том, собирает ли маршрутизатор сам информацию для вычисления входного дерева, или эта информация каким-то другим образом поступает к нему. Мы вскоре рассмотрим этот вопрос. Тем не менее принцип оптимальности и входное дерево — это те точки отсчета, относительно которых можно измерять эффективность различных алгоритмов маршрутизации.

5.2.2. Алгоритм нахождения кратчайшего пути

Начнем наше изучение алгоритмов выбора маршрутов с простого метода вычисления кратчайших путей, требующего полной информации о сети. Целью распределенного алгоритма является нахождение таких путей даже в тех случаях, когда маршрутизатор не располагает всеми сведениями о сети.

Идея заключается в построении графа сети, в котором каждый узел будет соответствовать маршрутизатору, а каждое ребро — линии связи или просто связи. При выборе маршрута между двумя маршрутизаторами алгоритм просто находит кратчайший путь между ними на графе.

Концепция **кратчайшего пути (shortest path)** требует некоторого пояснения. Один из способов измерения длины пути состоит в подсчете количества транзитных участков. В таком случае пути *ABC* и *ABE* на рис. 5.6 имеют одинаковую длину. Можно измерять расстояния в километрах. В таком случае окажется, что путь *ABC* значительно длиннее пути *ABE* (предполагается, что рисунок изображен с соблюдением масштаба).

Однако, помимо учета количества транзитных участков и физической длины линий, возможен учет и других параметров. Например, каждому ребру графа можно поставить в соответствие среднее время задержки стандартного тестового пакета, измеряемое каждый час. В таком графе кратчайший путь определяется как самый быстрый путь, а не путь с наименьшим числом ребер или наименьшей длиной в километрах.

В общем случае параметры ребер графа являются функциями расстояния, пропускной способности, средней загруженности, стоимости связи, измеренной величины задержки и других факторов. Изменяя весовую функцию, алгоритм может вычислять кратчайший путь с учетом любого количества критериев в различных комбинациях.

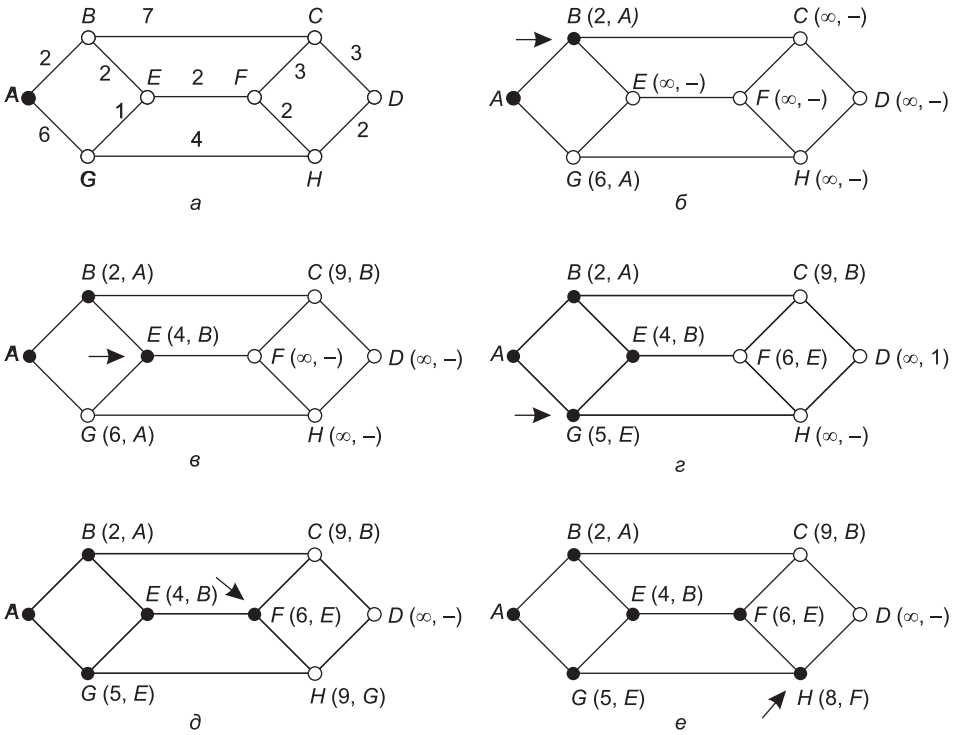


Рис. 5.6. Первые шесть шагов вычисления кратчайшего пути от A к D. Стрелка указывает на рабочий узел

Известно несколько алгоритмов вычисления кратчайшего пути между двумя узлами графа. Один из них был создан знаменитым Дейкстрой (Dijkstra) в 1959 году. Он находит кратчайшие пути между отправителем и всеми возможными адресами назначения в данной сети. Каждый узел помечается (в скобках) расстоянием до него от узла отправителя по наилучшему известному пути. Эти расстояния должны быть неотрицательными; если они основаны на реальных величинах — таких как пропускная способность и время задержки, — это условие будет выполнено. Вначале пути неизвестны, поэтому все узлы помечаются символом бесконечности. По мере работы алгоритма и нахождения путей отметки узлов изменяются, показывая оптимальные пути. Отметка может быть постоянной или экспериментальной. Вначале все отметки являются ориентировочными. Когда выясняется, что отметка действительно соответствует кратчайшему возможному пути, она становится постоянной и в дальнейшем не изменяется.

Чтобы показать, как работает этот алгоритм, рассмотрим взвешенный ненаправленный граф на рис. 5.6, *а*, где весовые коэффициенты соответствуют, например, расстояниям. Мы хотим найти кратчайший путь от *A* к *D*. Для начала мы черным кружком помечаем узел *A* как постоянный. Затем мы исследуем все соседние с ним узлы, указывая около них расстояние до узла *A*. Если отыскивается более короткий путь к какому-либо узлу, то вместе с указанием расстояния в отметке меняется и узел, через который прошел более короткий путь. Таким образом, позднее можно восста-

новить весь путь. Если бы в сети было более одного кратчайшего пути от *A* до *D* и мы хотели бы найти их все, нам нужно было бы запоминать все узлы, которые позволяют дойти до данного узла, пройдя одинаковое расстояние.

Рассмотрев все соседние с *A* узлы, мы помечаем ближайший узел как постоянный, как показано на рис. 5.6, *б*. Этот узел становится новым рабочим узлом.

Теперь мы повторяем ту же процедуру с узлом *B*, исследуя все его соседние узлы. Если сумма расстояния от узла *B* и значения отметки в узле *B* (расстояния от *A* до *B*) оказывается меньше, чем отметка у исследуемого узла (уже найденное другим путем расстояние от *A*), это значит, что найден более короткий путь, поэтому пометка узла изменяется.

После того как все соседние с рабочим узлы исследованы и временные отметки при необходимости изменены, по всему графу ищется узел с наименьшей временной отметкой. Этот узел помечается как постоянный и становится текущим рабочим узлом. На рис. 5.6 показаны первые шесть этапов работы алгоритма.

Чтобы понять, как работает алгоритм, посмотрим на рис. 5.6, *в*. На данном этапе узел *E* только что был отмечен как постоянный. Предположим, что существует более короткий путь, нежели *ABE*, например *AXYZE* (для некоторых *X* и *Y*). В этом случае есть две возможности — либо узел *Z* уже сделан постоянным, либо еще нет. Если да, значит, узел *E* уже проверялся, когда узел *Z* был сделан постоянным и, следовательно, рабочим узлом. В этом случае путь *AXYZE* уже исследовался.

Теперь рассмотрим случай, когда узел *Z* все еще помечен как временный. Если отметка узла *Z* больше или равна пометки узла *E*, путь *AXYZE* не может быть короче, чем путь *ABE*. Если же отметка узла *Z* меньше пометки узла *E*, то тогда узел *Z* должен был стать постоянным раньше узла *E*, и узел *E* проверялся бы с узла *Z*.

Этот алгоритм приведен в листинге 5.1. Глобальные переменные *n* и *dist* описывают граф и инициализируются раньше, чем вызывается *shortest_path*. Единственное отличие программы от описанного выше алгоритма заключается в том, что вычисление кратчайшего пути в программе начинается не от узла-источника *s*, а от конечного узла *t*.

Поскольку в однонаправленном графе кратчайшие пути от *t* к *s* те же самые, что и от *s* к *t*, не имеет значения, с какого конца начинать. Причина поиска пути в обратном направлении заключается в том, что каждый узел помечается предшествующим узлом, а не следующим. Когда найденный путь копируется в выходную переменную *path*, он инвертируется. В результате двух инверсий мы получаем путь, идущий в нужную сторону.

Листинг 5.1. Алгоритм Дейкстры вычисления кратчайшего пути по графу

```
#define MAX_NODES 1024              /* максимальное количество узлов */
#define INFINITY 1000000000         /* число, большее длины максимального пути */
int n, dist[MAX_NODES][MAX_NODES]; /* dist[i][j] - это расстояние от i до j */
void shortest_path(int s, int t, int path[])
{ struct state {                    /* рабочий путь */
    int predecessor;                /* предыдущий узел */
    int length;                     /* расстояние от источника до этого узла */
    enum {permanent, tentative} label; /* метка состояния */
  } state[MAX_NODES];
  int i, k, min;
```

продолжение ⇩

Листинг 3.7 (*продолжение*)

```
  struct state *p;
  for (p = &state[0]; p < &state[n]; p++) { /* инициализация состояния */
      p->predecessor = -1;
      p->length = INFINITY;
      p->label = tentative;
  }
  state[t].length = 0; state[t].label = permanent;
  k = t;                                  /* k - начальный рабочий узел */
  do {                                    /* Есть ли лучший путь от k? */
      for (i = 0; i < n; i++)             /* У этого графа n узлов */
          if (dist[k][i] != 0 && state[i].label == tentative) {
              if (state[k].length + dist[k][i] < state[i].length) {
                  state[i].predecessor = k;
                  state[i].length = state[k].length + dist[k][i];
              }
          }
      /* Поиск узла, помеченного как предварительный с наименьшей меткой */
      k = 0; min = INFINITY;
      for (i = 0; i < n; i++)
          if (state[i].label == tentative && state[i].length < min) {
              min = state[i].length;
              k =i;
          }
      state[k].label = permanent;
  } while (k != s);
  /* Копирование пути в выходной массив */
  i = 0; k = s;
  do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

5.2.3. Заливка

При реализации алгоритма маршрутизации любой маршрутизатор должен принимать решения на основании локальных сведений, а не полной информации о сети. Одним из простых локальных методов является **заливка (flooding)**, при которой каждый приходящий пакет посылается на все исходящие линии, кроме той, по которой он пришел.

Очевидно, что алгоритм заливки порождает огромное количество дублированных пакетов, даже бесконечное количество в сетях с замкнутыми контурами, если не принять специальных мер. Одна из таких мер состоит в помещении в заголовок пакета счетчика преодоленных им транзитных участков, уменьшаемого при прохождении каждого маршрутизатора. Когда значение этого счетчика падает до нуля, пакет удаляется. В идеальном случае счетчик транзитных участков должен вначале устанавливаться равным длине пути от отправителя до получателя. Если отправитель не знает расстояния до получателя, он может установить значение счетчика равным длине максимального пути (диаметру) в данной сети.

В результате заливки с использованием счетчика переходов количество отправленных копий пакета может расти экспоненциально, если маршрутизатор повторно отправляет пакеты, которые он уже видел. Хороший способ ограничения количества

тиражируемых пакетов заключается в учете проходящих через маршрутизатор пакетов. Это позволяет не посылать их повторно. Один из методов состоит в том, что каждый маршрутизатор кладет в каждый получаемый от своих хостов пакет порядковый номер. Все маршрутизаторы ведут список маршрутизаторов-источников, в котором сохраняются все порядковые номера пакетов, которые им встречались. Если пакет от данного источника с таким порядковым номером уже есть в списке, он дальше не распространяется и удаляется.

Чтобы предотвратить неограниченный рост размера списка, можно снабдить все списки счетчиком k, показывающим, что все порядковые номера вплоть до k уже встречались. И когда приходит пакет, можно очень легко проверить, был ли он уже передан, сравнивая его порядковый номер с k; при положительном ответе такой пакет отвергается. Кроме того, не нужно хранить весь список до k, поскольку этот счетчик очень действенно подытоживает его.

В большинстве случаев использование алгоритма заливки для отправки пакетов является непрактичным, но тем не менее иногда он оказывается очень полезным. Во-первых, он гарантированно доставляет пакет в каждый узел сети. Если пакет нужно доставить в одно конкретное место, этот метод может не оправдать себя. Однако он оказывается очень эффективным при широковещательной рассылке. В беспроводных сетях сообщения, передаваемые станцией, могут быть приняты любой другой станцией, находящейся в пределах радиуса действия передатчика, — и это фактически является заливкой. Некоторые алгоритмы используют это свойство.

Во-вторых, метод заливки отличается чрезвычайной надежностью. Даже если большая часть маршрутизаторов окажется полностью уничтоженной (например, если речь идет о военной сети связи в зоне вооруженных конфликтов), любой существующий путь для доставки сообщения будет найден. Кроме того, заливка практически не требует настройки. Маршрутизаторы должны лишь знать своих соседей. Это означает, что заливка может использоваться внутри другого алгоритма, более эффективного, но требующего более тщательной настройки. Также алгоритм заливки может служить эталоном при тестировании других алгоритмов выбора маршрутов, так как он всегда находит все возможные пути в сети, а следовательно, и кратчайшие. Ухудшить эталонные показатели времени доставки могут разве что накладные расходы, вызванные огромным количеством пакетов, формируемых самим алгоритмом заливки.

5.2.4. Маршрутизация по вектору расстояний

Компьютерные сети обычно используют динамические алгоритмы маршрутизации, которые являются более сложными, чем заливка, но в то же время и более эффективными, поскольку они позволяют находить кратчайшие пути для текущей топологии. Самой большой популярностью пользуются два динамических метода: маршрутизация по вектору расстояний и маршрутизация с учетом состояния каналов. В этом разделе мы изучим первый, в следующем — второй метод.

Алгоритмы **маршрутизации по вектору расстояний (distance vector routing)** работают, опираясь на таблицы (то есть векторы), поддерживаемые всеми маршрутизаторами и содержащие сведения о кратчайших известных путях к каждому из возможных адресатов и о том, какое соединение следует при этом использовать. Для

обновления данных этих таблиц производится обмен информацией с соседними маршрутизаторами. В результате маршрутизатор знает наилучший способ добраться до любого адреса назначения.

Алгоритм маршрутизации по вектору расстояний иногда называют по именам его создателей распределенным алгоритмом **Беллмана—Форда** (Bellman—Ford) (Bellman, 1957; Ford и Filkerson, 1962). Этот алгоритм изначально применялся в сети ARPANET и в Интернете был известен под названием RIP.

При маршрутизации по вектору расстояний таблицы, с которыми работают и которые обновляют маршрутизаторы, содержат записи о каждом маршрутизаторе сети. Каждая запись состоит из двух частей: предпочитаемый номер линии для данного получателя и предполагаемое расстояние или время прохождения пакета до этого получателя. В качестве меры расстояния можно использовать число транзитных участков или другие единицы измерения (о которых мы говорили при обсуждении длины кратчайшего пути).

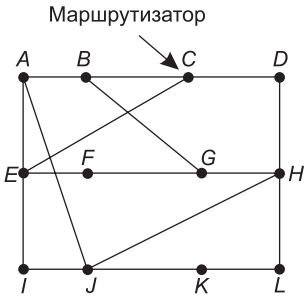
Предполагается, что маршрутизаторам известно расстояние до каждого из соседей. Если в качестве единицы измерения используется число транзитных участков, то расстояние равно одному транзитному участку. Если же дистанция измеряется временем задержки распространения, то маршрутизатор может измерить его с помощью специального пакета ECHO (эхо), в который получатель помещает время получения и который отправляет обратно как можно быстрее.

Предположим, что в качестве единицы измерения используется время задержки и этот параметр относительно каждого из соседей известен маршрутизатору. Через каждые *T* мс все маршрутизаторы посылают своим соседям список с приблизительными задержками для каждого получателя. Они, разумеется, также получают подобный список от всех своих соседей. Допустим, одна из таких таблиц пришла от соседа *X* и в ней указывается, что время распространения от маршрутизатора *X* до маршрутизатора *i* равно *Xi*. Если маршрутизатор знает, что время пересылки до маршрутизатора *X* равно *m*, тогда задержка при передаче пакета маршрутизатору *i* через маршрутизатор *X* составит *Xi* + *m*. Выполнив такие расчеты для всех своих соседей, маршрутизатор может выбрать наилучшие пути и поместить соответствующие записи в новую таблицу. Обратите внимание, что старая таблица в расчетах не используется.

Процесс обновления таблицы проиллюстрирован на рис. 5.7. На рис. 5.7, *а* показана сеть. Первые четыре столбца на рис. 5.7, *б* показывают векторы задержек, полученные маршрутизатором *J* от своих соседей. Маршрутизатор *A* считает, что время пересылки от него до маршрутизатора *B* равно 12 мс, 25 мс до маршрутизатора *C*, 40 мс до *D* и т. д. Предположим, что маршрутизатор *J* измерил или оценил задержки до своих соседей *A*, *I*, *H* и *K* как 8, 10, 12 и 6 мс соответственно.

Теперь рассмотрим, как *J* рассчитывает свой новый маршрут к маршрутизатору *G*. Он знает, что задержка до *A* составляет 8 мс и при этом *A* думает, что от него до *G* данные дойдут за 18 мс. Таким образом, *J* знает, что если он станет отправлять пакеты для *G* через *A*, то задержка составит 26 мс. Аналогично, он вычисляет значения задержек для маршрутов от него до *G*, проходящих через остальных его соседей (*I*, *H* и *K*), и получает соответственно 41 (31+10), 18 (6+12) и 37 (31+6). Лучшим значением является 18, поэтому именно оно помещается в таблицу в запись для получателя *G*. Вместе с числом 18 туда же помещается обозначение линии, по которой

проходит самый короткий маршрут до *G*, то есть *H*. Данный метод повторяется для всех остальных адресатов, и при этом получается новая таблица, показанная в виде правого столбца на рисунке.



а

K	A	I	H	K	Новая расчетная задержка от J	Линия
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	–
K	24	22	22	0	6	K
L	29	33	9	9	15	K

Задержка JA равна 8 | Задержка JI равна 10 | Задержка JH равна 12 | Задержка JK равна 6

Векторы, полученные от четырех соседей J

Новая таблица маршрутов для J

б

Рис. 5.7. Сеть (*а*); полученные от *A*, *I*, *H* и *K* векторы и новая таблица маршрутов для *J* (*б*)

Проблема счета до бесконечности

Установление маршрутов, соответствующих кратчайшим путям, в сети называется **конвергенцией (convergence)**. Алгоритм маршрутизации по вектору расстояний — простой метод, позволяющий маршрутизаторам совместно вычислять кратчайшие пути. Однако на практике он обладает серьезным недостатком: хотя правильный ответ в конце концов и находится, процесс его поиска может занять довольно много времени. В частности, такой алгоритм быстро реагирует на хорошие новости и очень лениво — на плохие. Рассмотрим маршрутизатор, для которого расстояние до маршрутизатора *X* достаточно велико. Если при очередном обмене векторами его сосед *A* сообщит ему, что от него до маршрутизатора *X* совсем недалеко, наш маршрутизатор просто переключится для передач маршрутизатору *X* на линию, проходящую через этого соседа. Таким образом, хорошая новость распространилась всего за один обмен информацией.

Чтобы увидеть, как быстро распространяются хорошие известия, рассмотрим линейную сеть из пяти узлов, показанную на рис. 5.8, в которой мерой расстояния служит количество транзитных участков. Предположим, что вначале маршрутизатор *A* выключен, и все остальные маршрутизаторы об этом знают. То есть они считают, что расстояние до *A* равно бесконечности.

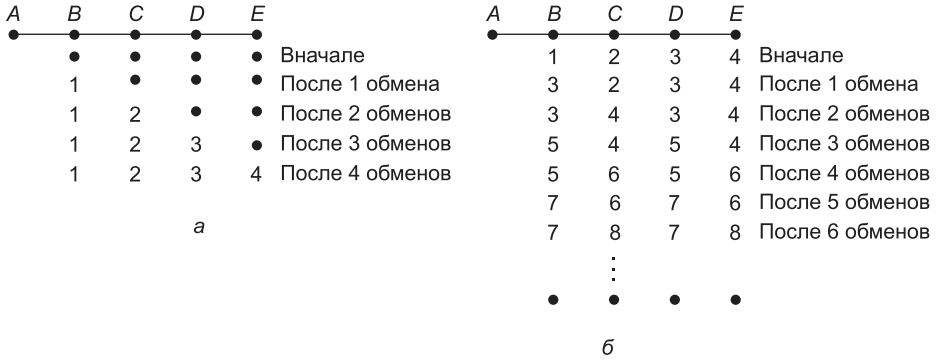


Рис. 5.8. Проблема счета до бесконечности

Когда в сети появляется *A*, остальные маршрутизаторы узнают об этом с помощью обмена векторами. Для простоты будем предполагать, что где-то в сети имеется гигантский гонг, в который периодически ударяют, чтобы инициировать одновременный обмен векторами. После первого обмена *B* узнает, что у его соседа слева нулевая задержка при связи с *A*, а *B* помечает в своей таблице маршрутов, что *A* находится слева на расстоянии одного транзитного участка. Все остальные маршрутизаторы в этот момент еще полагают, что *A* выключен. Значения задержек для *A* в таблицах на этот момент показаны во второй строке на рис. 5.8, *а*. При следующем обмене информацией *C* узнает, что у *B* есть путь к *A* длиной 1, поэтому он обновляет свою таблицу, указывая длину пути до *A*, равную 2, но *D* и *E* об этом еще не знают. Таким образом, хорошие вести распространяются со скоростью один транзитный участок за один обмен векторами. Если самый длинный путь в сети состоит из *N* транзитных участков, то через *N* обменов все маршрутизаторы подсети будут знать о включенных маршрутизаторах и заработавших линиях.

Теперь рассмотрим ситуацию на рис. 5.8, *б*, в которой все связи и маршрутизаторы изначально находятся во включенном состоянии. Маршрутизаторы *B*, *C*, *D* и *E* находятся на расстоянии 1, 2, 3 и 4 транзитных участков от *A* соответственно. Внезапно либо *A* отключается, либо происходит обрыв линии между *A* и *B* (что с точки зрения *B* одно и то же).

При первом обмене пакетами *B* не слышит ответа от *A*. К счастью, *C* говорит: «Не волнуйся. У меня есть путь к *A* длиной 2». *B* вряд ли догадывается, что путь от *C* к *A* проходит через *B*. *B* может только предполагать, что у *C* около 10 выходных связей с независимыми путями к *A*, кратчайшая из которых имеет длину 2. Поэтому теперь *B* думает, что может связаться с *A* через *C* по пути длиной 3. При этом первом обмене маршрутизаторы *D* и *E* не обновляют свою информацию об *A*.

При втором обмене векторами *C* замечает, что у всех его соседей есть путь к *A* длиной 3. Он выбирает один из них случайным образом и устанавливает свое расстояние до *A* равным 4, как показано в третьей строке на рис. 5.8, *б*. Результаты последующих обменов векторами также показаны на этом рисунке.

Теперь должно быть понятно, почему плохие новости медленно распространяются — ни один маршрутизатор не может установить значение расстояния, более чем на

единицу превышающее минимальное значение этого расстояния, хранящееся у его соседей. Таким образом, все маршрутизаторы будут до бесконечности увеличивать значение расстояния до выключенного маршрутизатора. Количество необходимых для завершения этого процесса обменов векторами можно ограничить, если установить значение этой «бесконечности» равным длине самого длинного пути плюс 1.

Неудивительно, что эта проблема называется **счетом до бесконечности (count-to-infinity)**. Было сделано много попыток решить ее, например можно запретить маршрутизатору сообщать о своих кратчайших путях соседям, от которых они получили эту информацию, с помощью правила расколотого горизонта с «отравляющим» ответом внесенного в RFC 1058. Однако на практике все эти эвристические правила с красивыми названиями оказались абсолютно бесполезными. Суть проблемы заключается в том, что когда *X* сообщает *Y* о том, что у него есть какой-то путь, у *Y* нет никакой возможности узнать, входит ли он сам в этот путь.

5.2.5. Маршрутизация с учетом состояния линий

Маршрутизация на основе векторов расстояний использовалась в сети ARPANET вплоть до 1979 года, когда ее сменил алгоритм маршрутизации с учетом состояния линий. Отказаться от прежнего алгоритма пришлось в первую очередь потому, что при изменении топологии сети алгоритм слишком долго приходил к устойчивому состоянию (вследствие проблемы счета до бесконечности). В результате он был заменен на совершенно новый алгоритм, ныне называемый **маршрутизацией с учетом состояния линий (link state routing)**. Сейчас в крупных сетях и сети Интернет используются его варианты — алгоритмы маршрутизации IS-IS и OSPF.

В основе алгоритма лежит относительно простая идея, ее можно изложить в *пяти требованиях к маршрутизатору*. Каждый маршрутизатор должен:

1) обнаруживать своих соседей и узнавать их сетевые адреса;
2) задавать метрику расстояния или стоимости связи с каждым из своих соседей;
3) создавать пакет, содержащий всю собранную информацию;
4) посылать этот пакет всем маршрутизаторам и принять все пакеты, отправленные другими маршрутизаторами;
5) вычислять кратчайший путь ко всем маршрутизаторам.

В результате каждому маршрутизатору высылается полная топология. После этого для обнаружения кратчайшего пути ко всем остальным маршрутизаторам каждый маршрутизатор может использовать алгоритм Дейкстры. Ниже мы рассмотрим каждый из этих пяти этапов более подробно.

Знакомство с соседями

Когда маршрутизатор загружается, его первая задача состоит в получении информации о своих соседях. Он достигает этой цели, посылая специальный пакет `HELLO` по всем линиям «точка-точка». При этом маршрутизатор на другом конце линии должен послать ответ, содержащий его имя. Имена маршрутизаторов должны быть совершенно уникальными, поскольку если удаленный маршрутизатор слышит,

что три маршрутизатора являются соседями маршрутизатора *F*, то не должно возникать разночтений по поводу того, один и тот же маршрутизатор *F* имеется в виду или нет.

Когда два или более маршрутизаторов соединены с помощью широковещательной связи (например, коммутатора, кольцевой сети или классической сети Ethernet), ситуация несколько усложняется. На рис. 5.9, *а* изображена широковещательная ЛВС, к которой напрямую подключены три маршрутизатора: *A*, *C* и *F*. Каждый из них, как показано на рисунке, соединен также с одним или несколькими дополнительными маршрутизаторами.

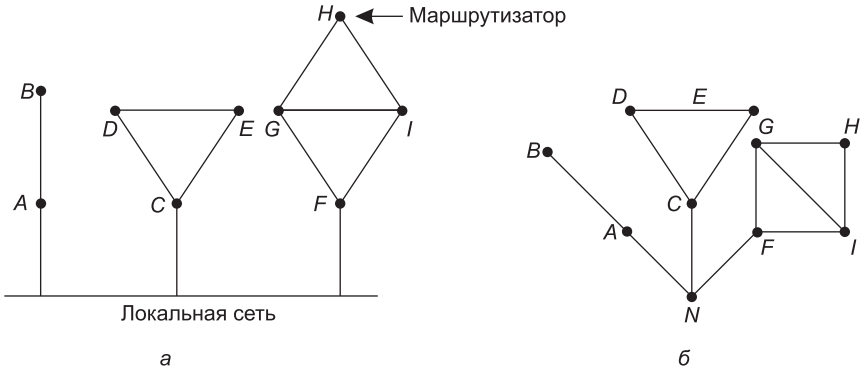


Рис. 5.9. Широковещательная ЛВС: *а* — девять маршрутизаторов и широковещательная локальная сеть; *б* — графовая модель той же системы

Широковещательная ЛВС обеспечивает связь между каждой парой подключенных маршрутизаторов. Однако моделирование такой сети в виде системы связей «точка-точка» существенно увеличивает размер топологии и приводит к нерациональной передаче сообщений. Более подходящий способ моделирования локальной сети состоит в том, что ЛВС рассматривается в виде узла графа, как и маршрутизаторы. Это показано на рис. 5.9, *б*. На рисунке сеть изображена в виде искусственного узла *N*, с которым соединены маршрутизаторы *A*, *C* и *F*. Для исполнения роли *N* в протоколе маршрутизации выбирается один из маршрутизаторов сети, который называется **отмеченным маршрутизатором (designated router)**. Возможность передачи пакетов от *A* к *C* по локальной сети отражается здесь наличием пути *ANC*.

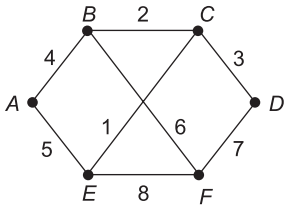
Задание стоимости связи

Алгоритм маршрутизации с учетом состояния линии требует, чтобы каждая связь обладала метрикой расстояния или стоимости, необходимой для вычисления кратчайшего пути. Стоимость пути до соседних маршрутизаторов может быть задана автоматически или определена оператором сети. Чаще всего стоимость обратно пропорциональна пропускной способности связи. Так, сеть Ethernet со скоростью 1 Гбит/с может иметь стоимость 1, а Ethernet со скоростью 100 Мбит/с — стоимость 10. Благодаря этому в качестве наилучших путей будут выбираться пути с более высокой пропускной способностью. Если узлы сети находятся на большом расстоянии друг от

друга, при вычислении стоимости может учитываться задержка соединения. В таком случае в качестве наилучших путей будут выбираться более короткие. Наиболее прямой способ определить эту задержку заключается в посылке по линии специального пакета ECHO, на который другая сторона обязана немедленно ответить. Измерив время двойного оборота этого пакета и разделив его на два, отправитель получает приемлемую оценку задержки.

Создание пакетов состояния линий

После того как информация, необходимая для обмена, собрана, следующий шаг, выполняемый каждым маршрутизатором, заключается в создании пакета, содержащего все эти данные. Пакет начинается с идентификатора отправителя, за которым следует порядковый номер и возраст (описываемый ниже), а также список соседей. Для каждого соседа также указывается соответствующая стоимость пути связи с ним. Пример сети приведен на рис. 5.10, *а*, на котором показаны стоимости для каждой линии. Соответствующие пакеты состояния линий для всех шести маршрутизаторов показаны на рис. 5.10, *б*.



а

Пакеты состояния линий

A	
Порядковый номер	
Возраст	
B	4
E	5

B	
Порядковый номер	
Возраст	
A	4
C	2
F	6

C	
Порядковый номер	
Возраст	
B	2
D	3
E	1

D	
Порядковый номер	
Возраст	
C	3
F	7

E	
Порядковый номер	
Возраст	
A	5
C	1
F	8

F	
Порядковый номер	
Возраст	
B	6
D	7
E	8

б

Рис. 5.10. Сеть (*а*); пакеты состояния линий для этой сети (*б*)

Создаются пакеты состояния линий несложно. Самое трудное заключается в выборе момента времени для их создания. Их можно создавать периодически через равные интервалы времени. Другой вариант состоит в создании пакетов, когда происходит какое-нибудь значительное событие, — например, линия или сосед выходит из строя или, наоборот, снова появляется в сети, либо существенно изменяет свои свойства.

Распространение пакетов состояния линий

Самая сложная часть алгоритма заключается в распространении пакетов состояния линий. Все маршрутизаторы должны принимать такие пакеты быстро и безотказно. Если разные маршрутизаторы используют разные версии топологии, это может привести к противоречиям, таким как петли в маршрутах, недоступность машин, а также другие проблемы.

Сначала мы опишем основной алгоритм распространения. Затем расскажем о некоторых улучшениях. Основная идея алгоритма отправки пакетов состояния линии на все маршрутизаторы состоит в использовании алгоритма заливки. Чтобы держать этот процесс под контролем, в каждый пакет помещают порядковый номер, увеличивающийся на единицу для каждого следующего пакета. Маршрутизаторы записывают все пары (источник, порядковый номер), которые им попадаются. Когда приходит новый пакет состояния линий, маршрутизатор ищет адрес его отправителя и порядковый номер пакета в своем списке. Если это новый пакет, он рассылается дальше по всем линиям, кроме той, по которой он пришел. Если же это дубликат, он удаляется. Если порядковый номер прибывшего пакета меньше, чем номер уже полученного пакета от того же отправителя, то такой пакет также удаляется как устаревший, поскольку очевидно, что у маршрутизатора есть более свежие данные.

С этим алгоритмом связано несколько проблем, но с ними можно справиться. Во-первых, если последовательный номер, достигнув максимально возможного значения числа, обнулится, возникнет путаница. Решение состоит в использовании 32-битных порядковых номеров. Даже если рассылать каждую секунду по пакету, то для переполнения 4-байтового целого числа понадобится 137 лет.

Во-вторых, если маршрутизатор выйдет из строя, будет потерян его порядковый номер. Если он будет снова загружен с нулевым порядковым номером, следующий отправленный с него пакет будет проигнорирован как устаревший.

В-третьих, может произойти искажение порядкового номера — например, вместо номера 4 будет принято число 65 540 (ошибка в одном бите); в этом случае пакеты с 5-го по 65 540-й будут игнорироваться некоторыми маршрутизаторами как устаревшие.

Решение этих проблем заключается в помещении в пакет после его порядкового номера возраста пакета и уменьшении его на единицу каждую секунду. Когда возраст уменьшается до нуля, информация от этого маршрутизатора удаляется. В нормальной ситуации новый пакет приходит, например, каждые 10 секунд; таким образом, сведения о маршрутизаторе устаревают, только когда маршрутизатор выключен (или в случае потери шести пакетов подряд, что маловероятно). Поле возраста также уменьшается на единицу каждым маршрутизатором во время начального процесса заливки, чтобы гарантировать, что ни один пакет не потеряется и не будет жить вечно.

Для повышения надежности этого алгоритма используются некоторые усовершенствования. Когда пакет состояния линий приходит на маршрутизатор для заливки, он не ставится сразу в очередь на отправку. Вместо этого он сохраняется в течение некоторого периода времени в области промежуточного хранения на случай появления новых связей или разрыва старых. Если за это время от того же отправителя успевает прийти еще один пакет, маршрутизатор сравнивает их порядковые номера. Более ста-

рый пакет удаляется. Если номера одинаковые, то удаляется дубликат. Для защиты от ошибок на линиях связи получение всех пакетов состояния линий подтверждается. Структура данных, используемая маршрутизатором *B* для работы с сетью, изображенной на рис. 5.10, *a*, показана на рис. 5.11. Каждая строка здесь соответствует недавно полученному, но еще не полностью обработанному пакету состояния линий. В таблице записываются адрес отправителя, порядковый номер, возраст и данные. Кроме того, в таблице содержатся флаги рассылки и подтверждений для каждой из трех линий маршрутизатора *B* (к *A*, *C* и *F*, соответственно). Флаги отсылки означают, что этот пакет следует отослать по соответствующей связи. Флаги подтверждений означают, что нужно подтвердить получение этого пакета по данной линии.

			Флаги отсылки			Флаги подтверждения			
Источник	Порядковый номер	Возраст	*A*	*C*	*F*	*A*	*C*	*F*	Данные
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

Рис. 5.11. Буфер пакетов маршрутизатора B с рисунка 5.10

Как видно из рис. 5.11, пакет состояния линий от маршрутизатора *A* пришел напрямую, поэтому он должен быть отправлен маршрутизаторам *C* и *F*, а подтверждение о его получении следует направить маршрутизатору *A*, что и показывают флаговые биты. Аналогично, пакет от *F* следует переслать маршрутизаторам *A* и *C*, а *F* отослать подтверждение.

Однако ситуация с третьим пакетом, полученным от маршрутизатора *E*, отличается. Он приходит дважды, по линиям *EAB* и *EFB*. Следовательно, его нужно отослать только *C*, но подтверждения необходимо выслать и *A* и *F*, как указывают биты.

Если в то время, когда оригинал еще находится в буфере, прибывает дубликат пакета, значение битов должно быть изменено. Например, если копия состояния маршрутизатора *C* прибудет от *F*, прежде чем четвертая строка таблицы будет разослана, шесть флаговых битов примут значение 100011, и это будет означать, что следует подтвердить получение пакета от *F*, но не пересылать его *F*.

Вычисление новых маршрутов

Собрав полный комплект пакетов состояния линий, маршрутизатор может построить полный граф сети, так как он располагает данными обо всех линиях. На самом деле, каждая линия представлена даже дважды, по одному значению для каждого направления. У разных направлений может быть разная стоимость. Поэтому результаты вычисления кратчайшего пути от *A* до *B* и от *B* до *A* могут не совпадать.

Теперь для построения кратчайших путей ко всем возможным адресатам может быть локально применен алгоритм Дейкстры. Результат вычислений сообщает маршрутизатору, какое соединение следует выбрать, чтобы добраться до нужного адреса назначения. Эта информация добавляется в таблицы маршрутов, после чего возобновляется нормальная работа маршрутизатора.

В отличие от маршрутизации по вектору расстояния, маршрутизация с учетом состояния линий требует большего количества вычислений и памяти. В сети, состоящей из n маршрутизаторов, у каждого из которых k соседей, количество памяти, необходимой для хранения входной информации, пропорционально kn, что как минимум соответствует размеру таблицы маршрутизации, содержащей все адреса назначения. Кроме того, даже при использовании эффективных структур данных время, требующееся для обработки информации, растет быстрее, чем kn. В больших сетях это может составлять проблему. Тем не менее во многих практических ситуациях маршрутизация с учетом состояния линий работает вполне удовлетворительно, поскольку ее не затрагивает проблема медленной конвергенции.

Маршрутизация с учетом состояния линий широко применяется в современных сетях, поэтому следует сказать несколько слов о некоторых примерах протоколов. Многие интернет-провайдеры используют протокол маршрутизации с учетом состояния линий **IS-IS (Intermediate System to Intermediate System — связь между промежуточными системами)** (Oran, 1990). Он был разработан достаточно давно для сети DECnet и был впоследствии принят Международной организацией по стандартизации ISO для использования вместе с протоколами OSI. После этого он был модифицирован для поддержки также и других протоколов, в частности, IP. Еще один из наиболее известных протоколов маршрутизации с учетом состояния линий — **OSPF (Open Shortest Path First — открытый алгоритм предпочтительного выбора кратчайшего маршрута)**. Он был разработан Специальной комиссией интернет-разработок (IETF) через несколько лет после IS-IS и включал многие новшества, применявшиеся при создании IS-IS. К этим новшествам относятся метод саморегуляции лавинного потока обновлений информации о состоянии линий, концепция отмеченного маршрутизатора в локальной сети, а также метод вычисления и поддержки расщепления пути и умножения метрик. Соответственно, между протоколами IS-IS и OSPF нет почти никакой разницы. Наиболее существенное различие между ними заключается в том, что в IS-IS возможна одновременная поддержка нескольких протоколов сетевого уровня (например, IP, IPX и AppleTalk). OSPF не обладает таким свойством, и это оказывается преимуществом в больших многопротокольных средах. Более подробно о протоколе OSPF будет рассказано в разделе 5.6.6.

Следует также сказать несколько общих слов об алгоритмах маршрутизации. Маршрутизация с учетом состояния линий, маршрутизация по вектору расстояния и другие алгоритмы предполагают, что маршрутизаторы выполняют вычисления маршрутов. Однако неисправности оборудования или программного обеспечения могут привести к очень серьезным проблемам сети. Например, если маршрутизатор заявит о существовании линии, которой у него в действительности нет, или наоборот, забудет о существовании имеющейся у него линии, граф сети окажется неверным. Если маршрутизатор не сможет переслать пакеты или повредит их при пересылке, также возникнет проблема. Наконец, если у маршрутизатора закончится свободная память, или он ошибется в расчетах маршрутов, также возможны различные неприят-

ности. При увеличении размера сети до нескольких десятков или сотен тысяч маршрутизаторов вероятность выхода из строя одного из них перестает быть пренебрежимо малой. Все, что можно здесь сделать, — это попытаться ограничить вред, наносимый практически неизбежным выходом из строя оборудования. Эти проблемы и методы их разрешения подробно обсуждаются в (Perlman, 1988).

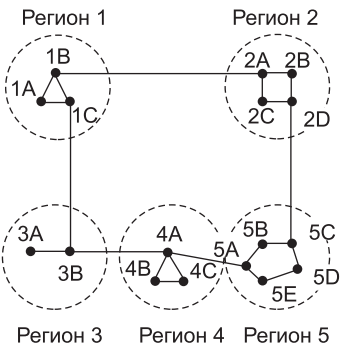
5.2.6. Иерархическая маршрутизация

Размер таблиц маршрутов, поддерживаемых маршрутизаторами, увеличивается пропорционально увеличению размеров сети. При этом требуется не только большее количество памяти для хранения этой таблицы, но и большее время центрального процессора для ее обработки. Кроме того, возрастает размер служебных пакетов, которыми обмениваются маршрутизаторы, что увеличивает нагрузку на линии. В определенный момент сеть может вырасти до таких размеров, при которых перестанет быть возможным хранение на маршрутизаторах записей обо всех остальных маршрутизаторах. Поэтому в больших сетях маршрутизация должна осуществляться иерархически, как это делается в телефонных сетях.

При использовании иерархической маршрутизации маршрутизаторы разбиваются на отдельные так называемые **регионы (regions)**. Каждый маршрутизатор знает все детали выбора маршрутов в пределах своей области, но ему ничего не известно о внутреннем строении других регионов. При объединении нескольких сетей естественно рассматривать их как отдельные регионы, при этом маршрутизаторы одной сети освобождаются от необходимости знать топологию других сетей.

В очень больших сетях двухуровневой иерархии может оказаться недостаточно. Может потребоваться группировать регионы в кластеры, кластеры в зоны, зоны в группы и т. д., пока у нас не иссякнет фантазия на названия для новых образований. В качестве примера многоуровневой иерархии рассмотрим маршрутизацию пакета, пересылаемого из университета Беркли (Berkeley), штат Калифорния в Малинди (Malindi) в Кении. Маршрутизатор в Беркли знает детали топологии в пределах Калифорнии, но трафик, направляющийся за пределы штата, он посылает маршрутизатору в Лос-Анджелесе. Маршрутизатор в Лос-Анджелесе может выбрать прямой маршрут для трафика в пределах США, но все пакеты, направляемые за рубеж, переправляет в Нью-Йорк. Нью-йоркский маршрутизатор направит трафик на маршрутизатор страны назначения, ответственный за прием трафика из-за границы. Он может располагаться, например, в Найроби. Наконец, направляясь вниз по дереву иерархии уже в пределах Кении, пакет попадет в Малинди.

На рис. 5.12 приведен количественный пример маршрутизации в двухуровневой иерархии с пятью регионами. Полная таблица маршрутизатора *1A*, как показано на рис. 5.12, *б*, состоит из 17 записей. При использовании иерархической маршрутизации, как показано на рис. 5.12, *в*, таблица, как и прежде, содержит сведения обо всех локальных маршрутизаторах, но записи обо всех остальных регионах концентрируются в пределах одного маршрутизатора, поэтому трафик во второй регион по-прежнему пойдет по линии *1B-2A*, а во все остальные регионы — по линии *1C-3B*. При иерархической маршрутизации размер таблицы маршрутов уменьшается с 17 до 7 строк. Чем крупнее выбираются регионы, тем больше экономится места в таблице.



а

Полная таблица для 1A

Назначение	Линия	Транзитные участки
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

б

Иерархическая таблица для 1A

Назначение	Линия	Транзитные участки
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

в

Рис. 5.12. Иерархическая маршрутизация

К сожалению, этот выигрыш памяти не достается бесплатно. Платой является увеличение длины пути. Например, наилучший маршрут от *1A* до *5C* проходит через регион 2, однако при использовании иерархической маршрутизации весь трафик в регион 5 направляется через регион 3, поскольку так лучше для большинства адресатов в регионе 5.

Когда единая сеть становится очень большой, возникает интересный вопрос: сколько уровней должна иметь иерархия? Для примера рассмотрим сеть с 720 маршрутизаторами. Если иерархии нет, то каждому маршрутизатору необходимо поддерживать таблицу из 720 строк. Если сеть разбить на 24 региона по 30 маршрутизаторов в каждом регионе, тогда каждому маршрутизатору потребуется 30 локальных записей плюс 23 записи об удаленных регионах, итого 53 записи. При выборе трехуровневой иерархии, состоящей из 8 кластеров по 9 регионов из 10 маршрутизаторов, каждому маршрутизатору понадобится 10 строк в таблице для локальных маршрутизаторов, 8 строк для маршрутизации в другие регионы в пределах своего кластера, плюс 7 строк для удаленных кластеров, итого 25 строк. Камоун (Kamoun) и Кляйнрок (Kleinrock) в 1979 году показали, что оптимальное количество уровней иерархии для сети, состоящей из N маршрутизаторов, равно $\ln N$. При этом потребуется $e \ln N$ записей для каждого маршрутизатора. Они также показали, что увеличение длины эффективного среднего пути, вызываемое иерархической маршрутизацией, довольно мало и обычно является приемлемым.

5.2.7. Широковещательная маршрутизация

В некоторых приложениях хостам требуется посылать сообщения на множество хостов или даже на все сразу. Можно привести такие примеры, как распространение прогнозов погоды, обновление биржевых курсов ценных бумаг, радиопрограммы в прямом эфире. Эффективнее всего отправлять соответствующие данные широковещательным способом, предоставляя возможность получить их всем заинтересованным хостам. Итак, **широковещанием (broadcasting)** называется рассылка пакетов по всем пунктам назначения одновременно. Для ее реализации применяются разнообразные методы.

Один из методов широковещательной маршрутизации не требует никаких особых способностей от сети и используется просто для того, чтобы рассылать отдельные пакеты по всем направлениям. Он не только является медленным и неэффективным с точки зрения пропускной способности, но и требует, чтобы у источника пакета был полный список всех хостов. На практике такой метод является нежелательным, хотя и отличается широким применением.

Один из более совершенных алгоритмов называется **многоадресной маршрутизацией (multidestination routing)**, при которой в каждом пакете содержится либо список адресатов, либо битовая карта, показывающая предпочитаемые хосты назначения. Когда такой пакет прибывает в маршрутизатор, последний проверяет список, содержащийся в пакете, определяя набор выходных линий, которые потребуются для дальнейшей рассылки. (Линия может потребоваться в том случае, если она входит в оптимальный путь к какому-то из адресатов списка.) Маршрутизатором создается копия пакета для каждой из используемых исходящих линий. В нее включаются только те адресаты, для доступа к которым требуется данная линия. Таким образом, весь список рассылки распределяется между исходящими линиями. После определенного числа пересылок каждый из пакетов будет содержать только один адрес назначения, как и обычный пакет. Многоадресная маршрутизация подобна использованию индивидуально адресуемых пакетов с той разницей, что в первом случае из нескольких пакетов, следующих по одному и тому же маршруту, только один «платит полную стоимость», а остальные едут бесплатно. В таком случае пропускная способность используется более эффективно. Однако этот метод все же требует начальных сведений обо всех адресах назначения, и, кроме того, чтобы понять, куда отправить один многоадресный пакет, маршрутизатор должен выполнить столько же действий, сколько и при отправке набора отдельных пакетов.

Мы уже знакомы с еще одним методом широковещательной маршрутизации: заливкой. Если заливка реализована с помощью порядковых номеров, она эффективно работает с каналами связи, поскольку маршрутизаторы используют относительно простое правило принятия решения. Хотя этот метод плохо подходит для обычных двухточечных соединений, он может быть хорошим претендентом для широковещания. Но оказывается, что ситуация может быть еще лучше, если кратчайшие пути для стандартных пакетов уже вычислены.

Идея **продвижения по встречному пути (reverse path forwarding)** замечательно проста и изящна (Dadal, Metcalfe, 1978). Когда прибывает широковещательный пакет, маршрутизатор проверяет, используется ли та связь, по которой он прибыл, для нормальной передачи пакетов *источнику широковещания*. В случае положительного от-

вета велика вероятность того, что широковещательный пакет прибыл по наилучшему маршруту и является, таким образом, первой копией, прибывшей на маршрутизатор. Тогда маршрутизатор рассылает этот пакет по всем связям, кроме той, по которой он прибыл. Однако если широковещательный пакет прибывает от того же источника по другой связи, он отвергается как вероятный дубликат.

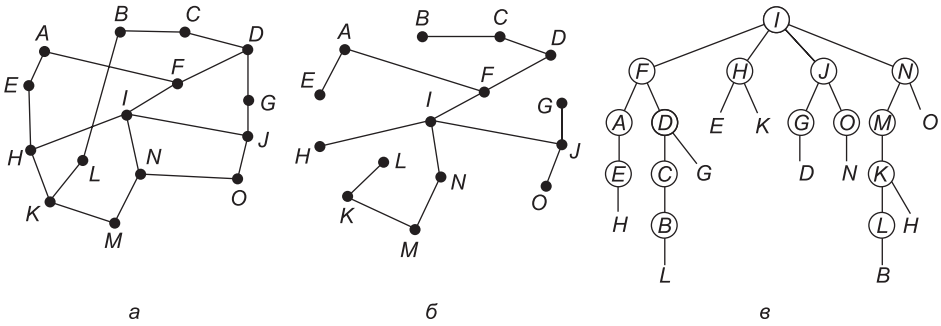


Рис. 5.13. Продвижение по встречному пути: *а* — сеть; *б* — входное дерево; *в* — дерево, построенное методом продвижения по встречному пути

Пример работы алгоритма продвижения по встречному пути показан на рис. 5.13. Слева изображена сеть, посередине — входное дерево для маршрутизатора *I* этой сети. На первом транзитном участке маршрутизатор *I* посылает пакеты маршрутизаторам *F*, *H*, *J* и *N*, являющимся вторым ярусом дерева. Все эти пакеты прибывают к ним по предпочитаемым линиям до *I* (по пути, совпадающему с входным деревом), что обозначается кружками вокруг символов на рис. 5.13, *в*. На втором этапе пересылки формируются восемь пакетов — по два каждым маршрутизатором, получившим пакет после первой пересылки. Все восемь пакетов попадают к маршрутизаторам, не получавшим ранее пакетов, а пять из них приходят по предпочитаемым линиям. Из шести пакетов, формируемых на третьем транзитном участке, только три прибывают по предпочитаемым линиям (на маршрутизаторы *C*, *E* и *K*). Остальные оказываются дубликатами. После пяти транзитных участков широковещание заканчивается с общим количеством переданных пакетов, равным 23, тогда как при использовании входного дерева потребовалось бы 4 транзитных участка и 14 пакетов.

Принципиальное преимущество метода продвижения по встречному пути заключается в его вполне приемлемой эффективности при простоте реализации. Как и при заливке, широковещательный пакет отправляется по всем каналам связи только один раз в каждом направлении, и при этом маршрутизатор должен всего лишь знать, как добраться до соответствующего адреса назначения, но не обязан помнить порядковые номера (или использовать другие механизмы, позволяющие остановить заливку) или хранить список всех адресов в пакете.

Последний алгоритм широковещательной маршрутизации выигрывает перед алгоритмом продвижения по встречному пути. Он в явном виде использует входное дерево или любое другое связующее дерево для маршрутизатора, инициировавшего широковещание. **Связующее дерево (spanning tree)** представляет собой подмножество сети, включающее в себя все маршрутизаторы, но не содержащее замкнутых путей. Если

каждый маршрутизатор знает, какие из его линий принадлежат связующему дереву, он может отправить приходящий пакет по всем линиям связующего дерева, кроме той, по которой пакет прибыл. Такой метод оптимальным образом использует пропускную способность сети, порождая минимальное количество пакетов, требующихся для выполнения работы. Так, если в примере на рис. 5.13 в качестве связующего дерева использовать входное дерево (рис. 5.13, *б*), минимальное число пакетов, необходимое для широковещания, составляет 14. Единственной проблемой этого метода является то, что каждому маршрутизатору необходимо обладать информацией о связующем дереве. Иногда такая информация доступна (например, в случае маршрутизации с учетом состояния линий все маршрутизаторы обладают полными сведениями о топологии сети и поэтому могут вычислить связующее дерево), но иногда — нет (при маршрутизации по векторам расстояний).

5.2.8. Многоадресная рассылка

Некоторые приложения (такие как многопользовательские игры или трансляции спортивных событий, передаваемые множеству получателей) отправляют пакеты группе адресов. Если такая группа не очень мала, отправка отдельного пакета на каждый адрес окажется весьма дорогостоящей операцией. С другой стороны, широковещание оказывается нерентабельным, если группа состоит, скажем, из 1000 машин в сети из миллиона узлов, причем большинство получателей не заинтересованы в данном сообщении (или, что еще хуже, явно заинтересованы, но было бы крайне нежелательно, чтобы эта информация до них дошла). Таким образом, требуется способ рассылки сообщений строго определенным группам, довольно большим по численности, но небольшим по сравнению со всей сетью.

Передача сообщения членам такой группы называется **многоадресной рассылкой (multicasting)**, а использующийся при этом алгоритм — **многоадресной маршрутизацией (multicast routing)**. Любая схема многоадресной рассылки предполагает возможность создания и удаления групп и определения списка маршрутизаторов, являющихся членами данной группы. Реализация данных задач, однако, не интересует алгоритм маршрутизации. Пока мы будем считать, что каждая группа определяется адресом рассылки и что каждый маршрутизатор знает, в какие группы он входит. Мы вернемся к вопросу принадлежности к группам, когда будем говорить о сетевом уровне Интернета в разделе 5.6. Схемы многоадресной рассылки основываются на принципах широковещательной маршрутизации, о которой мы уже говорили: пакеты, предназначенные членам группы, пересылаются по связующему дереву, и при этом ставится задача эффективного использования пропускной способности сети. Однако выбор наилучшего связующего дерева зависит от того, является ли группа плотной (когда получатели занимают большую часть всей сети) или разреженной (когда большая часть сети не принадлежит группе). В этом разделе мы рассмотрим оба случая.

Если группа является плотной, применение широковещания является неплохой идеей, поскольку в результате пакет будет успешно отправлен во все части сети. Однако минус этого алгоритма в том, что пакет получат и те маршрутизаторы, которые не являются членами группы. Решение, предложенное Дирингом и Черитоном (Deering, Cheriton, 1990), состоит в том, чтобы удалить из связующего дерева ветви,

не ведущие к членам группы. В результате получается эффективное многоадресное связующее дерево.

Для примера рассмотрим две группы, 1 и 2, в сети, изображенной на рис. 5.14, *а*. Как показано на рисунке, некоторые маршрутизаторы соединены с хостами, принадлежащими к одной или обеим группам. Связующее дерево для самого левого маршрутизатора показано на рис. 5.14, *б*. Такое дерево может быть использовано для широковещания, однако (как это видно на примере двух усеченных вариантов дерева, приведенных далее) для многоадресной рассылки оно оказывается избыточным. На рис. 5.14, *в* удалены все связи, не ведущие к хостам, являющимся членами группы 1. В результате получается многоадресное связующее дерево, по которому самый левый маршрутизатор может отправлять пакеты группе 1. Пакеты передаются исключительно по такому дереву — этот способ оказывается гораздо более экономичным, чем широковещание: новое дерево использует 7 связей вместо 10. На рис. 5.14, *г* показано усеченное многоадресное связующее дерево для группы 2. Оно использует 5 связей, что доказывает его эффективность. Следует обратить внимание на то, что для разных групп используются разные связующие деревья.

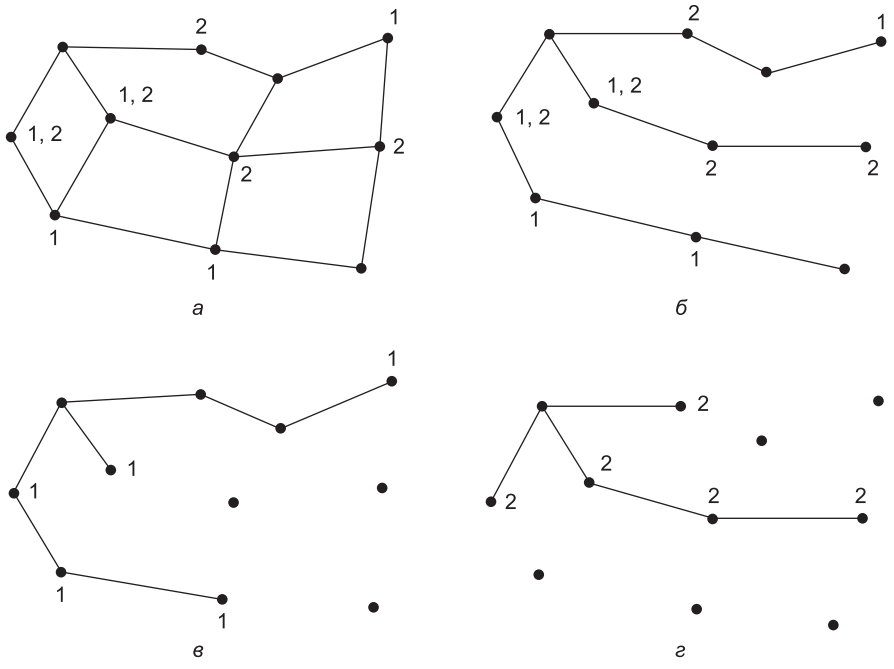


Рис. 5.14. Многоадресная рассылка: *а* — сеть; *б* — связующее дерево для самого левого маршрутизатора; *в* — многоадресное дерево для группы 1; *г* — многоадресное дерево для группы 2

Существует несколько способов усечения связующего дерева. Простейший способ может применяться при использовании маршрутизации с учетом состояния линий, когда каждому маршрутизатору известна полная топология сети, в том числе и состав групп. В таком случае каждый маршрутизатор может построить собственное усеченное связующее дерево для каждого отправителя, сначала создав обычное входное дерево,

а затем удалив из него все связи, не соединяющие входной (корневой) узел с членами данной группы. Одним из протоколов маршрутизации с учетом состояния линий, работающих по такому принципу, является **MOSPF (Multicast OSPPF — многоадресный OSPF)** (Moy, 1994).

При маршрутизации по векторам расстояний может быть применена другая стратегия усечения дерева. Для многоадресной рассылки здесь применяется алгоритм продвижения по встречному пути. Когда многоадресное сообщение получает маршрутизатор, у которого нет хостов, входящих в группу, и линий связи с другими маршрутизаторами, принимающими сообщения для данной группы, он может ответить сообщением PRUNE (отсечь), информируя соседа, отправившего сообщение, о том, что сообщения для данной группы ему больше посылать не нужно. Такой же ответ может дать маршрутизатор, у которого нет хостов, входящих в группу, если он получил сообщение PRUNE по всем линиям, по которым он осуществил многоадресную рассылку. В результате связующее дерево постепенно рекурсивно усекается. Примером протокола многоадресной маршрутизации, работающего по такому принципу, является **DVRMP (Distance Vector Multicast Routing Protocol — протокол многоадресной маршрутизации по вектору расстояний)** (Waitzman и др., 1988).

Усечение позволяет строить эффективные связующие деревья, состоящие только из тех связей, которые необходимы для соединения с членами группы. Недостаток данного метода заключается в том, что он требует от маршрутизаторов выполнения большого количества операций, особенно в больших сетях. Предположим, что в сети есть n групп, каждая из которых в среднем состоит из m членов. На каждом маршрутизаторе и для каждой группы должно храниться m усеченных связующих деревьев, то есть mn деревьев для всей сети. К примеру, на рис. 5.14, *в* изображено связующее дерево, по которому самый левый маршрутизатор отправляет пакеты группе 1. Связующее дерево, по которому самый правый маршрутизатор отправляет пакеты группе 1 (оно не показано на рисунке), будет выглядеть по-другому, поскольку пакеты будут передаваться непосредственно членам группы, а не через узлы в левой части графа. А это значит, что выбор направления, в котором маршрутизаторы должны передавать пакеты для группы 1, зависит от того, какой узел является отправителем. При большом количестве групп и отправителей для хранения всех деревьев потребуется много памяти.

Альтернативный метод при построении отдельного связующего дерева для группы использует **деревья с корнем в ядре (core-based trees)**(Ballardie и др., 1993). Согласно этому методу, все маршрутизаторы выбирают общий корень (также называемый **ядром — core** или **точкой встречи — rendezvouz point**), а для построения дерева все члены группы отправляют в этот корень специальный пакет. Конечное дерево формируется из путей, пройденных этими пакетами. На рис. 5.15, *а* показано такое дерево с основанием в ядре, построенное для группы 1. Для пересылки пакета этой группе отправитель передает сообщение ядру, откуда оно уже рассылается по дереву. Пример такой работы алгоритма продемонстрирован на рис. 5.15, *б* для отправителя, расположенного в правой части сети. Однако производительность этого метода может быть улучшена: дело в том, что для многоадресной рассылки не обязательно, чтобы пакеты, предназначенные для группы, приходили в ядро. Как только пакет достигает дерева, он может быть передан как в направлении корня, так и в любом другом направлении. Так алгоритм работает для отправителя, расположенного вверху рис. 5.15, *б*.

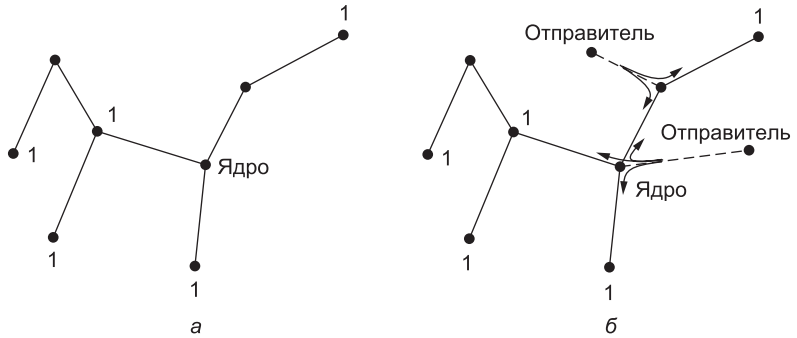


Рис. 5.15. Дерево с основанием в сердцевине для группы 1 (*а*); рассылка для группы 1 (*б*)

Использование общего дерева не является оптимальным для всех источников. В примере на рис. 5.15, *б* пакет, передаваемый от отправителя, расположенного в правой части сети, правому верхнему члену группы не напрямую, а через ядро, требует трех переходов. Степень неэффективности зависит от взаимного расположения ядра и отправителей, но чаще всего разумно располагать ядро посередине между отправителями. Если отправитель всего один (как, например, при трансляции видеозаписи членам группы), оптимальной идеей является использовать в качестве ядра самого отправителя.

Следует также отметить, что использование общего дерева позволяет существенно снизить затраты на хранение информации, а также уменьшить число отправленных сообщений и объем вычислений. Для каждой группы маршрутизатор должен хранить не *m* деревьев, а лишь одно. Кроме того, маршрутизаторы, не являющиеся частью дерева, не участвуют в передаче сообщений группе. Поэтому алгоритмы на основе общих деревьев (в частности, деревьев с основанием в ядре) используются при широковещании для разреженных групп в сети Интернет, являясь частью популярных протоколов, таких как **PIM (Protocol Independent Multicast — многоадресная рассылка, не зависящая от протокола)** (Fenner и др., 2006).

5.2.9. Произвольная маршрутизация

До сих пор мы рассматривали такие модели предоставления информации, в которых источник отправляет сообщение на один адрес (**одноадресная рассылка — unicast**), на все адреса (широковещание) или группе адресов (многоадресная рассылка). Иногда используется еще одна модель под названием **произвольная рассылка (anycast)**. При такой рассылке пакет отправляется ближайшему члену группы (Partridge и др., 1993). Методы нахождения соответствующих путей называются **произвольной маршрутизацией (anycast routing)**.

Зачем нам может понадобиться произвольная рассылка? Иногда узлы предоставляют услугу (например, сообщают время суток или передают контент), для которой важно одно: чтобы информация была правильной; при этом не имеет значения, какой узел ее предоставил — с этой задачей справился бы любой узел. Пример использования свободной рассылки в сети Интернет — DNS, о которой мы поговорим в главе 7.

К счастью, нам не придется придумывать новые алгоритмы для произвольной маршрутизации: стандартные методы — маршрутизация по вектору расстояния и маршрутизация с учетом состояния линий — позволяют создавать маршруты для произвольной рассылки. Предположим, что нам нужно передать данные членам группы 1. Вместо различных адресов все члены группы получат одинаковый адрес — «1». Алгоритм маршрутизации по вектору расстояния распределит векторы обычным способом, и узлы выберут кратчайший путь к адресу 1. В результате узлы отправят данные на ближайшее устройство с адресом 1. Соответствующие маршруты показаны на рис. 5.16 *а*. Этот метод работает успешно потому, что протокол маршрутизации не знает о существовании нескольких устройств с адресом 1, а значит, он считает их всех одним узлом, как показано в топологии на рис. 5.16, *б*.

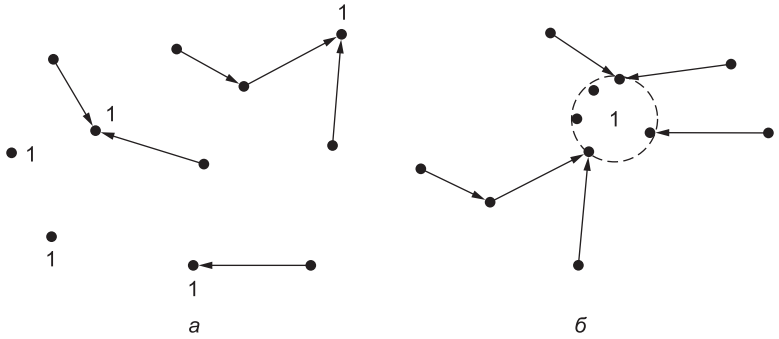


Рис. 5.16. Произвольная маршрутизация: *а* — маршруты для свободной рассылки; *б* — топология с точки зрения протокола маршрутизации

Такой метод будет работать и для маршрутизации с учетом состояния линий. Здесь, правда, следует добавить, что протокол маршрутизации не обязательно должен находить кажущиеся кратчайшими пути, проходящие чрез узел 1. Это привело бы к «прыжку через гиперпространство», потому что экземпляры узла 1 на самом деле расположены в различных частях сети. Однако сейчас протоколы маршрутизации с учетом состояния линий различают маршрутизаторы и хосты. Мы не говорили об этом раньше, поскольку в этом не было явной необходимости.

5.2.10. Алгоритмы маршрутизации для мобильных хостов

Миллионы людей пользуются компьютерами на ходу. Сюда относятся ситуации фактического передвижения, такие как использование беспроводных устройств в движущемся автомобиле, а также ситуации мобильного использования портативных компьютеров в нескольких разных местах. Термин **мобильные хосты (mobile hosts)** мы будем использовать для обозначения и тех, и других ситуаций, противопоставляя их стационарным, никогда не перемещающимся хостам. В последнее время люди все больше хотят оставаться в сети независимо от того, в какой части земного шара они находятся, и делать это так же легко, как дома. Такие мобильные хосты вызывают до-

полнительную сложность: чтобы направить пакет к мобильному хосту, сеть должна сначала найти его.

Здесь мы будем рассматривать модель мира, в которой у всех хостов есть постоянное **домашнее местоположение (home location)**, которое никогда не меняется. У каждого хоста также есть постоянный домашний адрес, которым можно воспользоваться для определения домашнего местоположения, аналогично тому, как телефонный номер 1-212-5551212 обозначает США (страна с кодом 1) и Манхеттен (212). Целью маршрутизации в системах с мобильными хостами является обеспечение возможности передачи пакетов мобильным хостам с помощью их постоянных домашних адресов. При этом пакеты должны эффективно достигать хостов, независимо от их расположения. Самое сложное здесь, конечно, — найти хост.

Следует сказать несколько слов об этой модели. Существует такой вариант: можно заново вычислять маршруты каждый раз при смене местоположения хоста или при изменении топологии. Тогда можно было бы просто воспользоваться алгоритмами маршрутизации, описанными ранее в этом разделе. Однако, учитывая, насколько быстро увеличивается количество хостов, становится ясно: в результате вся сеть будет бесконечно занята вычислением новых маршрутов. Использование домашних адресов значительно упрощает задачу.

Еще один вариант — обеспечить мобильность над сетевым уровнем, как это, по сути, сейчас и происходит с переносными компьютерами. При изменении местоположения они получают новые сетевые адреса; при этом старый и новый адреса никак не связаны друг с другом — сеть даже не знает о том, что они принадлежат одному и тому же компьютеру. В этой модели с помощью переносного компьютера можно просматривать содержимое Сети, однако другие хосты не могут отправлять пакеты на данный компьютер (например, осуществлять входящий звонок) без выполнения процедуры адресации высокого уровня (например, повторного входа в Skype при смене местоположения). Более того, связь не будет поддерживаться, если хост движется: в таком случае придется устанавливать новые соединения. С этими проблемами может справиться мобильность на сетевом уровне.

В основе мобильной маршрутизации в сети Интернет и сетях сотовой связи лежит следующая идея: мобильный хост должен сообщать о том, где он находится, хосту, имеющему домашнее местоположение. Этот хост, выполняющий операции от имени мобильного хоста, называется **внутренним агентом (home agent)**. Как только он узнает текущее местоположение мобильного хоста, он может перенаправлять пакеты, которые должны быть ему доставлены.

Рисунок 5.17 наглядно демонстрирует работу мобильной маршрутизации. Отправитель из северо-западного города Сиэтла хочет отправить пакет хосту, который обычно находится по другую сторону США, в Нью-Йорке. Нас интересует ситуация, при которой мобильный хост располагается не дома. Временно он находится в Сан-Диего. Для работы в сети мобильный хост в Сан-Диего должен получить локальный сетевой адрес. Это обычная процедура; о том, как она работает для сети Интернет, мы поговорим позже. Такой локальный адрес называется **адресом для передачи (care of address)**. Как только мобильный хост узнает этот адрес, он может сообщить внутреннему агенту свое местонахождение. Для этого он отправляет внутреннему агенту регистрационное сообщение (этап 1), содержащее адрес для передачи. На рис. 5.17 это

сообщение выделено пунктирной линией, чтобы показать, что оно является управляющим, а не информационным.

Далее отправитель посылает пакет с данными на мобильный хост с помощью постоянного адреса (этап 2). Сеть передает этот пакет на домашнее местоположение хоста, так как именно ему принадлежит этот домашний адрес. В Нью-Йорке внутренний агент перехватывает этот пакет, поскольку мобильный хост находится далеко от дома. Затем он формирует пакет с новым заголовком (**инкапсулирует** — **encapsulates**, или, как иногда говорят, «заворачивает» — **wraps**) **и отправляет все это на адрес для передачи** (этап 3). Такой прием называется **туннелированием** (**tunneling**). Поскольку он имеет важное значение в работе сети Интернет, далее мы поговорим о нем более подробно.

Когда этот пакет прибывает на адрес для передачи, мобильный хост разворачивает пакет и считывает данные, полученные от отправителя. Далее мобильный хост напрямую посылает отправителю ответный пакет (этап 4). Весь этот путь называется **треугольной маршрутизацией** (**triangle routing**); он может быть непрямым, если удаленное местоположение расположено на большом расстоянии от домашнего местоположения. На этапе 4 отправитель может узнать текущий адрес для передачи. Последующие пакеты могут направляться напрямую на мобильный хост, передаваясь по туннелю на адрес для передачи (этап 5), полностью минуя домашний адрес мобильного пользователя. Если соединение будет прервано из-за перемещения мобильного устройства, для связи с ним всегда можно использовать домашний адрес.

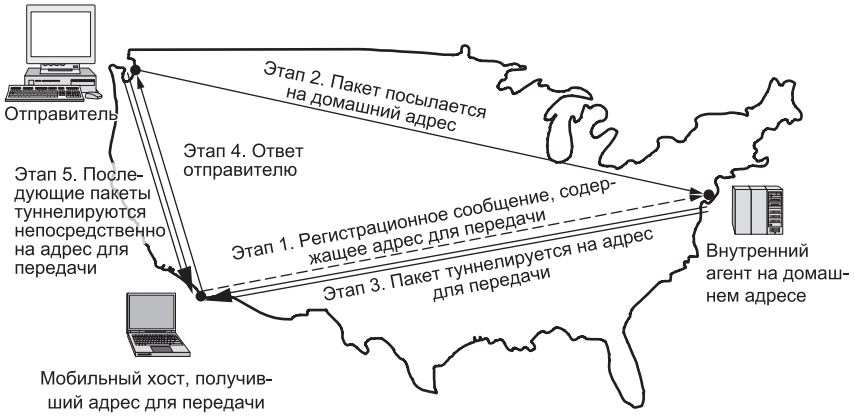


Рис. 5.17. Маршрутизация пакетов мобильным хостам

Мы не уделили внимания еще одному важному аспекту: безопасности. Как правило, когда хост или маршрутизатор получает сообщение вида «Начиная с этого момента, пожалуйста, пересылайте мне всю почту, адресуемую Стефани», у него могут возникнуть вопросы — например, с кем он разговаривал, соглашаться или нет на данное предложение. Поэтому, для того чтобы достоверность сообщений можно было проверить с помощью криптографических протоколов (о которых мы поговорим в главе 8), в сообщения добавляется информация о безопасности.

Существует много различных схем мобильной маршрутизации. Алгоритм, представленный выше, основан на мобильности IPv6 — той форме мобильности, которая

используется в сети Интернет (Джонсон и др., 2004), а также в IP-сетях сотовой связи, таких как UMTS. Для простоты в нашем примере отправитель рассматривался как стационарный узел, однако алгоритм позволяет обоим узлам быть мобильными хостами. Или же хост может быть частью мобильной сети, например сети в самолете. Такое расширение базовой схемы не требует дополнительных действий со стороны хостов (Devarapalli и др., 2005).

В некоторых схемах используется внешний (то есть удаленный) агент — это то же самое, что и внутренний агент, только с внешним расположением; аналогом внешнего агента в сетях сотовой связи является VLR (Visitor Location Register, гостевой регистр местоположения). Однако в большинстве более новых схем внешний агент не требуется; мобильные хосты сами являются своими внешними агентами. В любом случае, знание временного местоположения мобильного хоста ограничивается небольшим числом хостов (например, мобильное устройство, внутренний агент и отправители), поэтому маршрутизаторам крупной сети не придется повторно вычислять маршруты.

Более подробную информацию о мобильной маршрутизации можно найти в работах Perkins (1998, 2002) и Snoeren и Balakrishnan (2000).

5.2.11. Маршрутизация в произвольных сетях

Итак, мы рассмотрели, как производится маршрутизация в случаях, когда станции мобильны, а маршрутизаторы стационарны. Еще более занимательная ситуация возникает тогда, когда мобильны сами маршрутизаторы. Это может понадобиться, например, аварийным службам на месте землетрясения, военной технике на поле боя, морскому флоту, находящемуся в море, или группе людей с портативными компьютерами при отсутствии сети 802.11.

Во всех подобных случаях каждый узел использует беспроводное соединение и выступает одновременно в качестве маршрутизатора и хоста. Сети, состоящие из узлов, волею судеб оказавшихся недалеко друг от друга, называются **произвольными сетями (ad hoc networks)** или **мобильными произвольными сетями (MANET — Mobile Ad hoc NETworks)**. Давайте их вкратце рассмотрим. Более подробную информацию можно найти в книге (Perkins, 2001).

Основное отличие произвольных сетей от обычных проводных сетей состоит в том, что топология неожиданно исключается из рассмотрения. Узлы могут легко появляться в системе и так же легко из нее исчезать, появляясь в каком-то другом месте. В обычных сетях путь от маршрутизатора к какому-либо адресату продолжает оставаться реализуемым до тех пор, пока не произойдет какой-нибудь сбой (что, как мы надеемся, случается редко). В произвольных сетях топология постоянно меняется, а с ней меняется и предпочтительность (и даже реализуемость) путей. Причем это происходит спонтанно, безо всяких предупреждений. Надо ли говорить о том, что в таких условиях маршрутизация будет более трудной задачей, чем в стационарных сетях.

Известно множество алгоритмов выбора маршрута для произвольных сетей. Однако поскольку по сравнению с мобильными сетями произвольные сети пока не так широко используются на практике, не ясно, какие из этих протоколов наиболее удобны. Для примера рассмотрим один из наиболее популярных — алгоритм **AODV (Ad hoc On-demand Distance Vector — маршрутизация по требованию в произвольных**

сетях на основе вектора расстояний). Об этом можно прочитать у (Perkins и Royer, 1999). AODV является родственником метода векторов расстояний, адаптированным для работы в мобильной среде, где узлы обычно имеют ограниченную пропускную способность и время работы от автономных элементов питания. Давайте посмотрим, как этот алгоритм вычисляет маршруты и следит за их изменением.

Построение маршрута

В AODV маршруты к адресу назначения вычисляются «по требованию», то есть только тогда, когда кто-то хочет отправить пакет на этот адрес. Это позволяет избежать выполнения лишней работы: если топология сети изменится, а маршрут еще не использовался, маршрут не придется вычислять дважды. Топология произвольной сети в любой момент может быть описана с помощью графа соединенных друг с другом узлов. Два узла считаются соединенными (то есть между ними проведена дуга), если они могут связываться напрямую посредством радио. Для наших целей вполне подойдет простая, но адекватная модель, в которой каждый узел может связаться со всеми узлами, находящимися в пределах его зоны охвата. На практике сети устроены более сложным образом: соединению могут препятствовать здания, холмы и т. д.; в сети также возможны ситуации, когда узел *A* соединен с *B*, но *B* не соединен с *A*, поскольку у одного из них может быть более мощный передатчик, чем у другого. Однако для простоты мы будем считать, что все соединения симметричны.

Для описания алгоритма рассмотрим недавно сформированную произвольную сеть, изображенную на рис. 5.18. Предположим, что процессу, запущенному на узле *A*, необходимо отправить пакет на узел *I*. Алгоритм AODV на каждом узле ведет таблицу векторов расстояния, доступ к которой осуществляется с помощью поля адреса. Таблица содержит информацию об адресате, в том числе адрес ближайшего соседа, которому необходимо переслать пакет, чтобы он мог достичь пункта назначения. Сначала *A* просматривает эту таблицу и не находит записи для *I*. Значит, нужно найти маршрут, ведущий к этому узлу. Итак, алгоритм начинает заниматься поисками маршрутов только тогда, когда они реально требуются. Это и делает его алгоритмом «по требованию».

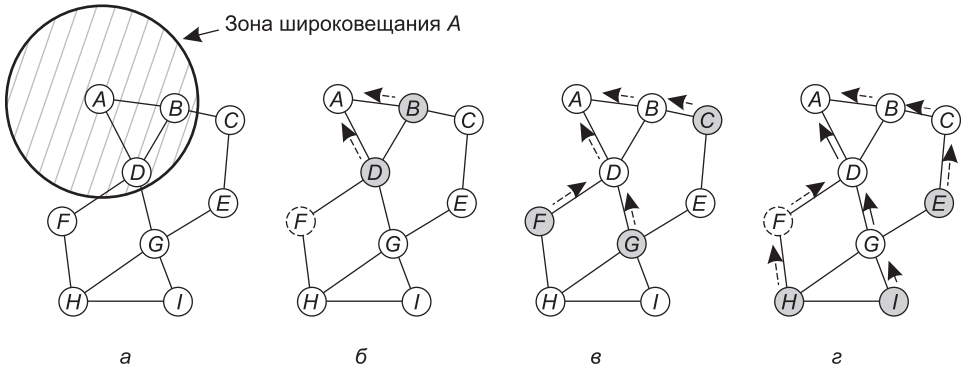


Рис. 5.18. Произвольная сеть: *а* — зона широковещания *A*; *б* — состояние после получения его узлами *B* и *D*; *в* — состояние после получения его узлами *C*, *F* и *G*; *г* — состояние после получения его узлами *E*, *H* и *I*. Затененными кружочками обозначены новые получатели. Пунктиром показаны возможные обратные маршруты. Сплошными линиями показан построенный маршрут

Для поиска *I* узел *A* генерирует пакет запроса маршрута ROUTE REQUEST и распространяет его по сети методом заливки (см. раздел 5.2.3). На рис. 5.18, *а* показано, что этот пакет достигает узлов *B* и *D*. Далее каждый узел повторно передает запрос, который в результате достигает узлов *F*, *G* и *C* (рис. 5.18, *в*) и узлов *H*, *E* и *I* (рис. 5.18, *г*). Чтобы избавиться от лишних копий пакета при заливке, используется порядковый номер. Так, например, *D* не принимает передачу от *B* (рис. 5.18, *в*), так как он уже передал запрос. Наконец запрос достигает узла *I*, который создает пакет ROUTE REPLY. Этот пакет пересылается отправителю по тому же пути, по которому он пришел. Чтобы это было возможно, каждый узел должен помнить, какой узел передал ему запрос. Информация об обратном маршруте, хранящаяся в памяти узлов, показана на рис. 5.18, *б*–*г* стрелками. При передаче запроса каждый узел также увеличивает значение счетчика транзитных участков. Это позволяет понять, насколько далеко от адреса назначения расположен узел. Ответы сообщают каждому отдельному узлу, какому из соседей следует передавать пакет, предназначенный для данного адреса: тому узлу, от которого пришел ответ. Во время обработки ответа узлы *G* и *D* записывают сведения о наилучшем маршруте в таблицы маршрутизации. Когда ответ достигает узла *A*, появляется новый маршрут — *ADGI*.

В больших сетях алгоритмом генерируется много широковещательных пакетов даже для адресатов, расположенных достаточно близко друг к другу. Чтобы уменьшить издержки, область широковещания может быть ограничена полем *Время жизни* (*Time to live*) IP-пакета. Это поле устанавливается отправителем и декрементируется при каждой пересылке. Когда его значение становится равным 0, пакет отвергается, а не распространяется дальше. При этом процесс поиска пути немного изменяется. Для обнаружения адресата отправитель рассылает пакет запроса маршрута со *Временем жизни*, равным 1. Если в течение разумного времени ответ не приходит, посылается еще один запрос со *Временем жизни*, равным 2. И так далее. Таким образом, поиск, начавшийся в какой-то локальной области, все больше расширяет свой охват.

Обслуживание маршрута

Поскольку узлы могут перемещаться и выключаться, топология сети может изменяться совершенно спонтанно. Например, если на рис. 5.18 узел *G* выключится, *A* не поймет, что путь к *I* (*ADGI*) больше не может быть реализован. Алгоритму нужно как-то с этим бороться. Периодически все узлы рассылают сообщение приветствия *Hello*. Ожидается, что все узлы, будучи истинными джентльменами, ответят на него. Если ответ не приходит, значит, сосед вышел из зоны действия или вышел из строя и больше не связан с данным узлом. Аналогичным образом, если он пытается послать пакет соседу, который не отвечает, он узнает, что связь с ним недоступна.

Эта информация используется для удаления нерабочих путей. Для каждого из возможных адресатов каждый узел *N* хранит историю о том, какие активные соседи снабжали узел пакетами для данных адресатов в течение последних ΔT секунд.

Когда какой-либо из соседей узла N становится недоступным, узел N проверяет свою таблицу маршрутизации — ведь теперь нужно понять, к каким адресатам лежал путь через ушедший узел. Всем оставшимся активным соседям сообщается, что такие

пути больше нельзя использовать и их следует удалить из таблиц маршрутизации. В нашем примере D удаляет из своей таблицы маршрутизации записи для G и I и сообщает об этом A, который в свою очередь удаляет I. В общем случае активные соседи передают эти новости своим активным соседям и т. д., пока все пути, зависевшие от ушедшего узла, не будут удалены из всех таблиц.

Теперь, когда неправильные маршруты удалены из сети, отправители могут вычислить новые действующие маршруты в соответствии с методом, описанным выше. Однако здесь есть одна сложность. Если вы помните, при изменении топологии сети протоколы векторов расстояний сталкиваются с проблемами медленной конвергенции и счета до бесконечности, в результате чего они могут перепутать старые неправильные маршруты и новые действующие маршруты.

Чтобы обеспечить быструю конвергенцию, маршрутам присваиваются порядковые номера, контролируемые получателем. Порядковый номер получателя — своего рода логические часы. Получатель увеличивает этот номер каждый раз при отправке нового пакета `ROUTE REPLY`. Чтобы запросить новый маршрут, отправитель добавляет в пакет `ROUTE REQUEST` порядковый номер получателя для последнего использовавшегося маршрута: это может быть или порядковый номер только что удаленного маршрута, или 0, то есть исходное значение. Запрос будет передаваться широковещательным способом до тех пор, пока не будет найден маршрут с большим порядковым номером. Промежуточные узлы сохраняют маршруты, имеющие больший порядковый номер или меньшее число промежуточных узлов для текущего порядкового номера.

Подобно тому, как это происходит в протоколе «по требованию», узлы хранят информацию только о тех маршрутах, которые используются в настоящий момент. Остальные маршруты хранятся лишь в течение определенного времени, после чего удаляются. Вычисление и хранение только действующих в настоящий момент маршрутов позволяет более эффективно использовать полосу пропускания и увеличивает время работы от элементов питания (в отличие от стандартного протокола векторов расстояний, который время от времени рассылает обновления).

Пока мы рассмотрели только один маршрут: от *A* до *I*. Для большей экономии ресурсов построение и обслуживание пересекающихся маршрутов производится совместно. К примеру, если *B* тоже захочет отправлять пакеты в *I*, он выполнит вычисление маршрута. Но в таком случае запрос сначала достигнет узла *D*, который уже знает о существовании маршрута, ведущего к *I*. И тогда узел *D* может отправить *B* ответ, содержащий нужный маршрут, тем самым избежав лишних вычислений.

Существует множество других схем маршрутизации в произвольных сетях. Еще один известный алгоритм «по требованию» называется **DSR (Dynamic Source Routing — динамическая маршрутизация от источника)** (Johnson и др., 2001). Другая стратегия, основанная на географии, используется в **GPSR (Greedy Perimeter Stateless Routing — «жадный» протокол маршрутизации по периметру без учета состояний)**. Если все узлы знают свое географическое местоположение, доставка пакета получателю может происходить без вычисления маршрута: пакет можно каждый раз отправлять в правильном направлении, возвращаясь назад в случае, если иначе путь приведет в тупик. Выбор протокола зависит в первую очередь от характеристик данной произвольной сети.

5.3. Алгоритмы борьбы с перегрузкой

Когда количество пакетов, передаваемых одновременно по сети (или ее части), превышает некий пороговый уровень, производительность сети начинает снижаться. Такая ситуация называется **перегрузкой (congestion)**. За борьбу с перегрузкой отвечают сетевой и транспортный уровни. Поскольку перегрузка происходит в сети, именно сетевой уровень непосредственно с ней сталкивается, и именно он должен в конечном итоге решить, что делать с лишними пакетами. Однако наиболее эффективный метод борьбы с перегрузкой — снижение нагрузки на сеть со стороны транспортного уровня. В таком случае сетевой и транспортный уровни должны работать вместе. В этой главе мы рассмотрим те аспекты перегрузки, которые относятся к сетевому уровню. Но чтобы картина была полной, мы обратимся к транспортным аспектам в главе 6.

На рис. 5.19 показано, как начинается перегрузка. Когда число пакетов, посылаемых хостами в сеть, не превышает ее пропускной способности, число доставленных пакетов пропорционально числу отправленных. Если отправить вдвое больше пакетов, вдвое больше пакетов будет и получено. Однако, когда нагрузка на сеть приближается к пропускной способности, большие объемы трафика постепенно заполняют буферы маршрутизаторов, и в результате некоторые пакеты теряются. Эти потерянные пакеты расходуют часть пропускной способности, поэтому число доставленных пакетов оказывается ниже идеальной кривой. Это означает, что сеть перегружена.

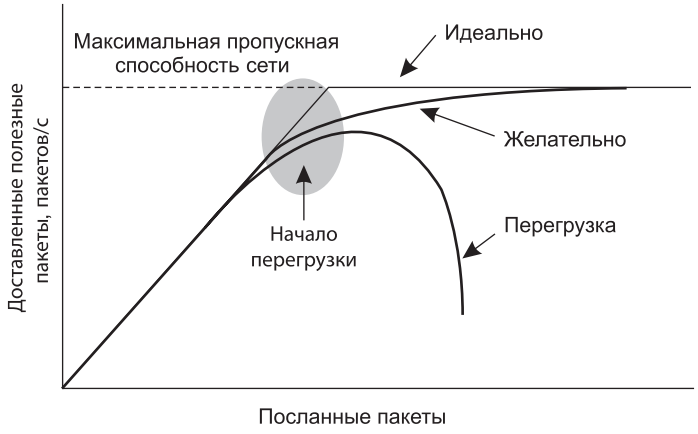


Рис. 5.19. При слишком высоком уровне трафика начинается перегрузка, и производительность сети резко снижается

Если сеть устроена не идеально, в ней может произойти **затор (congestion collapse)**, при котором производительность падает с ростом нагрузки на сеть (если нагрузка превышает пропускную способность). Это может произойти, если пакеты настолько задерживаются в сети, что в момент доставки они уже не нужны. Например, на ранних этапах работы сети Интернет время, которое пакет проводил в очереди, ожидая отправки непереданных пакетов (при скорости 56 Кбит/с), зачастую превышало то время, которое пакет мог находиться в сети. В результате его приходилось удалять. Еще один неприятный сценарий — ситуация, когда отправители повторно передают

пакеты, которые сильно задерживаются, полагая, что они утеряны. В таком случае пропускная способность сети используется неэффективно, поскольку по сети передаются копии одного и того же пакета. Чтобы наглядно продемонстрировать влияние этих факторов на производительность, мы отобразили на оси *y* (см. рис. 5.19) **полезную нагрузку (goodput)**, то есть скорость, с которой по сети передаются *полезные* пакеты.

В идеале сеть должна быть устроена так, чтобы перегрузки происходили в ней как можно реже и чтобы в случае перегрузок в ней не возникало заторов. К сожалению, перегрузок не всегда можно избежать. Если вдруг потоки пакетов начинают прибывать на маршрутизатор сразу по трем или четырем входным линиям и всем им нужна одна и та же выходная линия, то образуется очередь. Когда у маршрутизатора закончится свободная память для буферирования все прибывающих пакетов, их негде будет сохранять, и они начнут теряться. Увеличение объема памяти маршрутизаторов может в какой-то степени помочь, но Нэгл (Nagle) в 1987 году показал, что даже если у маршрутизаторов будет бесконечное количество памяти, ситуация с перегрузкой не улучшится, а, наоборот, ухудшится. Причина в том, что к тому времени, когда пакеты доберутся до начала очереди, они уже запоздают настолько, что источником будут высланы их дубликаты. В результате ситуация не улучшится, а ухудшится: в сети возникнет затор.

Линии с низкой пропускной способностью и маршрутизаторы, для которых скорость обработки пакетов ниже пропускной способности линии, также могут стать причиной перегрузки. В таком случае проблему можно решить, перенаправив трафик в обход «узкого места» на другие участки сети. С другой стороны, это приведет к перегрузке всех областей сети. В этой ситуации придется либо уменьшить нагрузку, либо разработать более быструю сеть.

Необходимо пояснить, в чем состоит разница между борьбой с перегрузкой и управлением потоком. Предотвращение перегрузки гарантирует, что сеть справится с предлагаемым ей трафиком. Это глобальный вопрос, включающий поведение всех хостов и маршрутизаторов. Управление потоком, напротив, относится к трафику между двумя конкретными станциями — отправителем и получателем. Задача управления потоком состоит в согласовании скорости передачи отправителя со скоростью, с которой получатель способен принимать поток пакетов.

Чтобы разница между этими двумя проблемами стала яснее, представьте себе сеть, состоящую из оптоволоконных линий с пропускной способностью в 100 Гбит/с, по которой суперкомпьютер пытается передать большой файл персональному компьютеру, способному принимать данные со скоростью 1 Гбит/с. Хотя перегрузки сети в данной ситуации не наблюдается, алгоритм управления потоком довольно часто заставляет суперкомпьютер приостанавливать передачу, чтобы персональный компьютер мог успевать принимать файл.

А вот другой пример. Рассмотрим сеть, состоящую из 1000 больших компьютеров, соединенных линиями с пропускной способностью в 1 Мбит/с. Одна половина компьютеров пытается передавать файлы другой половине со скоростью 100 Кбит/с. Здесь проблема заключается уже не в том, что медленные получатели не успевают принимать данные, посылаемые им быстрыми отправителями, а просто в неспособности сети пропустить весь предлагаемый трафик.

Причина, по которой управление потоком и борьбу с перегрузкой часто путают, заключается в том, что лучшее решение обеих проблем — добиться того, чтобы хост работал медленнее. Таким образом, хост может получить просьбу замедлить передачу в двух случаях: когда с передаваемым потоком не справляется получатель или когда с ним не справляется вся сеть. Мы еще будем рассматривать этот вопрос в главе 6.

Мы начнем изучение алгоритмов борьбы с перегрузкой с рассмотрения подходов, которые могут применяться в рамках различных временных интервалов. Затем мы познакомимся с методами предотвращения перегрузки, а также с различными алгоритмами борьбы с последствиями перегрузки.

5.3.1. Подходы к борьбе с перегрузкой

Наличие перегрузки означает, что нагрузка на сеть (временно) превышает возможности (сетевых) ресурсов. Соответственно, существует два возможных решения: увеличить ресурсы или снизить нагрузку. Как показано на рис. 5.20, эти решения обычно используют разные временные шкалы в зависимости от того, что требуется: предотвратить перегрузку или справиться с ней, если ее не удалось избежать.

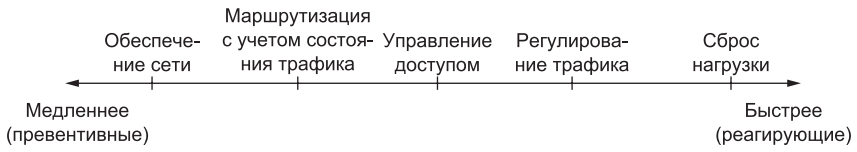


Рис. 5.20. Временные шкалы подходов к борьбе с перегрузкой

Самый простой способ избежать перегрузки — построить такую сеть, которая лучше всего соответствует передаваемому по ней трафику. Если часть пути, по которому обычно пересылаются большие объемы данных, обладает низкой пропускной способностью, возникновение перегрузки очень вероятно. Иногда при возникновении перегрузки возможно динамическое добавление ресурсов, например подключение свободных маршрутизаторов или резервных линий (обеспечивающих отказоустойчивость системы), или же приобретение дополнительной пропускной способности на свободном рынке. Как правило, наиболее загруженные связи и маршрутизаторы обновляются при первой возможности. Этот процесс называется **обеспечением (provisioning)** и использует временную шкалу месяцев, основываясь на долгосрочных данных о динамике трафика.

Чтобы максимально эффективно использовать пропускную способность сети, маршруты могут строиться в соответствии со специальными схемами трафика, которые меняются в течение суток по мере того, как пользователи просыпаются и ложатся спать в различных временных зонах. Можно, например, отвести трафик от загруженных линий, изменив весовые коэффициенты кратчайшего пути. Некоторые радиостанции используют вертолеты, чтобы получить сведения о перегрузках на дорогах и передать их радиослушателям, которые благодаря этому смогут объехать «горячие точки». Это называется **маршрутизацией с учетом состояния трафика (traffic-aware routing)**. В таком случае полезно также разделять трафик, направляя его по разным линиям.

Но иногда увеличение пропускной способности оказывается невозможным. Тогда единственным средством борьбы с перегрузкой является снижение нагрузки. В сети виртуальных каналов новые соединения могут быть отклонены, если они приведут к перегрузке сети. Это называется **управлением доступом (admission control)**.

Возможен и более сложный вариант: когда перегрузка неизбежна, сеть может послать сообщение обратной связи тому отправителю, чей трафик вызывает проблему. Сеть может попросить отправителя уменьшить трафик или же сделать это сама.

Этот подход **(регулирование трафика — traffic throttling)** требует ответа на два вопроса: как обнаружить зарождающуюся перегрузку и как сообщить об этом отправителю, который должен замедлить трафик. Решить первую проблему можно, если маршрутизаторы будут следить за средней нагрузкой на сеть, временем ожидания в очереди и числом утерянных пакетов. В любом случае увеличение значений параметров сигнализирует о приближении к перегрузке.

Для решения второй проблемы необходимо, чтобы маршрутизаторы входили в петлю обратной связи с отправителями. Чтобы данная схема работала, необходимо тщательно настроить временные параметры. Если каждый раз, когда два пакета приходят одновременно, какой-нибудь нервный маршрутизатор будет кричать «Стоп!», а простояв без работы 20 мкс, он же будет давать команду «Давай!», система будет находиться в состоянии постоянных незатухающих колебаний. С другой стороны, если маршрутизатор для большей надежности станет ждать 30 минут, прежде чем что-либо сообщить, то механизм борьбы с перегрузкой будет реагировать слишком медленно, чтобы приносить вообще какую-либо пользу. Своевременная доставка сообщений обратной связи — не такая уж простая задача. Кроме того, необходимо добиться того, чтобы маршрутизаторы отправляли больше сообщений, если сеть уже перегружена.

Наконец, если все остальные методы не работают, сеть вынуждена удалить пакеты, которые она не может доставить. Для этого используется общий термин **сброс нагрузки (load shedding)**. Если правильно выбрать удаляемые пакеты, можно предотвратить затор.

5.3.2. Маршрутизация с учетом состояния трафика

Первый подход, который мы рассмотрим, называется маршрутизацией с учетом состояния трафика. Схемы, рассмотренные нами в разделе 5.2, используют фиксированные весовые коэффициенты связей. Эти схемы могут адаптироваться к изменению топологии, но не к изменению нагрузки. Причина, по которой при вычислении маршрутов нагрузку следует принимать во внимание, состоит в том, что благодаря этому можно будет отвести поток трафика от «горячих точек», в которых в первую очередь возникает перегрузка.

Наиболее простой способ — сделать весовой коэффициент связи функцией от (фиксированной) пропускной способности связи и задержки распространения, а также (переменной) измеренной нагрузки или среднего времени ожидания в очереди. В результате пути с наименьшим весом будут при прочих равных параметрах наименее нагруженными.

Маршрутизация с учетом состояния трафика использовалась на ранних этапах развития сети Интернет в рамках этой модели (Khanna и Zinky, 1989). Однако здесь

есть небольшая опасность. Рассмотрим сеть, изображенную на рис. 5.21. Она разделена на две части — Восток и Запад, соединенные связями CF и EI. Предположим, что основной трафик между Западом и Востоком проходит по связи CF, в результате чего эта линия оказывается слишком нагруженной, что приводит к длительным задержкам. Учет времени ожидания в очереди при вычислении кратчайшего пути сделает связь EI более популярной. После внесения изменений в таблицы маршрутизации большинство трафика пойдет по связи EI, и она окажется слишком нагруженной. Поэтому при следующем обновлении таблиц кратчайшим путем снова станет CF. В конечном итоге значения в таблицах маршрутизации будут сильно колебаться, что приведет к ошибкам при выборе маршрутов и другим проблемам.

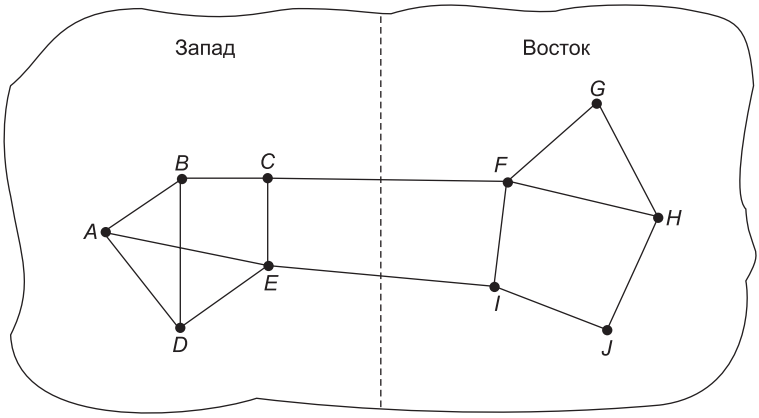


Рис. 5.21. Сеть, в которой восточная и западная части соединены двумя связями

Если оставить в стороне нагрузку и учитывать только пропускную способность и задержку распространения, такая проблема не возникает. Попытки учесть нагрузку, используя узкий диапазон весовых значений, лишь замедляют колебания маршрутов. Удачное решение проблемы основывается на двух методах. Первый — это многопутевая маршрутизация, при которой между отправителем и получателем может существовать несколько путей. Применительно к нашему примеру это означает, что пакеты могут передаваться по обеим связям между Востоком и Западом. Второй метод состоит в следующем: схема маршрутизации должна перемещать трафик по маршрутам настолько медленно, чтобы она сходилась (так, например, работает схема Gallagher, 1977).

Из-за всех этих трудностей протоколы маршрутизации сети Интернет обычно не строят маршруты на основании нагрузки на сеть. Вместо этого перегрузки регулируются вне протокола маршрутизации за счет изменения входных данных. Это называется **управлением трафиком (traffic engineering)**.

5.3.3. Управление доступом

Широко применяемым методом недопущения перегрузки в сетях виртуальных каналов является **управление доступом**. Идея этого метода проста: не нужно создавать новый виртуальный канал до тех пор, пока сеть не сможет обработать дополнительный трафик без перегрузки. То есть любые попытки добавить виртуальный канал могут

закончиться неудачно. Это лучшее решение, поскольку если пустить в сеть, которая в данный момент занята, дополнительных пользователей, то ситуация только ухудшится. Аналогичная ситуация происходит в телефонных системах: когда коммутатор оказывается перегруженным, вы поднимаете трубку и не слышите гудка.

Преимущество этого подхода оказывается существенным в случае, если добавление нового виртуального канала приведет к перегрузке. В телефонных сетях эта задача решается просто благодаря фиксированной пропускной способности для вызовов (64 Кбит/с для несжатого аудио). Но в компьютерных сетях используются виртуальные каналы всевозможных типов. Поэтому если мы хотим прибегнуть к управлению доступом, каналы должны включать в себя определенные характеристики трафика.

Часто в качестве характеристик трафика выступают скорость и форма. Однако вопрос о том, как просто и осмысленно описать эти характеристики, достаточно трудный: трафик обычно неравномерен, и поэтому средняя скорость не является особенно показательной. Например, обеспечить поиск в Сети гораздо сложнее, чем просмотр потокового видео такого же объема, поскольку блоки трафика с большей вероятностью приводят к перегрузке маршрутизаторов сети. Этот эффект чаще всего называют **дырявым ведром (leaky bucket)** или **маркерным ведром (token buket)**. Дырявое ведро использует два параметра, ограничивающих среднюю скорость и мгновенный выброс трафика. Алгоритм дырявого ведра широко используется для улучшения качества обслуживания, поэтому мы поговорим о нем более подробно в разделе 5.4.

Вооружившись характеристиками трафика, сеть принимает решение о добавлении нового виртуального канала. Чтобы перегрузка не произошла, сеть может, например, зарезервировать часть пропускной способности путей для каждого из виртуальных каналов. В таком случае характеристика трафика выступает в качестве договора об обслуживании, которое сеть обязуется предоставить пользователю. Мы предотвратили перегрузку, но слишком рано отклонились в сторону смежной темы качества обслуживания; к ней мы вернемся в следующем разделе.

Даже если сеть не берет на себя никаких обязательств, она может использовать характеристики трафика для управления доступом. В таком случае задача сводится к тому, чтобы оценить число виртуальных каналов, достаточное для обеспечения нужной пропускной способности сети и работы без перегрузок. Предположим, что все виртуальные каналы, способные передавать трафик со скоростью до 10 Мбит/с, используют один и тот же физический канал с пропускной способностью 100 Мбит/с. Сколько каналов можно использовать? Очевидно, что в случае 10 каналов нет никакого риска перегрузки, однако в нормальной ситуации это неэффективно, поскольку вряд ли все 10 каналов будут одновременно работать в полную силу. В действующих сетях для оценки числа возможных каналов используются статистические данные. Таким образом, за счет допустимого увеличения риска сеть выигрывает в производительности.

Можно совместить принципы управления доступом и маршрутизации с учетом состояния трафика, направляя маршруты в обход «горячих точек». Для примера рассмотрим сеть, показанную на рис. 5.22, в которой два маршрутизатора перегружены.

Предположим, что хост, соединенный с маршрутизатором *A*, хочет установить соединение с хостом, соединенным с маршрутизатором *B*. В нормальных условиях это соединение прошло бы через один из перегруженных маршрутизаторов. Чтобы этого избежать, сеть усекается, как показано на рис. 5.22, *б*. При этом из нее удаляются перегруженные маршрутизаторы и все их линии связи. Пунктирной линией показан

возможный маршрут виртуального канала в обход перегруженных маршрутизаторов. Подробное описание маршрутизации, чувствительной к нагрузке, можно найти в работе (Shaikh и др., 1999).

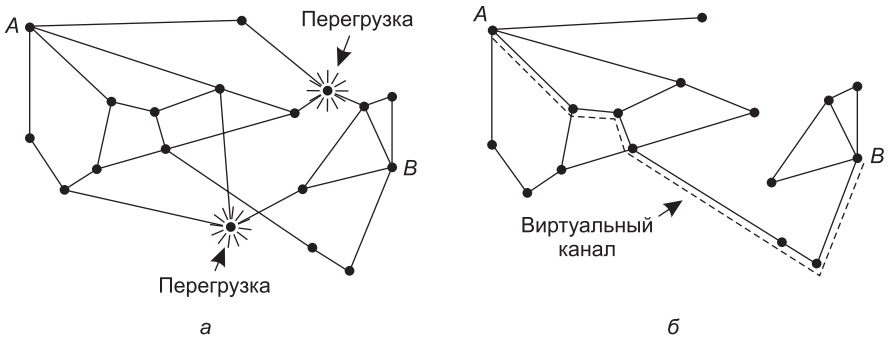


Рис. 5.22. Сеть: а — перегруженная; б — часть сети без перегрузки. Показан виртуальный канал между A и B

5.3.4. Регулирование трафика

В сети Интернет и многих других компьютерных сетях отправители передают столько трафика, сколько сеть в состоянии успешно доставить. В таком режиме сеть работает до тех пор, пока она не перегружена. Если перегрузка неизбежна, она просит отправителей снизить скорость передачи данных. Такая обратная связь — не исключительная, а вполне обычная ситуация, являющаяся частью работы системы. Такой режим работы называется **предотвращением перегрузки (congestion avoidance)** — в противопоставление ситуации, в которой сеть уже (слишком) перегружена.

Давайте рассмотрим несколько подходов к регулированию трафика, применяемых в дейтаграммных сетях и сетях виртуальных каналов. Каждый такой подход должен решать две проблемы. Во-первых, маршрутизаторы должны узнавать о перегрузке до того, как она произойдет. Для этого каждый маршрутизатор должен непрерывно отслеживать ресурсы, которые он использует. Здесь возможны три варианта: использование выходных линий, буферизация очереди пакетов данного маршрутизатора и число пакетов, утерянных вследствие неправильной буферизации. Наиболее эффективным является второй вариант. Средние показатели использования линий не отражают реальной картины при прерывистом трафике. Так, значение 50 % — это немного при сплошном трафике и слишком много при переменном трафике. Число утерянных пакетов становится известно слишком поздно: пакеты начинают теряться уже после возникновения перегрузки.

Время ожидания в очереди маршрутизатора явно отражает то, как перегрузка сказывается на пакетах. Будучи низким в большинстве случаев, этот показатель должен резко возрастать при скачке трафика, когда увеличивается число непереданных пакетов. Такую оценку времени ожидания в очереди (d) можно получить с помощью несложных вычислений, периодически замеряя мгновенную длину очереди s и рассчитывая новое значение переменной d по формуле:

$$d_{\text{новое}} = \alpha d_{\text{старое}} + (1 - \alpha)\, s,$$

где константа α определяет, насколько быстро маршрутизатор забывает свое прошлое. Это называется **ЭВСС (экспоненциально взвешенное скользящее среднее — Exponentialy Weighted Moving Average, EWMA)**. Оно сглаживает различные флуктуации и работает как фильтр низких частот. Как только значение d выходит за пороговый уровень, маршрутизатор узнает о начале перегрузки.

Вторая проблема состоит в том, что маршрутизаторы должны вовремя доставлять сообщения обратной связи отправителям, чей трафик вызывает перегрузку. Хотя перегрузка происходит внутри сети, ее устранение требует участия отправителей, пользующихся сетью. Чтобы доставить сообщение обратной связи, маршрутизатор должен установить соответствующих отправителей. Затем он должен аккуратно передать им уведомления, не отправляя лишних пакетов в уже перегруженную сеть. Разные алгоритмы используют разные механизмы обратной связи. О них мы и поговорим далее.

Сдерживающие пакеты

Самый простой способ сообщить отправителю о перегрузке — сказать об этом прямо. При таком подходе маршрутизатор выбирает перегружающий пакет и отправляет источнику **сдерживающий пакет (choke packet)**. Информация об источнике берется из задержанного пакета. Исходный пакет помечается (специальный бит в его заголовке устанавливается в единицу), чтобы он больше не порождал сдерживающих пакетов на пути следования, и отправляется дальше по своему обычному маршруту. Чтобы избежать роста нагрузки на сеть при перегрузке, маршрутизатор может отправлять сдерживающие пакеты только на низкой скорости.

Когда хост-отправитель получает сдерживающий пакет, он должен уменьшить трафик к указанному получателю, к примеру, на 50 %. В дейтаграммной сети выбор случайного пакета, скорее всего, приведет к тому, что сдерживающие пакеты дойдут до самых быстрых отправителей, поскольку именно их пакеты составляют большую часть очереди. Такая неявная обратная связь позволяет предотвратить перегрузку, не мешая тем отправителям, действия которых не вызывают проблем. По той же причине велика вероятность того, что множественные сдерживающие пакеты будут отправлены на нужный хост и адрес. В течение определенного промежутка времени — пока уменьшение трафика не подействует — хост будет игнорировать дополнительные пакеты. По истечении этого времени новые сдерживающие пакеты сообщат ему, что сеть все еще перегружена.

Примером сдерживающего пакета, применявшегося на ранних этапах сети Интернет, является сообщение `SOURCE QUENCH` (Postel, 1981). Оно не прижилось отчасти потому, что не были четко определены условия и результаты его рассылки. Сейчас в сети Интернет используется другая схема уведомлений, о которой мы поговорим далее.

Явное уведомление о перегрузке

Вместо того чтобы создавать дополнительные пакеты, маршрутизатор может добавить специальную метку (например, 1 или 0 в заголовке) в любой передаваемый пакет, тем самым сообщая о перегрузке. Когда пакет будет доставлен, получатель поймет, что сеть

перегружена, и добавит эту информацию в ответный пакет. После этого отправитель сможет, как и раньше, регулировать свой трафик.

Этот метод называется **явным уведомлением о перегрузке (ECN — Explicit Congestion Notification)** и используется в сети Интернет (Ramakrishnan, 1988). Это усовершенствованный вариант ранних протоколов уведомления о перегрузке, в частности бинарной схемы обратной связи Ramakrishnan и Jain (1988), применявшейся в архитектуре DECNET. На информацию о перегрузке в заголовке пакета отводится два бита. В момент отправки пакет не имеет метки (рис. 5.23). При проходе через перегруженный маршрутизатор пакет получает отметку о перегрузке. После этого получатель передает эти сведения отправителю, добавив явное уведомление о перегрузке в следующий ответный пакет. На рисунке этот процесс показан пунктирной линией, чтобы показать, что он происходит над IP-уровнем (например, на уровне TCP). Далее, как и в случае со сдерживающими пакетами, отправитель должен начать регулировать трафик.

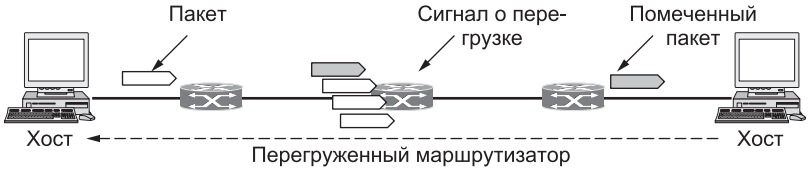


Рис. 5.23. Явное уведомление о перегрузке

Обратное давление на ретрансляционных участках

При больших скоростях передачи данных или при сильной удаленности хостов много новых пакетов может быть отправлено даже после передачи уведомлений о перегрузке, поскольку реакция на уведомление занимает некоторое время. Рассмотрим, к примеру, хост в Сан-Франциско (маршрутизатор *A* на рис. 5.24), посылающий поток данных на хост, расположенный в Нью-Йорке (маршрутизатор *D* на рис. 5.24), со скоростью OC-3 155 Мбит/с. Если у нью-йоркского хоста начнет кончаться буферная память, сдерживающему пакету потребуется около 40 мс на то, чтобы добраться обратно в Сан-Франциско и сообщить о том, что необходимо снизить объем трафика. Явное уведомление о перегрузке займет еще больше времени, поскольку оно доставляется через получателя. Распространение сдерживающего пакета схематично показано на второй, третьей и четвертой диаграммах рис. 5.24, *а*. За те 40 мс, пока этот пакет движется по сети, в сторону нью-йоркского маршрутизатора передается еще 6,2 Мбит данных, с которыми тоже надо как-то совладать. Только к седьмой диаграмме (рис. 5.24, *а*) маршрутизатор заметит начавшееся снижение потока.

Однако есть альтернативный метод, позволяющий бороться с этой проблемой. Он заключается в том, что сдерживающий пакет влияет на трафик каждого маршрутизатора, через который он проходит. Это показано на последовательности диаграмм на рис. 5.24, *б*. Как только сдерживающий пакет достигает точки *F*, поток данных от *F* в сторону *D* должен быть уменьшен. Таким образом, *F* резервирует для соединения большее количество буферной памяти: источник все еще продолжает заваливать это направление своими данными. Нагрузка на *D* мгновенно спадает,

как головная боль у страдальца, рекламирующего по телевизору чудодейственные пилюли. На следующем шаге сдерживающий пакет, продолжая свой путь, достигает *E* и приказывает уменьшить поток в сторону *F*. В результате в течение какого-то времени точке *E* приходится выдерживать повышенную нагрузку, но зато мгновенно освобождается от своего бремени точка *F*. Наконец, победный марш сдерживающего пакета приводит его к источнику всех бед — точке *A*, и теперь поток снижается на самом деле.

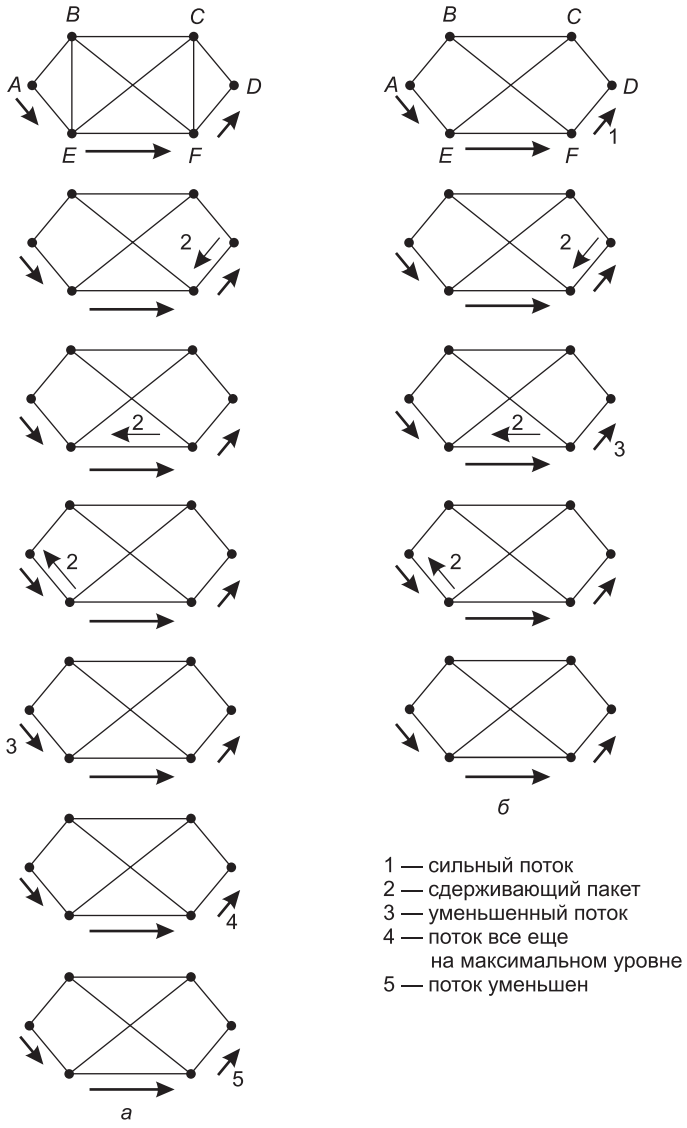


Рис. 5.24. Сдерживающий пакет влияет только на источник (*а*); Сдерживающий пакет влияет на все промежуточные участки (*б*)

Результатом применения метода сдерживания трафика на ретрансляционных участках является максимально быстрое устранение перегрузки в самой горячей точке за счет использования большего объема буферной памяти промежуточных маршрутизаторов. Таким образом, перегрузка пресекается без потери пакетов. Эта идея обсуждается более подробно у Mishra и др. (1996).

5.3.5. Сброс нагрузки

Когда ни один из выше приведенных методов не помогает в борьбе с перегрузкой, маршрутизаторы могут ввести в бой тяжелую артиллерию — сброс нагрузки. **Сбросом нагрузки** называется простое игнорирование маршрутизаторами пакетов, которые они не могут обработать. Своим происхождением этот термин обязан системам электроснабжения, где он означает отключение в случае перегрузок отдельных участков во избежание выхода из строя всей системы. Обычно такое происходит в жаркие летние дни, когда спрос на электроэнергию многократно превосходит предложение.

Ключевой проблемой для маршрутизатора, заваленного пакетами, является выбор пакета, который будет отвергнут. Выбор может зависеть от типа приложений, использующих данную сеть. Для передачи файла более старый пакет ценится выше нового, так как отвержение пакета номер 6 и сохранение пакетов с номерами, например, с 7 по 10 лишь заставит получателя выполнить лишнюю работу: поместить в буфер данные, которые он еще не может использовать. Для мультимедийных приложений, работающих в реальном времени, напротив, новый пакет важнее старого. Причина в том, что пакеты становятся ненужными, если они задерживаются и не приходят вовремя.

Первую стратегию (старое лучше нового) часто называют **винной стратегией**, а вторую (новое лучше старого) — **молочной стратегией**, так как большинство людей предпочтут пить свежее молоко и выдержанное вино, а не наоборот.

Более разумные алгоритмы сброса нагрузки требуют участия отправителей. В качестве примера можно привести пакеты, содержащие сведения о маршрутизации. Эти пакеты значительно важнее обычных пакетов с данными, поскольку они устанавливают маршруты; если они будут утеряны, может пострадать связность сети. Другой пример — алгоритмы сжатия видеосигнала (например, MPEG), которые периодически посылают полный кадр, а последующие кадры представляют собой карты изменений относительно последнего полного кадра. В таком случае потеря пакета, содержащего разностный сигнал, не так страшна, как потеря полного кадра, так как от этого полного кадра зависят последующие пакеты.

Для реализации интеллектуальной стратегии выбрасывания части информации приложения должны помечать свои пакеты, сообщая сети об их важности. Тогда маршрутизаторы смогут сначала выбросить пакеты наименее важного класса, затем следующего за ним и т. д.

Конечно, при отсутствии стимула все будут помечать свои пакеты не иначе как ОЧЕНЬ ВАЖНО — НИ В КОЕМ СЛУЧАЕ НЕ ВЫБРАСЫВАТЬ. Предотвращение неоправданного использования таких отметок часто достигается за счет сетевых ресурсов и денежных средств. Например, сеть может разрешить отправителям пересылать пакеты с большей скоростью, чем указано в договоре на предоставление услуг, если пакет будет помечаться как низкоприоритетный. Такая стратегия весьма удачна,

поскольку более эффективно использует свободные ресурсы, разрешая хостам пользоваться ими, пока это никому не мешает, но не закрепляя за ними этого права.

Случайное раннее обнаружение

С перегрузкой гораздо проще начать бороться в тот момент, когда она только началась, чем дать ей развиться до критических размеров и потом думать, что делать в сложившейся ситуации. Это соображение приводит к интересной модификации идеи сброса нагрузки, при которой отвержение пакетов происходит еще до того, как все буферное пространство будет заполнено скопившимися необработанными данными.

Причина, по которой эта идея имеет смысл, состоит в том, что большинство интернет-хостов узнают о перегрузке не через явные уведомления. В действительности единственным достоверным сигналом о перегрузке сети служит утеря пакетов. Трудно сконструировать такой маршрутизатор, который не удалял бы пакеты при перегрузке. Реакцией транспортных протоколов наподобие TCP на утерю пакетов при перегрузке является ответное снижение трафика от источника. Обоснование такой логики состоит в том, что TCP предназначен для проводных сетей, которые по сути своей являются очень надежными, и потеря пакетов в них чаще всего сигнализирует о переполнении буфера, а не об ошибках передачи. Чтобы TCP работал эффективно, беспроводные линии связи должны справляться с ошибками передачи на канальном уровне (так, чтобы на сетевом уровне они были не видны).

Эта ситуация и используется для уменьшения перегрузок. Если заставить маршрутизаторы сознательно терять пакеты еще до того, как ситуация станет безнадежной, то останется время на то, чтобы источник мог предпринять какие-то действия. Популярный алгоритм, реализующий данную идею, называется **случайным ранним обнаружением (RED — Random Early Detection)** (Floyd и Jacobson, 1993). Для определения условий, при которых следует начинать терять пакеты, маршрутизаторы постоянно высчитывают скользящее среднее длин своих очередей. Когда средняя длина очереди на какой-либо связи превышает пороговое значение, эта связь объявляется перегруженной и небольшая часть пакетов удаляется случайным образом. Именно случайный выбор пакетов увеличивает вероятность того, что самые быстрые отправители обнаружат утерю пакета; этот вариант является наилучшим, поскольку маршрутизатор не знает, какой именно источник является причиной большинства проблем в дейтаграммной сети. Отправитель заметит утерю пакета без всяких уведомлений, после чего транспортный протокол замедлит работу. Таким образом, утерянный пакет несет ту же информацию, что и сдерживающий пакет, но неявно, то есть маршрутизатор обходится без отправки реального сигнала.

Маршрутизаторы, использующие случайное раннее обнаружение, выигрывают в производительности перед маршрутизаторами, удаляющими пакеты при заполнении буфера, хотя иногда они требуют правильной настройки. Например, оптимальное число пакетов, которые необходимо удалить, зависит от числа отправителей, которых требуется оповестить о перегрузке. Однако по возможности лучше всего использовать явные уведомления о перегрузке. Они работают точно так же, но передают сообщения в явном виде, а не косвенно через утерю пакета; случайное раннее обнаружение используется в тех случаях, когда хосты не принимают явные уведомления.

5.4. Качество обслуживания

Методы, рассмотренные нами выше, направлены на уменьшение перегрузок и повышение производительности в сетях передачи данных. Однако существуют приложения (и клиенты), требующие от сети более строгих гарантий производительности, чем «лучшее, что можно сделать в данных условиях». В частности, для работы мультимедийных приложений часто необходимы минимальная пропускная способность и максимальное время задержки. В этом разделе мы продолжим изучение параметров производительности сетей, но больший упор сделаем на методах обеспечения высокого качества обслуживания, соответствующего требованиям конкретных приложений. Это та область, в которой Интернет сейчас стремительно развивается.

Одно из наиболее простых решений для предоставления высокого качества обслуживания заключается в создании сети с пропускной способностью, позволяющей передавать любые объемы трафика. Этот метод называется **избыточным обеспечением (overprovisioning)**. Такая сеть будет осуществлять трафик приложений без существенных потерь, а при условии хорошей схемы маршрутизации пакеты будут доставляться с низкой задержкой. Этим ограничиваются преимущества такого алгоритма с точки зрения производительности. Можно сказать, что телефонная сеть является системой с избыточным обеспечением. Довольно редко бывает, чтобы вы подняли трубку и не услышали гудка. Дело в том, что в систему заложена настолько большая пропускная способность, что превысить ее оказывается тяжело.

Проблема здесь одна: такое решение обходится очень дорого. По сути, проблема легко решается за счет финансовых затрат. Благодаря механизмам обеспечения качественного обслуживания сеть с меньшей пропускной способностью может удовлетворить требованиям приложения при низких затратах. Более того, избыточное обеспечение основывается на данных об ожидаемом трафике. Ситуация кардинально меняется, если схема трафика слишком сильно отличается от предполагаемой. Благодаря механизмам обеспечения качества обслуживания сеть может выполнить свои обязательства даже при резком увеличении объема трафика за счет отклонения некоторых запросов.

Чтобы обеспечить качество обслуживания, необходимо обратить внимание на следующие вопросы.

1. Что приложениям нужно от сети.
2. Как регулировать трафик, поступающий в сеть.
3. Как зарезервировать ресурсы на маршрутизаторах, необходимые для обеспечения производительности.
4. Может ли сеть принять больший объем трафика.

Ни один метод не в состоянии эффективно решить все эти проблемы. Поэтому для сетевого (и транспортного) уровня разработано множество различных методов. На практике для обеспечения качества обслуживания используются их комбинации. Поэтому мы рассмотрим два варианта, использующихся в сети Интернет: интегральное обслуживание и дифференцированное обслуживание.

5.4.1. Требования приложений

Последовательность пакетов, передающихся от источника к приемнику, называется **потоком** (Clark, 1988) (**flow**). При этом в сетях, ориентированных на соединение, поток могут составлять все пакеты соединения, а в сетях без установления соединения — все пакеты, отправленные от одного процесса к другому. Каждому потоку требуются определенные условия, которые можно охарактеризовать следующими четырьмя основными параметрами: пропускная способность, задержка, флуктуация и потери. Все вместе они формируют то, что называется **качеством обслуживания** (**QoS** — **Quality of Service**), необходимым потоку.

Некоторые часто используемые приложения и их требования к сети приведены в табл. 5.2. Следует отметить, что требования к сети являются менее жесткими, чем требования приложений, так как в некоторых случаях приложение может само улучшить уровень обслуживания, предоставленный сетью. В частности, для надежной передачи файлов сеть не обязана работать без потерь; кроме того, время задержки не должно быть одинаковым при передаче пакетов для аудио и видео. Потери можно компенсировать за счет повторной передачи, а для сглаживания флуктуаций можно сохранять пакеты в буфере получателя. Но при слишком низкой пропускной способности или слишком большой задержке приложения оказываются бессильны.

Таблица 5.2. Строгость требований некоторых приложений к качеству обслуживания

Приложение	Пропускная способность (bandwidth)	Задержка (delay)	Флуктуации (jitter)	Потери (loss)
Электронная почта	Низкая	Низкая	Слабая	Средняя
Передача файлов	Высокая	Низкая	Слабая	Средняя
Веб-доступ	Средняя	Средняя	Слабая	Средняя
Удаленный доступ	Низкая	Средняя	Средняя	Средняя
Аудио по заказу	Низкая	Низкая	Сильная	Низкая
Видео по заказу	Высокая	Низкая	Сильная	Низкая
Телефония	Низкая	Высокая	Сильная	Низкая
Видеоконференции	Высокая	Высокая	Сильная	Низкая

Приложения могут иметь различные потребности в пропускной способности: для электронной почты, аудио в различных форматах и удаленного доступа необходима небольшая пропускная способность, тогда как передача файлов и видео в различных форматах требуют большой пропускной способности.

С задержкой дело обстоит иначе. Приложения, занимающиеся передачей файлов, включая электронную почту и видео, не чувствительны к задержкам. Даже если все пакеты будут доставляться с задержкой в несколько секунд, ничего страшного не произойдет. Однако интерактивные приложения — например, обеспечивающие веб-доступ или удаленный доступ — к задержкам более критичны. Что касается приложений, работающих в реальном масштабе времени, их требования к задержкам очень строги. Если при телефонном разговоре все слова собеседников будут приходить с большой

задержкой, пользователи сочтут такую связь неприемлемой. С другой стороны, проигрывание видео- или аудиофайлов, хранящихся на сервере, допускает наличие некоторой задержки.

Колебание (то есть стандартное отклонение) времени задержки или времени прибытия пакета называется **флуктуацией (jitter)**. Первые три приложения (см. табл. 5.2) спокойно отнесутся к неравномерной задержке доставки пакетов, а при организации удаленного доступа этот фактор имеет более важное значение, поскольку при сильных флуктуациях обновления на экране будут появляться скачками. Видео- и особенно аудиоданные исключительно чувствительны к флуктуациям. Если пользователь просматривает видео, доставляемое на его компьютер по сети, и все кадры приходят с задержкой ровно 2,000 с, все нормально. Однако если время передачи колеблется от одной до двух секунд и приложению не удается скрыть флуктуации, то результат будет просто ужасен. При прослушивании звукозаписей будут заметны флуктуации даже в несколько миллисекунд.

Первые четыре приложения предъявляют высокие требования к потерям, так как для них, в отличие от аудио и видео, важен каждый бит информации. Обычно это достигается путем повторной передачи утерянных пакетов транспортным уровнем. Однако это неэффективно; было бы лучше, если бы сеть отвергала те пакеты, которые, скорее всего, будут утеряны. Для аудио- и видеоприложений утеря пакета не всегда требует повторной передачи: короткие паузы и пропущенные кадры могут остаться незамеченными пользователями.

Чтобы удовлетворить требования различных приложений, сеть может предлагать разное качество обслуживания. Важный пример — технология, использующаяся в сетях ATM, которая когда-то считалась перспективной, но впоследствии оказалась на заднем плане. Сети ATM поддерживают следующее:

1. Постоянная битовая скорость (например, телефония).
2. Переменная битовая скорость в реальном времени (например, сжатые видеоданные при проведении видеоконференций).
3. Переменная битовая скорость не в реальном времени (например, просмотр фильмов по заказу).
4. Доступная битовая скорость (например, передача файлов).

Такое разбиение по категориям может оказаться полезным и для других целей, и для других сетей. Постоянная битовая скорость — это попытка моделирования проводной сети путем предоставления фиксированной пропускной способности и фиксированной задержки. Битовая скорость может быть переменной, например, при передаче сжатого видео, когда одни кадры удается сжать в большей степени, нежели другие. Кадр, содержащий множество разноцветных деталей, сожмется, скорее всего, плохо, и на его передачу придется потратить много битов, тогда как кадр, запечатлевший белую стену, сожмется очень хорошо. Фильмы по заказу на самом деле проигрываются не в реальном времени: часто несколько секунд видеозаписи доставляется перед проигрыванием и хранится в буфере, поэтому флуктуации попросту приводят к колебаниям объема видео, которое было получено, но не воспроизведено. Приложениям типа электронной почты нужно принципиальное наличие хоть какой-нибудь битовой скорости, они не чувствительны к задержкам и флуктуациям, поэтому говорят, что этим приложениям требуется «доступная битовая скорость».

5.4.2. Формирование трафика

Прежде чем гарантировать пользователю определенное качество обслуживания, сеть должна знать примерный объем трафика. В телефонных сетях его оценить легко: обычный звонок (в несжатом формате) требует скорости 64 Кбит/с и одного 8-битного отсчета каждые 125 мкс. Однако в данных сетях трафик является **неравномерным (bursty)**. Трафик может прибывать неравномерно по трем причинам: непостоянная скорость трафика (например, при видеоконференциях со сжатием), взаимодействие пользователя и приложения (например, просмотр новой веб-страницы) и переключение компьютера между задачами. С неравномерным трафиком работать гораздо сложнее, чем с постоянным, так как при этом может произойти переполнение буферов и, как следствие, утеря пакетов.

Формирование трафика (traffic shaping) — способ регулировки средней скорости и равномерности потока входных данных. Приложения должны иметь возможность передавать такой трафик, который им нужен (включая неравномерный); при этом необходим простой и удобный способ описания возможных схем трафика. Когда устанавливается поток, пользователь и сеть (то есть клиент и оператор связи) договариваются об определенной схеме (то есть форме) трафика для данного потока. В результате клиент сообщает интернет-провайдеру: «Мой график передачи будет выглядеть следующим образом. Сможете ли вы это обеспечить?»

Иногда такое соглашение называется **соглашением об уровне обслуживания (SLA, Service Level Agreement)**, особенно если в нем оговариваются комплексные потоки и длительные периоды времени (например, весь трафик для данного клиента). До тех пор пока клиент выполняет свою часть условий соглашения и посылает пакеты не чаще оговоренного в договоре графика, интернет-провайдер обязуется доставлять их в определенный срок.

Формирование трафика снижает перегрузку и, таким образом, помогает сети выполнять свои обязательства. Но чтобы это работало, необходимо ответить на вопрос: как интернет-провайдер будет узнавать о том, что клиент соблюдает соглашение, и что делать, если клиент нарушит договор. Пакеты, отправка которых выходит за рамки условий соглашения, могут удаляться или помечаться как низкоприоритетные. Наблюдение за потоком трафика называется **политикой трафика (traffic policing)**.

Формирование трафика и политика трафика не так важны для равноранговой передачи и передачах других типов, при которых используется вся доступная пропускная способность. Однако они имеют важное значение для данных, передаваемых в реальном времени (например, при соединениях с передачей аудио- и видеосигнала), обладающих строгими требованиями к качеству обслуживания.

Алгоритмы дырявого и маркерного ведра

Нам уже знаком один способ ограничения передаваемого приложением объема данных — скользящее окно. Этот метод использует один параметр, который ограничивает данные, передаваемые в определенный момент времени, и косвенно ограничивает скорость. Теперь мы обратимся к более общим методам описания трафика и рассмотрим алгоритмы дырявого ведра и маркерного ведра. Хотя эти алгоритмы немного отличаются друг от друга, они приводят к одинаковому результату.

Представьте себе ведро с маленькой дырочкой в днище, как показано на рис. 5.25, *б*. Независимо от скорости, с которой вода наливается в ведро, выходной поток обладает постоянной скоростью *R*, когда в ведре есть вода, и нулевой скоростью, когда ведро пустое. Кроме того, когда ведро наполняется и вода занимает весь объем *B*, вся лишняя вода выливается через край и теряется.

Рис. 5.25. Алгоритмы дырявого и маркерного ведра: *а* — формирование пакетов; *б* — дырявое ведро; *в* — маркерное ведро

С помощью такого ведра можно формировать или приводить в порядок пакеты, поступающие в сеть (рис. 5.25, *а*). Принцип таков: каждый хост соединяется с сетью через интерфейс, содержащий дырявое ведро. Чтобы пакет можно было отправить в сеть, в ведре должно быть достаточно места для воды. Если в момент поступления пакета ведро заполнено, пакет либо помещается в очередь до тех пор, пока в ведре не освободится достаточно места, либо отвергается. Первый вариант встречается в таких случаях, когда формирование трафика производится операционной системой хоста. Второй вариант имеет место, когда проверка трафика, поступающего в сеть, осуществляется аппаратными средствами в сетевом интерфейсе интернет-провайдера. Этот метод был предложен Тернером (Turner, 1986) и называется **алгоритмом дырявого ведра (leaky bucket algorithm)**.

То же самое можно представить себе по-другому: в виде ведра, которое в данный момент наполняется (рис. 5.25, *в*). Вода вытекает из крана со скоростью *R*, а объем ведра, как и в предыдущем случае, равен *B*. Чтобы отправить пакет, необходимо, чтобы из ведра можно было вылить воду, или маркеры (так обычно называют содержимое ведра), а не налить ее туда. В ведре может содержаться ограниченное число маркеров (не более *B*); если ведро пустое, для отправки пакета необходимо подождать, пока не появятся новые маркеры. Этот алгоритм называется **алгоритмом маркерного ведра (token bucket algorithm)**.

Дырявое ведро и маркерное ведро ограничивают постоянную скорость потока, при этом пропуская кратковременные пики трафика (также ограниченные максимальным значением) без искусственных задержек. Чтобы снизить нагрузку на сеть, шейпер (формирователь) трафика сглаживает крупные пики. В качестве примера представьте, что компьютер может производить данные со скоростью 1000 Мбит/с (125 млн байт в секунду) и что первая связь сети также работает на этой скорости. Хост генерирует схему трафика, показанную на рис. 5.26, *а*. Эта схема является неравномерной. Сред-

няя скорость составляет 200 Мбит/с, хотя хост отправляет пиковый объем трафика в 16 000 Кбайт на максимальной скорости 1000 Мбит/с (за 1/8 секунды).

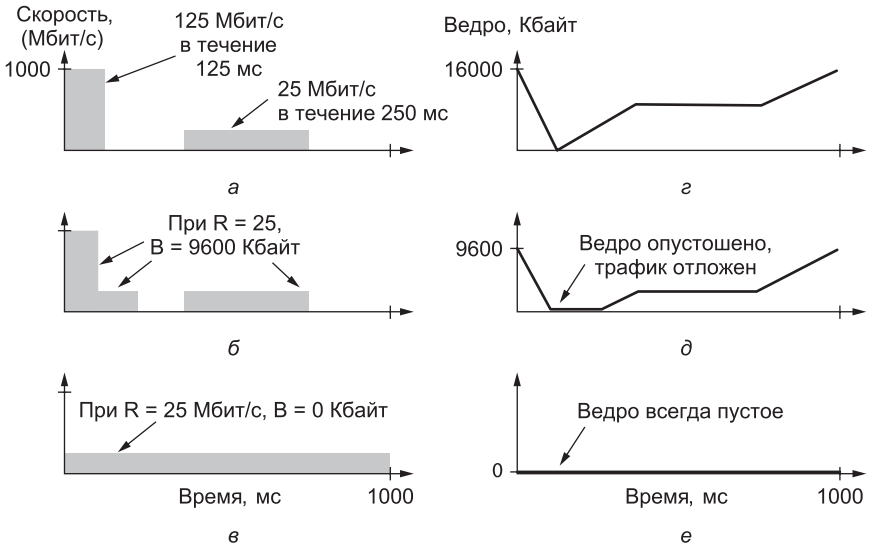


Рис. 5.26. Схема: *а* — трафик, передаваемый хостом; *б* — исходящий трафик, сформированный с помощью маркерного ведра со скоростью 200 Мбит/с и емкостью 9600 Кбайт и *в* — 0 Кбайт; *г* — уровень маркерного ведра при формировании трафика со скоростью 200 Мбит/с и емкостью 16 000 Кбайт, *д* — 9600 Кбайт и *е* — 0 Кбайт

Теперь предположим, что маршрутизаторы могут принимать данные на максимальной скорости только в течение небольших промежутков времени — до тех пор, пока буфер не заполнится. Размер буфера составляет 9600 Кбайт, что меньше, чем объем пикового трафика. В течение больших промежутков времени маршрутизаторы лучше всего работают, если скорость не превышает 200 Мбит/с (так как именно эта пропускная способность указана в соглашении с клиентом). Из этого следует, что если для передачи используется эта схема, часть трафика будет удалена, так как не поместится в буферы маршрутизаторов.

Чтобы избежать такой утери пакетов, можно сформировать трафик на хосте по методу маркерного ведра. Если скорость R равна 200 Мбит/с, а емкость B равна 9600 Кбайт, трафик попадает в границы возможностей сети. Исходящий трафик такого маркерного ведра показан на рис. 5.26, *б*. Хост может отправлять трафик на максимальной скорости в 1000 Мбит/с в течение небольшого промежутка времени — до тех пор, пока ведро не станет пустым. Затем он должен будет снизить скорость до 200 Мбит/с и отправить оставшуюся часть трафика. Идея в том, чтобы растянуть время отправки пикового трафика (пачки пакетов), если сеть не в состоянии обработать его в один прием. Уровень маркерного ведра показан на рис. 5.26, *д*. Вначале ведро наполнено, но после первой порции трафика оно становится пустым. Когда уровень достигает нуля, новые пакеты могут передаваться только с такой скоростью, с которой буфер наполняется; пока ведро снова не наполнится, отправка крупных объемов трафика

невозможна. Когда трафик не поступает, ведро постепенно наполняется; пока данные приходят со скоростью заполнения ведра, оно остается пустым.

Чтобы трафик был равномерным, его можно сформировать. На рис. 5.26, *в* показан исходящий трафик маркерного ведра со скоростью 200 Мбит/с и емкостью 0. Это крайний случай — трафик полностью выровнен. Крупные пачки не принимаются; трафик приходит с постоянной скоростью. Уровень воды в ведре, соответственно, всегда равен нулю (рис. 5.26, *е*). Трафик помещается в очередь хоста, и в каждый момент времени какой-то пакет ожидает отправки.

Наконец, на рис. 5.26, *д* показан уровень маркерного ведра со скоростью R = 200 Мбит/с и емкостью B = 16 000 Кбайт. Это самое маленькое маркерное ведро, через которое трафик проходит без изменений. Такое ведро маршрутизатор может использовать для проверки трафика, передаваемого хостом. Такое ведро можно расположить на одном из концов сети; если трафик соответствует маркерному ведру, указанному в соглашении об обслуживании, он сможет через него пройти. Если же хост будет отправлять данные слишком быстро или неравномерно, вода в маркерном ведре закончится, и сеть узнает о том, что трафик не соответствует условиям соглашения. После этого в зависимости от конфигурации сети лишние пакеты будут либо удалены, либо помечены как низкоприоритетные. А в нашем примере ведро становится пустым лишь на короткое время — после получения большого объема трафика. После этого оно восстанавливается и снова готово принять такой трафик.

Дырявое ведро и маркерное ведро просты в реализации. Давайте посмотрим на то, как работает маркерное ведро. Хотя до этого мы говорили о втекающей и вытекающей жидкости, нужно понимать, что на практике сеть имеет дело с дискретными величинами. Реализация алгоритма маркерного ведра подразумевает наличие переменной, считающей маркеры. Счетчик увеличивается на $R/\Delta T$. В нашем примере счетчик будет увеличиваться на 200 Кбит за 1 мс. Каждый раз при посылке трафика счетчик уменьшается; когда его значение доходит до нуля, передача пакетов останавливается.

Если все пакеты имеют одинаковый объем, уровень ведра можно измерять в пакетах (например, 200 Мбит — это 20 пакетов по 1250 байт). Однако чаще всего используются пакеты разного размера. Тогда уровень ведра исчисляется в байтах. Если число байт в ведре не является достаточным для отправки пакета, пакет вынужден ждать, пока ситуация не изменится (что может произойти не сразу, если скорость заполнения ведра мала).

При расчете длительности максимальной выходной пачки (до тех пор, пока ведро не станет пустым) нужно учитывать, что пока ведро опорожняется, появляются новые маркеры. При длительности пачки S с, емкости маркерного ведра B байт, скорости появления маркеров R байт/с и максимальной выходной скорости M байт/с очевидно, что максимальное количество переданных байтов в пачке будет равно $B + RS$ байт. Мы также знаем, что количество байтов, переданных в пачке с максимальной скоростью, равно MS. Таким образом,

$$B + RS = MS.$$

Решив это уравнение, получим: $S = B/(M - R)$. При наших параметрах B = = 9600 Кбайт, M = 125 Мбайт/с и R = 25 Мбайт/с получаем длительность пачки около 94 мс.

Недостатком алгоритма маркерного ведра является то, что скорость передачи крупных пачек снижается до постоянного значения R. Часто бывает желательно уменьшить пиковую скорость, не возвращаясь при этом к постоянному значению скорости (но и не увеличивая это значение, чтобы не пропустить в сеть дополнительный трафик). Один из способов получения более гладкого трафика состоит в помещении еще одного маркерного ведра после первого. Скорость второго ведра должна быть гораздо выше скорости первого. По сути, первое ведро определяет характеристики трафика и фиксирует его скорость, иногда позволяя отправлять крупные объемы данных. Второе ведро уменьшает максимальную скорость, с которой могут передаваться такие пачки. Например, если скорость второго ведра равна 500 Мбит/с, а емкость — 0, первая пачка поступит в сеть с максимальной скоростью 500 Мбит/с. Это меньше, чем наше предыдущее значение 1000 Мбит/с.

Управление такими схемами может оказаться непростым. Когда маркерные ведра используются для формирования трафика на хостах, пакеты вынуждены ждать в очереди до тех пор, пока ведро их не пропустит. Когда они применяются на маршрутизаторах сети для определения трафика, сеть имитирует алгоритм и гарантирует, что пакетов и байтов посылается не больше, чем разрешено. Тем не менее эти методы позволяют формировать сетевой трафик, приводя его к более управляемому виду и обеспечивая тем самым выполнение требований по качеству обслуживания.

5.4.3. Диспетчеризация пакетов

Возможность управления трафиком — это неплохой начальный шаг в деле обеспечения гарантированного качества обслуживания. Однако чтобы предоставить клиенту гарантии производительности, необходимо резервировать достаточное количество ресурсов. Для этого мы будем считать, что все пакеты в потоке следуют по одному и тому же пути. При распределении их случайным образом между несколькими маршрутизаторами невозможно что-либо гарантировать. Следовательно, между источником и приемником должно быть установлено нечто вроде виртуального канала, и все пакеты, принадлежащие данному потоку, должны следовать по указанному маршруту.

Алгоритмы распределения ресурсов маршрутизатора между пакетами потока и конкурирующими потоками называются **алгоритмами диспетчеризации пакетов (packet scheduling algorithm)**. Резервироваться могут три типа ресурсов:

1. Пропускная способность.
2. Буферное пространство.
3. Время центрального процессора.

Наиболее очевидно резервирование пропускной способности. Если потоку необходима скорость в 1 Мбит/с, а исходящая линия может работать со скоростью 2 Мбит/с, то направить три потока с такими параметрами по этой линии не удастся. То есть резервирование пропускной способности означает предотвращение предоставления канала большему числу абонентов, чем канал может обработать.

Вторым дефицитным ресурсом является буферное пространство. Когда прибывает пакет, он хранится в буфере маршрутизатора до тех пор, пока он не будет передан

по выбранной исходящей линии. Буфер нужен для того, чтобы хранить небольшие пачки трафика, пока потоки сражаются друг с другом. Если буферное пространство недоступно, входящий пакет приходится игнорировать, поскольку его просто негде сохранить. Для обеспечения хорошего качества обслуживания можно резервировать некоторую часть буферной памяти под конкретный поток, чтобы ему не пришлось бороться за буфер с другими потоками. И тогда при передаче потока ему всегда будет предоставляться выделенная часть буфера, вплоть до некоторого максимума.

Наконец, время центрального процесса также может быть очень ценным ресурсом. На что расходуется время работы процессора в маршрутизаторе? На обработку пакетов. Поэтому существует предельная скорость, с которой маршрутизатор может обрабатывать пакеты. Хотя большинство пакетов современные маршрутизаторы могут обрабатывать очень быстро, определенные типы пакетов (в частности, ICMP, о которых мы поговорим в разделе 5.6) все же требуют более длительной работы центрального процессора. Необходимо быть уверенным в том, что процессор не перегружен, — это залог своевременной обработки каждого пакета.

Алгоритмы диспетчеризации пакетов распределяют пропускную способность и другие ресурсы маршрутизатора, определяя, какие пакеты из буфера необходимо отправить по исходящей линии следующими. Простейший вариант диспетчера мы уже обсуждали, когда говорили о принципе работы маршрутизатора. Каждый маршрутизатор помещает пакеты в очередь (отдельную для каждой исходящей линии), где пакеты ждут отправки. При этом отправка пакетов происходит в том же порядке, в котором они пришли. Этот принцип называется **FIFO (First-In First-Out**, **первым пришел — первым ушел)**, или **FCFS (First-Come First-Serve**, **первым пришел — первым обслуживается)**.

Маршрутизаторы, работающие по принципу FIFO, при переполнении очереди обычно удаляют последние прибывшие пакеты. Поскольку только что прибывшие пакеты помещаются в конец очереди, такое поведение называется «**обрубанием хвоста» (tail drop)**. Это настолько простой и привычный вариант, что альтернативный метод не так-то просто себе представить. Тем не менее алгоритм случайного раннего обнаружения (RED), о котором мы говорили в разделе 5.3.5, удалял новые пакеты случайным образом, когда длина очередь начинала расти. Другие алгоритмы диспетчеризации, о которых мы здесь поговорим, используют для выбора удаляемых пакетов самые разные возможности.

Диспетчеризацию по принципу FIFO очень легко реализовать, однако она не позволяет обеспечить высокое качество обслуживания: если потоков несколько, один из них может влиять на производительность других. Если первый поток будет вести себя агрессивно и отправлять большие объемы трафика, его пакеты будут помещаться в очередь. Если пакеты обрабатываются в порядке поступления, агрессивный отправитель захватит большую часть мощности маршрутизаторов, отбирая ее у других потоков и тем самым уменьшая их качество обслуживания. Ситуация усугубится тем, что пакеты других потоков, прошедшие через маршрутизатор, скорее всего, придут с опозданием, поскольку им пришлось задержаться в очереди, ожидая отправки пакетов агрессивного потока.

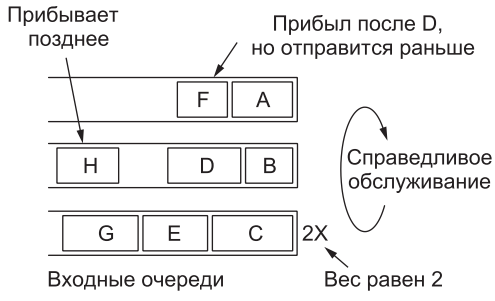
Чтобы обеспечить изоляцию потоков и предотвратить их конфликты, было разработано множество различных алгоритмов диспетчеризации пакетов (Bhatti

и Crowcroft, 2000). Одним из первых был алгоритм **справедливого обслуживания (fair queueing)** (Nagle, 1987). Суть его состоит в том, что маршрутизаторы организуют отдельные очереди для каждой исходящей линии, по одной для каждого потока. Как только линия освобождается, маршрутизатор начинает циклически сканировать очереди (рис. 5.27), выбирая первый пакет следующей очереди. Таким образом, если за данную исходящую линию борются n хостов, то каждый из них имеет возможность отправить свой пакет, пропустив $n-1$ чужих пакетов. Получается, что все потоки передают пакеты с одинаковой скоростью. Агрессивному хосту не поможет то, что в его очереди стоит больше пакетов, чем у остальных.

Рис. 5.27. Циклическое справедливое обслуживание

Однако и с этим алгоритмом связана одна проблема: предоставляемая им пропускная способность напрямую зависит от размера пакета используемого хостом: большая часть предоставляется хостам с большими пакетами и меньшая — хостам с небольшими пакетами. В книге (Demers и др., 1990) предлагается улучшенная версия, в которой циклический опрос производится с целью выхватывания не пакета, а байта. Идея заключается в вычислении виртуального времени, то есть номера цикла, после которого отправка пакета завершится. Каждый цикл выхватывает один байт из каждой очереди, содержащей пакеты для отправки. После этого пакеты отправляются в том порядке, в котором они заканчивались при опросе очередей.

На рис. 5.28 показана работа алгоритма и примеры значений времени окончания отправки пакетов для трех потоков. Если длина пакета равна L, его отправка завершится через L циклов. Отправка пакета начинается либо сразу после отправки предыдущего пакета, либо в момент прибытия этого пакета, если очередь пуста.



Пакет	Время прибытия	Длина	Время окончания отправки	Порядок отправки
A	0	8	8	1
B	5	6	11	3
C	5	10	10	2
D	8	9	20	7
E	8	8	14	4
F	10	6	16	5
G	11	10	19	6
H	20	8	28	8

б

Рис. 5.28. Работа алгоритма: *а* — взвешенное справедливое обслуживание; *б* — время окончания отправки для пакетов

Данные таблицы (рис. 5.28, *б*) показывают, что первые два пакета в первых двух очередях прибывают в порядке *A*, *B*, *D*, *F*. Пакет *A* прибывает на нулевом цикле, а его длина равна 8 байтам, поэтому отправка завершается на 8 цикле. Точно так же отправка пакета *B* завершается на цикле 11. Пакет *D* прибывает в момент отправки *B*. Его отправка заканчивается через 9 циклов после окончания отправки *B*, и в этот момент время равно 20. Аналогичным образом получается, что время завершения отправки *F* равно 16. При условии отсутствия новых пакетов порядок окончания отправки пакетов будет таким: *A*, *B*, *F*, *D* (хотя *F* прибывает после *D*). Может случиться так, что в верхний поток поступит небольшой пакет, который закончит отправку раньше *D*. Но он обойдет *D* только в том случае, если его передача еще не началась. При справедливом обслуживании передача пакета, передаваемого в настоящий момент, не прерывается. Этот алгоритм отправляет пакеты целиком и потому является лишь приближением схемы побайтовой передачи. Но это очень хорошее приближение, поскольку в каждый момент времени передается ровно один пакет.

Проблема данного алгоритма заключается в том, что он дает всем хостам одинаковые приоритеты. Во многих случаях желательно предоставлять, например, видеосерверам большую пропускную способность, чем обычным файл-серверам, чтобы они могли посылать два или более байт за цикл. Такая модификация алгоритма называется **взвешенным справедливым обслуживанием (WFQ — Weighted Fair Queueing)**. Если количество байт, передаваемое за цикл, считать весом потока, *W*, то формула вычисления времени окончания передачи будет выглядеть так:

$$F_i = \max(A_i, F_{i-1}) + L_i/W,$$

где A_i — время прибытия, F_i — время окончания отправки, а L_i — длина i-го пакета. Нижняя очередь (рис. 5.28, *а*) имеет вес 2, поэтому ее пакеты передаются быстрее. Это хорошо видно, если посмотреть на значения времени окончания отправки (см. рис. 5.28, *б*). Еще один фактор, который необходимо учитывать, — это сложность реализации. Метод взвешенного справедливого обслуживания помещает пакеты в очередь, сортируя их по времени окончания отправки. Для *N* потоков это требует по меньшей мере $O(\log N)$ операций для каждого пакета, что является трудновыполнимым при большом количестве потоков на высокоскоростных маршрутизаторах. Существует упрощенная схема **DRR (deficit round robin)**, работающая гораздо эффективнее — всего за $O(1)$ операций для каждого пакета (Shreedhar и Varghese, 1995). Она широко применяется для алгоритма взвешенного справедливого обслуживания.

Существуют другие алгоритмы диспетчеризации. К ним относится, например, приоритетная диспетчеризация, при которой каждый пакет обладает приоритетом. Высокоприоритетные пакеты отправляются раньше, чем низкоприоритетные; последние помещаются в буфер. Пакеты с одинаковым приоритетом отправляются по принципу FIFO. Существенным недостатком этого алгоритма является то, что высокоприоритетные пакеты препятствует отправке низкоприоритетных, которые в результате могут ждать своей очереди бесконечно долго. С этой точки зрения взвешенное справедливое обслуживание — более удачный вариант. Если присвоить высокоприоритетной очереди большой вес (скажем, 3), высокоприоритетные пакеты будут в основном проходить по быстрой линии (так как пакетов с высоким приоритетом сравнительно немного), но одновременно с этим часть низкоприоритетных пакетов тоже будет передаваться.

Фактически бинарная приоритетная система представляет собой две очереди, одна из которых имеет бесконечный вес.

Наконец, существует алгоритм диспетчеризации, при котором у каждого пакета есть временной штамп, определяющий порядок отправки. В реализации, предложенной Clark и др. (1992), временной штамп регистрирует информацию о том, насколько пакет отстает от графика или опережает его, проходя через маршрутизаторы сети. Пакеты, ждущие отправки в очереди, обычно отстают от графика; пакеты, передаваемые в первую очередь, обычно опережают график. Передача пакетов в порядке временных штампов — эффективный способ ускорить отправку медленных пакетов и замедлить отправку быстрых. При такой диспетчеризации все пакеты доставляются с приблизительно равной задержкой.

5.4.4. Управление доступом

Теперь, когда мы познакомились с основными составляющими качества обслуживания (QoS), самое время собрать все это воедино. Гарантии качества обслуживания выполняются через управление доступом. До этого мы рассматривали управление доступом как средство борьбы с перегрузкой, что является гарантией производительности, пусть даже не очень эффективной. Здесь мы предъявляем к гарантиям более серьезные требования, но модель остается той же. Пользователь передает в сеть поток, предъявляя определенные требования к качеству обслуживания. Сеть принимает или отвергает этот поток в зависимости от своих возможностей и обязательств перед другими клиентами. Если сеть принимает поток, она должна заранее зарезервировать ресурсы, чтобы при передаче трафика по этому потоку клиент получил необходимое качество обслуживания.

Резервирование может производиться на всех маршрутизаторах, расположенных в узлах пути, по которому следуют пакеты. Если на каком-то из них ресурсы не зарезервированы, там может возникнуть затор, и гарантии качества обслуживания рискуют оказаться невыполненными. Многие алгоритмы маршрутизации выбирают наилучший путь от отправителя до получателя и направляют весь трафик по этому пути. Однако это может привести к тому, что часть потоков будут отвергнуты из-за недостатка ресурсов на узлах наилучшего пути. Тогда, чтобы выполнить свои обязательства перед клиентом, сеть выберет другой путь для отправки крупного потока. Это называется **QoS-маршрутизацией** (**QoS routing**). Описание этих методов можно найти в работе Chen и Nahrstedt (1998). Если заранее распределять трафик для одного и того же адреса по нескольким путям, найти дополнительные ресурсы будет гораздо проще. Маршрутизаторы могут выбирать пути с одинаковой стоимостью и использовать маршрутизацию, пропорциональную или эквивалентную емкостям исходящих связей. Однако существуют и более сложные алгоритмы (Nelakuditi и Zhang, 2002).

Для выбранного пути процесс принятия решения об обработке или игнорировании потока сложнее, нежели простое сравнение запрашиваемых потоком ресурсов (пропускной способности, буферной памяти, времени центрального процессора) с имеющимися. Во-первых, хотя многие приложения и знают свои требования по пропускной способности, они понятия не имеют, какой объем буферной памяти и сколько тактов работы процессора им требуется. Следовательно, нужен, по крайней мере, иной способ

описания потоков и определения ресурсов, выделяемых маршрутизатором. Вскоре мы вернемся к этому вопросу.

Далее, приложения весьма отличаются по толерантности в отношении пропущенного предельного срока обработки. Поэтому приложение должно выбрать один из типов гарантий, предлагаемых сетью: от строгих до предельно лояльных. При прочих равных условиях строгие гарантии пользовались бы самой большой популярностью. Но проблема в том, что они стоят достаточно дорого, так как ограничивают поведение сети в наихудшем случае. Для приложений обычно достаточно тех же гарантий, что и для большинства пакетов; кроме того, эти гарантии позволяют добавить дополнительный поток, используя фиксированные мощности.

Наконец, некоторые приложения могут поторговаться за параметры пакетов, а некоторые не могут. Скажем, проигрыватель видео, предоставляющий обычно 30 кадров/с, может согласиться работать на 25 кадрах/с, если для 30 не хватает пропускной способности. Аналогично можно настраивать количество пикселов на кадр, полосу пропускания для аудиоданных и другие свойства потоков различных приложений.

Поскольку в спор по поводу того, что делать с потоком, вовлечено много сторон (отправитель, приемник и все маршрутизаторы на пути между ними), поток необходимо описывать крайне аккуратно с помощью параметров, о которых можно дискутировать. Набор таких параметров называется **спецификацией потока** (**flow specification**). В типичном случае отправитель (например, сервер видеоданных) создает спецификацию потока, указывая параметры, которые он хотел бы использовать для аргументации. По мере того как эта спецификация распространяется по пути следования потока, содержащаяся в нем информация анализируется всеми маршрутизаторами, которые модифицируют параметры так, как считают нужным. Эти модификации могут быть направлены только на уменьшение потока (например, указываемая в спецификации скорость передачи данных может быть понижена, но не повышена). Когда спецификация доходит до приемника, становятся понятны окончательные параметры.

В качестве содержимого спецификации потока рассмотрим пример (табл. 5.3), базирующийся на RFC 2210 и RFC 2211 для интегрального обслуживания — технологии QoS, о которой мы поговорим в следующем разделе. В спецификации содержится пять параметров. Первые два, *скорость маркерного ведра* и *размер маркерного ведра*, позволяют вычислить максимальную скорость, которую отправитель может поддерживать в течение длительного времени, усредненную по большому временному отрезку, а также максимальный объем пачки, передаваемой за короткий промежуток времени.

Таблица 5.3. Пример спецификации потока

Параметр	Единицы измерения
Скорость маркерного ведра	байт/с
Размер маркерного ведра	байт
Пиковая скорость передачи данных	байт/с
Минимальный размер пакета	байт
Максимальный размер пакета	байт

Третий параметр, *пиковая скорость передачи данных*, это максимальная допустимая скорость даже для коротких промежутков времени. Отправитель ни в коем случае не должен превышать это значение.

Наконец, последние два параметра определяют минимальный и максимальный размеры пакетов, включая заголовки транспортного и сетевого уровней (например, TCP и IP). Минимальный размер удобен, поскольку обработка каждого пакета занимает какое-то, пусть даже очень малое время. Маршрутизатор, может быть, готов принимать 10 000 пакетов в секунду по 1 Кбайт каждый, но не готов обрабатывать 100 000 пакетов в секунду по 50 байт, несмотря на то что во втором случае скорость передачи данных меньше, чем в первом. Максимальный размер пакета не менее важен, но уже по другой причине. Дело в том, что существуют определенные внутрисетевые ограничения, которые ни в коем случае не должны быть превышены. Например, если путь потока лежит через Ethernet, то максимальный размер пакета будет ограничен 1500 байтами, независимо от того, какого размера пакеты могут поддерживать другие части сети.

Интересно, каким образом маршрутизатор преобразует спецификацию потока в набор определенных резервируемых ресурсов? На первый взгляд может показаться, что если один из каналов маршрутизатора работает со скоростью 1 Гбит/с, а средний размер пакета равен 1000 бит, он может обрабатывать 1 млн пакетов в секунду. Однако это не так, поскольку из-за статистических флуктуаций нагрузки передача будет периодически приостанавливаться на некоторое время. Если для выполнения всей работы канал должен использовать всю свою мощность, перерыв длиной в несколько секунд станет причиной завала, который ему никогда не удастся разгрести.

Однако даже если нагрузка несколько меньше теоретической емкости, все равно могут образовываться очереди и возникать задержки. Рассмотрим ситуацию, в которой пакеты прибывают нерегулярно со средней скоростью прибытия λ пакетов в секунду. Пакеты имеют случайную длину и могут передаваться по каналу со средней скоростью обслуживания μ пакетов в секунду. Предположим, что как скорость прибытия, так и скорость обслуживания имеют пуассоновское распределение (такие системы называются системами массового обслуживания M/M/1, где «M» означает «марковский процесс», который в данном случае является еще и пуассоновским). Тогда, используя теорию массового обслуживания, можно доказать, что средняя задержка T, присущая пакету, равна

$$T = \frac{1}{\mu} \times \frac{1}{1 - \lambda/\mu} = \frac{1}{\mu} \times \frac{1}{1 - \rho},$$

где $\rho = \lambda/\mu$ — коэффициент использования центрального процессора. Первый сомножитель $1/\mu$ — это задержка при отсутствии конкуренции. Второй сомножитель представляет собой дополнительную задержку, возникающую благодаря конкурентной борьбе с другими потоками. Например, если λ = 950 000 пакетов/с, а μ = 1 000 000 пакетов/с, тогда ρ = 0,95 и средняя задержка каждого пакета составляет 20 мкс вместо 1 мкс. Эти подсчеты учитывают и задержку доставки, и задержку обработки: при малом трафике отношение $\lambda/\mu \approx 0$. Если на пути потока стоят, скажем, 30 маршрутизаторов, то одна только задержка обслуживания составит 600 мкс.

Один из методов соотнесения характеристик потока и ресурсов маршрутизатора, необходимых для выполнения гарантий пропускной способности и времени задержки, был предложен Parekh и Gallagher (1993, 1994). Согласно этому методу, отправитель должен сформировать трафик с помощью маркерных ведер (*R*, *B*), а маршрутизаторы должны использовать взвешенное справедливое обслуживание. Каждому потоку присваивается вес *W*, достаточный для того, чтобы опустошить маркерное ведро со скоростью *R* (рис. 5.29). Если, например, скорость потока составляет 1 Мбит/с, а мощности маршрутизатора и исходящей связи равны 1 Гбит/с, вес потока должен превышать одну тысячную от общей суммы весов всех потоков этого маршрутизатора для исходящей связи. Это обеспечит потоку минимальную пропускную способность. Если поток не может получить необходимую ему скорость, он не будет допущен в сеть.

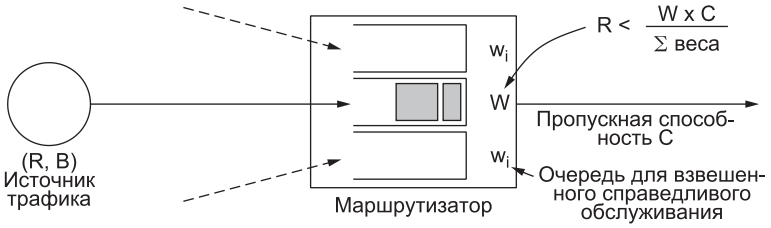


Рис. 5.29. Гарантии пропускной способности и задержки с использованием маркерных ведер и взвешенного справедливого обслуживания

Наибольшее время ожидания в очереди для данного потока является функцией максимальной емкости маркерного ведра. При равномерном трафике пакеты проходят через маршрутизатор с той же скоростью, с которой прибывают. При этом не происходит никаких задержек (не считая эффектов пакетирования). С другой стороны, если трафик передается пачками, то пачка максимального размера, B, может прибыть на маршрутизатор целиком. Тогда максимальная задержка, *D*, будет равна времени прохождения пакета через маршрутизатор при фиксированной пропускной способности, или *B*/*R* (опять же, не считая эффектов пакетирования). Если этот показатель слишком высокий, поток может запросить бóльшую пропускную способность.

Такие гарантии являются достаточно строгими: маркерные ведра ограничивают неравномерность трафика, а справедливое обслуживание изолирует пропускную способность, выделяемую для разных потоков. Это значит, что гарантии пропускной способности и задержки для потока будут выполнены, даже если другие потоки будут копить трафик и отправлять его одновременно.

Более того, результат не зависит от количества маршрутизаторов в узлах пути и от топологии сети. Каждый поток получает свою минимальную пропускную способность благодаря тому, что она зарезервирована на каждом маршрутизаторе. Более тонко дело обстоит с максимальной задержкой. В наихудшем случае, если крупный объем трафика поступит на первый маршрутизатор и будет соревноваться с трафиком других потоков, максимальная задержка будет равна *D*. Однако после этого трафик станет более равномерным, и поэтому на следующих маршрутизаторах такой задержки уже не будет. В результате общая задержка в очереди не будет превышать *D*.

5.4.5. Интегральное обслуживание

В 1995–1997 годах проблемная группа проектирования сети Интернет (IETF) прилагала множество усилий по продвижению архитектуры потокового мультимедиа. В результате появилось две дюжины документов RFC, начинающихся с префикса RFC и включающих порядковые номера 2205–2212. Общее название этих трудов — **потоковые алгоритмы** или **интегральное обслуживание (integrated services)**. Предлагаемая технология предназначена как для одноадресных, так и для многоадресных приложений. Примером первых может быть видеоклип на сайте новостей, доставляемый в виде потока пользователю, пожелавшему посмотреть его. Пример вторых — набор станций цифрового телевидения, осуществляющих широковещательное распространение своих программ в виде потоков IP-пакетов. Данной услугой может пользоваться большое число абонентов, расположенных в разных географических точках. Ниже мы более подробно рассмотрим многоадресную рассылку, поскольку одноадресная передача — это лишь особый случай многоадресной.

Во многих приложениях с многоадресной маршрутизацией группы пользователей могут меняться динамически. Например, люди могут подключаться к участию в видеоконференциях, но со временем это занятие им надоедает и они переключаются на мыльные оперы или спортивные каналы. В данном случае стратегия предварительного резервирования пропускной способности не совсем подходит, потому что при этом каждому источнику пришлось бы запоминать все изменения в составе аудитории. В системах, предназначенных для передачи телевизионного сигнала миллионам абонентов, такой подход и вовсе не годится.

RSVP — протокол резервирования ресурсов

Главная составляющая архитектуры интегрального обслуживания, открытая для пользователей сети, называется **протоколом резервирования ресурсов (RSVP — Resource reSerVation Protocol)**. Он описывается в документах RFC 2205–2210. Как следует из названия, протокол предназначен для резервирования ресурсов; другие протоколы применяются для описания передачи данных. RSVP позволяет нескольким отправителям посылать данные нескольким группам абонентов, разрешает отдельным получателям переключать каналы и оптимизирует использование пропускной способности, в то же время устраняя возникновение перегрузки.

Простейшая форма этого протокола использует многоадресную маршрутизацию с применением связующих деревьев, обсуждавшуюся ранее. Каждой группе назначается групповой адрес. Чтобы послать данные группе, отправитель помещает ее адрес в заголовки пакетов. Затем стандартный алгоритм многоадресной маршрутизации строит связующее дерево, покрывающее всех членов группы. Алгоритм маршрутизации не является частью протокола RSVP. Единственное отличие от нормальной многоадресной маршрутизации состоит в том, что группе периодически рассылается дополнительная информация, с помощью которой маршрутизаторы обновляют определенные структуры данных.

Для примера рассмотрим сеть, показанную на рис. 5.30, *а*. Хосты 1 и 2 являются многоадресными передатчиками, а хосты 3, 4 и 5 — многоадресными приемниками.

В данном примере передатчики и приемники разделены, однако в общем случае эти два множества могут перекрываться. Деревья многоадресной рассылки для хостов 1 и 2 показаны на рис. 5.30, *б* и *в* соответственно.

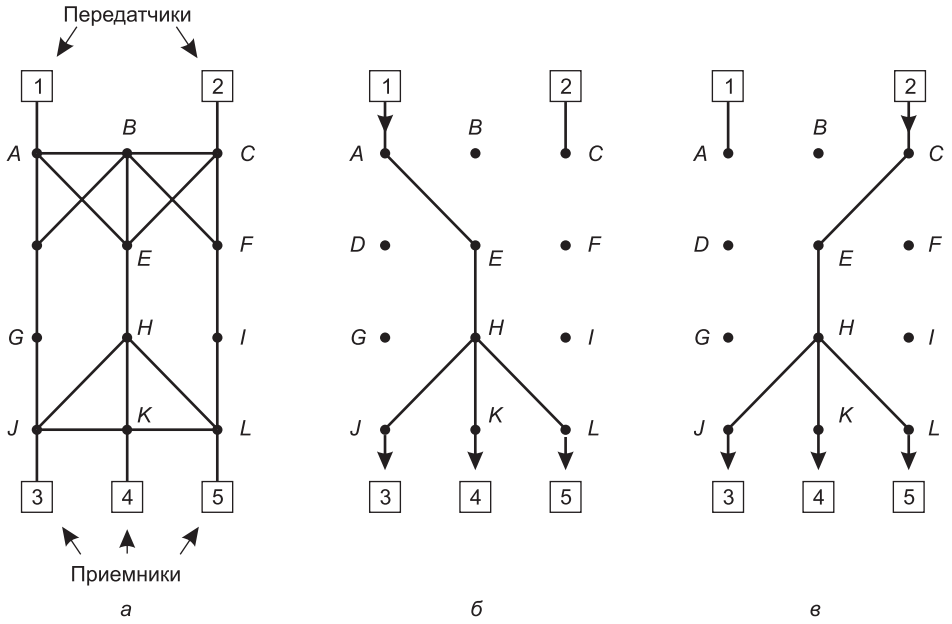


Рис. 5.30. Протокол резервирования ресурсов: *а* — сеть; *б* — связующее дерево многоадресной рассылки для хоста 1; *в* — связующее дерево многоадресной рассылки для хоста 2

Для улучшения качества приема и устранения перегрузки каждый получатель в группе может послать передатчику (вверх по дереву) запрос на резервирование. Запрос продвигается, используя обсуждавшийся ранее алгоритм обратного пути. На каждом транзитном участке маршрутизатор замечает запрос и резервирует необходимую пропускную способность. В предыдущем разделе мы увидели, как это делает диспетчер взвешенного справедливого обслуживания. Если пропускной способности недостаточно, он отвечает сообщением об ошибке. К тому моменту, как запрос доходит до передатчика, пропускная способность зарезервирована вдоль всего пути от отправителя к получателю.

Пример резервирования показан на рис. 5.31, *а*. Здесь хост 3 запросил канал к хосту 1. После создания канала поток пакетов от хоста 1 к хосту 3 может течь, не боясь попасть в затор. Рассмотрим, что произойдет, если теперь хост 3 зарезервирует канал к другому передатчику, хосту 2, чтобы пользователь мог одновременно смотреть две телевизионные программы. Зарезервированный второй канал показан на рис. 5.31, *б*. Обратите внимание: между хостом 3 и маршрутизатором *E* требуется наличие двух отдельных каналов, так как передаются два независимых потока.

Наконец, на рис. 5.31, *в* хост 5 решает посмотреть программу, передаваемую хостом 1, и также резервирует себе канал. Сначала требуемая пропускная способность резервируется до маршрутизатора *H*. Затем этот маршрутизатор замечает, что у него

уже есть канал от хоста 1, поэтому дополнительное резервирование выше по дереву не требуется. Обратите внимание на то, что хосты 3 и 5 могут запросить различную пропускную способность (например, у хоста 3 маленький экран, поэтому ему не нужно высокое разрешение), следовательно, маршрутизатор *H* должен зарезервировать пропускную способность, достаточную для удовлетворения самого жадного получателя.

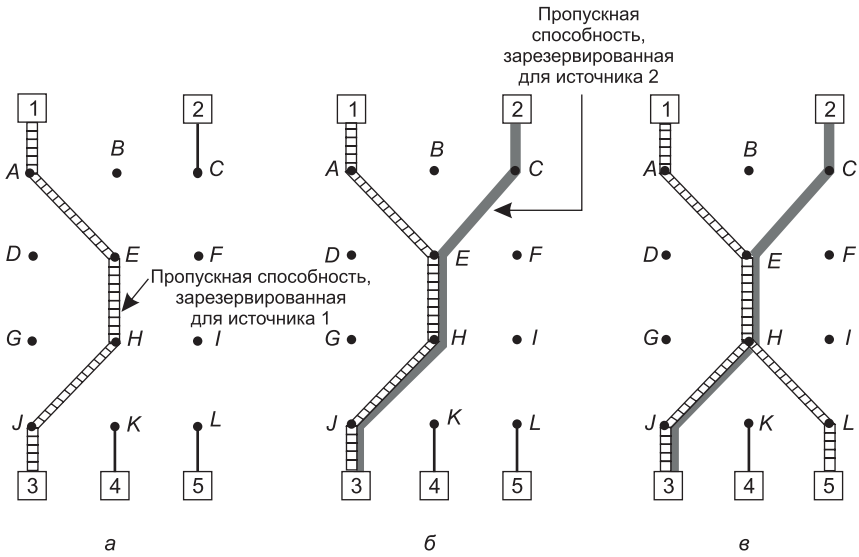


Рис. 5.31. Пример резервирования: *а* — хост 3 запрашивает канал к хосту 1; *б* — затем хост 3 запрашивает второй канал к хосту 2; *в* — хост 5 запрашивает канал к хосту 1

При подаче запроса на резервирование получатель может (по желанию) указать один или несколько источников, от которых он хотел бы получать сигнал. Он также может указать, является ли выбор источников фиксированным в течение времени резервирования, или он оставляет за собой право сменить источники. Данные сведения используются маршрутизаторами для оптимизации планирования пропускной способности. В частности, двум приемникам выделяется общий путь, только если они соглашаются не менять впоследствии свой источник.

В основе такой динамической стратегии лежит полная независимость зарезервированной пропускной способности от выбора источника. Получив зарезервированную пропускную способность, получатель может переключаться с одного источника на другой, сохраняя часть существующего пути, годящуюся для нового источника. Например, если хост 2 передает несколько видеопотоков в реальном времени (например, при многоканальном телевещании), хост 3 может переключаться между ними по желанию, не меняя своих параметров резервирования: маршрутизаторам все равно, какую программу смотрит получатель.

5.4.6. Дифференцированное обслуживание

Потоковые алгоритмы способны обеспечивать хорошее качество обслуживания одного или нескольких потоков за счет резервирования любых необходимых ресурсов на протяжении всего маршрута. Однако есть у них и недостаток. Им требуется предварительная договоренность при установке канала для каждого потока. Это не позволяет в достаточной мере расширять систему, в которой применяются данные алгоритмы. Скажем, в системах с тысячами или миллионами потоков интегральное обслуживание применить не удастся. Кроме того, потоковые алгоритмы работают с внутренней информацией о каждом потоке, хранящейся в маршрутизаторах, что делает их уязвимыми к выходу из строя маршрутизаторов. Наконец, программные изменения, которые нужно производить в маршрутизаторах, довольно значительны и связаны со сложными процессами обмена между маршрутизаторами при установке потоков. В результате с развитием интегрального обслуживания эти алгоритмы и их аналоги используются крайне мало.

По этим причинам IETF был создан упрощенный подход к повышению качества обслуживания. Его можно реализовать локально в каждом маршрутизаторе без предварительной настройки и без включения в процесс всех устройств вдоль маршрута. Подход известен как **ориентированное на классы** (в отличие от ориентированного на потоки) качество обслуживания. Проблемной группой IETF была стандартизована специальная архитектура под названием **дифференцированное обслуживание (differentiated services)**, описываемая в документах RFC 2474, RFC 2475 и во многих других. Ниже мы опишем ее.

Дифференцированное обслуживание может предоставляться набором маршрутизаторов, образующих административный домен (например, интернет-провайдер или телефонную компанию). Администрация определяет множество классов обслуживания и соответствующие правила маршрутизации. Пакеты, приходящие от абонента, пользующегося дифференцированным обслуживанием, получают метку с информацией о классе. Эти сведения записываются в поле *Дифференцированное обслуживание* пакетов IPv4 и IPv6 (см. раздел 5.6). Классы определяют **пошаговое поведение (per hop behaviors)**, так как они отвечают за то, что будет происходить с пакетом на маршрутизаторе, а не во всей сети. Пакетам с пошаговым поведением предоставляется улучшенное обслуживание (например, премиум-обслуживание) по сравнению с остальными пакетами (обычное обслуживание). К трафику класса могут предъявляться определенные требования, касающиеся его формы. Например, от него может потребоваться, чтобы он представлял собой «дырявое ведро» с определенной скоростью просачивания данных через «дырочку». Оператор, привыкший брать деньги за все, может взимать дополнительную плату за каждый пакет, обслуживаемый по высшему классу, либо может установить абонентскую плату за передачу N таких пакетов в месяц. Обратите внимание: здесь нет никакой предварительной настройки, резервирования ресурсов и трудоемких согласований параметров для каждого потока, как при интегральном обслуживании. Это делает дифференцированное обслуживание относительно просто реализуемым.

Обслуживание, ориентированное на классы, возникает и в других областях. Например, службы доставки посылок могут предлагать на выбор несколько уровней

обслуживания: доставка на следующий день, через день или через два дня. В самолетах обычно бывают первый класс, бизнесс-класс и второй класс. То же самое касается поездов дальнего следования. Даже в парижской подземке до недавних пор были вагоны двух разных классов. Что касается нашей тематики, то классы пакетов могут отличаться друг от друга задержками, флуктуациями времени доставки, вероятностью быть проигнорированными в случае коллизии, а также другими параметрами (коих, впрочем, не больше, чем у кадров Ethernet).

Чтобы разница между обслуживанием, ориентированным на классы, и обслуживанием, ориентированным на потоки, стала яснее, рассмотрим пример: интернет-телефонию. При потоковом алгоритме обслуживания каждому телефонному соединению предоставляются собственные ресурсы и гарантии. При обслуживании, ориентированном на классы, все телефонные соединения совместно получают ресурсы, зарезервированные для телефонии данного класса. Эти ресурсы, с одной стороны, не может отнять никто извне (соединения других классов, потоки систем просмотра веб-страниц и т. п.), с другой стороны, ни одно телефонное соединение не может получить никакие ресурсы в частное пользование.

Срочная пересылка

Выбор классов обслуживания зависит от решения оператора, однако поскольку пакеты зачастую необходимо пересылать между сетями, управляемыми разными операторами, проблемной группой IETF были определены классы обслуживания, не зависящие от сети. Простейший из них — класс **срочной пересылки (expedited forwarding)**, с него и начнем. Он описывается документом RFC 3246.

Итак, идея, на которой построена срочная пересылка, очень проста. Существует два класса обслуживания: обычный и срочный. Ожидается, что подавляющая часть объема трафика будет использовать обычный класс обслуживания. Однако есть ограниченная доля пакетов, которые необходимо передавать в срочном порядке. Их нужно пересылать между сетями так, будто кроме них в сети больше нет вообще никаких пакетов. Тогда они получат обслуживание с низкими потерями, низкой задержкой и низкой флуктуацией — как раз то, что нужно для IP-телефонии. Графическое представление такой двухканальной системы показано на рис. 5.32. Имейте в виду, что физическая линия здесь только одна. Два логических пути — это своеобразный способ резервирования пропускной способности для разных классов обслуживания, а вовсе не протягивание второго провода для передачи данных рядом с основным.

Данную стратегию можно реализовать следующим образом. Пакеты разделяются на обычные и срочные, после чего они получают соответствующие отметки. Это может выполнять хост-источник или входной (первый) маршрутизатор. Преимущество первого варианта в том, что источник располагает большей информацией о распределении пакетов по потокам. Классификация пакетов может производиться сетевым ПО или операционной системой, что позволяет избежать изменений в существующих приложениях. Например, сейчас VoIP-пакеты все чаще помечаются хостами как срочные. Если такие пакеты передаются по корпоративной сети или через интернет-провайдера, поддерживающих срочную пересылку, они оказываются в привилегированном положении. В противном случае отметка не будет иметь никаких негативных последствий.

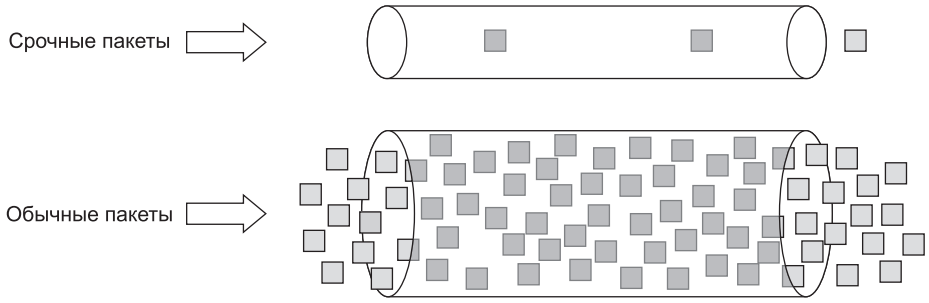


Рис. 5.32. Срочные пакеты движутся по свободной от трафика сети

Конечно же, если пакет получает метку на хосте, входной маршрутизатор, скорее всего, захочет проверить, не выходит ли объем срочного трафика за установленные пределы. В сети маршрутизаторы могут использовать две очереди для каждой исходящей линии — для обычных и для срочных пакетов. Прибывший пакет ставится в очередь, соответствующую его классу обслуживания. Срочная очередь получает более высокий приоритет, чем обычная; это можно сделать, к примеру, с помощью диспетчера приоритетов. Таким образом, срочный трафик будет думать, что сеть пустынна и безжизненна, хотя на самом деле она может быть загружена чрезвычайно сильно.

Гарантированная пересылка

Более совершенная схема управления классами обслуживания называется схемой **гарантированной пересылки (assured forwarding)**. Эта стратегия описывается в документе RFC 2597. Гарантированная пересылка подразумевает наличие четырех классов приоритетов, каждый из которых обладает своими ресурсами. Первые три класса можно назвать золотым, серебряным и бронзовым. Кроме того, определены три класса игнорирования пакетов, попавших в затор (низкий, средний и высокий). Итого получается 12 сочетаний, то есть 12 классов обслуживания.

На рис. 5.33 показан один из способов обработки пакетов при гарантированной пересылке. На первом шаге пакеты разбиваются на четыре класса приоритетов. Как и раньше, эта процедура может выполняться на хосте-источнике (как показано на рисунке) или на первом маршрутизаторе. Скорость высокоприоритетных пакетов может быть ограничена оператором в рамках соглашения о предоставлении услуг.

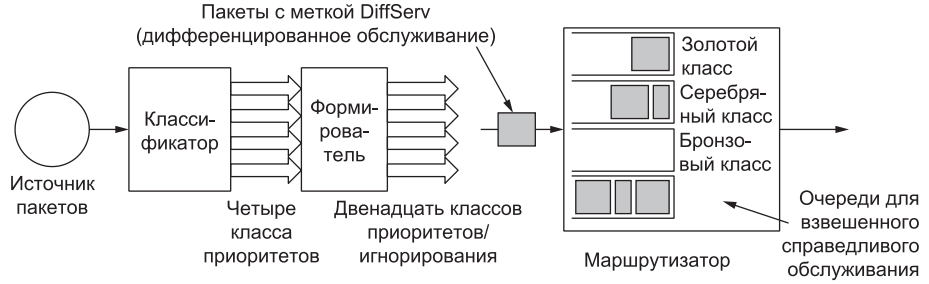


Рис. 5.33. Возможная реализация гарантированной пересылки потока данных

Следующий шаг — определение классов игнорирования пакетов. Для этого пакеты каждого класса приоритетов проходят проверку с помощью маркерного ведра или похожей схемы. При этом пакетам небольшого размера присваивается низкий класс игнорирования, пакетам среднего размера — средний класс, а пакетам большого размера — высокий. Информация о классах приоритетов и игнорирования кодируется в каждом пакете.

Наконец пакеты проходят обработку на маршрутизаторах сети, где диспетчер определяет их классы. Чаще всего для четырех классов приоритетов используется метод взвешенного справедливого обслуживания: чем выше класс, тем выше вес. В результате высокоприоритетные пакеты получают большую часть пропускной способности, однако отправка низкоприоритетных пакетов не останавливается. К примеру, вес каждого класса приоритетов может быть вдвое больше, чем вес более низкого класса. В пределах одного класса приоритетов пакеты с высоким классом игнорирования удаляются в первую очередь. Это может понадобиться, например, при случайном раннем обнаружении (RED), о котором мы говорили в разделе 5.3.5. Случайное раннее обнаружение начнет удалять пакеты еще до того, как в буфере маршрутизатора закончится место. Пакеты с низким классом игнорирования все еще будут приниматься, а с высоким — отвергаться.

5.5. Объединение сетей

До сих пор мы неявно предполагали наличие единой однородной сети, в которой каждая машина использует одни и те же протоколы на каждом уровне. К сожалению, данное предположение слишком оптимистично. Существует множество различных сетей, включая персональные, локальные, региональные и глобальные. Мы уже говорили о сети Ethernet, кабельном Интернете, стационарных и мобильных телефонных сетях, стандартах 802.11, 802.16 и др. В этих сетях на каждом уровне широко применяются многочисленные и разнообразные протоколы. В следующих разделах будет уделено особое внимание вопросам, возникающим при объединении двух или более сетей, формирующих **интерсеть (internetwork)**, или проще — **интернет (internet)**[1].

Если бы все использовали одну и ту же сетевую технологию, объединить сети было бы гораздо проще. В большинстве случаев доминирующая сеть (например, Ethernet) существует. Некоторые ученые считают разнообразие типов сетей временным явлением, которое скоро перестанет иметь место, как только все, наконец, поймут, как замечательна сеть [вставьте свою любимую сеть]. Не стоит на это рассчитывать. Время показало, что такие рассуждения — всего лишь принятие желаемого за действительное. Разным типам сетей приходится сталкиваться с разными трудностями, поэтому, например, Ethernet и спутниковые сети всегда будут отличаться. К примеру, использование уже существующих систем (кабельной, телефонной сети и сети электропитания) при создании сетей передачи данных приводит к дополнительным

[1] Соответственно, Интернет (Internet, или, как иногда говорят «сеть Интернет» или «сеть Internet») — это самая известная интерсеть или, «говоря проще» — самый известный интернет. — *Примеч. ред.*

ограничениям, которые изменяют свойства сети. Но при этом однородность должна сохраняться.

Если разные сети будут существовать всегда, то лучше бы нам вообще никогда не потребовалось их объединять. Но это крайне маловероятно. Боб Меткальф выдвинул такой принцип: ценность сети, состоящей из N узлов, пропорциональна числу соединений между узлами, или N^2 (Gilder, 1993). Это значит, что большие сети всегда ценнее маленьких, и поэтому объединение сетей всегда будет иметь смысл.

Главный пример такого объединения — Интернет. Целью объединения всех этих сетей является предоставление пользователям возможности общаться с пользователями любой другой из этих сетей. Стоимость оплаты интернет-услуг часто зависит от доступной пропускной способности. Но на самом деле вы платите за возможность обмена пакетами с другими хостами, также подключенными к Интернету. Интернет не был бы так популярен, если бы пакеты можно было отправлять только хостам, находящимся в том же городе.

Поскольку сети зачастую различаются довольно сильно, передача пакетов из одной сети в другую далеко не всегда является простой задачей. Помимо проблем с неоднородностью, нам придется решать вопросы, возникающие вследствие увеличения размера такой интерсети. Чтобы понять, с чем мы можем столкнуться, необходимо сначала узнать, чем сети отличаются друг от друга. После этого мы рассмотрим подход, успешно применяющийся в IP (Internet Protocol), протоколе сетевого уровня сети Интернет. Здесь же мы расскажем о методах туннелирования в сетях, маршрутизации в интерсетях и фрагментации пакетов.

5.5.1. Различия сетей

Сети могут отличаться друг от друга довольно сильно и по разным параметрам. Некоторые из параметров, такие как методы модуляции или форматы кадров, нас сейчас не интересуют, поскольку они относятся к физическому и канальному уровням. В таблице 5.4 приведен список некоторых параметров, которые могут встретиться на сетевом уровне. Именно сглаживание этих различий делает обеспечение работы объединенной сети значительно более сложным делом, чем обеспечение работы одной сети.

Когда пакетам приходится пересекать несколько сетей, может возникнуть много проблем, связанных с интерфейсами между сетями. Во-первых, должна существовать возможность пересылки пакета от отправителя получателю. Что делать, если отправитель находится в сети Ethernet, а получатель — в сети WiMAX? Даже если мы можем задать адрес назначения WiMAX в сети Ethernet, пакеты нужно будет переправить из сети, не требующей соединения, в сеть, ориентированную на соединение. Тогда может понадобиться срочно создать новое соединение, что приведет к задержке и неэффективному использованию ресурсов, так как это соединение не будет активно использоваться.

Существует еще много различий, к которым необходимо приспособиться. Как, например, передать пакет группе, некоторые члены которой находятся в сети, не поддерживающей многоадресную рассылку? Различия в максимальном размере пакетов в разных сетях также составляют большую головную боль. Как передать 8000-байтовый пакет по сети, в которой максимальный размер пакета равен 1500 байтам? Когда

пакеты из ориентированной на соединение сети должны пересечь не требующую соединений сеть, их порядок может нарушиться. Для отправителя это может оказаться (неприятной) неожиданностью — впрочем, как и для получателя.

Таблица 5.4. Некоторые аспекты отличия сетей

Аспект	Возможные значения
Предлагаемый сервис	Ориентированные на соединение или не требующие соединения
Адресация	Разного размера, плоская или иерархическая
Широковещание	Присутствует или отсутствует (а также многоадресная рассылка)
Размер пакета	У каждой сети есть свой максимум
Упорядочивание	Упорядоченная и неупорядоченная доставка
Качество обслуживания	Может присутствовать и отсутствовать. Много разновидностей
Надежность	Различные уровни потерь
Безопасность	Правила секретности, шифрование и т. д.
Параметры	Различные тайм-ауты, спецификация потока и др.
Тарификация	По времени соединения, за пакет, побайтно или никак

С различиями таких типов все же можно справиться. Например, шлюз на стыке двух сетей может имитировать многоадресную рассылку, создавая копии пакетов для разных адресов назначения. Крупные пакеты могут разбиваться на части, а затем снова объединяться. Принимающее устройство может помещать пакеты в буфер, а затем доставлять их в правильном порядке.

Однако сети могут различаться и в таких аспектах, с которым справиться гораздо труднее. Самый яркий пример — качество обслуживания. Если одна сеть предоставляет хороший уровень обслуживания, а другая — наилучший уровень из возможных, невозможно создать сквозную пропускную способность и задержку. На самом деле это возможно, только если вторая сеть работает в медленном режиме или почти не используется, что маловероятно. Трудности вызывают и механизмы безопасности, но как минимум шифрование в целях конфиденциальности и целостности данных можно обеспечить в тех сетях, где они не поддерживаются. И наконец, различия в тарификации могут стать причиной неожиданно высоких счетов за обычные операции — ситуация, в которой часто оказываются пользователи мобильных телефонов в зоне роуминга.

5.5.2. Способы объединения сетей

Существует два основных способа соединения разных сетей. Можно создать специальные устройства, которые умеют конвертировать пакеты из любой сети в любую другую. Или, как хорошие специалисты в этой области, мы можем решить эту проблему так: добавив еще один уровень косвенной адресации, мы создадим надсетевой уровень. В обоих случаях устройства должны помещаться в граничных областях сетей.

Уже давно идею создания общего слоя, сглаживающего различия существующих на данный момент сетей, выдвинули Серф и Кан (1974). Этот подход имел невероятный успех и нашел свое применение в протоколах IP и TCP. Сейчас, почти сорок лет спустя, IP составляет основу современной сети Интернет. За это достижение в 2004 году Серф и Кан были удостоены премии Тьюринга, которую неофициально называют Нобелевской премией в области информатики. IP использует универсальный формат пакетов, который распознается всеми маршрутизаторами и может быть передан по практически любой сети. Действие IP распространяется также и на телефонные сети. Кроме того, сейчас IP работает в сенсорных сетях и на малых устройствах, хотя раньше считалось, что такая поддержка невозможна из-за ограничений на ресурсы.

Мы уже говорили о нескольких типах устройств для соединения сетей. Среди них повторители, концентраторы, мосты, коммутаторы и шлюзы. Повторители и концентраторы просто переносят биты с одного кабеля на другой. Чаще всего это аналоговые устройства, ничего не смыслящие в протоколах вышележащих уровней. Мосты и коммутаторы работают на канальном уровне. Они могут использоваться для построения сетей, осуществляя по ходу дела минимальные преобразования протоколов, например, между сетями Ethernet со скоростями 10, 100 и 1000 Мбит/с. В этом разделе в центре нашего внимания будут устройства сетевого взаимодействия, работающие на сетевом уровне, то есть маршрутизаторы. Шлюзы, являющиеся высокоуровневыми интеркоммуникационными устройствами, будут рассмотрены позднее.

Давайте для начала посмотрим, как взаимодействие через общий сетевой слой может использоваться для объединения разнородных сетей. На рис. 5.34, *а* изображена интерсеть, состоящая из сетей 802.11, MPLS и Ethernet. Пусть источник в сети 802.11 хочет отправить пакет приемнику в сети Ethernet. Так как технологии отправки в этих сетях различны, и вдобавок пакету придется пройти через сеть другого типа (MPLS), на границах сетей пакеты должны быть дополнительно обработаны.

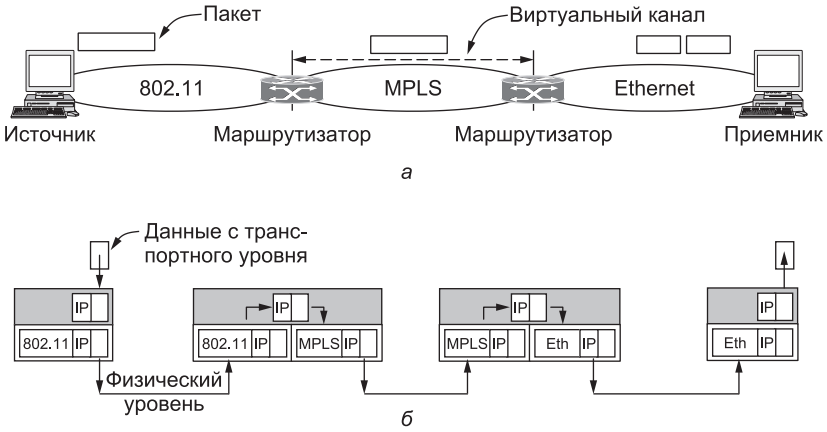


Рис. 5.34. Взаимодействие через сетевой слой: *а* — пакет проходит через разные сети; *б* — выполнение протоколов на сетевом и канальном уровне

Поскольку обычно разные сети используют разные способы адресации, пакет содержит адрес сетевого слоя, который может идентифицировать любой хост любой из

трех сетей. Сначала пакет приходит на границу сетей 802.11 и MPLS. Сеть 802.11 не требует соединения, а MPLS, наоборот, ориентирована на соединение. Это значит, что необходимо создать виртуальный канал. Когда пакет пройдет по этому каналу, он достигнет границы сети Ethernet. На этом этапе пакет может оказаться слишком большим, так как 802.11 работает с кадрами большего размера, чем Ethernet. В таком случае пакет разделяется на фрагменты, каждый из которых отправляется по отдельности. Фрагменты снова собираются вместе по прибытии на адрес назначения. Так заканчивается путешествие пакета.

Выполнение протоколов для этого путешествия показано на рис. 5.34, *б*. Отправитель получает данные от транспортного уровня и создает пакет с заголовком общего сетевого уровня — в данном случае IP. Этот заголовок содержит адрес конечного пункта назначения, который определяет, что пакет должен быть отправлен через первый маршрутизатор. Пакет вставляется в кадр 802.11 с адресом первого маршрутизатора, после чего он передается. На маршрутизаторе из его поля данных извлекается пакет. Далее маршрутизатором анализируется содержащийся в пакете IP-адрес. Этот адрес нужно отыскать в таблице маршрутизации. В соответствии с ним принимается решение об отправке пакета на второй маршрутизатор. Чтобы пакет прошел эту часть пути, нужно добавить виртуальный канал MPLS, ведущий ко второму маршрутизатору, а пакет должен быть упакован в кадр с заголовками MPLS. На противоположном конце заголовок MPLS удаляется, а на основании адреса определяется следующий транзитный участок сетевого уровня. Этот участок ведет к конечному адресу назначения. Так как пакет является слишком длинным для сети Ethernet, он разделяется на две части, каждая из которых помещается в поле данных кадра Ethernet и передается на адрес назначения. Там заголовки кадров считываются, и содержимое исходного пакета восстанавливается. Пакет достиг пункта назначения. Следует отметить, что между случаем коммутации (установки моста) и маршрутизации есть существенная разница. При применении маршрутизатора пакет извлекается из кадра, и для принятия решения используется адрес, содержащийся именно в пакете. Коммутатор (мост) пересылает весь пакет, обосновывая свое решение значением MAC-адреса. Коммутаторы не обязаны вникать в подробности устройства протокола сетевого уровня, с помощью которого производится коммутация. А маршрутизаторы — обязаны.

К сожалению, объединение сетей — не такая простая задача, как может показаться. Когда были внедрены мосты, предполагалось, что они будут соединять разные типы сетей или, по крайней мере, разные типы локальных сетей, преобразовывая кадры одной ЛВС в кадры другой. Однако на практике это не работает. Причина проста: невозможно справиться с отличиями в свойствах ЛВС, такими как максимальный размер пакета и наличие/отсутствие классов приоритетов. Поэтому сейчас мосты используются в основном для объединения сетей одного типа на канальном уровне; для объединения разных сетей на сетевом уровне используются маршрутизаторы.

Объединение сетей нашло свое применение в построении крупных сетей. Однако условием являлось наличие общего сетевого уровня. За многие годы существования компьютерных сетей появилось множество различных протоколов. Заставить всех согласиться применять только один формат практически невозможно, особенно учитывая тот факт, что каждая компания считает самым большим своим достижением изобретение, внедрение и раскручивание собственного формата. Помимо IP, кото-

рый на данный момент является практически универсальным сетевым протоколом, существуют IPX, SNA и AppleTalk. Ни один из этих протоколов не используется повсеместно. Новые протоколы будут появляться всегда. Наиболее подходящий пример сейчас — протоколы IPv4 и IPv6. Притом что оба они являются версиями IP, они не совместимы (иначе не было бы необходимости в разработке IPv6).

Маршрутизатор, поддерживающий несколько протоколов, называется мультипротокольным маршрутизатором (**multiprotocol router**). Он должен либо преобразовывать протоколы, либо обеспечивать соединение на уровне протокола более высокого уровня. Ни один из двух вариантов не отвечает всем требованиям сети. Соединение на более высоком уровне, например с использованием TCP, предполагает, что TCP должен быть реализован во всех сетях (что далеко не всегда так). Более того, в таком случае сетью могут пользоваться только приложения, использующие TCP (к которым не относятся многие программы, работающие в реальном времени).

Альтернативный вариант — преобразование протоколов различных сетей. Однако если форматы пакетов не являются близкими родственниками с одинаковыми информационными полями, такое преобразование всегда будет неполным и часто обречено на ошибку. К примеру, длина IPv6-адресов составляет 128 бит. И как бы ни старался маршрутизатор, такой адрес ни за что не поместится в 32-битное поле адреса IPv4. Проблема использования IPv4 и IPv6 в одной сети оказалась серьезным препятствием к внедрению IPv6. (И честно говоря, по этой же причине потребителей так и не удалось убедить в том, что они должны использовать именно IPv6.) Но еще более серьезные проблемы возникают, если требуется выполнять преобразования между принципиально разными протоколами — например, между ориентированным на соединение протоколом и протоколом, не требующим соединения. Поэтому такие преобразования практически никогда не выполняются. Пожалуй, и IP выигрывает только за счет того, что служит чем-то вроде наименьшего общего делителя. Он требует немногого от сетей, однако предоставляет только наилучший уровень обслуживания из возможных.

5.5.3. Туннелирование

Объединение сетей в общем случае является исключительно сложной задачей. Однако есть частный случай, реализация которого вполне осуществима даже для разных сетевых протоколов. Это случай, при котором хост-источник и хост-приемник находятся в сетях одного типа, но между ними находится сеть другого типа. Например, представьте себе международный банк, у которого имеется одна сеть IPv6 в Париже и такая же сеть в Лондоне, а между ними находится IPv4, как показано на рис. 5.35.

Метод решения данной проблемы называется **туннелированием (tunneling)**. Чтобы послать IP-пакет хосту в Лондоне, хост в Париже формирует пакет, содержащий лондонский IPv6-адрес и отправляет его на мультипротокольный маршрутизатор, соединяющий парижскую сеть IPv6 и сеть IPv4. Получив пакет IPv6, маршрутизатор помещает его в другой пакет с IPv4-адресом маршрутизатора, соединяющего сеть IPv4 и лондонскую сеть IPv6. Когда пакет попадает на этот адрес, лондонский многопротокольный маршрутизатор извлекает исходный IPv6-пакет и посылает его дальше на хост назначения.

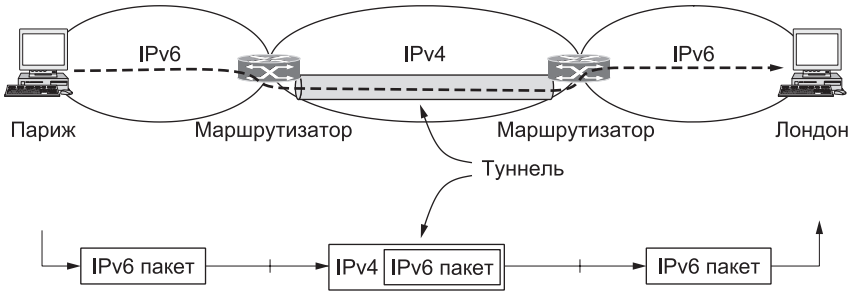


Рис. 5.35. Туннелирование пакета из Парижа в Лондон

Путь через сеть IPv4 можно рассматривать как большой туннель, простирающийся от одного многопротокольного маршрутизатора до другого. IPv6-пакет, помещенный в красивую упаковку, просто перемещается от одного конца туннеля до другого. Ему не нужно беспокоиться о взаимодействии с сетью IPv4. Это же касается и хостов Парижа и Лондона. Переупаковкой пакета и переадресацией занимаются многопротокольные маршрутизаторы. Для этого им нужно уметь разбираться в IPv6- и IPv4-пакетах. В результате весь путь от одного многопротокольного маршрутизатора до другого работает как один транзитный участок.

Чтобы сделать этот пример еще проще и понятнее, рассмотрим в качестве аналогии водителя автомобиля, направляющегося из Парижа в Лондон. По территории Франции автомобиль двигается по дороге сам. Но, достигнув Ла-Манша, он погружается в высокоскоростной поезд и транспортируется под проливом по туннелю (автомобилям не разрешается передвигаться по туннелю самим). Таким образом, автомобиль перевозится как груз (рис. 5.36). На другом конце туннеля автомобиль спускается с железнодорожной платформы на английское шоссе и снова продолжает путь своими силами. Точно такой же принцип применяется при туннелировании пакетов при прохождении через чужеродную сеть.

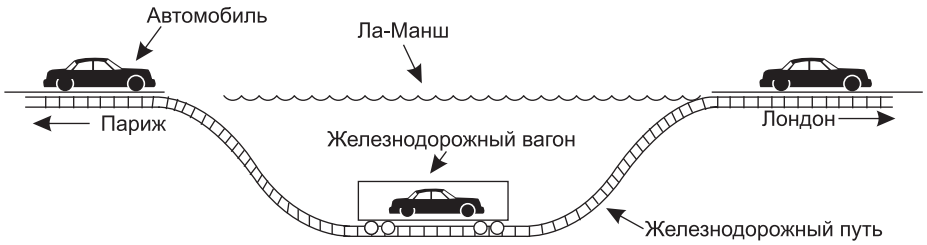


Рис. 5.36. Туннелирование автомобиля из Парижа в Лондон

Туннелирование широко используется для соединения изолированных хостов и сетей через сеть-посредник. В результате появляется новая сеть, которая как бы накладывается на старую. Такая сеть называется **оверлейной сетью (overlay)**. Использование сетевого протокола с новым свойством (как в нашем примере, где сети IPv6 соединяются через IPv4) — достаточно распространенная причина. Недостатком туннелирования является то, что пакет не может быть доставлен ни на один из

хостов, расположенных в сети-посреднике. Однако этот недостаток становится преимуществом в сетях **VPN (Virtual Private Network — виртуальная частная сеть)**. VPN — обычная оверлейная сеть, использующаяся в качестве меры безопасности. Более подробно о VPN мы поговорим в главе 8.

5.5.4. Маршрутизация в объединенных сетях

Маршрутизация в интерсетях сталкивается с теми же основными проблемами, что и маршрутизация в единой сети, но связана с некоторыми дополнительными сложностями. Рассмотрим, например, две сети, использующие разные алгоритмы маршрутизации: маршрутизацию с учетом состояния линий и маршрутизацию по вектору расстояний. Так как в первом случае алгоритму требуется информация о топологии сети, а во втором — нет, не очень понятно, как вычислять кратчайшие пути в такой интерсети.

Более серьезные проблемы возникают, если работу сети обеспечивают разные операторы. Во-первых, они могут по-разному представлять себе, что такое хороший путь: одни обращают больше внимания на время задержки, другие — на стоимость маршрута. В результате стоимость кратчайших путей указывается в разных единицах: соответственно, в миллисекундах или денежных единицах. Из-за невозможности сравнения весов вычисление кратчайших путей становится затруднительным.

Более того, оператор может не предоставлять другим операторам доступ к сведениям о весах и путях в своей сети, потому что в них может содержаться секретная информация (например, стоимость), важная в конкурентной борьбе.

Наконец, интерсеть может быть гораздо больше, чем любая из ее составляющих. В такой ситуации могут потребоваться алгоритмы маршрутизации, учитывающие иерархию, даже если в отдельных сетях такой необходимости нет.

Таким образом, нам необходим двухуровневый алгоритм маршрутизации. В пределах каждой сети для маршрутизации используется **внутридоменный (intradomain)** или **внутренний шлюзовый протокол (interior gateway protocol**; термин «шлюз» употреблялся раньше вместо термина «маршрутизатор»). Это может быть обычный протокол, учитывающий состояние линий. Между сетями применяется **междоменный (interdomain)** или **внешний шлюзовый протокол (exterior gateway protocol)**. Сети могут использовать разные внутридоменые протоколы, но междоменный протокол обязан быть общим. В сети Интернет междоменный протокол называется **BGP (Border Gateway Protocol — пограничный межсетевой протокол)**. О нем мы поговорим в следующем разделе.

Здесь следует рассказать еще об одном важном понятии. Так как все сети управляются независимо, они часто называются **АС — автономными системами (AS — Autonomous System)**. Хорошая умозрительная модель АС — сеть ISP. На практике сеть ISP может состоять из нескольких АС, если они управляются независимо. Но разница не очень существенна.

Эти два уровня не являются в строгом смысле иерархическими. Если крупную международную сеть объединить с небольшой региональной сетью, пути могут быть близкими к оптимальным. Однако для вычисления маршрутов в интерсети отдельные сети предоставляют сравнительно мало информации о своих маршрутах. Это позволяет справиться со всеми сложностями. В результате улучшается масштабирование

сети, а операторы свободны в выборе протоколов маршрутизации внутри своих сетей. Кроме того, при этом не требуется сравнивать веса, использующиеся в различных сетях, и предоставлять доступ к секретной внутренней информации.

До сих пор мы практически ничего не рассказали о том, как вычисляются маршруты в интерсети. В Интернете определяющим фактором является сотрудничество между сетями ISP. Каждая сеть может назначить цену за трафик другой сети. Еще один фактор заключается в том, что если маршрутизация в объединенных сетях требует пересечения границ государств, в игру могут неожиданно вступить законы этих государств. Так, в Швеции персональные данные граждан находятся под строжайшей защитой. Все эти внешние факторы объединяются в понятие **политики маршрутизации (routing policy)**. В соответствии с ней автономные сети выбирают маршруты. К этому понятию мы еще вернемся, когда будем говорить о BGP.

5.5.5. Фрагментация пакетов

Все сети и каналы накладывают ограничения на размер своих пакетов. Эти пределы вызваны различными предпосылками, среди которых есть следующие:

1. Аппаратные (например, размер кадра Ethernet).
2. Операционная система (например, все буферы имеют размер 512 байт).
3. Протоколы (например, количество бит в поле длины пакета).
4. Соответствие какому-либо международному или национальному стандарту.
5. Желание снизить количество пакетов, пересылаемых повторно из-за ошибок передачи.
6. Желание предотвратить ситуацию, когда один пакет слишком долгое время занимает канал.

Результатом действия всех этих факторов является то, что разработчики не могут выбирать максимальный размер пакета по своему усмотрению. Максимальный размер поля полезной нагрузки составляет 1500 байт для сети Ethernet и 2272 байта для 802.11. Протокол IP более щедр: размер пакета может достигать 65 515 байт.

Обычно хосты стараются отправлять крупные пакеты, так как это уменьшает издержки — например, позволяет сэкономить на заголовках. Очевидно, возникает проблема, когда большой пакет хочет пройти по сети, в которой максимальный размер пакетов слишком мал. Эта проблема сохраняла свою актуальность долгое время, а ее решения разрабатывались во многом на основании опыта, полученного в сети Интернет.

Одно из решений состоит в предотвращении возникновения самой проблемы. Но сказать это проще, чем сделать. Обычно отправитель не знает, по какому пути пойдет пакет, и поэтому он также не может знать, какого размера должен быть пакет, чтобы он добрался до места назначения. Такой размер пакета называется **путевым значением MTU (Path Maximum Transmission Unit — максимальный размер пакета для выбранного пути)**. Не важно, знает ли отправитель путевое значение MTU. В сети, не требующей соединения (такой как Интернет), маршруты пакетов в любом случае выбираются независимо. Это значит, что путь может неожиданно измениться, а тогда изменится и путевое значение MTU.

Альтернативное решение проблемы заключается в разрешении шлюзам разбивать пакеты на **фрагменты (fragments)** и посылать каждый фрагмент в виде отдельного пакета сетевого уровня. Однако, как вам скажет любой родитель маленького ребенка, преобразование большого объекта в небольшие фрагменты существенно проще, чем обратный процесс. (Физики даже дали этому эффекту специальное название: второй закон термодинамики.) В сетях с коммутацией пакетов также существует проблема с восстановлением пакетов из фрагментов.

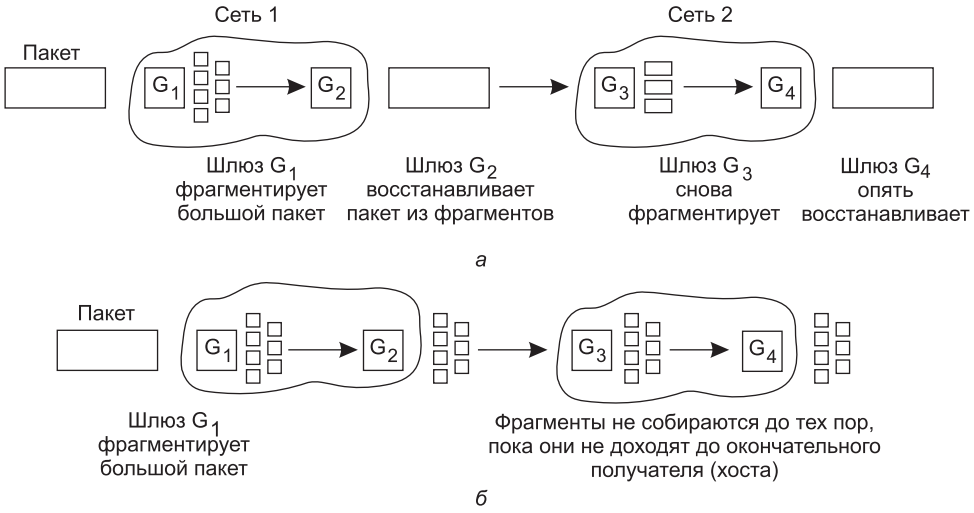


Рис. 5.37. Фрагментация: *а* — прозрачная; *б* — непрозрачная

Для восстановления исходных пакетов из фрагментов применяются две противоположные стратегии. Первая стратегия заключается в том, чтобы фрагментация пакета, вызванная сетью с пакетами малых размеров, оставалась прозрачной для обоих хостов, обменивающихся этим пакетом. Этот вариант показан на рис. 5.37, *а*. Когда на G1 приходит пакет слишком большого размера, он разбивается на фрагменты. Каждый фрагмент адресуется одному и тому же выходному маршрутизатору, G2, восстанавливающему из этих фрагментов исходный пакет. Таким образом, прохождение данных через сеть с небольшим размером пакетов оказывается прозрачным. Соседние сети даже не догадываются о том, что у них под боком пакеты страшным образом нарезаются, а потом снова склеиваются. Прозрачная фрагментация проста, но тем не менее создает некоторые проблемы. Во-первых, выходной маршрутизатор должен уметь определять момент получения последней части пакета, поэтому каждый фрагмент должен содержать либо поле счетчика, либо признак конца пакета. Кроме того, тот факт, что для последующего восстановления пакета все фрагменты должны выходить через один и тот же маршрутизатор, ограничивает возможности маршрутизации. Таким образом, налагается запрет на использование фрагментами различных путей к окончательному получателю, и в результате может оказаться потерянной часть производительности. Но еще более важным является вопрос о том, какую работу должен будет выполнить при этом маршрутизатор. Возможно, ему потребуется помесить

в буфер все прибывшие фрагменты и понять, в какой момент их можно выбросить, если какие-то фрагменты не дошли. Наконец, процессы фрагментации и последующей сборки пакетов при прохождении каждой сети с малым размером пакетов приводят к дополнительным накладным расходам. Другая стратегия фрагментации состоит в отказе от восстановления пакета из фрагментов на промежуточных маршрутизаторах. Как только пакет оказывается разбитым на отдельные фрагменты, с каждым фрагментом обращаются как с отдельным пакетом. Все фрагменты проходят через маршрутизаторы, как показано на рис. 5.37, *б*. Задача восстановления оригинального пакета возложена на получающий хост.

Основное преимущество непрозрачной фрагментации состоит в том, что маршрутизаторы выполняют меньше работы. Так работает IP. При этом фрагменты пакета должны нумероваться таким образом, чтобы можно было восстановить исходный поток данных. В случае IP каждому фрагменту сообщается номер пакета (который есть у каждого пакета), абсолютное смещение внутри пакета (в байтах) и флажок, показывающий, является ли этот фрагмент последним в пакете. Пример такой фрагментации показан на рис. 5.38. Хотя схема очень проста, она обладает рядом важных преимуществ. По прибытии в место назначения фрагменты можно помещать в буфер в любом порядке. Кроме того, при пересечении сети с меньшим значением MTU фрагменты могут быть разделены на более мелкие фрагменты. Такая ситуация показана на рис. 5.38, *в*. При повторной передаче (если все фрагменты не дошли до адресата) пакет может быть разбит на фрагменты по-другому. И наконец, размер фрагментов может быть произвольным, вплоть до одного байта плюс заголовок. В любом случае пакет будет восстановлен: номер пакета и смещение помогут расположить данные в правильном прядке, а флажок укажет на конец пакета.

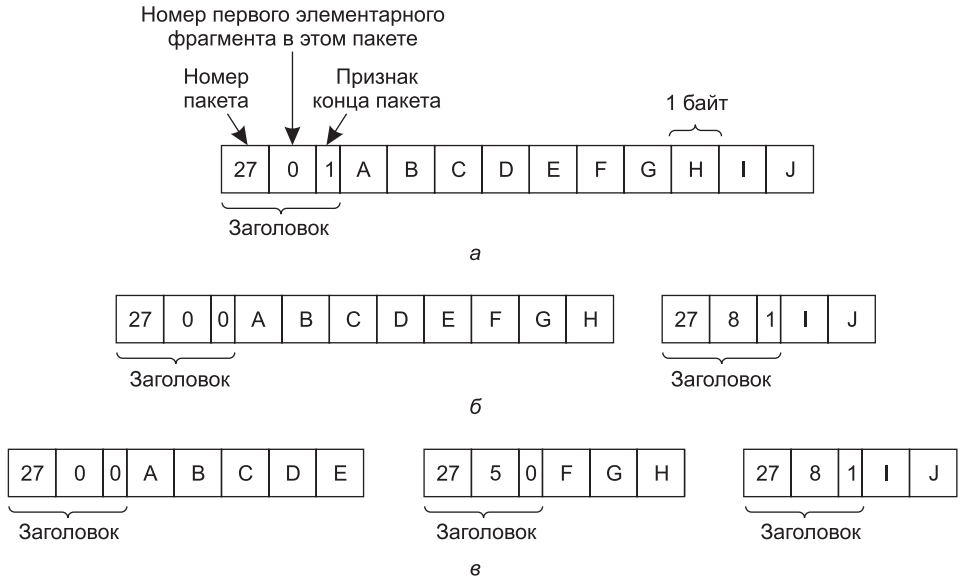


Рис. 5.38. Фрагментация при элементарном размере 1 байт: *а* — исходный пакет, содержащий 10 байт данных; *б* — фрагменты после прохождения через сеть с максимальным размером 8 байт; *в* — фрагменты после прохождения через шлюз размера 5

К сожалению, у этой схемы есть некоторые недостатки. Во-первых, это может быть более затратным, чем прозрачная фрагментация, так как заголовки фрагментов иногда передаются по связям, где без них можно обойтись. Однако настоящая проблема заключается в самом существовании фрагментов. Kent и Mogul (1987) считают, что фрагментация работает в ущерб производительности, так как при утере одного фрагмента теряется весь пакет (не говоря уже затратах на заголовки). Для хостов фрагментация оказалась тяжелым бременем, чем предполагалось вначале.

Таким образом, следует вернуться к первоначальной идее и полностью избавиться от фрагментации в сети — стратегия, использующаяся в современном Интернете. Этот процесс называется **Path MTU discovery (поиск путевого значения MTU)** (Mogul и Deering, 1990). Он работает так. При отправке каждого IP-пакета в его заголовок помещается информация о том, что фрагментация не разрешена. Если маршрутизатор получает слишком большой пакет, он отправляет источнику сообщение об ошибке и удаляет его (рис. 5.39). Используя информацию, содержащуюся в сообщении об ошибке, источник перераспределяет данные так, чтобы пакеты смогли пройти через маршрутизатор. Если такая же проблема возникнет на каком-то из следующих маршрутизаторов, ситуация повторится.

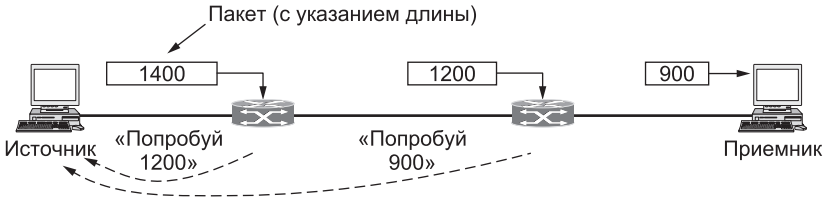


Рис. 5.39. Поиск путевого значения MTU

Преимущество Path MTU discovery состоит в том, что в результате источник знает необходимый размер пакета. При изменении маршрута новое путевое значение MTU станет известно источнику из новых сообщений об ошибке. Однако фрагментация между отправителем и получателем будет все равно необходима, если только путевое значение MTU не будет вычислено на более высоком уровне и передано IP. Чтобы это было возможно, TCP и IP обычно используются вместе (в виде TCP/IP). И хотя для некоторых протоколов это пока не реализовано, фрагментация все же была перенесена из сети на хосты.

Недостатком технологии Path MTU discovery является то, что отправка пакета может вызвать дополнительную задержку. Эта задержка может увеличиться в несколько раз в зависимости от того, сколько раз отправителю придется повторять отправку (меняя размер пакета), прежде чем пакет достигнет места назначения. Следовательно, возникает вопрос: существуют ли более эффективные схемы? Вероятно, да. Рассмотрим, к примеру, вариант, при котором слишком большие пакеты просто обрезаются. В таком случае получатель узнает путевое значение MTU настолько быстро, насколько это вообще возможно (по размеру полученного пакета), а также получает некоторую часть данных.

5.6. Сетевой уровень в Интернете

Теперь самое время подробно поговорить о сетевом уровне в Интернете. Но прежде чем перейти к деталям, неплохо было бы познакомиться с принципами, которые были основополагающими в прошлом, при его разработке, и которые обеспечили сегодняшний успех. В наше время люди все чаще пренебрегают ими. Между тем, эти принципы пронумерованы и включены в документ RFC 1958, с которым полезно ознакомиться (а для разработчиков протоколов он должен быть просто обязательным для прочтения, с экзаменом в конце). Этот документ использует идеи, которые были изложены в книгах (Clark, 1988; Saltzer и др., 1984). Ниже мы приведем 10 основных принципов, начиная с самых главных.

1. **Убедитесь в работоспособности.** Нельзя считать разработку (или стандарт) законченной, пока не проведен ряд успешных соединений между прототипами. Очень часто разработчики сначала пишут тысячестраничное описание стандарта, утверждают его, а потом обнаруживается, что он еще очень сырой или вообще неработоспособен. Тогда пишется версия 1.1 стандарта. Так бывает, но так быть не должно.
2. **Упрощайте.** Если есть сомнения, всегда самый простой выбор является самым лучшим. Уильям Оккам (William Occam) декларировал этот принцип еще в XIV веке («Бритва Оккама»). Его можно выразить следующим образом: *Борись с излишествами*. Если какое-то свойство не является абсолютно необходимым, забудьте про него, особенно если такого же эффекта можно добиться комбинированием уже имеющихся свойств.
3. **Всегда делайте четкий выбор.** Если есть несколько способов реализации одного и того же, необходимо выбрать один из них. Увеличение количества способов — это порочный путь. В стандартах часто можно встретить несколько опций, режимов или параметров. Почему так получается? Лишь потому, что при разработке было несколько авторитетных мнений на тему того, что является наилучшим. Разработчики должны избегать этого и сопротивляться подобным тенденциям. Надо просто уметь говорить «Нет».
4. **Используйте модульный принцип.** Этот принцип напрямую приводит к идее стеков протоколов, каждый из которых работает на одном из независимых уровней. Таким образом, если обстоятельства складываются так, что необходимо изменить один из модулей или уровней, то это не затрагивает других частей системы.
5. **Предполагайте разнородность.** В любой большой сети могут сосуществовать различные типы оборудования, средства передачи данных и различные приложения. Сетевая технология должна быть достаточно гибкой, простой и обобщенной, чтобы работать в таких условиях.
6. **Избегайте статичности свойств и параметров.** Если есть какие-то обязательные параметры (например, максимальный размер пакета), то лучше заставить отправителя и получателя договариваться об их конкретных значениях, чем жестко закреплять их.
7. **Проект должен быть хорошим, но он не может быть идеальным.** Очень часто разработчики создают хорошие проекты, но не могут предусмотреть какие-нибудь

причудливые частные случаи. Не стоит портить то, что сделано хорошо и работает в большинстве случаев. Вместо этого имеет смысл переложить все бремя ответственности за «улучшения» проекта на тех, кто предъявляет свои странные требования.

8. **Тщательно продумывайте отправку данных, но будьте снисходительны при приеме данных.** Другими словами, посылайте только те пакеты, которые полностью соответствуют всем требованиям стандартов. При этом надо иметь в виду, что приходящие пакеты не всегда столь идеальны и их тоже нужно обрабатывать.
9. **Продумайте масштабируемость.** Если в системе работают миллионы хостов и миллиарды пользователей, о централизации можно забыть. Нагрузка должна быть распределена максимально равномерно между имеющимися ресурсами.
10. **Помните о производительности и цене.** Никого не заинтересует сеть низкопроизводительная или дорогостоящая.

Теперь перейдем от общих принципов к деталям построения сетевого уровня Интернета. На сетевом уровне Интернет можно рассматривать как набор сетей или **автономных систем** (АС), соединенных друг с другом. Структуры как таковой Интернет не имеет, но все же есть несколько магистралей. Они собраны из высокопроизводительных линий и быстрых маршрутизаторов. Самые крупные магистрали (к которым необходимо присоединиться, чтобы получить доступ к остальной части сети Интернет) называются сетями **Tier 1**. К магистралям присоединены интернет-провайдеры, обеспечивающие доступ к Интернету домам и предприятиям, центры обработки данных и станции колокации с большим числом серверов, а также региональные сети (сети среднего уровня). Центры обработки данных обрабатывают большинство данных, передаваемых по сети Интернет. К региональным сетям присоединяются другие интернет-провайдеры, локальные сети многочисленных университетов и компаний, а также прочие периферийные сети. Схема этой квазииерархической структуры показана на рис. 5.40.

Вся эта конструкция «склеивается» благодаря протоколу сетевого уровня, **IP** (**Internet Protocol** — **протокол сети Интернет**). В отличие от большинства ранних протоколов сетевого уровня, IP с самого начала разрабатывался как протокол межсетевого обмена. Вот как можно описать данный протокол сетевого уровня: его работа заключается в приложении максимума усилий (тем не менее без всяких гарантий) по транспортировке дейтаграмм от отправителя к получателю, независимо от того, находятся эти машины в одной и той же сети или нет.

Соединение в сети Интернет представляет собой следующее. Транспортный уровень берет поток данных и разбивает его на дейтаграммы. Теоретически размер каждой дейтаграммы может достигать 64 Кбайт, однако на практике они обычно не более 1500 байт (укладываются в один кадр Ethernet). IP-маршрутизаторы направляют каждый пакет на следующий маршрутизатор до тех пор, пока он не достигнет места назначения. После этого сетевой уровень передает данные транспортному уровню, вставляющему их во входной поток получающего процесса. Когда фрагменты приходят в пункт назначения, сетевой уровень собирает их в исходную дейтаграмму. Затем эта дейтаграмма передается транспортному уровню.

В примере на рис. 5.40 пакет, посланный одним из хостов домашней сети, пересечет на своем пути четыре сети и пройдет через множество IP-маршрутизаторов, прежде чем доберется до сети предприятия, в которой расположен хост-получатель. На практике такие ситуации встречаются довольно часто, причем существуют и более длинные пути. С точки зрения связности сеть Интернет является избыточной: между магистралями и интернет-провайдерами обычно существует несколько точек соединения. Отсюда и множественные пути между хостами. Задача протокола IP — решить, какие из этих путей лучше использовать.

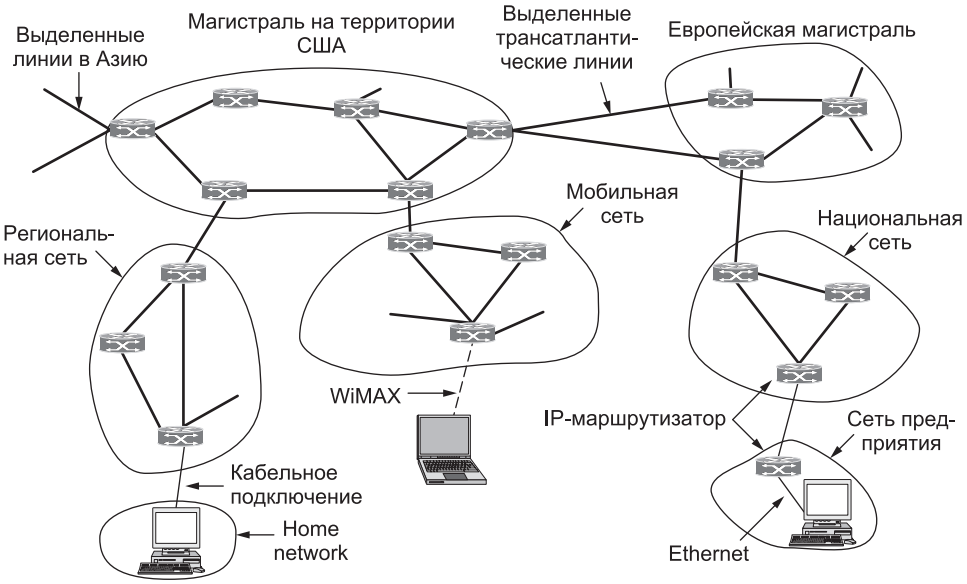


Рис. 5.40. Интернет представляет собой набор соединенных друг с другом сетей

5.6.1. Протокол IP версии 4

Начнем изучение сетевого уровня Интернета с формата IPv4-дейтаграмм. IPv4-дейтаграмма состоит из заголовка и основной или полезной части. Заголовок содержит обязательную 20-байтную часть, а также необязательную часть переменной длины. Формат заголовка показан на рис. 5.41. Биты передаются слева направо и сверху вниз, то есть старший бит поля *Версия* передается первым. (Такой порядок байтов называется «big-endian» — «со старшего конца слова». На компьютерах с порядком байтов «little-endian» — «с младшего конца слова», таких как Intel x86, требуется программное преобразование, как при передаче, так и при приеме.) Сейчас уже совершенно ясно, что для IP лучше было использовать порядок «little-endian», но на момент создания протокола это не было столь очевидным.

Поле *Версия* содержит версию протокола, к которому принадлежит дейтаграмма. Сейчас в сети Интернет доминирует версия 4, поэтому с нее мы и начали обсуждение IP. Включение версии в начало каждой дейтаграммы позволяет использовать

разные версии протокола в течение долгого времени. Вообще-то протокол **IPv6, следующая** версия IP, был разработан более десяти лет назад, но применять его начинают только сейчас. О нем мы поговорим позже в этом разделе. Широкое распространение протокол IPv6 получит тогда, когда у каждого из 2^{31} жителей Китая будет настольный ПК, ноутбук и IP-телефон. Что касается нумерации, то ничего странного в ней нет, просто в свое время существовал мало кому известный экспериментальный протокол реального масштаба времени IPv5.

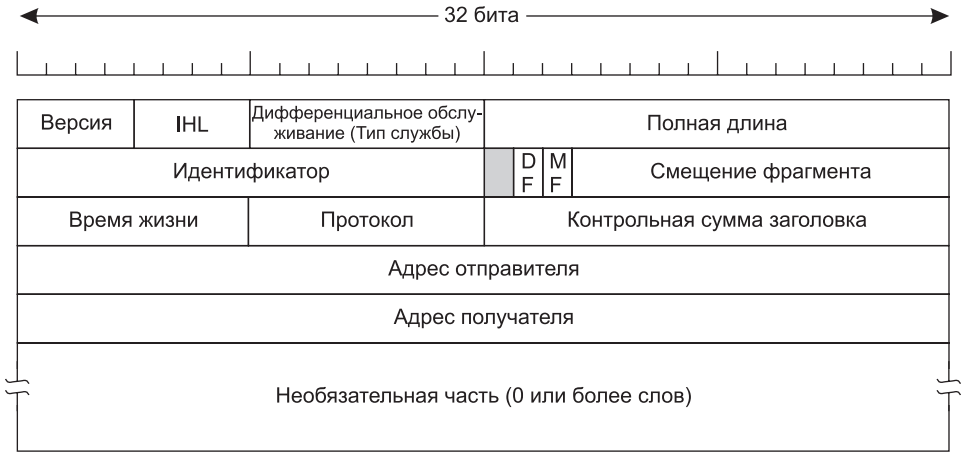


Рис. 5.41. Заголовок IP-дейтаграммы IPv4

Длина заголовка является переменной величиной, для хранения которой выделено поле *IHL* (в нем указано число 32-разрядных слов). Минимальное значение длины (при отсутствии необязательного поля) равно 5. Максимальное значение этого 4-битового поля равно 15, что соответствует заголовку длиной 60 байт; таким образом, максимальный размер необязательного поля равен 40 байтам. Для некоторых приложений, например для записи маршрута, по которому должен быть переслан пакет, 40 байт слишком мало. В данном случае дополнительное поле оказывается бесполезным.

Поле *Дифференцированное обслуживание* — одно из немногих полей, смысл которых с годами слегка изменился. Изначально это поле называлось *Тип службы*. Оно было (впрочем, и до сих пор) предназначено для различения классов обслуживания. Возможны разные комбинации надежности и скорости. Для оцифрованного голоса скорость доставки важнее точности. При передаче файла, наоборот, передача без ошибок важнее быстрой доставки. В поле *Тип службы* 3 бита использовались для задания приоритета, еще 3 бита показывали, что беспокоит хост больше всего: задержка, пропускная способность или надежность. Но поскольку никто не знал, что делать со всеми этими битами, они не использовались в течение многих лет. Когда появилось дифференцированное обслуживание, IETF сдалась и нашла этому полю другое применение. В результате первые 6 бит задают класс обслуживания; о срочном и гарантированном обслуживании мы уже говорили ранее в этой главе. В последние два бита помещаются явные уведомления о перегрузке; о таких уведомлениях шла речь в разделе, посвященном борьбе с перегрузкой.

Поле *Полная длина* содержит длину всей дейтаграммы, включая как заголовок, так и данные. Максимальная длина дейтаграммы 65 535 байт. В настоящий момент этот верхний предел достаточен, однако в будущем могут понадобиться дейтаграммы большего размера.

Поле *Идентификатор* позволяет хосту-получателю определить, какому пакету принадлежат полученные им фрагменты. Все фрагменты одного пакета содержат одно и то же значение идентификатора.

Следом идет один неиспользуемый бит, что достаточно странно, так как место в IP-заголовке принято использовать экономно. В качестве первоапрельской шутки Белловин (2003) предложил использовать его для обнаружения вредоносного трафика. Тогда обеспечить безопасность сети было бы гораздо проще: пакеты с таким «злым» битом можно было бы просто удалять, зная, что они отправлены злоумышленниками. К сожалению, безопасность не так проста.

Далее следуют два однобитных поля, относящиеся к фрагментации. Бит *DF* означает Don't Fragment (Не фрагментировать). Это команда маршрутизатору, запрещающая ему фрагментировать пакет. Изначально предполагалось, что это поле будет помогать хостам, которые не могут восстановить пакет из фрагментов. Сейчас оно используется при определении путевого значения MTU, которое равно максимальному размеру пакета, передаваемого по пути без фрагментации. Пометив дейтаграмму битом *DF*, отправитель гарантирует, что либо дейтаграмма дойдет единым блоком, либо отправитель получит сообщение об ошибке.

Бит *MF* означает More Fragments (Продолжение следует). Он устанавливается во всех фрагментах, кроме последнего. По этому биту получатель узнает о прибытии последнего фрагмента дейтаграммы.

Поле *Смещение фрагмента* указывает положение фрагмента в исходном пакете. Длина всех фрагментов в байтах, кроме длины последнего фрагмента, должна быть кратна 8. Так как на это поле выделено 13 бит, максимальное количество фрагментов в дейтаграмме равно 8192, что дает максимальную длину пакета вплоть до пределов поля *Полная длина*. Поля *Идентификатор*, *MF* и *Смещение фрагмента* используются при реализации схемы фрагментации, описанной в разделе 5.5.5.

Поле *Время жизни* представляет собой счетчик, ограничивающий время жизни пакета. Предполагалось, что он будет отсчитывать время в секундах, таким образом, допуская максимальное время жизни пакета в 255 с. На каждом маршрутизаторе это значение должно было уменьшаться как минимум на единицу плюс время стояния в очереди. Однако на практике этот счетчик просто считает количество переходов через маршрутизаторы. Когда значение этого поля становится равным нулю, пакет отвергается, а отправителю отсылается пакет с предупреждением. Таким образом, удается избежать вечного странствования пакетов, что может произойти, если таблицы маршрутизаторов по какой-либо причине испортятся.

Собрав пакет из фрагментов, сетевой уровень должен решить, что с ним делать. Поле *Протокол* сообщит ему, какому процессу транспортного уровня передать этот пакет. Это может быть TCP, UDP или что-нибудь еще. Нумерация процессов глобально стандартизирована по всему Интернету. Номера протоколов вместе с некоторыми другими были сведены в RFC 1700, однако теперь доступна интернет-версия в виде базы данных, расположенной по адресу *www.iana.org*.

Поскольку заголовок содержит важную информацию (в частности, адрес), для ее защиты существует отдельное поле *Контрольная сумма заголовка*. Алгоритм вычисления суммы просто складывает все 16-разрядные полуслова заголовка в дополнительном коде, преобразуя результат также в дополнительный код. Таким образом, проверяемая получателем контрольная сумма заголовка (вместе с этим полем) должна быть равна нулю. Подобная контрольная сумма полезна для обнаружения ошибок, возникающих во время прохождения пакета по сети. Обратите внимание на то, что значение *Контрольной суммы заголовка* должно подсчитываться заново на каждом транзитном участке, так как по крайней мере одно поле постоянно меняется (поле *Время жизни*). Для ускорения расчетов применяются некоторые хитрости.

Поля *Адрес отправителя* и *Адрес получателя* указывают IP-адреса сетевых интерфейсов отправителя и получателя. Интернет-адреса будут обсуждаться в следующем разделе.

Поле *Необязательная часть* было создано для того, чтобы с появлением новых вариантов протокола не пришлось вносить в заголовок поля, отсутствующие в нынешнем формате. Оно же может служить пространством для различного рода экспериментов, испытания новых идей. Кроме того, оно позволяет не включать в стандартный заголовок редко используемую информацию. Размер поля *Необязательная часть* может варьироваться. В начале поля всегда располагается однобайтный идентификатор. Иногда за ним может располагаться также однобайтное поле длины, а затем один или несколько информационных байтов. В любом случае, размер поля *Необязательная часть* должен быть кратен 4 байтам. Изначально было определено пять разновидностей этого поля, перечисленных в табл. 5.5.

Таблица 5.5. Некоторые типы необязательного поля IP-дейтаграммы

Тип	Описание
Безопасность	Указывает уровень секретности дейтаграммы
Строгая маршрутизация от источника	Задает полный путь следования дейтаграммы
Свободная маршрутизация от источника	Задает список маршрутизаторов, которые нельзя миновать
Запомнить маршрут	Требует от всех маршрутизаторов добавлять свой IP-адрес
Временной штамп	Требует от всех маршрутизаторов добавлять свой IP-адрес и текущее время

Параметр *Безопасность* указывает уровень секретности дейтаграммы. Теоретически, военный маршрутизатор может использовать это поле, чтобы запретить маршрутизацию пакета через территорию определенных государств. На практике все маршрутизаторы игнорируют этот параметр, так что его единственное практическое применение состоит в помощи шпионам в поиске ценной информации.

Параметр *Строгая маршрутизация от источника* задает полный путь следования дейтаграммы от отправителя до получателя в виде последовательности IP-адресов. Дейтаграмма обязана следовать именно по этому маршруту. Наибольшая польза этого параметра заключается в том, что с его помощью системный менеджер может послать

экстренные пакеты, когда таблицы маршрутизатора повреждены, или замерить временные параметры сети.

Параметр *Свободная маршрутизация от источника* требует, чтобы пакет прошел через указанный список маршрутизаторов в указанном порядке, но при этом по пути он может проходить через любые другие маршрутизаторы. Обычно этот параметр указывает лишь небольшое количество маршрутизаторов. Например, чтобы заставить пакет, посылаемый из Лондона в Сидней, двигаться не на восток, а на запад, можно указать в этом параметре IP-адреса маршрутизаторов в Нью-Йорке, Лос-Анджелесе и Гонолулу. Этот параметр наиболее полезен, когда по политическим или экономическим соображениям следует избегать прохождения пакетов через определенные государства.

Параметр *Запомнить маршрут* требует от всех маршрутизаторов, встречающихся по пути следования пакета, добавлять свой IP-адрес к полю *Необязательная часть*. Этот параметр позволяет системным администраторам вылавливать ошибки в алгоритмах маршрутизации («Ну почему все пакеты, посылаемые из Хьюстона в Даллас, сначала попадают в Токио?»). Когда была создана сеть ARPANET, ни один пакет не проходил больше чем через девять маршрутизаторов, поэтому размер 40 байт для этого параметра был слишком большим. Как уже говорилось, сегодня размер поля *Необязательная часть* оказывается слишком мал.

Наконец, параметр *Временной штамп* действует полностью аналогично параметру *Запомнить маршрут*, но кроме 32-разрядного IP-адреса, каждый маршрутизатор также записывает 32-разрядную запись о текущем времени. Этот параметр также применяется в основном для измерения параметров сети.

В последнее время необязательные поля оказываются не в почете. Маршрутизаторы либо игнорируют их, либо обрабатывают неэффективно, отодвигая в сторону как нечто необычное. Иными словами, они поддерживаются лишь частично и используются достаточно редко.

5.6.2. IP-адреса

Определяющим признаком IPv4 является его 32-битный адрес. У каждого хоста и маршрутизатора в Интернете есть IP-адрес, который может использоваться в полях *Адрес отправителя* и *Адрес получателя* IP-пакетов. Важно отметить, что IP-адрес, на самом деле, не имеет отношения к хосту. Он имеет отношение к сетевому интерфейсу, поэтому хост, соединенный с двумя сетями, должен иметь два IP-адреса. Однако на практике большинство хостов подключены к одной сети, следовательно, имеют один адрес. Маршрутизаторы, наоборот, обычно имеют несколько интерфейсов и, следовательно, несколько IP-адресов.

Префиксы

В отличие от Ethernet-адресов IP-адреса имеют иерархическую организацию. Первая часть 32-битного адреса имеет переменную длину и задает сеть, а последняя часть указывает на хост. Сетевая часть совпадает для всех хостов одной сети (например, ЛВС Ethernet). Таким образом, сети соответствует непрерывный блок пространства IP-адресов. Этот блок называется **префиксом (prefix)**.

IP-адреса обычно записываются в виде четырех десятичных чисел (которые соответствуют отдельным байтам), разделенных точками (dotted decimal notation). Например, шестнадцатеричный адрес 80D00297 записывается как 128.208.2.151. Префикс задается наименьшим IP-адресом в блоке и размером блока. Размер определяется числом битов в сетевой части; оставшиеся биты в части хоста могут варьироваться. Таким образом, размер является степенью двойки. По традиции он пишется после префикса IP-адреса в виде слэша и длины сетевой части в битах. В нашем примере префикс содержит 2^8 адресов и поэтому для сетевой части отводится 24 бита. Пишется это так: 128.208.2.0/24.

Поскольку длина префикса не выводится из IP-адреса, протоколы маршрутизации вынуждены передавать префиксы на маршрутизаторы. Иногда префиксы задаются с помощью указания длины (например, «/16»). Длина префикса соответствует двоичной маске, в которой единицы указывают на сетевую часть. Такая маска называется **маской подсети (subnet mask)**. Выполнение операции И между маской и IP-адресом позволяет выделить сетевую часть. В нашем примере (рис. 5.42) маска подсети выглядит так: 255.255.255.0.

Рис. 5.42. Префикс IP-адреса и маска подсети

У иерархических адресов есть существенные преимущества и недостатки. Важное преимущество префиксов состоит в том, что маршрутизаторы могут направлять пакеты, используя только сетевую часть адреса, поскольку каждой сети соответствует свой уникальный адресный блок. Маршрутизатору не нужно учитывать часть адреса, задающую хост, так как пакеты для всех хостов одной сети передаются в одном направлении. Пакеты направляются на хосты только после того, как попадают в соответствующую сеть. В результате таблицы маршрутизации становятся гораздо меньше. Притом что число хостов в Интернете приближается к миллиарду, это очень существенное преимущество, так как иначе таблицы маршрутизации были бы невероятно большими. Но благодаря иерархии им приходится хранить маршруты лишь для 300 000 префиксов.

Хотя иерархия упрощает маршрутизацию в Интернете, она обладает двумя недостатками. Во-первых, IP-адрес хоста зависит от его местоположения в сети. Адреса Ethernet можно использовать в любой точке мира, а IP-адрес принадлежит конкретной сети, и поэтому маршрутизаторы могут доставить пакет, предназначенный для данного адреса, только в данную сеть. Для того чтобы хосты могли перемещаться из одной сети в другую, сохраняя свой IP-адрес, необходимы новые решения, такие как мобильный IP.

Второй недостаток состоит в том, что неправильная иерархия может привести к неэффективному использованию адресов. Если адреса приписываются сетям (слиш-

ком) крупными блоками, большое количество адресов будет выделено, но не будет использоваться. Если бы адресов было много, этот факт не имел бы такого значения. Однако уже более десяти лет назад стало ясно, что свободное адресное пространство в Интернете заполняется с невероятной скоростью. Протокол IPv6 решил эту проблему, но до тех пор, пока он не будет применяться повсеместно, эффективное выделение адресов будет вызывать серьезные затруднения.

Подсети

Во избежание конфликтов, номера сетям назначаются некоммерческой **корпорацией по присвоению имен и номеров**, **ICANN (Internet Corporation for Assigned Names and Numbers)**. В свою очередь, ICANN передала полномочия по присвоению некоторых частей адресного пространства региональным органам, занимающимся выделением IP-адресов провайдерам и другим компаниям. Таким образом компании получают блоки IP-адресов.

Однако с этого история только начинается, так как с ростом компаний им требуются новые IP-адреса. Как мы уже говорили, маршрутизация по префиксу требует, чтобы у всех хостов сети был один и тот же номер сети. Это свойство IP-адресации может вызвать проблемы при росте сети. Например, представьте, что университет создал сеть с префиксом /16 из нашего примера, используемую факультетом информатики в качестве Ethernet. Год спустя факультету электротехники понадобилось подключиться к Интернету, а затем и факультету искусств. Какие IP-адреса будут использовать эти факультеты? Если запросить дополнительные блоки, то получится, что новые сети не будут частью сети университета; к тому же, это может быть дорого и неудобно. Более того, префикс /16 позволяет подключить более 60 000 хостов. Возможно, такой префикс был выбран с учетом того, что сеть может вырасти. Но пока этого не произошло, выделять университету новые блоки будет неэффективно. Требуется новая архитектура.

Проблема решилась предоставлением сети возможности разделения на несколько частей с точки зрения внутренней организации. Это называется **разбиением на подсети (subnetting)**. Сети, полученные в результате разбиения крупной сети (например, локальные сети в сети Ethernet), называются **подсетями (subnets)**. Как уже упоминалось в главе 1, подобное использование этого термина конфликтует со старым понятием «подсети», обозначающим множество всех маршрутизаторов и линий связи в сети.

На рис. 5.43 показано, как разбиение на подсети может быть полезным в нашем примере. Единая сеть /16 разделена на части. Части не обязаны быть одинаковыми, но адреса должны быть распределены с учетом длины оставшейся части хоста. В нашем случае половина блока (/17) выделяется факультету информатики, четверть (/18) — факультету электротехники, восьмая часть (/19) — факультету искусств. Оставшаяся восьмая часть не используется. Еще один способ понять, как блок разбивается на части, — посмотреть на префиксы в двоичной нотации.

Информатика	10000000	11010000	1|xxxxxxx	xxxxxxxx
Электротехника	10000000	11010000	00|xxxxxx	xxxxxxxx
Искусства	10000000	11010000	011|xxxxx	xxxxxxxx

Вертикальная черта (|) обозначает границу между номером подсети и номером хоста.

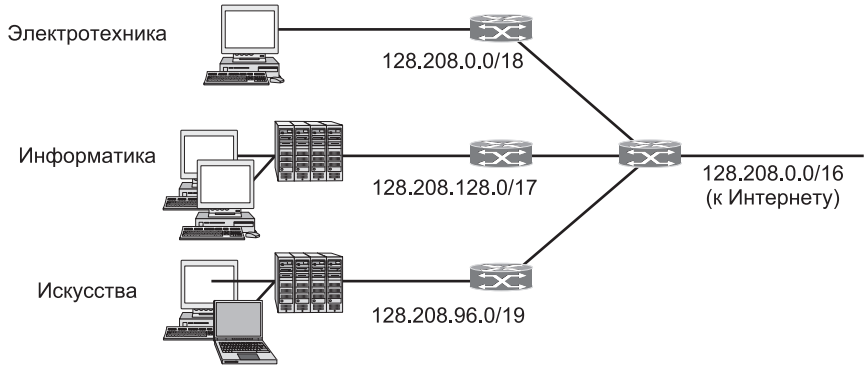


Рис. 5.43. Разделение IP-префикса при разбиении на подсети

Как центральный маршрутизатор узнает, в какую из подсетей направить пришедший пакет? Здесь и могут пригодиться специфические свойства префиксов. Одним из способов является поддержание каждым маршрутизатором таблицы из 65 536 записей, говорящих о том, какую исходящую линию использовать для доступа к каждому из хостов. Но это сведет на нет основное преимущество иерархии, касающееся размеров таблиц маршрутизации. Вместо этого можно сделать так, чтобы маршрутизаторы знали маски этих подсетей.

Когда приходит пакет, маршрутизатор просматривает адрес назначения и определяет, к какой подсети он относится. Для этого маршрутизатор может выполнить операцию И от этого адреса и маски каждой из подсетей, сравнивая результат с соответствующим префиксом. Пусть у нас есть пакет с адресом 128.208.2.151. Чтобы проверить, относится ли он к факультету информатики, прибавим к нему (используя логическое И) маску 255.255.128.0, таким образом отрезав первые 17 бит (то есть 128.208.0.0). Далее сравним полученный результат с префиксом (128.208.128.0). Они не совпадают. Для факультета электротехники аналогичным образом берем первые 18 бит адреса и получаем 128.208.0.0. Это значение совпадает с префиксом, поэтому пакет передается на интерфейс, ведущий к сети факультета электротехники.

Разделение на подсети можно впоследствии изменить. Для этого потребуется обновить сведения о сетевых масках подсетей на всех маршрутизаторах сети университета. За пределами сети разделение на подсети незаметно, поэтому нет нужды с появлением каждой подсети обращаться в ICANN или изменять какие-либо внешние базы данных.

CIDR — бесклассовая междоменная маршрутизация

Даже при эффективном выделении IP-адресов проблема разрастания сети сохраняется.

Если маршрутизатор находится на границе сети какой-либо организации (например, университета), он должен хранить информацию обо всех подсетях, чтобы знать, по какой линии следует передавать пакеты для этой сети. Если адрес назначения

находится за пределами данной организации, он может использовать простое правило по умолчанию: отправлять пакеты по линии, соединяющей эту организацию с остальной сетью Интернет. Остальные адреса назначения, очевидно, находятся поблизости.

Маршрутизаторы интернет-провайдеров и магистралей не могут позволить себе такую роскошь. Они должны знать путь к любой сети, поэтому для них не может существовать простого правила по умолчанию. Про такие магистральные маршрутизаторы говорят, что они находятся в свободной от умолчаний зоне (**default-free zone**) сети Интернет. Никто не знает точно, сколько всего сетей подключено к Интернету, но очевидно, что их много — возможно, порядка миллиона. Из них можно составить очень большую таблицу. Может быть, и не очень большую с точки зрения компьютерных стандартов, но представьте себе, что маршрутизатор должен просматривать ее при отправке каждого пакета, а за секунду он отправляет миллионы таких пакетов. Для обработки пакетов с такой скоростью требуются специализированные аппаратные средства и быстродействующая память; обычный компьютер для этого не подойдет.

Различные алгоритмы маршрутизации требуют, чтобы каждый маршрутизатор обменивался информацией о доступных ему адресах с другими маршрутизаторами. Чем больше размер таблицы, тем больше данных необходимо передавать и обрабатывать. С ростом размера таблицы время обработки растет как минимум линейно. Чем больше данных приходится передавать, тем выше вероятность потери (в лучшем случае временной) части информации по дороге, что может привести к нестабильности работы алгоритмов выбора маршрутов.

Проблема таблиц маршрутизаторов может быть решена при помощи увеличения числа уровней иерархии, как это происходит в телефонных сетях. Например, если бы каждый IP-адрес содержал поля страны, штата или провинции, города, сети и номера хоста. В таком случае, каждому маршрутизатору нужно будет знать, как добраться до каждой страны, до каждого штата или провинции своей страны, каждого города своей провинции или штата и до каждой сети своего города. К сожалению, такой подход потребует существенно более 32 бит для адреса, а адресное поле будет использоваться неэффективно (для княжества Лихтенштейн будет выделено столько же разрядов, сколько для Соединенных Штатов).

К счастью, способ уменьшить размер таблиц маршрутизации все же существует. Применим тот же принцип, что и при разбиении на подсети: маршрутизатор может узнавать о расположении IP-адресов по префиксам различной длины. Но вместо того чтобы разделять сеть на подсети, мы объединим несколько коротких префиксов в один длинный. Этот процесс называется **агрегацией маршрута** (**route aggregation**). Длинный префикс, полученный в результате, иногда называют **суперсетью** (**supernet**), в противоположность подсетям с разделением блоков адресов.

При агрегации IP-адреса содержатся в префиксах различной длины. Один и тот же IP-адрес может рассматриваться одним маршрутизатором как часть блока /22 (содержащего 2^{10} адресов), а другим — как часть более крупного блока /20 (содержащего 2^{12} адресов). Это зависит от того, какой информацией обладает маршрутизатор. Такой метод работает и для разбиения на подсети и называется **CIDR** (**Classless InterDomain Routing** — **бесклассовая междоменная маршрутизация**). Последняя на сегодняшний день версия описана в RFC 4632 (Fuller и Li, 2006). Название иллюстрирует отличие

от адресов, кодирующих иерархию с помощью классов, о которой мы в скором времени поговорим.

Чтобы лучше понять алгоритм маршрутизации, рассмотрим пример. Допустим, у нас есть блок из 8192 адресов, начиная с 194.24.0.0. Допустим также, что Кембриджскому университету требуется 2048 адресов и ему выделяются адреса от 194.24.0.0 до 194.24.7.255, а также маска 255.255.248.0. Это будет префикс /21. Затем Оксфордский университет запрашивает 4096 адресов. Так как блок из 4096 адресов должен располагаться на границе, кратной 4096, то ему не могут быть выделены адреса, начинающиеся с 194.24.8.0. Вместо этого он получает адреса от 194.24.16.0 до 194.24.31.255 вместе с маской 255.255.240.0. Наконец, Эдинбургский университет просит выделить ему 1024 адреса и получает адреса от 194.24.8.0 до 194.24.11.255 и маску 255.255.252.0. Все эти присвоенные адреса и маски сведены в табл. 5.6.

Таблица 5.6. Набор присвоенных IP-адресов

Университет	Первый адрес	Последний адрес	Количество	Форма записи
Кембридж	194.24.0.0	194.24.7.255	2048	194.24.0.0/21
Эдинбург	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(Свободно)	194.24.12.0	194.24.15.255	1024	194.24.12.0/22
Оксфорд	194.24.16.0	194.24.16.255	4096	194.24.16.0/20

После этого всем маршрутизаторам, находящимся в свободной от умолчаний зоне, сообщаются IP-адреса трех новых сетей. Маршрутизаторы, находящиеся рядом с этими университетами (например, в Лондоне — рис. 5.44), возможно, захотят отправлять пакеты на эти префиксы по разным исходящим линиям. Тогда они запишут эти адреса в свои таблицы маршрутизации.

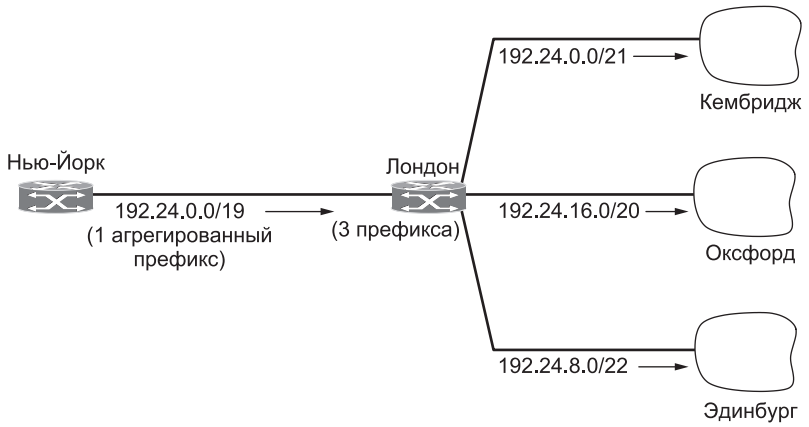


Рис. 5.44. Агрегация IP-префиксов

Теперь посмотрим на эту троицу университетов с точки зрения отдаленного маршрутизатора в Нью-Йорке. Все IP-адреса, относящиеся к этим трем префиксам, должны отправляться из Нью-Йорка (или из США) в Лондон. Процесс маршрутизации

в Лондоне узнает об этом и объединяет три префикса в одну **агрегированную запись** 194.24.0.0/19 и передает ее в Нью-Йорк. Этот префикс содержит 8 Кбайт адресов и объединяет три университета плюс 1024 свободных адреса. Агрегация позволила объединить три префикса в один, благодаря чему уменьшилось количество префиксов, о которых должен знать маршрутизатор в Нью-Йорке, и количество записей в его таблице маршрутизации.

Если агрегация включена, она производится автоматически. Этот процесс зависит от того, где какие префиксы расположены, а не от администратора, который выделяет сетям адреса. В Интернете агрегация используется очень активно, снижая размер таблиц маршрутизации примерно до 200 000 префиксов.

Дальше все становится еще интереснее: префиксы могут пересекаться. Согласно правилу, пакеты передаются в направлении самого специализированного блока, или **самого длинного совпадающего префикса (longest matching prefix)**, в котором находится меньше всего IP-адресов. Поведение маршрутизатора в Нью-Йорке (рис. 5.45) показывает, насколько гибкой является такой тип маршрутизации. Для отправки пакетов в наши три университета Нью-Йоркский маршрутизатор использует один агрегированный префикс. Но что делать, если тот блок адресов, который раньше был свободным, теперь принадлежит сети в Сан-Франциско? Например, Нью-Йоркский маршрутизатор может хранить четыре префикса: один для Сан-Франциско и три для Лондона. Маршрутизация по самому длинному совпадающему префиксу позволяет обойтись двумя (см. рис. 5.45). Один общий префикс используется для того, чтобы направлять трафик, предназначенный для всего блока, в Лондон. Еще один специфический префикс позволяет направлять его часть в Сан-Франциско. В соответствии с правилом самого длинного совпадающего префикса, пакеты, предназначенные для IP-адресов в Сан-Франциско, будут переданы по исходящей линии, ведущей в Сан-Франциско. Пакеты, предназначенные для более крупной сети, будут направлены в Лондон.

По сути CIDR работает так. Когда прибывает пакет, необходимо определить, относится ли данный адрес к данному префиксу; для этого просматривается таблица маршрутизации. Может оказаться, что по значению подойдет несколько записей. В этом случае используется самый длинный префикс. То есть если найдено совпадение для маски /20 и /24, то для выбора исходящей линии будет использоваться запись, соответствующая /24. Однако этот процесс был бы трудоемким, если бы таблица маршрутизации просматривалась запись за записью. Вместо этого был разработан сложный алгоритм для ускорения процесса поиска адреса в таблице (Ruiz-Sanchez и др., 2001). В маршрутизаторах, предполагающих коммерческое использование, применяются специальные чипы VLSI, в которые данные алгоритмы встроены аппаратно.

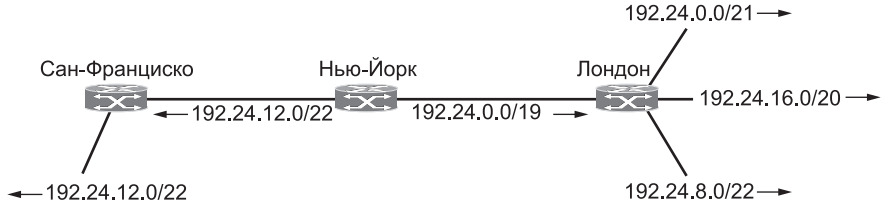


Рис. 5.45. Маршрутизация по самому длинному совпадающему префиксу на Нью-Йоркском маршрутизаторе

Полноклассовая и специализированная адресация

Чтобы лучше представлять себе основные преимущества CIDR, рассмотрим метод, который использовался раньше. До 1993 года IP-адреса разделялись на 5 категорий (рис. 5.46). Такое распределение называется **полноклассовой адресацией (classful addressing)**.

Рис. 5.46. Форматы IP-адреса

Форматы классов A, B, C и D позволяют задавать адреса до 128 сетей с 16 млн хостов в каждой, 16 384 сетей с 64 536 хостов или 2 млн сетей (например, ЛВС) с 256 хостами (хотя некоторые из них могут быть специализированными). Предусмотрен класс для многоадресной рассылки, при которой дейтаграммы рассылаются одновременно на несколько хостов. Адреса, начинающиеся с 1111, зарезервированы для будущего применения. Неплохо было бы использовать их уже сейчас, когда адресное пространство IPv4 практически закончилось. Но так как в течение долгого времени они были запрещены, многие хосты будут их отвергать — не так уж просто заставить старые хосты изменить свои привычки.

Такая структура является иерархической. Но в отличие от CIDR здесь используются фиксированные размеры блоков адресов. Существует 2 млрд адресов, но благодаря иерархической организации адресного пространства это число сократилось на миллионы. В частности, одним из виновников этого является класс сетей B. Для большинства организаций класс A с 16 млн адресов — это слишком много, а класс C с 256 адресами — слишком мало. Класс B с 65 536 адресами — это то, что нужно. В интернет-фольклоре такая дилемма известна под названием **проблемы трех медведей**.

Но на самом деле, и класс B слишком велик для большинства контор, которые устанавливают у себя сети. Исследования показали, что более чем в половине случаев сети класса B включают в себя менее 50 хостов. Безо всяких сомнений, всем этим организациям хватило бы и сетей класса C, однако почему-то все уверены, что в один прекрасный день маленькое предприятие вдруг разрастется настолько, что сеть выйдет за пределы 8-битного адресного пространства хостов. Сейчас, оглядываясь назад, кажется, что лучше было бы использовать в классе C 10-битную адресацию (до

1022 хостов в сети). Если бы это было так, то, возможно, большинство организаций приняло бы разумное решение устанавливать у себя сети класса C, а не B. Таких сетей могло бы быть полмиллиона, а не 16 384, как в случае сетей класса B.

Нельзя обвинять в создавшейся ситуации проектировщиков Интернета за то, что они не увеличили (или не уменьшили) адресное пространство сетей класса B. В то время, когда принималось решение о создании трех классов сетей, Интернет был инструментом научно-исследовательских организаций США (плюс несколько компаний и военных организаций, занимавшихся исследованиями с помощью сети). Никто тогда не предполагал, что Интернет станет коммерческой системой коммуникации общего пользования, соперничающей с телефонной сетью. Тогда кое-кто сказал, ничуть не сомневаясь в своей правоте: «В США около 2000 колледжей и университетов. Даже если все они подключатся к Интернету и к ним присоединятся университеты из других стран, мы никогда не превысим число 16 000, потому что высших учебных заведений по всему миру не так уж много. Зато мы будем кодировать номер хоста целым числом байт, что ускорит процесс обработки пакетов» (в то время это выполнялось исключительно программными средствами). Быть может, в один прекрасный день кто-то скажет, обвиняя разработчиков телефонной сети: «Вот идиоты! Почему они не включили номер планеты в телефонный номер?» Когда телефонные сети создавались, никто не думал, что это понадобится.

Для решения этих проблем стали создаваться подсети, обеспечивающие гибкий механизм выделения блоков адресов отдельным организациям. Позднее в употребление вошла новая схема, позволяющая уменьшить размер таблиц маршрутизации — CIDR. Сейчас биты, указывающие на класс сети (A, B или C), не используются, хотя ссылки на них в литературе встречаются все еще довольно часто.

Отказ от классов усложнил процесс маршрутизации. В старой системе, построенной на классах, маршрутизация происходила следующим образом. По прибытии пакета на маршрутизатор копия IP-адреса, извлеченного из пакета и сдвинутого вправо на 28 бит, давала 4-битный номер класса. С помощью 16-альтернативного ветвления пакеты рассортировывались на A, B, C (а также D и E): восемь случаев было зарезервировано для A, четыре для B, два для C. Затем при помощи маскировки по коду каждого класса определялся 8-, 16- или 32-битный сетевой номер, который и записывался с выравниванием по правым разрядам в 32-битное слово. Сетевой номер отыскивался в таблице A, B или C, причем для A и B применялась индексация, а для C — хэш-функция. По найденной записи определялась выходная линия, по которой пакет и отправлялся в дальнейшее путешествие. Этот метод гораздо проще, чем нахождение наиболее длинного совпадающего префикса, при котором простой поиск по таблице невозможен, так как префиксы IP-адресов могут иметь произвольную длину.

Адреса класса D и сейчас используются в Интернете для многоадресной рассылки. Вообще-то правильнее было бы сказать, что они *начинают* использоваться для этих целей, так как до недавнего времени многоадресная рассылка не имела широкого распространения в Интернете.

Некоторые адреса имеют особое назначение (рис. 5.47). **IP-адрес 0.0.0.0 — наименьший** адрес — используется хостом только при загрузке. Он означает «эта сеть» или «этот хост». IP-адреса с нулевым номером сети обозначают текущую сеть. Эти адреса позволяют машинам обращаться к хостам собственной сети, не зная ее номера (но

они должны знать маску сети, чтобы знать количество используемых нулей). Адрес, состоящий только из единиц, или 255.255.255.255 — наибольший адрес — обозначает все хосты в указанной сети. Он обеспечивает широковещание в пределах текущей (обычно локальной) сети. Адреса, в которых указана сеть, но в поле номера хоста одни единицы, обеспечивают широковещание в пределах любой удаленной локальной сети, соединенной с Интернетом. Однако многие сетевые администраторы отключают эту возможность по соображениям безопасности. Наконец, все адреса вида 127.xx.yy.zz зарезервированы для тестирования сетевого программного обеспечения методом обратной петли (loopback). Отправляемые по этому адресу пакеты не попадают на линию, а обрабатываются локально как входные пакеты. Это позволяет пакетам быть «посланными» на хост, когда отправитель на этом же хосте не знает его (и своего) номера и даже если хост его не имеет, что может пригодиться для тестирования.

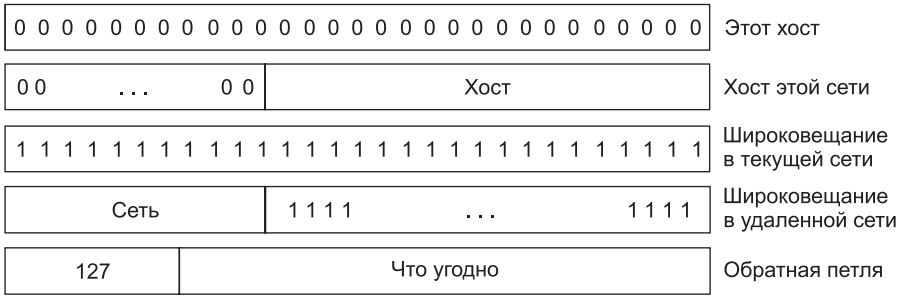


Рис. 5.47. Специальные IP-адреса

NAT — трансляция сетевого адреса

IP-адреса являются дефицитным ресурсом. У провайдера может быть /16-адрес (бывший класс B), дающий возможность подключить 65 534 хоста. Если клиентов становится больше, начинают возникать проблемы.

Этот дефицит ресурсов стал причиной появления методов эффективного использования IP-адресов. Идея одного из подходов заключается в том, что IP-адрес выделяется компьютеру, который в данный момент включен и подключен к сети; когда этот компьютер становится неактивным, его адрес присваивается новому соединению. В таком случае один /16-адрес будет обслуживать до 65 534 *активных* пользователей.

Такая стратегия в некоторых ситуациях работает хорошо — например, при подключении к сети через телефон, для мобильных и других ПК, где соединение или питание могут временно отсутствовать. Но если заказчиком является организация, эта стратегия, как правило, не подходит. Дело в том, что корпоративные клиенты предпочитают, чтобы компьютеры работали постоянно. Некоторые компьютеры являются рабочими станциями сотрудников, и данные с них резервируются в ночное время, а некоторые служат веб-серверами и поэтому должны уметь незамедлительно отвечать на любой удаленный запрос. Такие организации обладают линией доступа, обеспечивающей постоянное соединение с Интернетом.

Проблема усугубляется еще и тем, что все большее число частных пользователей желают иметь ADSL или кабельное соединение с Интернетом, поскольку при этом

отсутствует повременная оплата (взимается только ежемесячная абонентская плата). Многие такие пользователи имеют дома два и более компьютера (например, по одному на каждого члена семьи) и хотят, чтобы все машины имели постоянный выход в Интернет. Решение таково: необходимо установить (беспроводной) маршрутизатор и объединить все компьютеры в домашнюю локальную сеть. А уже маршрутизатор должен быть подключен к провайдеру. С точки зрения провайдера, в этом случае семья будет выступать в качестве аналога маленькой фирмы с несколькими компьютерами. Добро пожаловать в Jones, Inc! Если использовать обычные схемы, то каждый компьютер будет сохранять свой IP-адрес в течение всего дня. Но для интернет-провайдеров с несколькими тысячами клиентов (многие из которых представляют собой целые организации или семьи, также похожие на небольшие организации) такая ситуация недопустима, так как IP-адресов попросту не хватит. Проблема дефицита IP-адресов отнюдь не теоретическая и отнюдь не относится к отдаленному будущему. Она уже актуальна и бороться с ней приходится здесь и сейчас. В долгосрочной перспективе решением будет тотальный перевод всего Интернета на протокол IPv6 со 128-битной адресацией. Этот переход действительно постепенно происходит, но процесс идет настолько медленно, что затягивается на годы. Видя это, многие поняли, что нужно срочно найти какое-нибудь решение хотя бы на ближайшее время. Такое решение было найдено и сейчас широко используется: это метод **трансляции сетевого адреса**, **NAT (Network Address Translation)**, описанный в RFC 3022. Суть его мы рассмотрим ниже, а более подробную информацию можно найти в (Dutcher, 2001).

Основная идея трансляции сетевого адреса состоит в присвоении каждой фирме или семье одного IP-адреса (или, по крайней мере, небольшого числа адресов) для интернет-трафика. *Внутри* абонентской сети каждый компьютер получает уникальный IP-адрес, используемый для маршрутизации внутреннего трафика. Однако, как только пакет покидает пределы абонентской сети и направляется к провайдеру, выполняется трансляция адреса, при которой уникальный внутренний IP-адрес становится общим публичным IP-адресом. Для реализации этой схемы было создано три диапазона так называемых частных IP-адресов. Они могут использоваться внутри сети по ее усмотрению. Единственное ограничение заключается в том, что пакеты с такими адресами ни в коем случае не должны появляться в самом Интернете. Вот эти три зарезервированных диапазона:

```
10.0.0.0     -  10.255.255.255/8      (16 777 216 хостов)
172.16.0.0   -  172.31.255.255/12     (1 048 576 хостов)
192.168.0.0  -  192.168.255.255/16    (65 536 хостов)
```

Итак, первый диапазон может обеспечить адресами 16 777 216 хостов (кроме всех 0 и всех 1, как обычно), и именно его обычно предпочитают абоненты, причем даже для небольших сетей.

Работа метода трансляции сетевых адресов показана на рис. 5.48. В пределах территории абонента у каждой машины имеется собственный уникальный адрес вида 10.*x.y.z*. Тем не менее, перед тем как пакет выходит за пределы владений абонента, он проходит через **NAT-блок (NAT box)**, транслирующий внутренний IP-адрес источника (10.0.0.1 на рисунке) в реальный IP-адрес, полученный абонентом от провайдера (198.60.42.12 для нашего примера). NAT-блок обычно представляет собой единое устройство с межсетевым экраном, обеспечивающим безопасность путем строгого

отслеживания входящего и исходящего трафика абонентской сети. Межсетевые экраны мы будем изучать отдельно в главе 8. NAT-блок может быть также интегрирован с маршрутизатором или модемом ADSL.

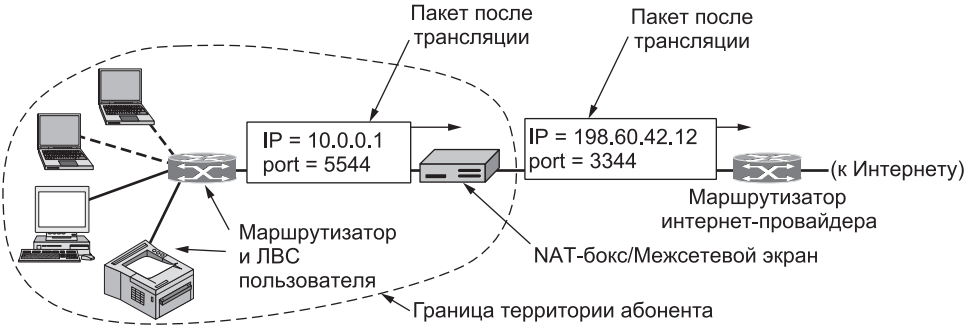


Рис. 5.48. Расположение и работа NAT-блока

Мы до сих пор обходили одну маленькую деталь: когда приходит ответ на запрос (например, от веб-сервера), он ведь адресуется 198.60.42.12. Как же NAT-блок узнает, каким внутренним адресом заменить общий адрес компании? Вот в этом и состоит главная проблема использования трансляции сетевых адресов. Если бы в заголовке IP-пакета было свободное поле, его можно было бы использовать для запоминания адреса того, кто посылал запрос. Но в заголовке остается неиспользованным всего один бит. В принципе, можно было бы создать такое поле для истинного адреса источника, но это потребовало бы изменения IP-кода на всех машинах по всему Интернету. Это не лучший выход, особенно если мы хотим найти быстрое решение проблемы нехватки IP-адресов.

На самом деле, происходит вот что. Разработчики NAT подметили, что большая часть полезной нагрузки IP-пакетов — это либо TCP, либо UDP. Когда мы будем в главе 6 рассматривать TCP и UDP, мы увидим, что оба формата имеют заголовки, содержащие номера портов источника и приемника. Ниже мы обсудим, что значит порт TCP, но надо иметь в виду, что с портами UDP связана точно такая же история. Номера портов представляют собой 16-разрядные целые числа, показывающие, где начинается и где заканчивается TCP-соединение. Место хранения номеров портов используется в качестве поля, необходимого для работы NAT.

Когда процесс желает установить TCP-соединение с удаленным процессом, он связывается со свободным TCP-портом на собственном компьютере. Этот порт становится **портом источника (source port)**, который сообщает TCP-коду информацию о том, куда направлять пакеты данного соединения. Процесс также определяет **порт назначения (destination port)**. Посредством порта назначения сообщается, кому отдать пакет на удаленной стороне. Порты с 0 по 1023 зарезервированы для хорошо известных сервисов. Например, 80-й порт используется веб-серверами, соответственно, на них могут ориентироваться удаленные клиенты. Каждое исходящее сообщение TCP содержит информацию о порте источника и порте назначения. Вместе они служат для идентификации процессов на обоих концах, использующих соединение.

Проведем аналогию, которая несколько прояснит принцип использования портов. Допустим, у компании есть один общий телефонный номер. Когда люди набирают его,

они слышат голос оператора, который спрашивает, с кем именно они хотели бы соединиться, и подключают их к соответствующему добавочному телефонному номеру. Основной телефонный номер является аналогией IP-адреса абонента, а добавочные на обоих концах аналогичны портам. Для адресации портов используется 16-битное поле, которое идентифицирует процесс, получающий входящий пакет.

С помощью поля *Порт источника* мы можем решить проблему отображения адресов. Когда исходящий пакет приходит в NAT-блок, адрес источника вида 10.*x.y.z* заменяется настоящим IP-адресом. Кроме того, поле *Порт источника* TCP заменяется индексом таблицы перевода NAT-блока, содержащей 65 536 записей. Каждая запись содержит исходный IP-адрес и номер исходного порта. Наконец, пересчитываются и вставляются в пакет контрольные суммы заголовков TCP и IP. Необходимо заменять поле *Порт источника*, потому что машины с местными адресами 10.0.0.1 и 10.0.0.2 могут случайно пожелать воспользоваться одним и тем же портом (5000-м, например). Так что для однозначной идентификации процесса отправителя одного поля *Порт источника* оказывается недостаточно.

Когда пакет прибывает на NAT-блок со стороны провайдера, извлекается значение поля *Порт источника* заголовка TCP. Оно используется в качестве индекса таблицы отображения NAT-блока. По найденной в этой таблице записи определяются внутренний IP-адрес и настоящий *Порт источника* TCP. Эти два значения вставляются в пакет. Затем заново подсчитываются контрольные суммы TCP и IP. Пакет передается на главный маршрутизатор абонента для нормальной доставки с адресом вида 10.*x.y.z*.

Хотя описанная выше схема частично решает проблему нехватки IP-адресов, сетевые пуристы из IP-сообщества рассматривают NAT как некую заразу, распространяющуюся по Земле. И их можно понять.Во-первых, сам принцип трансляции сетевых адресов никак не вписывается в архитектуру IP, которая подразумевает, что каждый IP-адрес уникальным образом идентифицирует только одну машину в мире. Вся программная структура Интернета построена на использовании этого факта. При трансляции сетевых адресов получается, что тысячи машин могут (и так происходит в действительности) иметь адрес 10.0.0.1.

Во-вторых, NAT нарушает «сквозной» принцип, согласно которому каждый хост должен уметь отправлять пакет любому другому хосту в любой момент времени. Поскольку отображение адресов в NAT-блоке задается исходящими пакетами, входящие пакеты не принимаются до тех пор, пока не отправлены исходящие. На деле это означает, что пользователь домашней сети с NAT может создать TCP/IP-соединение с удаленным сервером, но удаленный пользователь не может подключиться к игровому серверу в домашней сети. Чтобы это было возможным, необходимы специальные настройки или технология **NAT Traversal**.

В-третьих, NAT превращает Интернет из сети без установления соединения в нечто подобное сети, ориентированной на соединение. Проблема в том, что NAT-блок должен поддерживать таблицу отображения для всех соединений, проходящих через него. Запоминать состояние соединения — дело сетей, ориентированных на соединение, но никак не сетей без установления соединений. Если NAT-блок ломается и теряются его таблицы отображения, то про все TCP-соединения, проходящие через него, можно забыть. При отсутствии трансляции сетевых адресов выход из строя или перезагрузка маршрутизатора не оказывает никакого эффекта на TCP-соединение.

Отправляющий процесс просто выжидает несколько секунд и посылает заново все неподтвержденные пакеты. При использовании NAT Интернет становится таким же восприимчивым к сбоям, как сеть с коммутацией каналов.

В-четвертых, NAT нарушает одно из фундаментальных правил построения многоуровневых протоколов: уровень k не должен строить никаких предположений относительно того, что именно уровень $k + 1$ поместил в поле полезной нагрузки. Этот принцип определяет независимость уровней друг от друга. Если когда-нибудь на смену TCP придет TCP-2, у которого будет другой формат заголовка (например, 32-битная адресация портов), то трансляция сетевых адресов потерпит фиаско. Вся идея многоуровневых протоколов состоит в том, чтобы изменения в одном из уровней никак не могли повлиять на остальные уровни. NAT разрушает эту независимость.

В-пятых, процессы в Интернете вовсе не обязаны использовать только TCP или UDP. Если пользователь машины *A* решит придумать новый протокол транспортного уровня для общения с пользователем машины *B* (это может быть сделано, например, для какого-нибудь мультимедийного приложения), то ему придется как-то бороться с тем, что NAT-блок не сможет корректно обработать поле *Порт источника* TCP.

В-шестых, некоторые приложения предусматривают использование множественных TCP/IP-соединений или UDP-портов. Так, **FTP**, стандартный **Протокол передачи файлов (File Transfer Protocol)**, вставляет IP-адрес в тело пакета, чтобы затем получатель извлек его оттуда и обработал. Так как NAT об этом ничего не знает, он не сможет переписать адрес или как-то иначе его обработать. Это значит, что FTP и другие приложения, такие как протокол интернет-телефонии H.323 (мы будем изучать его в главе 7), могут отказаться работать при трансляции сетевых адресов, если только не будут приняты специальные меры. Часто существует возможность улучшить метод NAT и заставить его корректно работать в таких случаях, но невозможно же дорабатывать его всякий раз, когда появляется новое приложение.

Наконец, поскольку поле *Порт источника* является 16-разрядным, то на один IP-адрес может быть отображено примерно 65 536 местных адресов машин. На самом деле, это число несколько меньше: первые 4096 портов зарезервированы для служебных нужд. В общем, если есть несколько IP-адресов, то каждый из них может поддерживать до 61 440 местных адресов.

Эти и другие проблемы, связанные с трансляцией сетевых адресов, обсуждаются в RFC 2993. Несмотря на все трудности, NAT представляет собой единственный работающий механизм борьбы с дефицитом IP-адресов. Поэтому он широко используется на практике, особенно в домашних сетях и сетях небольших организаций. В NAT сейчас существуют межсетевые экраны и средства обеспечения конфиденциальности, поскольку по умолчанию блокируются все незапрошенные входящие пакеты. Поэтому маловероятно, что эта технология уйдет из употребления после внедрения IPv6.

5.6.3. Протокол IP версии 6

В течение многих лет IP оставался чрезвычайно популярным протоколом. Он работал просто прекрасно, и основное подтверждение тому — экспоненциальный рост сети Интернет. К сожалению, протокол IP скоро стал жертвой собственной популярности: стало подходить к концу адресное пространство. И хотя для более эффективного ис-

пользования адресного пространства стали применяться CIDR и NAT, до 2012 года IPv4-адреса будут выделяться с помощью ICANN. Надвигающуюся угрозу заметили почти 20 лет назад, и вопрос о том, что с этим делать, вызвал в интернет-сообществе бурю дискуссий и расхождений.

В этом разделе мы поговорим как об этой проблеме, так и о ее решениях. В перспективе хорошей идеей является переход к более длинным адресам. **IPv6 (IP version 6 — IP версии 6)** — новая разработка, призванная заменить старую. Протокол IPv6 использует 128-битные адреса; в обозримом будущем дополнительного увеличения длины адреса, скорее всего, не потребуется. Однако оказалось, что внедрить IPv6 не так уж просто. Это принципиально новый протокол сетевого уровня, несовместимый с IPv4, несмотря на многочисленные сходства с ним. Кроме того, компании и отдельные пользователи не понимают, почему им стоит перейти на IPv6. Поэтому сейчас он занимает лишь крохотную часть Интернета (около 1 %) несмотря на то что признан стандартом еще в 1998 году. Что будет дальше, станет ясно в ближайшие несколько лет, когда будут распределены последние из оставшихся IPv4-адресов. Интересно, станут ли люди продавать свои адреса на eBay? Или на черном рынке? Как знать.

Помимо перечисленных выше проблем с выделением адресов, имеются и другие, угрожающе вырастающие на горизонте. До недавних пор Интернетом пользовались в основном университеты, высокотехнологичные предприятия и правительственные организации США (особенно Министерство обороны). С лавинообразным ростом интереса к Интернету, начавшимся в середине 90-х годов, в третьем тысячелетии, скорее всего, им будет пользоваться гораздо большее количество пользователей с принципиально разными требованиями. Во-первых, пользователи смартфонов могут подключаться к Интернету для доступа к домашним базам данных. Во-вторых, при неминуемом сближении компьютерной промышленности, средств связи и индустрии развлечений, возможно, очень скоро каждый телефон и телевизор планеты станет узлом Интернета, что в результате приведет к появлению миллиардов машин, используемых для аудио и видео по заказу. В таких обстоятельствах становится очевидно, что протокол IP должен эволюционировать и стать более гибким.

Предвидя появление этих проблем, проблемная группа проектирования Интернета IETF начала в 1990 году работу над новой версией протокола IP, в которой никогда не должна возникнуть проблема нехватки адресов, а также будут решены многие другие проблемы. Кроме того, новая версия протокола должна была быть более гибкой и эффективной. Были сформулированы следующие основные цели.

1. Поддержка миллиардов хостов даже при неэффективном использовании адресного пространства.
2. Уменьшение размера таблиц маршрутизации.
3. Упрощение протокола для ускорения обработки пакетов маршрутизаторами.
4. Более надежное обеспечение безопасности (аутентификации и конфиденциальности).
5. Необходимость обращать большее внимание на тип сервиса, в частности, при передаче данных реального времени.
6. Упрощение работы многоадресных рассылок с помощью указания областей рассылки.

7. Возможность изменения положения хоста без необходимости изменять его адрес.
8. Возможность дальнейшего развития протокола в будущем.
9. Возможность сосуществования старого и нового протоколов в течение нескольких лет.

Разработка IPv6 дала шанс улучшить возможности IPv4 исходя из потребностей современного Интернета. Чтобы найти протокол, удовлетворяющий всем этим требованиям, IETF издал в RFC 1550 приглашение к дискуссиям и предложениям. Был получен двадцать один ответ. Далеко не все варианты содержали предложения, полностью удовлетворяющие этим требованиям. В декабре 1992 года были рассмотрены семь серьезных предложений. Их содержание варьировалось от небольших изменений в протоколе IP до полного отказа от него и замены совершенно другим протоколом.

Одно из предложений состояло в использовании вместо IP протокола CLNP, который с его 160-разрядным адресом обеспечивал бы достаточное адресное пространство на веки вечные — этого пространства хватило бы, если бы каждая молекула воды в мировом океане захотела создать свою небольшую сеть (порядка 2^5 адресов). Кроме того, это решение объединило бы два основных сетевых протокола. Однако все же сочли, что при подобном выборе придется признать, что кое-что в мире OSI было сделано правильно, что было бы политически некорректным в интернет-кругах. Протокол CLNP, на самом деле, очень мало отличается от протокола IP. Окончательный выбор был сделан в пользу протокола, отличающегося от IP значительно сильнее, нежели CLNP. Еще одним аргументом против CLNP была его слабая поддержка типа сервиса, требовавшегося для эффективной передачи мультимедиа.

Три лучших предложения были опубликованы в журнале *IEEE Network Magazine* (Deering, 1993; Francis, 1993; Katz и Ford, 1993). После долгих обсуждений, переработок и борьбы за первое место была выбрана модифицированная комбинированная версия Диринга (Deering) и Фрэнсиса (Francis), называемая в настоящий момент протоколом **SIPP (Simple Internet Protocol Plus — Простой интернет-протокол Плюс)**. Новому протоколу было дано обозначение **IPv6**.

Протокол IPv6 прекрасно справляется с поставленными задачами. Он обладает достоинствами протокола IP и лишен некоторых его недостатков (либо обладает ими в меньшей степени), к тому же наделен некоторыми новыми особенностями. В общем случае, протокол IPv6 несовместим с протоколом IPv4, но зато совместим со всеми остальными протоколами Интернета, включая TCP, UDP, ICMP, IGMP, OSPF, BGP и DNS, для чего иногда требуются небольшие изменения для работы с более длинными адресами. Основные особенности протокола IPv6 обсуждаются ниже. Дополнительные сведения о нем можно найти в RFC с 2460 по 2466.

Прежде всего, у протокола IPv6 поля адресов длиннее, чем у IPv4. Они имеют длину 128 бит, что решает основную проблему, поставленную при разработке протокола, — обеспечить практически неограниченный запас интернет-адресов. Мы еще кратко упомянем об адресах чуть позднее.

Второе заметное улучшение протокола IPv6 по сравнению с IPv4 состоит в более простом заголовке пакета. Он состоит всего из 7 полей (вместо 13 у протокола IPv4). Таким образом, маршрутизаторы могут быстрее обрабатывать пакеты, что повышает производительность. Краткое описание заголовков будет приведено ниже.

Третье усовершенствование заключается в улучшенной поддержке необязательных параметров. Подобное изменение действительно было существенным, так как в новом заголовке требуемые прежде поля стали необязательными (потому что они и так использовались не часто). Кроме того, изменился способ представления необязательных параметров, что упростило для маршрутизаторов пропуск не относящихся к ним параметров и ускорило обработку пакетов.

В-четвертых, протокол IPv6 демонстрирует большой шаг вперед в области безопасности. У проблемной группы проектирования Интернета IETF была полная папка вырезок из газет с сообщениями о том, как 12-летние мальчишки с помощью своего персонального компьютера по Интернету вломились в банк или военную базу. Было ясно, что надо как-то улучшить систему безопасности. Аутентификация и конфиденциальность являются ключевыми чертами нового IP-протокола. После модификации IPv4 разница с точки зрения безопасности стала не так уж и велика.

Наконец, в новом протоколе было уделено больше внимания качеству обслуживания. Различные нерешительные попытки по реализации качества обслуживания предпринимались и в прошлом, но при росте мультимедийного трафика в Интернете ощущение их актуальности возрастает.

Основной заголовок IPv6

Заголовок IPv6 показан на рис. 5.49. Поле *Версия* содержит число 6 для IPv6 (и 4 для IPv4). На период перехода с IPv4 на IPv6, который, вероятно, займет около десяти лет, маршрутизаторы по значению этого поля смогут отличать пакеты нового стандарта от старого. Подобная проверка потребует нескольких тактов процессора, что может оказаться нежелательным в некоторых ситуациях, тем более что в заголовке с информацией о канале передачи данных обычно указан протокол для демультиплексирования, так что эту проверку можно пропустить. Например, в Ethernet у поля *Tun* существует несколько разных значений, определяющих полезные данные IPv4- или IPv6-пакета. Дискуссия между лагерями, руководствующимися принципами «Делай правильно» и «Делай быстро», несомненно, будет долгой и энергичной.

Поле *Дифференцированное обслуживание* (изначально *Класс трафика*) используется для того, чтобы различать пакеты с разными требованиями к доставке в реальном времени. Оно используется для обеспечения качества обслуживания вместе с архитектурой дифференцированного обслуживания (точно так же, как и одноименное поле IPv4). Кроме того, так же как и в IPv4, младшие 2 бита отводятся под явные уведомления о перегрузке.

Поле *Метка потока* применяется для того, чтобы отправитель и получатель могли сообщить сети об определенных свойствах пакетов и требованиях к их обработке; при этом между ними устанавливается псевдосоединение. Например, поток пакетов между двумя процессами на разных хостах может обладать строгими требованиями к задержкам, что потребует резервирования пропускной способности. Поток устанавливается заранее и получает идентификатор. Когда прибывает пакет с отличным от нуля содержимым поля *Метка потока*, все маршрутизаторы смотрят в свои таблицы, чтобы определить, какого рода особая обработка ему требуется. Таким образом, новый протокол пытается соединить достоинства подсетей различных типов: гибкость дейтаграмм и гарантии виртуальных каналов.

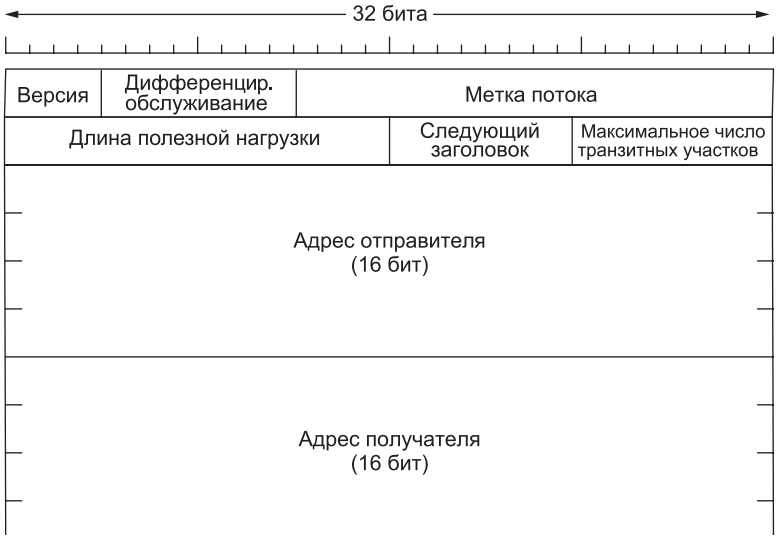


Рис. 5.49. Фиксированный заголовок IPv6 (обязательные поля)

С целью обеспечения должного качества обслуживания каждый поток описывается адресом источника, адресом назначения и номером потока. Это значит, что для каждой пары IP-адресов можно создать до 2^{20} активных потоков. Кроме того, два потока с одинаковыми номерами, но различными адресами отправителя или получателя считаются разными потоками и различаются маршрутизаторами по адресам. Ожидается, что номера каналов будут выбираться случайным образом, а не назначаться подряд начиная с 1, что облегчит маршрутизаторам их распознавание.

Поле *Длина полезной нагрузки* сообщает, сколько байт следует за 40-байтовым заголовком, показанным на рис. 5.49. В заголовке IPv4 аналогичное поле называлось *Полная длина* и определяло весь размер пакета. В новом протоколе 40 байт заголовка учитываются отдельно. Это значит, что теперь полезная нагрузка может занимать 65 535 байт вместо 65 515.

Поле *Следующий заголовок* раскрывает секрет возможности использования упрощенного заголовка. Дело в том, что после обычного 40-байтового заголовка могут идти дополнительные (необязательные) расширенные заголовки. Это поле сообщает, какой из шести дополнительных заголовков (на текущий момент) следует за основным. В последнем IP-заголовке поле *Следующий заголовок* сообщает, какой обрабатывающей программе протокола транспортного уровня (то есть TCP или UDP) передать пакет.

Поле *Максимальное число транзитных участков* не дает пакетам вечно блуждать по сети. Оно имеет практически то же назначение, что и поле *Время жизни* в заголовке протокола IPv4. Это поле уменьшается на единицу на каждом транзитном участке. Теоретически, в протоколе IPv4 это поле должно было содержать секунды времени жизни пакета, однако ни один маршрутизатор не использовал его подобным образом, поэтому имя поля было приведено в соответствие со способом его применения.

Следом идут поля *Адрес отправителя* и *Адрес получателя*. В исходном предложении Диринга (протоколе SIPP) использовались 8-байтовые адреса, но при рас-

смотрении проекта было решено, что 8-байтовых адресов хватит лишь на несколько десятилетий, в то время как 16-байтовых адресов должно хватить навечно. Другие возражали, что 16 байт для адресов слишком много, тогда как третьи настаивали на 20-байтных адресах для совместимости с дейтаграммным протоколом OSI. Еще одна фракция ратовала за адреса переменной длины. После продолжительных споров, содержащих непечатную лексику, было решено, что наилучшим компромиссным решением являются 16-байтовые адреса фиксированной длины.

Для написания 16-байтовых адресов была выработана новая нотация. Адреса в IPv6 записываются в виде восьми групп по четыре шестнадцатеричных цифры, разделенных двоеточиями, например:

8000:0000:0000:0000:0123:4567:89AB:CDEF

Поскольку многие адреса будут содержать большое количество нулей, были разрешены три метода сокращенной записи адресов. Во-первых, могут быть опущены ведущие нули в каждой группе, например 0123 можно записывать как 123. Во-вторых, одна или более групп, полностью состоящих из нулей, могут заменяться парой двоеточий. Таким образом, приведенный выше адрес принимает вид:

8000::123:4567:89AB:CDEF

Наконец, адреса IPv4 могут записываться как пара двоеточий, после которой пишется адрес в старом десятичном формате, например:

::192.31.20.46

Возможно, об этом нет необходимости говорить столь подробно, но количество всех возможных 16-байтовых адресов очень велико — 2^{128}, что приблизительно равно 3×1023 IP-адресов на квадратный метр. Кто изучал химию, может заметить, что это число больше числа Авогадро. Хотя в планы разработчиков не входило предоставление собственного IP-адреса каждой молекуле на поверхности Земли, они оказались не так далеко от обеспечения такой услуги.

На практике не все адресное пространство используется эффективно, как, например, не используются абсолютно все комбинации телефонных номеров. Например, телефонные номера Манхеттена (код 212) почти полностью заняты, тогда как в штате Вайоминг (код 307) они почти не используются. В RFC 3194 Дьюранд (Durand) и Хуйтема (Huitema) приводят свои вычисления. Утверждается, что если ориентироваться на использование телефонных номеров, то даже при самом пессимистическом сценарии все равно получается более 1000 IP-адресов на квадратный метр поверхности Земли (включая как сушу, так и море). При любом более вероятном сценарии обеспечиваются триллионы адресов на квадратный метр. Таким образом, маловероятно, что в обозримом будущем обнаружится нехватка адресов.

Полезно сравнить заголовок IPv4 (см. рис. 5.41) с заголовком IPv6 (рис. 5.49), чтобы увидеть, что осталось от старого стандарта. Поле *IHL* исчезло, так как заголовок IPv6 имеет фиксированную длину. Поле *Протокол* также было убрано, поскольку поле *Следующий заголовок* сообщает, что следует за последним IP-заголовком (то есть UDP или TCP-сегмент).

Были удалены все поля, относящиеся к фрагментации, так как в протоколе IPv6 используется другой подход к фрагментации. Во-первых, все хосты, поддерживающие протокол IPv6, должны динамически определять нужный размер пакета. Для этого используется технология Path MTU discovery, о которой мы говорили в разделе 5.5.5. Вкратце, когда хост посылает слишком большой IPv6-пакет, вместо того чтобы его фрагментировать, то маршрутизатор, неспособный переслать пакет дальше, посылает обратно сообщение об ошибке. Получив это сообщение, хост должен прекратить всю передачу этому адресату. Гораздо правильнее будет научить все хосты посылать пакеты требуемого размера, нежели учить маршрутизаторы фрагментировать их на лету. Кроме того, минимальный размер пакета был увеличен с 576 до 1280, чтобы можно было передавать 1024 байт данных, плюс множество заголовков.

Наконец, поле *Контрольная сумма* было удалено, так как ее подсчет значительно снижает производительность. Поскольку в настоящее время все шире используются надежные линии связи, а на канальном и транспортном уровнях подсчитываются свои контрольные суммы, наличие еще одной контрольной суммы не стоило бы тех затрат производительности, которых требовал бы ее подсчет. В результате перечисленных удалений получился простой, быстрый и в то же время гибкий протокол сетевого уровня с огромным адресным пространством.

Дополнительные заголовки

В опущенных полях заголовка иногда возникает необходимость, поэтому в протоколе IPv6 была представлена новая концепция (необязательного) **дополнительного заголовка**. На сегодня определены шесть типов дополнительных заголовков, которые перечислены в табл. 5.7. Все они являются необязательными, но в случае использования более чем одного дополнительного заголовка они должны располагаться сразу за фиксированным заголовком, желательно в указанном порядке.

Таблица 5.7. Дополнительные заголовки IPv6

Дополнительный заголовок	Описание
Параметры маршрутизации	Разнообразная информация для маршрутизаторов
Параметры получателя	Дополнительная информация для получателя
Маршрутизация	Частичный список транзитных маршрутизаторов на пути пакета
Фрагментация	Управление фрагментами дейтаграмм
Аутентификация	Проверка подлинности отправителя
Шифрованные данные	Информация о зашифрованном содержимом

У некоторых заголовков формат фиксированный, другие содержат переменное количество полей переменной длины. Для них каждый пункт кодируется в виде тройки (тип, длина, значение). *Тип* представляет собой однобайтовое поле, содержащее код параметра. Первые два бита этого поля сообщают, что делать с пакетом маршрутизаторам, не знающим, как обрабатывать данный параметр. Возможны четыре следующих

варианта: пропустить параметр, игнорировать пакет, игнорировать пакет и отослать обратно ICMP-пакет, а также то же самое, что и предыдущий вариант, но не отсылать обратно ICMP-пакет в случае многоадресной рассылки (чтобы один неверный многоадресный пакет не породил миллионы ICMP-донесений).

Поле *Длина* также имеет размер 1 байт. Оно сообщает, насколько велико значение (от 0 до 255 байт). Поле *Значение* содержит необходимую информацию размером до 255 байт.

Заголовок параметров маршрутизации содержит информацию, которую должны исследовать маршрутизаторы на протяжении всего пути следования пакета. Пока что был определен один вариант использования этого параметра: поддержка дейтаграмм, превышающих 64 Кбайт. Формат заголовка показан на рис. 5.50. При этом полю *Длина полезной нагрузки* в фиксированном заголовке присваивается значение 0.

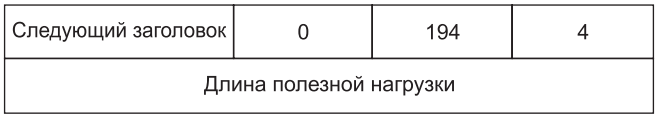


Рис. 5.50. Дополнительный заголовок для больших дейтаграмм

Как и все дополнительные заголовки, он начинается с байта, означающего тип следующего заголовка. Следующий байт содержит длину дополнительного заголовка в байтах, не считая первых 8 байт, являющихся обязательными. С этого начинаются все расширения.

Следующие два байта указывают, что данный параметр содержит размер дейтаграммы (код 194) в виде 4-байтового числа. Размеры меньше 65 536 не допускаются, так как могут привести к тому, что первый же маршрутизатор проигнорирует данный пакет и отошлет обратно ICMP-сообщение об ошибке. Дейтаграммы, использующие подобные расширения заголовка, называются **джамбограммами** (**jumbograms**, от слова «jumbo», означающего нечто большое и неуклюжее). Использование джамбограмм важно для суперкомпьютерных приложений, которым необходимо эффективно передавать по Интернету гигабайты данных.

Заголовок расширяется на поля, которые должны инетерпретироваться только хостом-получателем. В начальной версии IPv6 задавались только «нулевые» настройки для дополнения этого заголовка до кратности 8 байтам, и сам заголовок не использовался. Это делалась для того, чтобы программное обеспечение новых роутеров и хостов могло их обработать, если кто-нибудь подумает о дополнительных настройках в будущем.

Маршрутный заголовок содержит информацию об одном или нескольких маршрутизаторах, которые следует посетить по пути к получателю. Это очень сильно напоминает свободную маршрутизацию стандарта IPv4 в том, что указанные в списке маршрутизаторы должны быть пройдены строго по порядку, тогда как не указанные проходятся между ними. Формат маршрутного заголовка показан на рис. 5.51.

Первые четыре байта дополнительного маршрутного заголовка содержат четыре однобайтовых целых числа. Поля *Следующий заголовок* и *Длина дополнительного заголовка* были описаны ранее. В поле *Тип маршрутизации* описывается формат оставшейся части заголовка. Если он равен 0, это означает, что далее следует зарезервированное 32-разрядное слово, а за ним — некоторое число адресов IPv6. В бу-

дущем, возможно, будут по мере необходимости изобретаться какие-то новые поля. Наконец, в поле *Число оставшихся сегментов* указывается, сколько адресов из списка еще осталось посетить. Его значение уменьшается при прохождении каждого адреса. Когда оно достигает нуля, пакет оставляется на произвол судьбы — никаких указаний относительно его дальнейшего маршрута не дается. Обычно в этот момент пакет уже находится достаточно близко к месту назначения, и оптимальный маршрут очевиден.

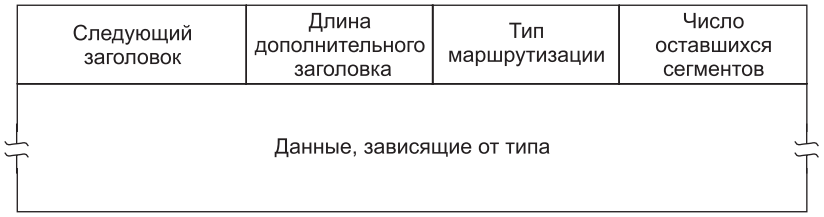


Рис. 5.51. Дополнительный заголовок для маршрутизации

Заголовок фрагментации определяет фрагментацию способом, схожим с протоколом IPv4. Заголовок содержит идентификатор дейтаграммы, номер фрагмента и бит, информирующий о том, является ли этот фрагмент последним. В отличие от IPv4 в протоколе IPv6 фрагментировать пакет может только хост-источник. Маршрутизаторы фрагментировать пересылаемые пакеты не могут. Хотя это изменение можно считать отказом от оригинальной философии IP, оно вполне в духе современного применения IPv4. Вдобавок оно упрощает и ускоряет работу маршрутизаторов. Как уже было сказано, маршрутизатор отвергает слишком большие пакеты, посылая в ответ ICMP-пакет, указывающий хосту-источнику на необходимость заново передать пакет, выполнив его фрагментацию на меньшие части.

Заголовок аутентификации предоставляет механизм подтверждения подлинности отправителя пакета. Шифрование данных, содержащихся в поле полезной нагрузки, обеспечивает конфиденциальность: прочесть содержимое пакета сможет только тот, для кого предназначен пакет. Для выполнения этих задач в заголовках используются криптографические методы, которые будут рассмотрены в главе 8.

Полемика

При той открытости, с которой происходил процесс разработки протокола IPv6, и при убежденности многочисленных разработчиков в собственной правоте неудивительно, что многие решения принимались в условиях весьма жарких дискуссий. О некоторых из них будет рассказано ниже. Все кровавые подробности описаны в соответствующих RFC.

О спорах по поводу длины поля адреса уже упоминалось. В результате было принято компромиссное решение: 16-байтовые адреса фиксированной длины.

Другое сражение разгорелось из-за размера поля *Максимальное количество транзитных участков*. Один из противостоящих друг другу лагерей считал, что ограничение количества транзитных участков числом 255 (это явно следует из использования 8-битного поля) является большой ошибкой. В самом деле, маршруты из 32 транзитных участков уже стали обычными, а через 10 лет могут стать обычными более длин-

ные маршруты. Сторонники этого лагеря заявляли, что использование полей адресов огромного размера было решением дальновидным, а крохотных счетчиков транзитных участков — недальновидным. Самый страшный грех, который, по их мнению, могут совершить специалисты по вычислительной технике, — это выделить для чего-нибудь недостаточное количество разрядов.

В ответ им было заявлено, что подобные аргументы можно привести для увеличения любого поля, что приведет к разбуханию заголовка. Кроме того, назначение поля *Максимальное количество транзитных участков* состоит в том, чтобы не допустить слишком долгого странствования пакетов, и 65 535 транзитных участков — это очень много. К тому же, по мере роста Интернета будет создаваться все большее количество междугородных линий, что позволит передавать пакеты из любой страны в любую страну максимум за шесть транзитных пересылок. Если от получателя или отправителя до соответствующего международного шлюза окажется более 125 транзитных участков, то, видимо, что-то не в порядке с магистралями этого государства. В итоге эту битву выиграли сторонники 8-битового счетчика.

Еще одним предметом спора оказался максимальный размер пакета. Обладатели суперкомпьютеров настаивали на размере пакетов, превышающем 64 Кбайт. Когда суперкомпьютер начинает передачу, он занимается серьезным делом и не хочет, чтобы его прерывали через каждые 64 Кбайта. Аргумент против больших пакетов заключается в том, что если пакет размером в 1 Мбайт будет передаваться по линии T1 со скоростью 1,5 Мбит/с, то он займет линию на целых 5 с, что вызовет слишком большую задержку, заметную для интерактивных пользователей. В данном вопросе удалось достичь компромисса: нормальные пакеты ограничены размером 64 Кбайт, но с помощью дополнительного заголовка можно пересылать дейтаграммы огромного размера.

Еще одним спорным вопросом оказалось удаление контрольной суммы IPv4. Кое-кто сравнивал этот ход с удалением тормозов из автомобиля. При этом автомобиль становится легче и может двигаться быстрее, но если случится что-нибудь неожиданное, то могут быть проблемы.

Аргумент против контрольных сумм состоял в том, что каждое приложение, действительно заботящееся о целостности своих данных, все равно считает контрольную сумму на транспортном уровне, поэтому наличие еще одной на сетевом уровне является излишним (кроме того, контрольная сумма подсчитывается еще и на уровне передачи данных). Более того, эксперименты показали, что вычисление контрольной суммы составляло основные затраты протокола IPv4. Это сражение было выиграно лагерем противников контрольной суммы, поэтому в протоколе IPv6, как мы знаем, контрольной суммы нет.

Вокруг мобильных хостов также разгорелся спор. Если мобильный компьютер окажется на другом конце Земли, сможет ли он продолжать использовать прежний IPv6-адрес, или он должен будет использовать схему с внутренним и внешним агентами? Появилось много желающих создать в протоколе IPv6 явную поддержку мобильных хостов. Эти попытки потерпели поражение, поскольку ни по одному конкретному предложению не удалось достичь консенсуса.

Вероятно, самые жаркие баталии разгорелись вокруг вопроса безопасности. Все были согласны, что это необходимо. Спорным было то, где и как следует реализовывать безопасность. Во-первых, где. Аргументом за размещение системы безопасности

на сетевом уровне было то, что при этом она становится стандартной службой, которой могут пользоваться все приложения безо всякого предварительного планирования. Контраргумент заключался в том, что по-настоящему защищенным приложениям подходит лишь сквозное шифрование, когда шифрование осуществляется самим источником, а дешифровка — непосредственным получателем. Во всех остальных случаях пользователь оказывается в зависимости от, возможно, содержащей ошибки реализации сетевого уровня, над которой у него нет контроля. В ответ на этот аргумент можно сказать, что приложение может просто отказаться от использования встроенных в IP функций защиты и выполнять всю эту работу самостоятельно. Возражение на этот контраргумент состоит в том, что пользователи, не доверяющие сетям, не хотят платить за не используемую ими функцию, реализация которой утяжеляет и замедляет работу протокола, даже если сама функция отключена.

Другой аспект вопроса расположения системы безопасности касается того факта, что во многих (но не во всех) странах приняты строгие экспортные законы, касающиеся криптографии. В некоторых странах, особенно во Франции и Ираке, строго запрещено использование криптографии даже внутри страны, чтобы у населения не могло быть секретов от полиции. В результате любая реализация протокола IP, использующая достаточно мощную криптографическую систему, не может быть экспортирована за пределы Соединенных Штатов (и многих других стран). Таким образом, приходится поддерживать два набора программного обеспечения — один для внутреннего использования и другой для экспорта, против чего решительно выступает большинство производителей компьютеров.

Единственный вопрос, по которому не было споров, состоял в том, что никто не ожидает, что IPv4-интернет будет выключен в воскресенье, а в понедельник утром будет включен уже IPv6-интернет. Вместо этого вначале появятся «островки» IPv6, которые будут общаться по туннелям (см. раздел 5.5.3). По мере своего роста, острова IPv6 будут объединяться в более крупные острова. Наконец, все острова объединятся, и Интернет окажется полностью трансформированным.

Так, по крайней мере, выглядел план. Внедрение IPv6 оказалось его ахиллесовой пятой. Этот протокол до сих пор очень мало используется, хотя большинство операционных систем полностью поддерживают его. Как правило, IPv6 применяется в тех случаях, когда оператору какой-либо сети — например, мобильной — требуются дополнительные IP-адреса. Чтобы упростить переход к новому протоколу, было придумано много разных стратегий. Среди них методы автоматической настройки туннелей, обеспечивающих передачу IPv6-пакетов через сеть IPv4, и технологии, позволяющие хостам автоматически находить конечную точку туннеля. Двухстековые хосты поддерживают как IPv4, так и IPv6 и могут выбирать протокол в зависимости от адреса назначения пакета. Эти стратегии помогут ускорить процесс перехода к IPv6, когда он будет неизбежным. Более подробно об этом можно прочитать в книге (Davies, 2008).

5.6.4. Управляющие протоколы Интернета

Помимо протокола IP, используемого для передачи данных, в Интернете есть несколько дополнительных управляющих протоколов, применяемых на сетевом уровне, к которым относятся ICMP, ARP и DHCP. В данном разделе мы рассмотрим их все

по очереди, описывая те версии, которые соответствуют IPv4 (так как именно они сейчас широко применяются). У ICMP и DHCP существуют аналогичные версии для IPv6; эквивалентом APR является NDP (Neighbor Discovery Protocol — протокол обнаружения соседей).

ICMP — протокол управляющих сообщений Интернета

За работой Интернета следят маршрутизаторы. Если во время обработки пакета маршрутизатором случается что-то неожиданное, о происшествии сообщается по протоколу **ICMP (Internet Control Message Protocol — протокол управляющих сообщений Интернета)**, используемому также для тестирования Интернета. Протоколом ICMP определено около дюжины типов сообщений. Каждое ICMP-сообщение вкладывается в IP-пакет. Наиболее важные из них приведены в табл. 5.8.

Таблица 5.8. Основные типы ICMP-сообщений

Тип сообщения	Описание
Адресат недоступен	Пакет не может быть доставлен
Время истекло	Время жизни пакета упало до нуля
Проблема с параметром	Неверное поле заголовка
Гашение источника	Сдерживающий пакет
Переадресовать	Научить маршрутизатор географии
Запрос отклика и отклик	Проверить, жива ли машина
Запрос временного штампа и ответ	То же, что и отклик, но с временным штампом
Объявление маршрутизатора/запрос к маршрутизатору	Найти близлежащий маршрутизатор

Сообщение АДРЕСАТ НЕДОСТУПЕН (DESTINATION UNREACHABLE) используется, когда маршрутизатор не может обнаружить пункт назначения, или когда пакет с битом *DF* (не фрагментировать) не может быть доставлен, так как путь ему преграждает сеть с маленьким размером пакетов.

Сообщение ВРЕМЯ ИСТЕКЛО (TIME EXCEEDED) посылается, когда пакет игнорируется, так как его счетчик *Время жизни* уменьшился до нуля. Это событие является признаком того, что пакеты двигаются по замкнутым контурам или что установлено слишком низкое значение таймера.

Хитрый способ использования этого сообщения, предложенный Ван Джейкобсоном в 1987 году, — утилита **traceroute**. Она находит маршрутизаторы, расположенные в узлах пути от хоста к адресу назначения. При этом ей не требуется особый уровень поддержки. Метод состоит в отправке на адрес назначения последовательности пакетов с временем жизни 1, 2, 3 и т. д. Маршрутизаторы, на которых счетчики достигают нуля, располагаются в том порядке, в котором пакет проходит их при пересылке. Эти маршрутизаторы послушно отправляют обратно на хост сообщения ВРЕМЯ ИСТЕКЛО. По этим сообщениям хост определяет их IP-адреса и получает информацию о пути. Сообщение ВРЕМЯ ИСТЕКЛО, конечно, предназначалось не для этого. Но вполне возможно, что это самый полезный инструмент отладки сети всех времен.

Сообщение ПРОБЛЕМА ПАРАМЕТРА (PARAMETER PROBLEM) указывает на то, что обнаружено неверное значение в поле заголовка. Это является признаком наличия ошибки в программном обеспечении хоста, отправившего этот пакет, или промежуточного маршрутизатора.

Сообщение ГАШЕНИЕ ИСТОЧНИКА (SOURCE QUENCH) ранее использовалось для усмирения хостов, которые отправляли слишком много пакетов. Хост, получивший такое сообщение, должен был снизить обороты. В настоящее время подобное сообщение редко используется, так как при возникновении перегрузки подобные пакеты только подливают масла в огонь, еще больше загружая сеть. К тому же, не ясно, как на них отвечать. Теперь борьба с перегрузкой в Интернете осуществляется в основном на транспортном уровне; при этом сигналом перегрузки является утеря пакетов. Это будет подробно обсуждаться в главе 6.

Сообщение ПЕРЕАДРЕСОВАТЬ (REDIRECT) посылается хосту, отправившему пакет, когда маршрутизатор замечает, что пакет адресован неверно. Таким образом маршрутизатор предлагает хосту обновить маршрут.

Сообщения ЗАПРОС ОТКЛИКА (ECHO) и ОТКЛИК (ECHO REPLY) посылаются, чтобы определить, достижим ли и жив ли в данный момент конкретный адресат. Получив сообщение ЗАПРОС ОТКЛИКА, хост должен отправить обратно сообщение ОТКЛИК. Эти сообщения используются утилитой **ping**, которая проверяет, включен ли хост и подключен ли он к Интернету.

Сообщения ЗАПРОС ВРЕМЕННОГО ШТАМПА (TIMESTAMP REQUEST) и ОТКЛИК С ВРЕМЕННЫМ ШТАМПОМ (TIMESTAMP REPLY) имеют то же назначение, но при этом в ответе проставляется время получения сообщения и время отправления ответа. Это сообщение используется для измерения производительности сети.

Сообщения ОБЪЯВЛЕНИЕ МАРШРУТИЗАТОРА (ROUTER ADVERTISEMENT) и ЗАПРОС К МАРШРУТИЗАТОРУ (ROUTER SOLICITATION) позволяют хостам находить близлежащие маршрутизаторы. Хосту необходимо знать IP-адрес хотя бы одного из таких маршрутизаторов, чтобы он мог передавать пакеты за пределы локальной сети.

Кроме перечисленных сообщений определены и другие. Их полный список хранится в Интернете по адресу *www.iana.org/assignments/icmp-parameters*.

ARP — протокол разрешения адресов

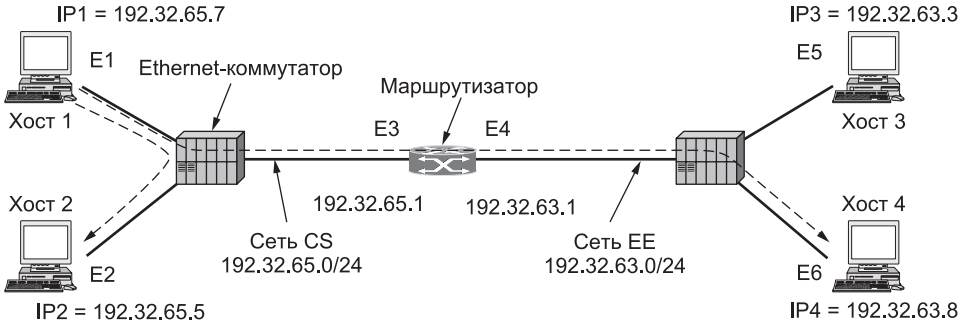
Хотя у каждой машины в Интернете есть один или более IP-адресов, их недостаточно для отправки пакетов. Сетевые карты канального уровня, такие как Ethernet-карты, не понимают интернет-адресов. Например, каждая когда-либо выпущенная сетевая карта Ethernet имеет 48-разрядный Ethernet-адрес. Производители сетевых карт Ethernet запрашивают у IEEE блок адресов, что гарантирует уникальность Ethernet-адресов (это позволяет избежать конфликтов при наличии одинаковых сетевых карт в одной ЛВС). Сетевые карты отправляют и принимают кадры, основываясь на 48-разрядных Ethernet-адресах. О 32-разрядных IP-адресах им ничего не известно.

Таким образом, возникает вопрос: как устанавливается соответствие IP-адресов и адресов уровня передачи данных, таких как Ethernet-адреса? Чтобы понять, как это работает, рассмотрим показанный на рис. 5.52 пример, в котором изображен небольшой университет с двумя сетями /24. На рисунке мы видим две коммутируемые сети Ethernet: одна сеть (CS) на факультете кибернетики, с префиксом 192.31.65.0/24,

а другая ЛВС (EE) — с префиксом 192.31.63.0/24 на электротехническом факультете. Они соединены IP-маршрутизатором. У каждой машины сетей Ethernet и у каждого интерфейса на маршрутизаторе есть уникальный Ethernet-адрес (на рисунке — от E1 до E6), и уникальный IP-адрес в сети CS или EE.

Рассмотрим, как пользователь хоста 1 посылает пакет пользователю хоста 2 в сети CS. Допустим, отправителю известно имя получателя, например *eagle.cs.uni.edu*. Сначала надо найти IP-адрес для хоста 2. Этот поиск осуществляется службой имен доменов DNS (Domain Name System), которую мы рассмотрим в главе 7. На данный момент мы просто предположим, что служба DNS возвращает IP-адрес для хоста 2 (192.31.65.5).

Теперь программное обеспечение верхнего уровня хоста 1 создает пакет со значением 192.31.65.5 в поле *Адрес получателя* и передает его IP-программе для пересылки. Программное обеспечение протокола IP может посмотреть на адрес и увидеть, что адресат находится в сети CS (то есть в его собственной сети), но ему нужно как-то определить Ethernet-адрес получателя. Одно из решений состоит в том, чтобы хранить в системе конфигурационный файл, в котором были бы перечислены соответствия всех локальных IP-адресов Ethernet-адресам. Такое решение, конечно, возможно, но в организациях с тысячами машин обновление этих файлов потребует много времени и подвержено ошибкам.



Кадр	IP-адрес отправителя	Ethernet-адрес отправителя	IP-адрес получателя	Ethernet-адрес получателя
От хоста 1 к 2 в сети CS	IP1	E1	IP2	E2
От хоста 1 к 4 в сети CS	IP1	E1	IP4	E3
От хоста 1 к 4 в сети EE	IP1	E4	IP4	E6

Рис. 5.52. Две коммутируемые ЛВС Ethernet, соединенные маршрутизатором

Более удачное решение заключается в рассылке хостом 1 по сети Ethernet широковещательного пакета с вопросом: «Кому принадлежит IP-адрес 192.31.65.5?» Этот пакет будет получен каждой машиной сети CS Ethernet и каждая проверит его IP-адрес. Только хост 2 ответит на вопрос своим Ethernet-адресом E2. Таким образом, хост 1 узнает, что IP-адрес 192.31.65.5 принадлежит хосту с Ethernet-адресом E2. Протокол, который задает подобный вопрос и получает ответ на него, называется **ARP (Address Resolution Protocol — протокол разрешения адресов)** и описан в RFC 826. Он работает почти на каждой машине в Интернете.

Преимущество протокола ARP над файлами конфигурации заключается в его простоте. Системный администратор должен всего лишь назначить каждой машине IP-адрес и решить вопрос с маской подсети. Все остальное сделает протокол ARP.

Затем программное обеспечение протокола IP хоста 1 создает Ethernet-кадр для E2, помещает в его поле полезной нагрузки IP-пакет, адресованный 192.31.65.5, и посылает его по сети Ethernet. IP- и Ethernet-адреса этого пакета приведены на рис. 5.52. Сетевая карта Ethernet хоста 2 обнаруживает кадр, замечает, что он адресован ей, считывает его и вызывает прерывание. Ethernet-драйвер извлекает IP-пакет из поля полезной нагрузки и передает его IP-программе, которая, видя, что пакет адресован правильно, обрабатывает его.

Существуют различные методы повышения эффективности протокола ARP. Во-первых, машина, на которой работает протокол ARP, может запоминать результат преобразования адреса на случай, если ей придется снова связываться с той же машиной. В следующий раз она найдет нужный адрес в своем кэше, сэкономив, таким образом, на рассылке широковещательного пакета. Скорее всего, хосту 2 понадобится отослать ответ на пакет, что также потребует от него обращения к ARP для определения адреса отправителя. Этого обращения можно избежать, если отправитель включит в ARP-пакет свои IP- и Ethernet-адреса. Когда широковещательный ARP-пакет прибудет на хост 2, пара (192.31.65.7, E1) будет сохранена хостом 2 в ARP-кэше для будущего использования. Более того, эту пару адресов могут сохранить у себя все машины сети Ethernet.

Чтобы разрешить изменение соответствий адресов, например, если хост использует новый IP-адрес (но Ethernet-адрес остается прежним), записи в ARP-кэше должны устаревать за несколько минут. Существует хороший способ поддержания актуальности информации об адресах в кэше, улучшая при этом производительность. Идея в том, что каждая машина может рассылать свою пару адресов во время настройки. Обычно эта широковещательная рассылка производится в виде ARP-пакета, запрашивающего свой собственный IP-адрес. Ответа на такой запрос быть не должно, но все машины могут запомнить эту пару адресов. Это называется **добровольным ARP-сообщением (gratuitous ARP)**. Если ответ все же (неожиданно) придет, это будет означать, что двум машинам назначен один и тот же IP-адрес. Эти машины не смогут пользоваться сетью, пока проблема не будет решена системным администратором.

Посмотрим снова на рис. 5.52. Пусть на этот раз хост 1 хочет послать пакет хосту 4 (192.31.63.8) в сети EE. Хост 1 увидит, что IP-адрес получателя не относится к сети CS. Он знает, что такие внешние пакеты нужно передавать на маршрутизатор, который иногда называют **шлюзом по умолчанию (default gateway)**. По соглашению принято, что шлюз по умолчанию имеет наименьший адрес сети (198.31.65.1). Но чтобы отправить кадр на этот маршрутизатор, хост 1 должен знать еще и Ethernet-адрес интерфейса маршрутизатора в сети CS. Поэтому он отправляет широковещательный ARP-пакет для 198.31.65.1 и узнает E3. После этого он отправляет кадр. Аналогичным образом пакеты передаются от одного маршрутизатора к другому на всем пути до места назначения.

Когда сетевая карта Ethernet получает этот кадр, она передает пакет на обработку программным средствам IP. По сетевым маскам маршрутизатор понимает, что пакет должен быть доставлен на хост 4 в сети EE. Если ему неизвестен Ethernet-адрес

хоста 4, он снова будет использовать ARP. В таблице на рис. 5.52 приведен список Ethernet- и IP-адресов из кадров сетей CS и EE. Обратите внимание на то, что для одного кадра в разных сетях Ethernet-адреса меняются, а IP-адреса — нет (так как они указывают на конечную точку во всех объединенных сетях).

Существует способ передать пакет от хоста 1 хосту 4 так, чтобы отправитель не знал, что получатель находится в другой сети. Для этого нужно, чтобы маршрутизатор отвечал на ARP-запросы сети CS для хоста 4, передавая при этом свой Ethernet-адрес E3. Хост 4 не ответит, так как не увидит широковещательного пакета (маршрутизаторы не переправляют широковещательные пакеты Ethernet-уровня). В результате маршрутизатор получит кадры для 192.32.63.8 и передаст их в сеть EE. Такой метод называется **ARP-прокси (ARP-proxy)**. Он используется в особых случаях, когда хосту требуется сымитировать свое присутствие в какой-либо сети. Например, если портативному компьютеру нужен еще один узел, который принимал бы пакеты для него, когда компьютер находится вне домашней сети.

Протокол динамической конфигурации узла DHCP

Протокол ARP (как и другие интернет-протоколы) предполагает, что хосты обладают базовыми сведениями, например, знают свой IP-адрес. Но как хосты получают эту информацию? Можно настраивать их вручную, это очень трудоемкий процесс, к тому же часто приводящий к ошибкам. Есть более удобный способ: он называется **DHCP (Dynamic Host Configuration Protocol — протокол динамической настройки хостов)**.

Каждая сеть должна иметь DHCP-сервер, отвечающий за настройки. При запуске у каждого компьютера есть встроенный в сетевую карту Ethernet-адрес или другой адрес канального уровня, но нет IP-адреса. Для отыскания своего IP-адреса компьютер широковещательным способом распространяет специальный пакет `DHCP DISCOVER`. Он должен прибыть на DHCP-сервер. Если этот сервер не подключен к сети напрямую, пакет будет ретранслирован на DHCP-сервер независимо от того, где он находится.

Когда сервер получает пакет, он выделяет свободный IP-адрес и отправляет его обратно с помощью пакета `DHCP OFFER` (который также может ретранслироваться). Чтобы это было возможным, даже если у хоста нет IP-адреса, сервер определяет хост по его Ethernet-адресу (который содержится в пакете `DHCP DISCOVER`).

Встает вопрос: на какое время можно выдавать в автоматическом режиме IP-адреса из пула? Если хост покинет сеть и не освободит захваченный адрес, этот адрес будет навсегда утерян. С течением времени будет теряться все больше адресов. Для предотвращения этих неприятностей нужно выдавать IP-адреса не навсегда, а на определенное время. Такая технология называется **лизингом (leasing)**. Перед окончанием срока действия лизинга хост должен послать на DHCP-сервер запрос о продлении срока пользования IP-адресом. Если такой запрос не был сделан или в просьбе было отказано, хост не имеет права продолжать использование выданного ранее адреса.

Протокол DHCP описан в RFC 2131 и 2132. Он широко используется в Интернете для настройки ряда параметров и приписывания IP-адресов. Помимо сетей предприятий и домашних сетей, DHCP используют интернет-провайдеры. С его помощью они настраивают устройства через интернет-соединение, чтобы абонентам не приходилось узнавать эту информацию у своего провайдера по телефону. Чаще всего с помощью

DHCP передается маска сети, IP-адрес шлюза по умолчанию, а также IP-адреса DNS и серверов времени. DHCP во многом заменил своих предшественников (RARP и BOOTP), функциональность которых оставляла желать лучшего.

5.6.5. Коммутация меток и MPLS

До сих пор, путешествуя по сетевому уровню Интернета, мы говорили в основном о пакетах и дейтаграммах, передаваемых IP-маршрутизаторами. Сейчас все более популярной (особенно у провайдеров) становится еще одна технология, позволяющая передавать интернет-трафик по сети. Она называется **MPLS (MultiProtocol Label Switching — мультипротокольная коммутация меток)** и находится в опасной близости к коммутации каналов. Хотя многие члены интернет-сообщества испытывают неприязнь к сетям, ориентированным на соединение, похоже, что эта идея снова становится популярной. Как однажды сказал Йоги Берра, и снова это дежавю. Но в том, как маршруты создаются в Интернете и в сетях, ориентированных на соединение, есть существенная разница. Так что этот метод — не то же самое, что коммутация каналов.

MPLS присваивает пакету специальную метку, и пересылка производится не по адресу, а по этой метке. Если добавлять метки во внутреннюю таблицу, выбор выходного канала будет соответствовать поиску в этой таблице. Это существенно ускоряет пересылку. Эта идея легла в основу MPLS, которая изначально разрабатывалась как патентованная технология, известная под разными именами — например, **коммутация меток (tag switching)**. В конечном счете, проблемная группа IETF занялась стандартизацией идей. Стандарт описан в документе RFC 3031 и многих других. Главными преимуществами, проверенными временем, являются гибкая маршрутизация и быстрая передача пакетов, позволяющая обеспечить необходимое качество обслуживания.

Первая проблема состоит вот в чем: куда поставить метку? Поскольку IP-пакеты не предназначены для виртуальных каналов, в их заголовке не предусмотрено место для номеров виртуальных каналов. Следовательно, нужно добавлять новый заголовок MPLS в начало IP-пакета. На линии между маршрутизаторами, используется протокол, включающий заголовки PPP, MPLS, IP и TCP, как показано на рис. 5.53.

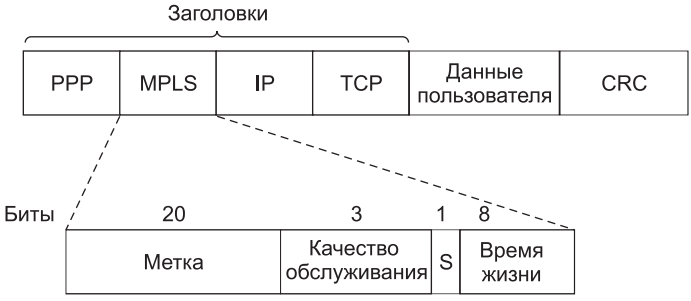


Рис. 5.53. Передача TCP-сегмента с использованием IP, MPLS и PPP

Обычно в заголовок MPLS входит четыре поля, наиболее важное из которых — поле *Метка*, значением которой является индекс. Поле *Качество обслуживания* указывает на применяемый класс обслуживания. Поле *S* связано со стеком меток (речь об

этом пойдет ниже). Поле *Время жизни* показывает, сколько еще раз пакет может быть отправлен. Его значение уменьшается на каждом маршрутизаторе; если оно равно 0, пакет игнорируется. Благодаря этому исключаются бесконечные циклы в случае сбоя маршрутизации.

MPLS располагается между протоколом сетевого уровня IP и протоколом канального уровня PPP. Этот уровень нельзя назвать третьим, так как метки задаются на основе IP-адреса или другого адреса сетевого уровня. Но это и не второй уровень, хотя бы потому, что MPLS контролирует передачу пакета на нескольких транзитных участках, а не на одном. Поэтому иногда этот протокол называют протоколом уровня 2.5. Это яркий пример того, что реальные протоколы не всегда вписываются в нашу идеальную уровневую модель.

Так как заголовки MPLS не являются частью пакетов сетевого уровня и также не имеют отношения к кадрам канального уровня, MPLS является методом, не зависящим от обоих этих уровней. Кроме всего прочего, это свойство означает, что можно создать такие коммутаторы MPLS, которые могут пересылать как IP-пакеты, так и не-IP-пакеты, в зависимости от того, что необходимо в каждом конкретном случае. Именно отсюда следует «мультипротокольность» метода, отраженная в его названии. MPLS может также передавать IP-пакеты через не-IP-сети.

Когда пакет, расширенный за счет заголовка MPLS, прибывает на **маршрутизатор коммутации меток (LSR, Label Switched Router)** извлеченная из него метка используется в качестве индекса таблицы, по которой определяется исходящая линия и значение новой метки. Смена меток используется во всех сетях с виртуальными каналами. Метки имеют только локальное значение, и два разных маршрутизатора могут снабдить независимые пакеты одной и той же меткой, если их нужно направить на одну и ту же линию третьего маршрутизатора. Поэтому, чтобы метки можно было различить на приемном конце, их приходится менять при каждом переходе. Мы видели этот механизм в действии — он был графически изображен на рис. 5.3. В MPLS используется такой же метод.

Вообще-то, иногда различают *передачу (forwarding)* и *коммутацию (switching)*. Передача — это процесс нахождения адреса, наиболее совпадающего с адресом назначения, в таблице маршрутизации, чтобы решить, куда отправлять пакет. Примером IP-передачи является алгоритм поиска наиболее длинного совпадающего префикса. При коммутации в таблице маршрутизации производится поиск по меткам, извлеченным из пакета. Это проще и быстрее. Правда, такие определения не очень-то универсальны.

Так как большинство хостов и маршрутизаторов не понимают MPLS, мы могли бы задаться вопросом, как и когда метки прикрепляются к пакетам. Это происходит в тот момент, когда пакет достигает границы MPLS-сети. **Пограничный маршрутизатор (LER, Label Edge Router)** проверяет IP-адрес назначения и другие поля, определяя, по какому MPLS-пути должен пойти пакет, и присваивает пакету соответствующую метку. По ней пакет передается в сети MPLS. На другой границе MPLS-сети метка уже не нужна, и она удаляется, после чего IP-пакет становится открытым для другой сети. Этот процесс показан на рис. 5.54. Одним из отличий от традиционных виртуальных каналов является уровень агрегации. Конечно, можно каждому потоку, проходящему через MPLS-сеть, предоставить собственный набор меток. Однако более распространенным приемом является группировка потоков, заканчивающихся на данном

маршрутизаторе или в данной ЛВС, и использование одной метки для всех таких потоков. О потоках, сгруппированных вместе и имеющих одинаковые метки, говорят, что они принадлежат одному **классу эквивалентности пересылок (FEC — Forwarding Equivalence Class)**. В такой класс входят пакеты, не только идущие по одному и тому же маршруту, но и обслуживаемые по одному классу (в терминах дифференцированного обслуживания). Такие пакеты воспринимаются при пересылке одинаково.

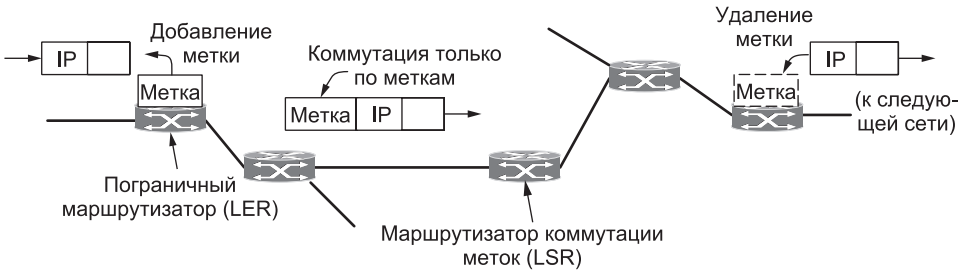


Рис. 5.54. Передача IP-пакета через MPLS-сеть

При традиционной маршрутизации с использованием виртуальных каналов невозможно группировать разные пути с разными конечными пунктами в один виртуальный канал, потому что адресат не сможет их различить. В MPLS пакеты содержат не только метку, но и адрес назначения, поэтому в конце помеченного пути заголовок с меткой может быть удален, и дальнейшая маршрутизация может осуществляться традиционным способом — с использованием адреса назначения сетевого уровня.

На самом деле MPLS идет дальше. Этот протокол может работать одновременно на нескольких уровнях, используя несколько меток. Представьте себе несколько пакетов с разными метками (например, если сеть должна по-разному их обрабатывать), которые должны пройти один и тот же путь до определенного адреса. В таком случае мы можем задать для всех пакетов один путь. Когда пакеты, уже имеющие метку, попадают на начало пути, в начало записывается новая метка. Это называется стеком меток. Самая наружная метка служит проводником пакета по пути. В конце пути она удаляется, и оставшиеся метки (если они есть) ведут пакет дальше. Бит *S* (см. рис. 5.53) позволяет маршрутизатору, удаляющему метку, узнать, остались ли у пакета еще метки. Единичное значение бита говорит о том, что метка — последняя в стеке, а нулевое значение говорит об обратном.

Наконец, остается понять, как устроены таблицы передачи по меткам. В этом вопросе MPLS существенно отличается от традиционных схем с виртуальными каналами. В традиционных сетях пользователь, желающий установить соединение, посылает установочный пакет для создания пути и соответствующей ему записи в таблице. В MPLS этого не происходит, потому что в этом методе вообще отсутствует установочная фаза для каждого соединения (в противном случае пришлось бы менять слишком большую часть существующего программного обеспечения Интернета).

Вместо этого информация, необходимая для передачи, задается специальным протоколом, который совмещает функции протокола маршрутизации и протокола установления соединения. Этот управляющий протокол полностью отделен от передачи меток, что позволяет использовать множество различных управляющих протоколов.

Имеется несколько вариантов этого подхода. Один из них работает следующим образом. При загрузке маршрутизатора выявляется, для каких маршрутов он является пунктом назначения (например, какие префиксы принадлежат его интерфейсам). Для них создается один или несколько FEC, каждому из них выделяется метка, значение которой сообщается соседям. Соседи, в свою очередь, заносят эти метки в свои таблицы пересылки и посылают новые метки своим соседям. Процесс продолжается до тех пор, пока все маршрутизаторы не получат представление о маршрутах. По мере формирования путей могут резервироваться ресурсы, что позволяет обеспечить надлежащее качество обслуживания. В других вариантах устанавливаются другие пути (например, пути управления трафиком, учитывающие неиспользуемую пропускную способность) или создаются пути «по требованию», предоставляющие нужный уровень обслуживания.

Хотя основные идеи MPLS довольно просты, детали этого метода чрезвычайно сложны, причем существует множество вариаций, находящихся в стадии активной разработки. Дополнительную информацию можно найти в книгах (Davie и Farrel, 2008; Davie и Rekhter, 2000).

5.6.6. Протокол внутреннего шлюза OSPF

Итак, мы завершили изучение процесса передачи пакетов в Интернете. Пришло время перейти к новой теме — маршрутизации в Интернете. Как уже упоминалось, Интернет состоит из большого количества независимых сетей или автономных систем (АС), которыми управляют различные организации — компании, университеты, провайдеры. Каждая автономная система может использовать собственный внутренний, или **внутридоменный**, алгоритм маршрутизации (**intradomain routing**). Тем не менее более-менее распространенных стандартных протоколов совсем немного. В данном разделе будет рассмотрена внутридоменная маршрутизация и популярный протокол OSPF. Алгоритм маршрутизации внутри автономной системы называется **протоколом внутреннего шлюза** (**interior gateway protocol**). В следующем разделе мы обсудим вопрос маршрутизации между независимыми сетями, или **междоменной** маршрутизации (**interdomain routing**). В данном случае все сети должны использовать один и тот же алгоритм маршрутизации или **протокол внешнего шлюза** (**exterior gateway protocol**). В Интернете используется протокол BGP (Border Gateway Protocol — пограничный межсетевой протокол).

Изначально в качестве протокола внутридоменной маршрутизации использовалась схема маршрутизации по вектору расстояний, основанная на распределенном алгоритме Беллмана—Форда (Bellman—Ford) и унаследованная из ARPANET. В первую очередь это RIP (Routing Information Protocol — протокол маршрутной информации), использующийся до сих пор. Он хорошо работал в небольших системах, но по мере увеличения автономных систем стали проявляться его недостатки, такие как проблема счета до бесконечности и медленная сходимость, поэтому в мае 1979 года он был заменен протоколом состояния каналов. В 1988 году проблемная группа проектирования Интернета (IETF, Internet Engineering Task Force) начала работу над протоколом, учитывающим состояние линий, для внутридоменной маршрутизации. Этот протокол под названием **OSPF** (**Open Shortest Path First — открытый алгоритм предпочтительного выбора кратчайшего маршрута**) был признан стандартом в 1990 году. Идея

была заимствована из протокола **IS-IS (Intermediate System to Intermediate System — связь между промежуточными системами)**, ставшего стандартом ISO. Эти два протокола скорее похожи, чем различны. Более подробное описание см. в RFC 2328. Они являются основными протоколами внутридоменной маршрутизации и в настоящее время поддерживаются многочисленными производителями маршрутизаторов. OSPF чаще используется в корпоративных сетях, IS-IS — в сетях интернет-провайдеров. Ниже будет дано краткое описание работы протокола OSPF.

Учитывая большой опыт работы с различными алгоритмами, группа разработчиков согласовывала свои действия с длинным списком требований, которые необходимо было удовлетворить. Во-первых, этот алгоритм должен публиковаться в открытой литературе, откуда буква «O» (Open — открытый) в OSPF. Из этого следовало, что патентованный алгоритм, принадлежащий одной компании, не годится. Во-вторых, новый протокол должен был уметь учитывать широкий спектр различных параметров, включая физическое расстояние, задержку и т. д. В-третьих, этот алгоритм должен был быть динамическим, а также автоматически и быстро адаптирующимся к изменениям топологии.

В-четвертых (это требование впервые было предъявлено именно к OSPF), он должен был поддерживать выбор маршрутов, основываясь на типе сервиса. Новый протокол должен был уметь по-разному выбирать маршрут для трафика реального времени и для других видов трафика. В то время IP-пакет содержал поле *Тип сервиса*, но ни один из имевшихся протоколов маршрутизации не использовал его. Это поле было и в OSPF, но и здесь никто его не использовал. Поэтому в результате его убрали. Возможно, это требование опередило свое время, так как было выдвинуто до появления дифференцированного обслуживания, вернувшего к жизни классы обслуживания.

В-пятых, новый протокол должен был уметь распределять нагрузку на линии. Это связано с предыдущим пунктом. Большинство протоколов посылало все пакеты по одному лучшему маршруту, даже если таких маршрутов два. Следующий по оптимальности маршрут не использовался совсем. Между тем, во многих случаях распределение нагрузки по нескольким линиям дает лучшую производительность.

В-шестых, необходима поддержка иерархических систем. К 1988 году Интернет вырос настолько, что ни один маршрутизатор не мог вместить сведения о его полной топологии. Таким образом, требовалась разработка нового протокола.

В-седьмых, требовался необходимый минимум безопасности, защищающий маршрутизаторы от обманывающих их студентов-шутников, присылающих неверную информацию о маршруте. Наконец, требовалась поддержка для маршрутизаторов, соединенных с Интернетом по туннелю. Предыдущие протоколы справлялись с этим неудовлетворительно.

Протокол OSPF поддерживает двухточечные линии (например, SONET) и широковещательные сети (большинство ЛВС). Он также поддерживает сети с множественными маршрутизаторами, каждый из которых может напрямую соединяться с любым другим (**сети множественного доступа — multi-access networks**), даже если в них невозможно широковещание. Предыдущие протоколы не так хорошо справлялись с этой проблемой.

На рис. 5.55, *а* показан пример автономной системы. Здесь не указаны хосты, так как обычно они не играют большой роли в OSPF; маршрутизаторы и сети (которые

могут содержать хосты) гораздо более важны. Большинство маршрутизаторов соединено с другими маршрутизаторами двухточечными линиями, а также с сетями, к хостам которых им необходим доступ. Но R3, R4 и R5 соединены широковещательной ЛВС, например коммутируемой сетью Ethernet.

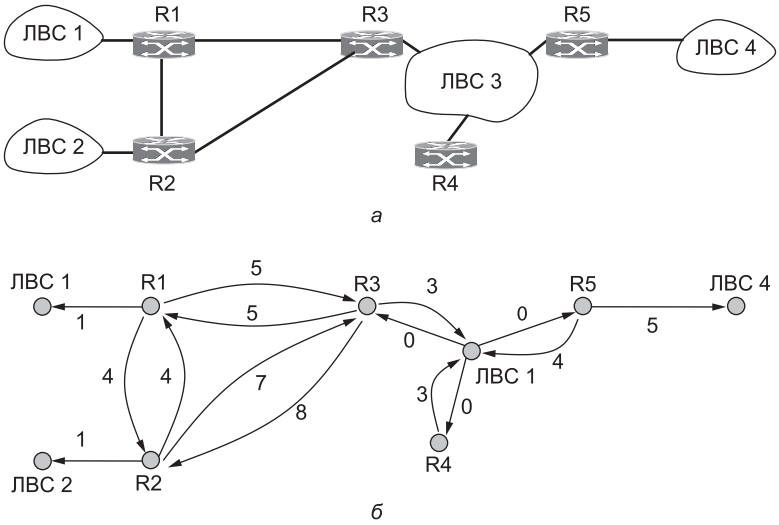


Рис. 5.55. Сеть множественного доступа: *а* — автономная система; *б* — представление (*а*) в виде графа

В основе работы протокола OSPF лежит обобщенное представление о множестве сетей, маршрутизаторов и связей в виде направленного графа, в котором каждой дуге поставлена в соответствие ее цена (может выражаться в таких физических параметрах, как расстояние, задержка и т. д.). Двухточечное соединение между двумя маршрутизаторами представляется в виде пары дуг, по одной в каждом направлении. Их весовые коэффициенты могут быть различными. Широковещательная сеть представляется в виде узла для самой сети, а также в виде узла для каждого маршрутизатора. Дуги, идущие от сетевого узла к узлам маршрутизаторов, обладают нулевым весом. Но они все равно важны, так как без них не будет существовать путь через сеть. Другие сети, состоящие исключительно из хостов, имеют только дуги, направленные к ним, и не имеют обратных дуг. То есть маршруты к хостам возможны, а через них — нет.

На рис. 5.55, *б* сеть, изображенная на рис. 5.55, *а*, представлена в виде графа. По сути, как раз это и делает OSPF. Когда представление в виде графа получено, маршрутизаторы могут вычислить кратчайшие пути до всех остальных узлов с помощью метода, учитывающего состояние линий. Возможно, что некоторые пути будут одинаково короткими. Тогда OSPF запоминает оба пути и использует эту информацию для разделения трафика. Это помогает распределить нагрузку более равномерно и называется **ECMP (Equal Cost MultiPath — использование множества равноценных маршрутов)**.

Многие автономные системы в Интернете сами по себе довольно велики и управлять ими непросто. Поэтому протокол OSPF позволяет делить их на пронумерованные **области**, то есть на сети или множества смежных сетей. Области не должны перекры-

ваться, но не обязаны быть исчерпывающими, то есть некоторые маршрутизаторы могут не принадлежать ни одной области. Если маршрутизатор полностью принадлежит какой-то области, он называется **внутренним маршрутизатором (internal router)**. Область является обобщением отдельной сети. За пределами области видны ее адреса, но не ее топология. Это упрощает масштабирование процесса маршрутизации.

У каждой автономной системы есть **магистральная область (backbone area)**, называемая областью 0. Маршрутизаторы, расположенные в этой области, называются **магистральными маршрутизаторами (backbone routers)**. Все области соединены с магистралью, например, туннелями, так что по магистрали можно попасть из любой области автономной системы в ее любую другую область. Туннель представляется на графе в виде дуги и обладает определенной ценой. Как и в случае других областей, топология магистрали за ее пределами не видна.

Маршрутизатор, соединенный одновременно с двумя и более областями, называется **пограничным маршрутизатором области (area border router)**. Он также должен быть частью магистрали. Его задача — собирать сведения об адресах одной области и передавать их другим областям. Эти сведения включают стоимость передачи, но не всю информацию о топологии области. Зная стоимость передачи, хосты других областей могут выбрать тот маршрутизатор границы области, через который они войдут в эту область. Отсутствие информации о топологии уменьшает трафик и упрощает процесс вычисления кратчайших путей для маршрутизаторов, находящихся вне данной области. Но если вне области есть только один пограничный маршрутизатор области, эти сведения передавать бессмысленно. Все пути, ведущие за пределы области, начинаются с указания: «Идите на пограничный маршрутизатор». Такая область называется **тупиковой областью (stub area)**.

Последний тип маршрутизаторов — **пограничные маршрутизаторы автономной системы (AS boundary router)**. Они передают внутрь области сведения о путях к внешним адресам на другие АС. Внешние пути становятся адресами, до которых можно добраться через пограничный маршрутизатор автономной системы; при этом указывается стоимость передачи. Внешний путь может быть передан на один или более таких маршрутизаторов. Связь между автономными системами, областями и различными типами маршрутизаторов показана на рис. 5.56. Один маршрутизатор может играть несколько ролей — например, быть и пограничным маршрутизатором области, и магистральным маршрутизатором.

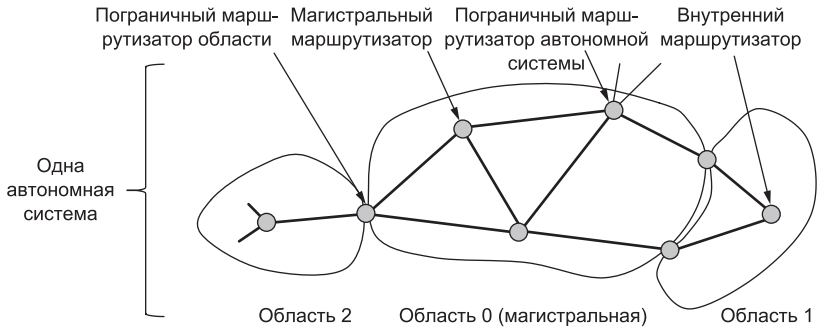


Рис. 5.56. Взаимосвязь между автономными системами, магистралями и областями в OSPF

При нормальной работе алгоритма у всех маршрутизаторов, принадлежащих к одной области, имеется одна и та же база данных состояния каналов и один алгоритм выбора кратчайшего пути. Работа маршрутизаторов заключается в расчете кратчайшего пути от себя до всех остальных маршрутизаторов этой области. Маршрутизатор, соединенный с несколькими областями, должен иметь базы данных для каждой из них. Кратчайший путь для каждой области вычисляется отдельно.

Для отправителя и получателя из одной области выбирается наилучший внутриобластной маршрут (полностью лежащий в этой области). Для отправителя и получателя из разных областей межобластной маршрут проходит от источника к магистрали, затем по магистрали к области назначения, а затем уже к адресу назначения. Такой алгоритм приводит к конфигурации типа «звезда», в которой магистраль исполняет роль концентратора, а области являются лучами звезды. Так как при выборе маршрута учитывается его стоимость, разные маршрутизаторы могут попадать на магистраль через разные пограничные маршрутизаторы области. Пакеты направляются от отправителя к получателю в натуральном виде. Они не упаковываются в другие пакеты и не туннелируются, кроме случаев, когда они направляются в области, с которыми магистраль соединена по туннелю. Кроме того, пути к внешним адресам могут включать внешнюю стоимость (от пограничного маршрутизатора по внешнему пути) или только внутреннюю (в пределах АС).

При загрузке маршрутизатор рассылает сообщения ПРИВЕТСТВИЕ (HELLO) по всем своим двухточечным линиям, производя многоадресную рассылку по локальным сетям для групп, состоящих из всех остальных маршрутизаторов. С помощью получаемых ответов каждый маршрутизатор знакомится со своими соседями. Все маршрутизаторы одной ЛВС являются соседями.

Протокол OSPF работает при помощи обмена информацией между *смежными* маршрутизаторами, что не то же самое, что *соседние* маршрутизаторы. В частности, общение каждого маршрутизатора с каждым маршрутизатором локальной сети неэффективно. Поэтому один маршрутизатор выбирается **назначенным маршрутизатором (designated router — DR)**. Он считается **смежным (adjacent)** со всеми остальными маршрутизаторами данной ЛВС и обменивается с ними информацией. Соседние маршрутизаторы, не являющиеся смежными, не обмениваются информацией друг с другом. На случай выхода из строя основного назначенного маршрутизатора всегда поддерживается в готовом состоянии запасной назначенный маршрутизатор **(backup designated router — BDR)**.

При нормальной работе каждый маршрутизатор периодически рассылает методом заливки сообщение ОБНОВЛЕНИЕ СОСТОЯНИЯ КАНАЛОВ (LINK STATE UPDATE) всем своим смежным маршрутизаторам. Это сообщение содержит сведения о состоянии маршрутизатора и предоставляет информацию о цене, используемую в базах данных. В ответ на эти сообщения посылаются подтверждения, что повышает их надежность. Каждое сообщение получает последовательный номер, так что маршрутизатор может распознать, что новее: пришедшее сообщение или сообщение, хранимое им. Маршрутизаторы также рассылают эти сообщения, когда включается или выключается канал или изменяется его цена.

Сообщение ОПИСАНИЕ БАЗЫ ДАННЫХ (DATABASE DESCRIPTION) содержит порядковые номера всех записей о состоянии линий, которыми владеет отправитель. Сравнивая соб-

ственные значения со значениями отправителя, получатель может определить, у кого информация новее. Эти сообщения посылаются при восстановлении линии.

Каждый маршрутизатор может запросить информацию о состоянии линий у своего партнера с помощью сообщения ЗАПРОС О СОСТОЯНИИ КАНАЛА (LINK STATE REQUEST). В результате каждая пара смежных маршрутизаторов выясняет, чьи сведения являются более свежими, и, таким образом, по области распространяется наиболее новая информация. Все эти сообщения посылаются в виде IP-пакетов. Пять типов сообщений приведены в табл. 5.9.

Таблица 5.9. Пять типов сообщений протокола OSPF

Тип сообщения	Описание
Приветствие	Используется для знакомства с соседями
Обновление состояния каналов	Сообщает соседям информацию о каналах отправителя
Подтверждение состояния каналов	Подтверждает обновление состояния каналов
Описание базы данных	Сообщает о том, насколько свежей информацией располагает отправитель
Запрос состояния каналов	Запрашивает информацию у партнера

Подведем итоги. С помощью механизма заливки каждый маршрутизатор информирует все остальные маршрутизаторы своей области о своих связях с другими маршрутизаторами и сетями и о стоимости этих связей. Эта информация позволяет всем маршрутизаторам построить граф своей области и рассчитать кратчайшие пути. Маршрутизаторы магистральной области также занимаются этим. Кроме того, магистральные маршрутизаторы получают информацию от пограничных маршрутизаторов областей, с помощью которой они вычисляют оптимальные маршруты от каждого магистрального маршрутизатора до всех остальных маршрутизаторов. Эта информация рассылается обратно пограничным маршрутизаторам областей, которые распространяют ее в своих областях. С помощью этой информации внутренний маршрутизатор может выбрать оптимальный путь до внешнего адреса и, в том числе, выбрать пограничный маршрутизатор области, имеющий выход к магистрали.

5.6.7. Протокол внешнего шлюза BGP

В пределах одной автономной системы наиболее часто используются протоколы OSPF и IS-IS. При выборе маршрута между различными автономными системами используется другой протокол, **BGP (Border Gateway Protocol — пограничный межсетевой протокол)**. Для выбора маршрута между различными автономными системами действительно требуется другой протокол, так как цели внутридоменного и междоменого протоколов различны. Задача внутридоменного протокола ограничивается максимально эффективной передачей пакетов от отправителя к получателю. Политикой этот протокол не интересуется.

Междоменный протокол, напротив, вынужден заниматься политикой (Metz, 2001). Например, корпоративной автономной системе может понадобиться возможность

принимать и посылать пакеты на любой сайт Интернета. Однако прохождение через автономную систему пакета, отправитель и получатель которого находятся за пределами данного государства, может быть нежелательно, даже если кратчайший путь между отправителем и получателем пролегает через эту автономную систему («Это их заботы, а не наши»). С другой стороны, может оказаться желательным транзит трафика для других автономных систем, возможно, соседних, которые специально заплатили за эту услугу. Например, телефонные компании были бы рады оказывать подобные услуги, но только своим клиентам. Протоколы внешнего шлюза вообще и протокол BGP в частности разрабатывались для возможности учета различных стратегий при выборе маршрута между автономными системами.

Типичные стратегии выбора маршрутов учитывают политические, экономические факторы, а также соображения безопасности. К типичным примерам ограничений при выборе маршрутов относятся следующие.

1. Не передавать коммерческий трафик в образовательные сети.
2. Никогда не прокладывать через Ирак маршрут, начинающийся в Пентагоне.
3. Использовать TeliaSonera вместо Verizon, так как она дешевле.
4. Не использовать AT&T в Австралии, так как производительность будет низкой.
5. Трафик, начинающийся или заканчивающийся в Apple, не должен проходить через Google.

Как вы, наверное, уже поняли, политика маршрутизации может носить в высшей степени индивидуальный характер. Часто стратегии не разглашаются по соображениям безопасности. Однако существуют схемы, объясняющие такой выбор этой компании. Часто эти схемы берутся в качестве отправной точки.

Реализация политики маршрутизации заключается в том, чтобы решить, какой трафик может перемещаться по каким каналам, соединяющим автономные системы. Один из довольно популярных вариантов выглядит так: провайдер-клиент платит другому провайдеру за отправку пакетов на любой адрес в сети Интернет и получение пакетов с любого адреса. В таком случае говорят, что провайдер-клиент покупает у другого провайдера транзитные услуги. Эта ситуация аналогична тому, как обычный пользователь домашней сети покупает у провайдера услугу доступа к Интернету. Чтобы это работало, провайдер должен объявить абоненту маршруты на все адреса в сети Интернет по каналу, который их соединяет. Тогда абонент будет знать маршрут, по которому можно отправить пакет куда угодно. И наоборот, абонент должен объявить провайдеру маршруты на все адреса в пределах своей сети. Тогда провайдер сможет передавать трафик только на эти адреса — абоненту не нужен трафик, предназначенный для кого-то другого.

Пример предоставления транзитных услуг приведен на рис. 5.57. Перед нами четыре автономные системы, соединенные между собой. Соединение часто осуществляется через **точку обмена интернет-трафиком** (**IXP**, **Internet eXchange Point**), инфраструктуру, с которой соединяются многие интернет-провайдеры, чтобы получить доступ к другим интернет-провайдерам. AS2, AS3 и AS4 — клиенты AS1. Они покупают у AS1 транзитные услуги. Таким образом, пакет, отправленный источником A на C, передается из AS2 в AS1. а затем в AS4. Объявления о маршрутах перемещаются в противоположном направлении. Чтобы источник мог передать пакет C через AS1,

AS4 объявляет C в качестве адреса назначения своему провайдеру (AS1). После этого AS1 объявляет маршрут до C другим своим клиентам, включая AS2, чтобы они могли отправлять трафик на C через AS1.

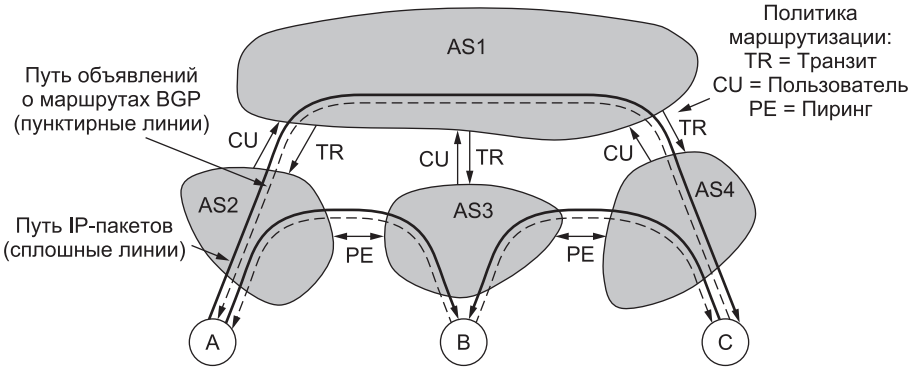


Рис. 5.57. Политика маршрутизации между четырьмя автономными системами

На рис. 5.57 все остальные АС покупают транзитные услуги у AS1. Благодаря этому они могут связаться с любым хостом в сети Интернет. Но за эту возможность им приходится платить. Пусть AS2 и AS3 обмениваются большим количеством трафика. Если эти сети соединены, то они могут выбрать другую политику — передавать трафик напрямую и бесплатно. Это уменьшит количество трафика, передаваемого AS1 от их имени, и сократит их расходы. Такая политика называется **пирингом (peering)**.

Для реализации пиринга две АС должны передавать друг другу объявления о маршрутах для своих адресов. Это позволяет AS2 отправлять пакеты AS3 из A в B, и наоборот. Однако следует обратить внимание на то, что пиринг не транзитивен. На рис. 5.57 сети AS3 и AS4 тоже используют политику пиринга, и поэтому пакеты из C в B моут передаваться напрямую в AS4. Но что если пакет нужно отправить из C в A? AS3 объявляет AS4 маршрут до B, но не объявляет маршрута до A. Поэтому трафик не сможет пройти из AS4 в AS3, а затем в AS2, хотя физический путь существует. Но именно это и выгодно AS3. Она хочет обмениваться трафиком с AS4, но не хочет, чтобы AS4 передавала через нее трафик для остального Интернета. Вместо этого AS4 придется пользоваться транзитными услугами AS1, которая, кстати, и передаст пакет из C в A.

Теперь, когда мы знаем о транзитных услугах и пиринге, посмотрим на транзитные соглашения A, B и C. К примеру, A покупает доступ к Интернету у AS2. Сеть A может состоять из одного компьютера, а может быть сетью компании с множеством ЛВС. Но в любом случае A не нужно использовать BGP, так как она является **тупиковой сетью (stub network)**, соединенной с остальным Интернетом только одной линией. Отправлять пакеты за пределы сети можно только через эту единственную линию. Такой путь можно задать в качестве пути по умолчанию. Именно поэтому A, B и C на рисунке не представлены в виде автономных систем с внутридоменной маршрутизацией.

С другой стороны, некоторые сети компаний подключены к нескольким интернет-провайдерам. Это делается с целью повышения надежности, так как при отказе одного из провайдеров трафик может быть передан через другого провайдера. Этот метод называется **многолинейным подключением (multihoming)**. В таком случае компания,

скорее всего, будет использовать протокол междоменной маршрутизации (например, BGP), чтобы АС знали, через какого провайдера следует передавать трафик.

Существует множество вариантов политик маршрутизации, и это хорошая иллюстрация того, как деловые отношения и контроль над объявлениями о маршрутизации приводят к созданию новых политик. Теперь мы перейдем к обсуждению того, как BGP-маршрутизаторы передают друг другу объявления о маршрутах и выбирают пути для передачи пакетов.

BGP по сути является вариантом протокола маршрутизации по вектору расстояний, однако он значительно отличается от других подобных протоколов, например протокола **RIP (Routing Information Protocol — протокол маршрутной информации)**. Как мы уже видели, вместо минимального расстояния при выборе маршрута учитывается политика. Еще одно отличие состоит в том, что вместо того чтобы периодически сообщать всем своим соседям свои расчеты цены передачи до каждого возможного адресата, каждый BGP-маршрутизатор передает соседям точную информацию об используемых им маршрутах. Этот подход называется **протоколом векторов маршрутов (Path Vector Protocol)**. Путь состоит из следующего маршрутизатора (совсем не обязательно смежного: он может находиться в другой части сети провайдера) и последовательности АС, или **пути АС (AS path)**, через которые проходит маршрут (в обратном порядке). Пары BGP-маршрутизаторов общаются друг с другом, устанавливая TCP-соединения. Таким образом обеспечивается надежная связь и скрываются детали устройства сети, по которой проходит трафик.

На рис. 5.58 показан пример того, как маршрутизаторы обмениваются объявлениями о маршрутах. Здесь мы видим три автономных сети, причем средняя предоставляет транзитные услуги правому и левому интернет-провайдеру. Объявление о маршруте к префиксу C начинается на AS3. Когда оно доходит до R2c (вверху), оно состоит из AS3 и следующего маршрутизатора R3a. Внизу этот маршрут имеет тот же путь АС, но другой следующий маршрутизатор, так как он пришел по другому каналу. Это объявление распространяется дальше и пересекает границу AS1. На маршрутизаторе R1a (вверху) путь АС состоит из AS2 и AS3, а следующий маршрутизатор — R2a.

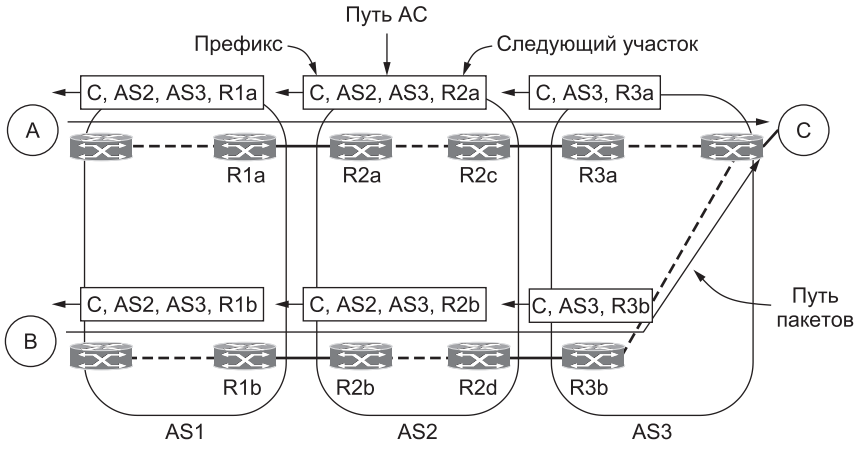


Рис. 5.58. Распространение объявлений о BGP-маршруте

Хранение полного пути для маршрута упрощает обнаружение и устранение циклов. Правило выглядит так: каждый маршрутизатор, отправляющий маршрут за пределы своей АС, добавляет в начало маршрута номер своей АС. (Именно поэтому список имеет обратный порядок.) Когда маршрутизатор получает маршрут, он проверяет, есть ли в пути АС номер его собственной АС. Положительный ответ означает, что в маршруте есть цикл; в таком случае объявление отвергается. Но несмотря на такие меры предосторожности, в конце 1990-х было обнаружено, что BGP все же сталкивается с одним из вариантов проблемы счета до бесконечности (Labovitz и др., 2001). Хотя от долговременных циклов удается избавиться, из-за медленной конвергенции маршрутизаторов могут возникать временные циклы.

Задавать путь списком АС — довольно-таки грубый вариант. АС может быть как небольшой компанией, так и международной магистральной сетью. По маршруту этого узнать невозможно. А BGP даже не пытается этого сделать, так как в разных АС могут использоваться разные внутридоменые протоколы, вследствие чего стоимость передачи трафика невозможно сравнить. Но если бы их можно было сравнить, с большой вероятностью возникла бы другая проблема: часто АС скрывают данные о своих внутренних параметрах. Это одно из основных отличий междоменной маршрутизации от внутридоменной.

До сих пор мы говорили о том, как объявление маршрута передается по каналу между двумя интернет-провайдерами. Но нужно также уметь отправлять BGP-маршруты в другие части сети провайдера, чтобы оттуда они могли быть переданы другим провайдерам. Эту задачу должен решать внутридоменный протокол. Но поскольку BGP очень хорошо работает в крупных сетях, для этого часто используется его вариант. Он называется **внутренним BGP (iBGP — internal BGP)**, в противоположность обычному, или **внешнему BGP (eBGP — external BGP)**.

Правило распространения маршрутов внутри сети провайдера выглядит так: маршрутизатор, расположенный на границе сети, узнает обо всех маршрутах, о которых знают другие пограничные маршрутизаторы (в целях согласованности их действий). Если какому-то пограничному маршрутизатору становится известен префикс к IP 128.208.0.0/16, все другие маршрутизаторы тоже о нем узнают. Тогда до этого префикса можно будет добраться из любой точки сети провайдера, причем не важно, как пакет попал в эту сеть.

Во избежание беспорядка этот процесс не показан на рис. 5.58. Но в качестве примера можно привести такой вариант: маршрутизатор R2b узнает, что он может добраться до C либо через R2c (вверху), либо через R2d (внизу). Данные о следующем маршрутизаторе будут обновляться, пока маршрут будет перемещаться по сети провайдера. Благодаря этому маршрутизаторы на разных концах сети будут знать, через какой маршрутизатор можно выйти за пределы сети на другом ее конце. Если посмотреть на самые левые маршруты, то можно увидеть, что для них следующий маршрутизатор находится в той же сети, а не в соседней.

Теперь мы можем перейти к обсуждению важного вопроса: как BGP-маршрутизаторы выбирают маршрут до данного адреса? Каждый BGP-маршрутизатор узнает маршрут до данного адреса от соединенного с ним маршрутизатора в соседней сети провайдера, а также от других пограничных маршрутизаторов (которые узнают другие

возможные маршруты от своего соседнего маршрутизатора). Далее каждый маршрутизатор решает, какой из этих маршрутов лучше всего использовать. В конечном итоге получается, что политику выбора маршрута определяет интернет-провайдер. Однако такое объяснение является слишком общим и не вполне удовлетворительным, так что давайте поговорим о нескольких возможных стратегиях.

Первая стратегия заключается в том, что маршруты через сети, соединенные напрямую, обладают большим приоритетом, чем транзитные маршруты. Первые являются бесплатными, вторые — нет. Существует еще один похожий вариант: наибольший приоритет присваивается маршрутам клиента. При этом трафик передается напрямую тому, кто за него платит.

Другая стратегия по умолчанию использует такое правило: чем короче путь АС, тем он лучше. Преимущества этой стратегии являются спорными, так как путь через три маленьких АС может оказаться короче, чем путь через одну большую АС. Однако в среднем выбор кратчайших путей дает неплохие результаты, так что решение остается за вами.

Наконец, существует стратегия, согласно которой предпочтение отдается маршруту с наименьшей стоимостью в пределах сети провайдера. Пример ее реализации показан на рис. 5.58. Пакеты из A в C покидают AS1 через верхний маршрутизатор R1a. Пакет из B выходит через нижний маршрутизатор R1b. Так происходит потому, что A и B выбирают наименее затратный способ выхода из AS1. А поскольку они расположены в разных частях сети провайдера, наилучшие пути для них различаются. То же самое происходит, когда пакеты проходят через AS2. На последнем участке AS3 должна переправить пакет из B по своей сети.

Эта стратегия называется ранним выходом или **методом «горячей картошки» (hot-potato routing)**. Интересно, что при такой маршрутизации пути обычно бывают несимметричными. Рассмотрим маршрут пакета, переданного из C в B. Пакет сразу же выйдет из AS3 через верхний маршрутизатор. При переходе из AS2 в AS1 он останется наверху, а уже потом будет долго путешествовать внутри AS1. Это зеркальное отражение пути от B к C.

Из всего этого следует, что каждый BGP-маршрутизатор выбирает свой наилучший путь из нескольких возможных. Но вопреки первоначальным ожиданиям, нельзя сказать, что BGP отвечает за маршруты между разными АС, а OSPF — за маршруты внутри одной АС. BGP и протокол внутреннего шлюза связаны более сложным образом. В частности, это значит, что BGP может найти наилучшую точку выхода за пределы сети провайдера и выбор точки будет различаться для разных уголков сети, как в случае стратегии «горячей картошки». Также это значит, что BGP-маршрутизаторы из разных частей одной АС могут использовать разные пути АС до одного и того же места назначения. При этом провайдер должен позаботиться о том, чтобы при такой свободе выбора маршруты были совместимы, но это уже относится к практике.

Как это ни удивительно, мы лишь слегка коснулись вопроса использования BGP. Для более подробной информации см. спецификацию BGP версии 4 в RFC 4271 и другие RFC. Но помните, что большинство трудностей касаются политики маршрутизации, о которой не говорится в спецификации протокола BGP.

5.6.8. Многоадресная рассылка в Интернете

Обычно IP-связь устанавливается между одним отправителем и одним получателем. Однако для некоторых приложений возможность послать сообщение одновременно большому количеству получателей является полезной. Такими приложениями могут быть, например, обновление реплицируемой распределенной базы данных, передача биржевых сводок брокерам и цифровые телеконференции (с участием нескольких собеседников).

Протокол IP поддерживает многоадресную рассылку при использовании адресов класса D. Каждый адрес класса D соответствует группе хостов. Для обозначения номера группы может быть использовано 28 бит, что делает возможным одновременное существование 250 млн групп. Когда процесс посылает пакет по адресу класса D, протокол прилагает максимальные усилия по его доставке всем членам группы, однако не дает гарантий доставки. Некоторые члены группы могут не получить пакета.

Диапазон IP-адресов 224.0.0.0/24 зарезервирован для многоадресной рассылки в локальной сети. В таком случае протокол маршрутизации не требуется. Пакеты передаются всей ЛВС с помощью широковещания с указанием группового адреса. Все хосты в этой сети получают пакет, но обрабатывают его только члены группы. За пределы ЛВС пакет не передается. Вот некоторые примеры групповых адресов:

- 224.0.0.1 — все системы локальной сети;
- 224.0.0.2 — все маршрутизаторы локальной сети;
- 224.0.0.5 — все OSPF-маршрутизаторы локальной сети;
- 224.0.0.251 — все DNS-серверы локальной сети.

В других случаях члены группы могут располагаться в разных сетях. Тогда протокол маршрутизации необходим. Но для начала маршрутизаторы, передающие многоадресные сообщения, должны знать, какие хосты являются членами группы. Процесс может попросить свой хост присоединиться к какой-либо группе или покинуть группу. Каждый хост следит за тем, членами каких групп являются его процессы в текущий момент. Когда последний процесс хоста покидает группу, хост больше не является членом этой группы. Примерно раз в минуту каждый многоадресный маршрутизатор совершает аппаратную (то есть на уровне передачи данных) многоадресную рассылку хостам на своей локальной сети (по адресу 224.0.0.1) с просьбой сообщить о группах, к которым в данный момент принадлежат их процессы. Многоадресные маршрутизаторы не обязательно должны находиться там же, где обычные маршрутизаторы. Каждый хост посылает обратно ответы для всех интересующих его адресов класса D. Эти пакеты запросов и ответов используются протоколом **IGMP (Internet Group Management Protocol** — протокол управления группами в Интернете). Он описан в RFC 3376.

Для построения связующего дерева, позволяющего получить пути от отправителей до всех членов группы, используются различные протоколы многоадресной маршрутизации. О соответствующих алгоритмах мы уже говорили в разделе 5.2.8. Внутри автономной системы чаще всего используется **протокольно-независимая многоадресная рассылка (PIM**, **Protocol Independent Multicast)**. Существует несколько вариантов PIM. При PIM в плотном режиме (Dense Mode PIM) создается усеченное дерево, построенное методом продвижения по встречному пути. Такой вариант под-

ходит в ситуациях, когда члены группы находятся во всех частях сети — например, при распространении файлов на множественные серверы. При PIM в разреженном режиме (Sparse Mode PIM) создаются связующие деревья, похожие на деревья с основанием в ядре. Такой вариант может использоваться, к примеру, когда поставщик контента транслирует ТВ-сигнал абонентам своей IP-сети. Вариант этой схемы под названием Source-Specific Multicast PIM работает лучше в ситуациях с одним источником. Наконец, если члены группы находятся в нескольких АС, для создания многоадресных маршрутов необходимо использовать специальные расширения BGP или туннелирование.

5.6.9. Мобильный IP

Многие пользователи Интернета обладают портативными компьютерами и заинтересованы в возможности оставаться в подключенном к сети состоянии, даже находясь в пути. К сожалению, система адресации IP такова, что реализовать это оказывается гораздо сложнее, чем кажется (вскоре мы поговорим об этом подробнее). Когда потребность в мобильных хостах значительно возросла, проблемная группа проектирования Интернета (IETF, Internet Engineering Task Force) создала рабочую группу для поиска решения проблемы. Созданная рабочая группа быстро сформулировала набор целей, которых хотелось бы достичь, независимо от способа решения. Основными целями были признаны следующие.

1. Каждый мобильный хост должен иметь возможность использовать свой домашний IP-адрес где угодно.
2. Изменения программного обеспечения фиксированных хостов недопустимы.
3. Изменения программного обеспечения и таблиц маршрутизаторов недопустимы.
4. Большая часть пакетов, направляемых мобильным хостам, должны доставляться напрямую.
5. Не должно быть никаких дополнительных расходов, когда мобильный хост находится дома.

Рабочая группа выработала решение, описанное в разделе 5.2.10. Суть его, напомним, заключалась в том, что везде, где требуется предоставить возможность перемещения в пространстве, следует создать **внутреннего агента (home agent)**. Когда мобильный хост прибывает на новое место, он получает новый IP-адрес (также называемый адресом для передачи). После этого мобильный хост сообщает внутреннему агенту о своем местоположении с помощью адреса для передачи. Если в момент прибытия пакета, предназначенного этому хосту, в домашнюю сеть хост находится в другом месте, внешний агент туннелирует этот пакет на хост, используя адрес для передачи. Мобильный хост может посылать ответные пакеты, но при этом адресом отправителя будет домашний адрес. Это решение удовлетворяет всем перечисленным выше требованиям за исключением рециркулирующих пакетов мобильных хостов.

Чтобы получить адрес для передачи в новой сети, мобильный хост может использовать DHCP. Или же, если IP-адресов не хватает, этот хост может отправлять и получать пакеты через внешнего агента, у которого уже есть IP-адрес в этой сети. Чтобы найти внешнего агента, хост использует тот же ICMP-механизм, что и при поиске

внутреннего агента. После того как хост получил IP-адрес или нашел внешнего агента, он может по сети сообщить внутреннему агенту о своем текущем местоположении.

Внутренний агент должен перехватывать пакеты, предназначенные для мобильного хоста, только при отсутствии этого хоста в домашней сети. Сделать это можно с помощью ARP. Чтобы отправить пакет на IP-хост по сети Ethernet, маршрутизатор должен знать Ethernet-адрес хоста. Обычно маршрутизатор отправляет ARP-сообщение, в котором спрашивает что-то вроде «какой Ethernet-адрес у 160.80.40.20?» Если мобильный хост находится в домашней сети, он отвечает на запрос, содержащий его IP-адрес, сообщением со своим Ethernet-адресом. Если он находится в другом месте, на этот запрос отвечает внутренний агент, сообщая свой Ethernet-адрес. В результате маршрутизатор отправляет пакеты для 160.80.40.20 домашнему агенту. Как вы, возможно, помните, это называется ARP-прокси.

Чтобы быстро обновлять таблицы отображения адресов при перемещении хоста туда и обратно, используется другой метод, называемый **добровольным ARP-сообщением (gratuitous ARP)**. Суть в том, что мобильный хост или внутренний агент отправляет самому себе ARP-запрос для мобильного IP-адреса, который заставляет маршрутизатор заменить в своей таблице запись о хосте.

При туннелировании пакета от внутреннего агента к мобильному хосту к пакету добавляется IP-заголовок с адресом для передачи. Когда пакет пребывает на этот адрес, внешний IP-заголовок удаляется.

Как и в случае со многими интернет-протоколами, дьявол кроется в деталях, и особенно в тех мелочах, которые касаются совместимости с другими протоколами. Здесь есть две сложности. Во-первых, NAT-блоку необходим доступ к TCP- и UDP-заголовкам, расположенным после IP-заголовка. Изначальный вариант туннелирования для мобильных IP не использовал эти заголовки и поэтому не работал с NAT-блоками. Позже было принято решение изменить способ добавления заголовков, включив UDP-заголовок.

Вторая сложность состоит в том, что некоторые провайдеры проверяют, соответствует ли IP-адрес источника тому расположению, где, по мнению протокола маршрутизации, источник должен находиться. Этот процесс называется **фильтрацией на входе (ingress filtering)** и является мерой безопасности, позволяющей отвергать трафик, поступивший с сомнительного адреса, так как он может представлять угрозу. Однако если мобильный хост находится не в домашней сети, его пакеты будут приходить с IP-адреса другой сети и поэтому, скорее всего, будут отвергнуты. Чтобы обойти эту проблему, мобильный хост может туннелировать пакеты внутреннему агенту с помощью адреса для передачи. Пакеты, отправленные отсюда, не будут казаться подозрительными. Минус такой схемы в том, что такой путь является далеко не прямым.

Еще одним вопросом является безопасность. Когда внутренний агент получает просьбу пересылать все пакеты, приходящие на имя Натальи, на некий IP-адрес, он не должен подчиняться, пока не убедится, что источником этого запроса является Наталья, а не кто-то пытающийся выдать себя за Наталью. Для этого применяются протоколы криптографической аутентификации, которые будут рассматриваться в главе 8.

Протоколы мобильности для IPv6 основаны на IPv4. Приведенная выше схема сталкивается с проблемой треугольной маршрутизации, при которой пакеты вынуждены делать крюк, проходя через внутреннего агента. IPv6 использует оптимизацию

маршрута, позволяющую установить прямые пути между мобильным хостом и другими IP-адресами после того, как первые пакеты прошли длинный путь. Мобильный IPv6 описан в RFC 3775.

В сети Интернет появляется еще один вид мобильности. В некоторых самолетах есть встроенная беспроводная сеть, с помощью которой пассажиры с переносными компьютерами могут подключаться к Интернету. В самолете есть маршрутизатор, подключенный к остальному Интернету по беспроводной связи. (А вы думали, через кабель?) Так что по сути это летающий маршрутизатор, то есть мобильной является вся сеть. Схемы, реализующие мобильность сетей, работают так, что переносные компьютеры не замечают, что самолет движется. Они думают, что это просто другая сеть. Некоторые компьютеры используют мобильный IP, чтобы сохранять связь с домашней сетью, и в таком случае мы имеем дело с двумя уровнями мобильности. Мобильность сети для IPv6 описана в RFC 3963.

5.7. Резюме

Сетевой уровень предоставляет услуги транспортному уровню. В его основе могут лежать либо виртуальные каналы, либо дейтаграммы. В обоих случаях его основная работа состоит в выборе маршрута пакетов от источника до адресата. В сетях с виртуальными каналами решение о выборе маршрута осуществляется при установке виртуального канала. В дейтаграммных сетях оно принимается для каждого пакета.

В компьютерных сетях применяется большое количество алгоритмов маршрутизации. Заливка — это простой алгоритм передачи пакетов по всем путям. Большинство алгоритмов определяют кратчайший путь и адаптируются к изменениям топологии сети. Основными алгоритмами являются маршрутизация по векторам расстояний и маршрутизация с учетом состояния линий. В большинстве имеющихся сетей применяется один из этих алгоритмов. К другим важным методам маршрутизации относятся использование иерархии в больших сетях, маршрутизация для мобильных хостов, широковещание, многоадресная маршрутизация и произвольная маршрутизация.

Сети подвержены перегрузкам, приводящим к увеличению задержки и утере пакетов. Разработчики сетей пытаются предотвратить перегрузку разнообразными способами, включающими создание сетей с достаточной пропускной способностью, выбор не перегруженных маршрутов, временный запрет входящего трафика, отправку сообщения источнику с просьбой замедлить передачу трафика и сброс нагрузки.

Следующим шагом после разрешения проблем с перегрузкой является попытка достижения гарантированного качества обслуживания. Некоторые приложения заботятся в первую очередь о пропускной способности, а не о задержке и флуктуациях. Комплексные методы, используемые для этого, включают формирование трафика, резервирование ресурсов на маршрутизаторах, контроль доступа. Подходы, разработанные для обеспечения хорошего качества обслуживания, включают в себя интегрированное обслуживание IETF (включая RSVP) и дифференцированное обслуживание.

Разные сети могут отличаться друг от друга весьма значительно, поэтому при попытке их объединения могут возникать определенные сложности. Если в сетях различаются ограничения на максимальный размер пакета, может быть применена фрагмен-

тация. Разные сети могут использовать разные внутренние протоколы маршрутизации, но внешний протокол маршрутизации должен быть общим. Иногда проблемы могут быть решены при помощи туннелирования пакета сквозь сильно отличающуюся сеть, но если отправитель и получатель находятся в сетях разных типов, этот подход не может быть применен.

В Интернете существует большое разнообразие протоколов, относящихся к сетевому уровню. К ним относятся протокол дейтаграмм — IP и соответствующие управляющие протоколы — ICMP, ARP и DHCP. В некоторых сетях IP-пакеты передает ориентированный на соединение протокол MPLS. Один из основных протоколов маршрутизации, использующихся внутри сетей, — это OSPF. Между сетями для маршрутизации используется протокол BGP. Интернету перестает хватать IP-адресов, поэтому была разработана новая версия протокола IP, IPv6, которая в настоящее время внедряется, но происходит это чрезвычайно медленно.

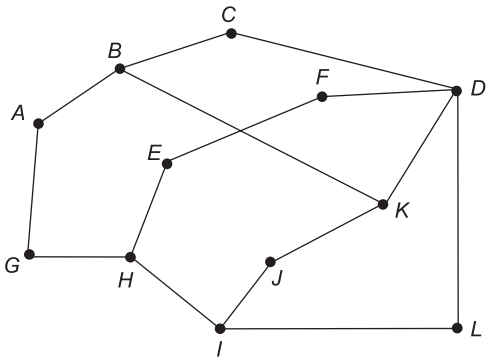
Вопросы

1. Приведите два примера приложений, для которых сервис, ориентированный на соединение, является приемлемым. Затем приведите два примера приложений, для которых наилучшим будет сервис без использования соединений.
2. Дейтаграммные сети выбирают маршрут для каждого отдельного пакета независимо от других. В подсетях виртуальных каналов каждый пакет следует по заранее определенному маршруту. Означает ли это, что подсетям виртуальных каналов не требуется способность выбирать маршрут для отдельного пакета от произвольного источника произвольному адресату. Поясните свой ответ.
3. Приведите три примера параметров протокола, о значениях которых можно договориться при установке соединения.
4. Предполагая, что все маршрутизаторы и хосты работают нормально и что их программное обеспечение не содержит ошибок, есть ли вероятность, хотя бы небольшая, что пакет будет доставлен неверному адресату?
5. Укажите простой эвристический метод нахождения двух путей от данного источника к данному адресату, гарантирующий сохранение связи при обрыве любой линии (если такие два пути существуют). Маршрутизаторы считаются достаточно надежными, поэтому рассматривать возможность выхода их из строя не нужно.
6. Рассмотрите подсеть на рис. 5.10, *а*. Используется алгоритм дистанционно-векторной маршрутизации. На маршрутизатор *C* только что поступили следующие векторы: от *B*: (5, 0, 8, 12, 6, 2); от *D*: (16, 12, 6, 0, 9, 10); от *E*: (7, 6, 3, 9, 0, 4). Измеренные задержки до *B*, *D* и *E* составляют 6, 3 и 5 соответственно. Какой будет новая таблица маршрутизатора *C*? Укажите используемые выходные линии и ожидаемое время задержки.
7. В сети, состоящей из 50 маршрутизаторов, значения стоимости записываются как 8-битовые номера, а маршрутизаторы обмениваются векторами дважды в секунду. Какая пропускная способность на каждой (дуплексной) линии съедается работой распределенного алгоритма маршрутизации? Предполагается, что каждый маршрутизатор соединен тремя линиями с другими маршрутизаторами.
8. На рис. 5.11 логическое ИЛИ двух наборов ACF-битов равно 111 для каждого ряда. Является ли это просто случайностью или же это сохраняется во всех подсетях при любых условиях?

9. Какие размеры регионов и кластеров следует выбрать для минимизации таблиц маршрутизации при трехуровневой иерархической маршрутизации, если количество маршрутизаторов равно 4800? Рекомендуется начать с гипотезы о том, что решение в виде k кластеров по k регионов из k маршрутизаторов близко к оптимальному. Это означает, что число k примерно равно корню кубическому из 4800 (около 16). Методом проб и ошибок подберите все три параметра так, чтобы они были близки к 16.
10. В тексте утверждалось, что когда мобильного хоста нет в его домашней сети, пакеты, посланные на адрес его домашней ЛВС, перехватываются внутренним агентом этой ЛВС. Как этот перехват осуществляет внутренний агент в IP-сети на основе локальной сети 802.3?
11. Сколько широковещательных пакетов формируется маршрутизатором *B* на рис. 5.5 с помощью:

    1) пересылки в обратном направлении?

    2) входного дерева?
12. Рассмотрите рис. 5.13, *а*. Допустим, добавляется одна новая линия между *F* и *G*, но входное дерево, показанное на рис. 5.13, *б*, остается без изменений. Какие изменения нужно внести в рис. 5.13, *в*?
13. Рассчитайте многоадресное связующее дерево для маршрутизатора *C* в подсети, показанной ниже, для группы, состоящей из маршрутизаторов *A*, *B*, *C*, *D*, *E*, *F*, *I* и *K*.



14. Допустим, узел *B* на рис. 5.18 только что перезагрузился и не имеет никакой информации о маршрутизации в своих таблицах. Внезапно у него появляется необходимость в маршруте к узлу *H*. Он рассылает широковещательным способом наборы *TTL* на 1, 2, 3 и т. д. Сколько раундов потребуется на поиск пути?
15. В качестве возможного механизма борьбы с перегрузкой в подсети, использующей виртуальные каналы, маршрутизатор может воздержаться от подтверждения полученного пакета, пока, во-первых, он знает, что его последняя передача по виртуальному каналу была получена успешно, и, во-вторых, у него есть свободный буфер. Для простоты предположим, что маршрутизаторы используют протокол с ожиданием и что у каждого виртуального канала есть один буфер, выделенный ему для каждого направления трафика. Передача пакета (данных или подтверждения) занимает *T* с. Путь пакета проходит через *n* маршрутизаторов. С какой скоростью пакеты доставляются адресату? Предполагается, что ошибки очень редки, а связь между хостом и маршрутизатором почти не отнимает времени.
16. Дейтаграммная подсеть позволяет маршрутизаторам при необходимости выбрасывать пакеты. Вероятность того, что маршрутизатор отвергнет пакет, равна *p*. Рассмотрите маршрут,

проходящий от хоста к хосту через два маршрутизатора. Если любой из маршрутизаторов отвергнет пакет, у хоста-отправителя, в конце концов, истечет интервал ожидания, и он попытается переслать пакет еще раз. Если обе линии (хост-маршрутизатор и маршрутизатор-маршрутизатор) считать за транзитные участки, то чему равно среднее число

1) транзитных участков, преодолеваемых пакетом за одну передачу?

2) передач для одного пакета?

3) транзитных участков, необходимых для получения пакета?

17. В чем состоит основная разница между методом ECN и методом RED?
18. Сеть использует для формирования трафика схему маркерного ведра. Новый маркер помещается в ведро каждые 5 мкс. Каждый маркер подходит для короткого пакета размером 48 байт. Чему равна максимальная скорость передачи данных в сети (не считая битов заголовка)?
19. Компьютер, подключенный к сети, скорость передачи в которой равна 6 Мбит/с, регулируется маркерным ведром. Маркерное ведро наполняется со скоростью 1 Мбит/с. Его начальная емкость составляет 8 Мбит. Как долго сможет передавать компьютер на полной скорости в 6 Мбит/с?
20. Сеть на рис. 5.30 использует RSVP в деревьях групповой рассылки для хостов 1 и 2. Допустим, хост 3 запрашивает канал с пропускной способностью 2 Мбайт/с для потока от хоста 1 и еще один канал с пропускной способностью 1 Мбайт/с для потока от хоста 2. Одновременно хост 4 запрашивает канал с пропускной способностью 2 Мбайт/с для потока от хоста 1, а хост 5 запрашивает канал с пропускной способностью 1 Мбайт/с для потока от хоста 2. Какую суммарную пропускную способность необходимо зарезервировать для удовлетворения перечисленных запросов на маршрутизаторах *A*, *B*, *C*, *E*, *H*, *J*, *K* и *L*?
21. Маршрутизатор может обрабатывать 2 млн пакетов в секунду. Нагрузка составляет в среднем 1,5 млн пакетов в секунду. Сколько времени уйдет на формирование очередей и обслуживание пакетов процессорами, если путь от источника до приемника содержит 10 маршрутизаторов?
22. Допустим, пользователь получает дифференцированный сервис со срочной пересылкой. Есть ли гарантия того, что срочные пакеты будут испытывать меньшую задержку, чем обычные? Ответ поясните.
23. Допустим, хост *A* соединен с маршрутизатором *R1*. Тот, в свою очередь, соединен с другим маршрутизатором, *R2*, а *R2* — с хостом *B*. Сообщение TCP, содержащее 900 байт данных и 20 байт TCP-заголовка, передается IP-программе, установленной на хосте *A*, для доставки его хосту *B*. Каковы будут значения полей *Общая длина*, *Идентификатор*, *DF*, *MF* и *Сдвиг фрагмента* IP-заголовка каждого пакета, передающегося по трем линиям. Предполагается, что на линии *A*–*R1* максимальный размер кадра равен 1024 байта, включая 14-байтный заголовок кадра, на линии *R1*–*R2* максимальный размер кадра составляет 512 байт, включая 8-байтный заголовок кадра, и на линии *R2*-*B* максимальный размер кадра составляет 512 байт, включая 12-байтный заголовок кадра.
24. Маршрутизатор обрабатывает IP-пакеты, общая длина которых (включая данные и заголовок) равна 1024 байт. Предполагая, что пакеты живут в течение 10 с, сосчитайте максимальную скорость линии, с которой может работать маршрутизатор без опасности зацикливания в пространстве идентификационных номеров IP-дейтаграммы.
25. IP-дейтаграмма, использующая параметр *Строгая маршрутизация от источника*, должна быть фрагментирована. Копируется ли этот параметр в каждый фрагмент, или достаточно поместить его в первый фрагмент? Поясните свой ответ.

26. Допустим, вместо 16 бит в адресе класса B для обозначения номера сети отводилось бы 20 бит. Сколько было бы тогда сетей класса B?
27. Преобразуйте IP-адрес, шестнадцатеричное представление которого равно C22F 1582, в десятичный формат, разделенный точками.
28. Маска подсети сети Интернета равна 255.255.240.0. Чему равно максимальное число хостов в ней?
29. IP-адреса подходят для обозначения сетей, а Ethernet-адреса — нет. Как вы думаете, почему это так?
30. Существует множество адресов, начинающихся с IP-адреса 198.16.0.0. Допустим, организации *A*, *B*, *C* и *D* запрашивают соответственно 4000, 2000, 4000 и 8000 адресов. Для каждой из них укажите первый и последний выданные адреса, а также маску вида *w.x.y.z/s*.
31. Маршрутизатор только что получил информацию о следующих IP-адресах: 57.6.96.0/21, 57.6.112.0/21 и 57.6.120.0/21. Если для них используется одна и та же исходящая линия, можно ли их агрегировать? Если да, то во что? Если нет, то почему?
32. Набор IP-адресов с 29.18.0.0 по 19.18.128.255 агрегирован в 29.18.0.0/17. Тем не менее остался пробел из 1024 не присвоенных адресов с 29.18.60.0 по 29.18.63.255, которые внезапно оказались присвоены хосту, использующему другую исходящую линию. Необходимо ли теперь разделить агрегированный адрес на составляющие, добавить в таблицу новый блок, а потом посмотреть, можно ли что-нибудь агрегировать? Если нет, тогда что можно сделать?
33. Маршрутизатор содержит следующие записи (CIDR) в своей таблице маршрутизации:

Адрес/маска	Следующий переход
135.46.56.0/22	Интерфейс 0
135.46.60.0/22	Интерфейс 1
192.53.40.0/23	Маршрутизатор 1
По умолчанию	Маршрутизатор 2

Куда направит маршрутизатор пакеты со следующими IP-адресами?

1) 135.46.63.10
2) 135.46.57.14
3) 135.46.52.2
4) 192.53.40.7
5) 192.53.56.7

34. Многие компании придерживаются стратегии установки двух и более маршрутизаторов, соединяющих компанию с провайдером, что гарантирует некоторый запас прочности на случай, если один из маршрутизаторов выйдет из строя. Применима ли такая политика при использовании NAT? Ответ поясните.
35. Вы рассказали товарищу про протокол ARP. Когда вы закончили объяснения, он сказал: «Ясно. ARP предоставляет услуги сетевому уровню, таким образом, он является частью канального уровня.» Что вы ему ответите?
36. Опишите способ сборки пакета из фрагментов в пункте назначения.
37. В большинстве алгоритмов сборки IP-дейтаграмм из фрагментов используется таймер, чтобы из-за потерянного фрагмента буфер, в котором производится повторная сборка, не оказался занят остальными фрагментами дейтаграммы. Предположим, дейтаграмма

разбивается на четыре фрагмента. Первые три фрагмента прибывают к получателю, а четвертый задерживается в пути. У получателя истекает период ожидания, и три фрагмента, хранившиеся в его памяти, удаляются. Немного позднее наконец приползает последний фрагмент. Как следует с ним поступить?

38. В IP контрольная сумма покрывает только заголовок, но не данные. Почему, как вы полагаете, была выбрана подобная схема?
39. Особа, живущая в Бостоне, едет в Миннеаполис и берет с собой свой персональный компьютер. К ее удивлению, локальная сеть в Миннеаполисе является беспроводной локальной сетью IP, поэтому ей нет необходимости подключать свой компьютер. Нужно ли, тем не менее, проходить процедуру с внутренним и внешним агентом, чтобы электронная почта и другой трафик прибывали правильно?
40. Протокол IPv6 использует 16-байтовые адреса. На какое время хватит этих адресов, если каждую пикосекунду назначать блок в 1 млн адресов?
41. Поле *Протокол*, используемое в заголовке IPv4, отсутствует в фиксированном заголовке IPv6. Почему?
42. Должен ли протокол ARP быть изменен при переходе на шестую версию протокола IP? Если да, то являются ли эти изменения концептуальными или техническими?
43. Напишите программу, моделирующую маршрутизацию методом заливки. Каждый пакет должен содержать счетчик, уменьшаемый на каждом маршрутизаторе. Когда счетчик уменьшается до нуля, пакет удаляется. Время дискретно, и каждая линия обрабатывает за один интервал времени один пакет. Создайте три версии этой программы: с заливкой по всем линиям; с заливкой по всем линиям, кроме входной линии; с заливкой только k лучших линий (выбираемых статически). Сравните заливку с детерминированной маршрутизацией ($k = 1$) с точки зрения задержки и использования пропускной способности.
44. Напишите программу, моделирующую компьютерную сеть с дискретным временем. Первый пакет в очереди каждого маршрутизатора преодолевает по одному транзитному участку за интервал времени. Число буферов каждого маршрутизатора ограничено. Прибывший пакет, для которого нет свободного места, игнорируется и повторно не передается. Вместо этого используется сквозной протокол с тайм-аутами и пакетами подтверждения, который, в конце концов, вызывает повторную передачу пакета маршрутизатором-источником. Постройте график производительности сети как функции интервала сквозного времени ожидания при разных значениях частоты ошибок.
45. Напишите функцию, осуществляющую пересылку в IP-маршрутизаторе. У процедуры должен быть один параметр — IP-адрес. Имеется доступ к глобальной таблице, представляющей собой массив из троек значений. Каждая тройка содержит следующие целочисленные значения: IP-адрес, маску подсети и исходящую линию. Функция ищет IP-адрес в таблице, используя CIDR, и возвращает номер исходящей линии.
46. Используя программы *traceroute* (UNIX) или *tracert* (Windows), исследуйте маршрут от вашего компьютера до различных университетов мира. Составьте список трансокеанских линий. Вот некоторые адреса:

    www.berkeley.edu (Калифорния)
    www.mit.edu (Массачусетс)
    www.vu.nl (Амстердам)
    www.ucl.ac.uk (Лондон)
    www.usyd.edu.au (Сидней)
    www.u-tokyo.ac.jp (Токио)
    www.uct.ac.za (Кейптаун)

Глава 6
Транспортный уровень

Вместе с сетевым уровнем транспортный уровень составляет сердцевину всей иерархии протоколов. Сетевой уровень обеспечивает сквозную доставку пакетов при помощи дейтаграмм и виртуальных каналов. Основанный на сетевом уровне транспортный уровень отвечает за передачу данных от процесса на машине-источнике до процесса на машине-адресате, предоставляя необходимый уровень надежности вне зависимости от физических характеристик использующихся сетей. Он создает абстракции, с помощью которых приложения могут работать в сети. Без транспортного уровня вся концепция многоуровневых протоколов потеряет смысл. В данной главе мы подробно рассмотрим транспортный уровень, включая его сервисы и выбор подходящей схемы API, позволяющей решить проблемы надежности, подключения и перегрузок, а также протоколы (такие как TCP и UDP) и производительность.

6.1. Транспортный сервис

В следующих разделах мы познакомимся с транспортным сервисом. Мы рассмотрим виды сервисов, предоставляемых прикладному уровню. Чтобы наш разговор не был слишком абстрактным, мы разберем два набора базовых операций транспортного уровня. Сначала рассмотрим простой (но не применяемый на практике) набор, просто чтобы показать основные идеи, а затем — реально применяемый в Интернете интерфейс.

6.1.1. Услуги, предоставляемые верхним уровням

Конечная цель транспортного уровня заключается в предоставлении эффективных, надежных и экономичных услуг (сервисов) передачи данных своим пользователям, которыми обычно являются процессы прикладного уровня. Для достижения этой цели транспортный уровень пользуется услугами, предоставляемыми сетевым уровнем. Программа и/или аппаратура, выполняющая работу транспортного уровня, называется **транспортной подсистемой** или **транспортным объектом (transport entity)**. Транспортная подсистема может располагаться в ядре операционной системы, в библиотечном модуле, загруженном сетевым приложением, в отдельном пользовательском процессе или даже в сетевой интерфейсной плате. Первые два варианта чаще всего встречаются в сети Интернет. Логическая взаимосвязь сетевого, транспортного и прикладного уровней проиллюстрирована на рис. 6.1.

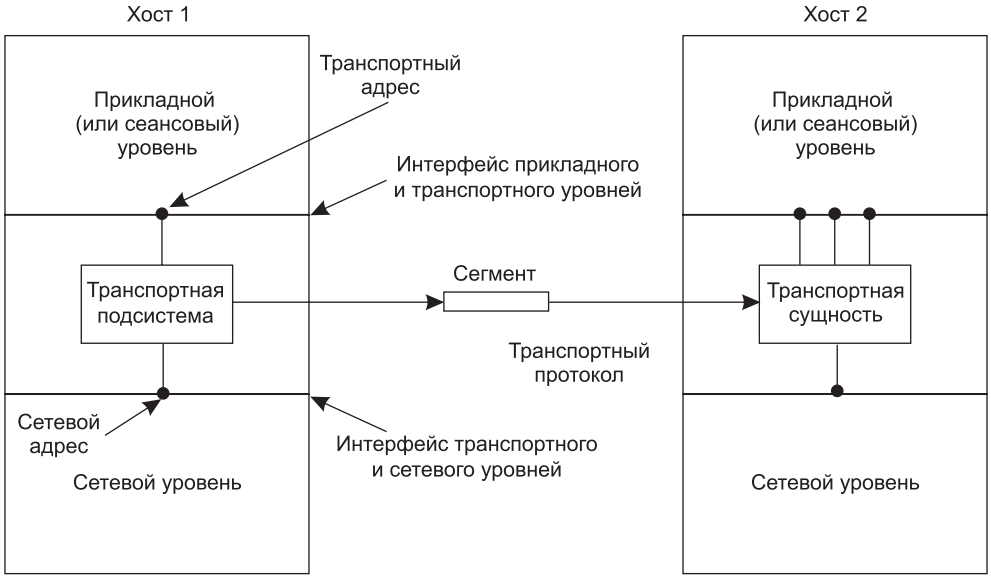


Рис. 6.1. Сетевой, транспортный и прикладной уровни

Сервисы транспортного уровня, как и сервисы сетевого уровня, делятся на сервисы с установлением соединения и сервисы без установления соединения. Транспортный сервис с установлением соединения во многом похож на аналогичный сетевой сервис. В обоих случаях соединение проходит три этапа: установление соединения, передача данных и разъединение. Адресация и управление потоком на разных уровнях также схожи. Более того, похожи друг на друга и сервисы без установления соединения разных уровней. Однако следует обратить внимание на то, что реализовать комбинацию транспортного сервиса без установления соединения и сетевого сервиса с установлением соединения часто достаточно трудно, так как для этого придется разрывать только что созданное соединение сразу же после отправки пакета.

Возникает закономерный вопрос: если сервис транспортного уровня так схож с сервисом сетевого уровня, то зачем нужны два различных уровня? Почему не достаточно одного уровня? Это довольно тонкий, но очень важный вопрос. Программное обеспечение транспортного уровня запускается целиком на пользовательских машинах, а сетевой уровень запускается в основном на маршрутизаторах, которые управляются оператором связи (по крайней мере, в глобальных сетях). Что произойдет, если сетевой уровень будет предоставлять сервис с установлением соединения, но этот сервис будет ненадежным? Что если он часто будет терять пакеты? Что случится, если маршрутизаторы будут время от времени выходить из строя?

В этом случае пользователи столкнутся с большими проблемами. У них нет контроля над сетевым уровнем, поэтому они не смогут решить проблему плохого обслуживания, используя хорошие маршрутизаторы или совершенствуя обработку ошибок канального уровня (просто потому, что маршрутизаторы им не принадлежат). Единственная возможность заключается в использовании для улучшения качества обслуживания еще одного уровня, расположенного над сетевым. Если в сети без

установления соединения пакеты теряются или передаются с искажениями, транспортная подсистема обнаруживает проблему и выполняет повторную передачу. Если в сети с установлением соединения транспортная подсистема узнает, что ее сетевое соединение было внезапно прервано, без каких-либо сведений о том, что случилось с передаваемыми в этот момент данными, она может установить новое соединение с удаленной транспортной подсистемой. С помощью нового сетевого соединения она может послать запрос к равноранговому объекту и узнать, какие данные дошли до адресата, а какие нет, после чего, зная, где это произошло, продолжить передачу данных.

По сути, благодаря наличию транспортного уровня транспортный сервис может быть более надежным, чем лежащая ниже сеть. Более того, базовые операции транспортного сервиса могут быть разработаны таким образом, что они будут независимы от базовых операций сетевого сервиса, которые могут значительно варьироваться от сети к сети (например, сервис Ethernet без установления соединения может значительно отличаться от сервиса сети WiMAX с установлением соединения). Если спрятать сетевой сервис за набором базовых операций транспортного сервиса, то для изменения сетевого сервиса потребуется просто заменить один набор библиотечных процедур другими, делающими то же самое, но с помощью других сервисов более низкого уровня.

Благодаря наличию транспортного уровня прикладные программы могут использовать стандартный набор базовых операций и сохранять работоспособность в самых различных сетях. Им не придется учитывать имеющееся разнообразие интерфейсов сетей и уровней надежности. Если бы все реальные сети работали идеально и у всех сетей был один набор базовых операций, который гарантированно никогда не мог бы быть изменен, то транспортный уровень, вероятно, был бы не нужен. Однако в реальном мире он выполняет ключевую роль изолирования верхних уровней от деталей технологии, устройства и несовершенства сети.

Именно по этой причине часто проводится разграничение между уровнями с первого по четвертый и уровнями выше четвертого. Нижние четыре уровня можно рассматривать как **поставщика транспортных услуг (transport service provider)**, а верхние уровни — как **потребителя транспортных услуг (transport service user)**. Разделение на поставщика и пользователя оказывает серьезное влияние на устройство уровней и помещает транспортный уровень на ключевую позицию, поскольку он формирует основную границу между поставщиком и пользователем надежного сервиса передачи данных. Именно этот уровень виден приложениям.

6.1.2. Базовые операции транспортного сервиса

Чтобы пользователи могли получить доступ к транспортному сервису, транспортный уровень должен совершать некоторые действия по отношению к прикладным программам, то есть предоставлять интерфейс транспортного сервиса. У всех транспортных сервисов есть свои интерфейсы. В этом разделе мы вначале рассмотрим простой (но гипотетический) пример транспортного сервиса и его интерфейсов, просто чтобы узнать основные принципы и понятия. Следующий раздел будет посвящен реальному примеру.

Транспортный сервис подобен сетевому, но имеет и некоторые существенные отличия. Главное отличие состоит в том, что сетевой сервис предназначен для моделирования сервисов, предоставляемых реальными сетями со всеми их особенностями. Реальные сети могут терять пакеты, поэтому в общем случае сетевой сервис ненадежен.

Транспортный сервис с установлением соединения, напротив, является надежным. Конечно, реальные сети содержат ошибки, но именно транспортный уровень как раз и должен обеспечивать надежный сервис поверх ненадежной сети.

В качестве примера рассмотрим два процесса, работающих на одной машине, соединенных каналом в системе UNIX (или с помощью любого средства, позволяющего обеспечить взаимодействие процессов). Эти процессы предполагают, что соединение между ними абсолютно идеально. Они не желают знать о подтверждениях, потерянных пакетах, перегрузках и т. п. Им требуется стопроцентно надежное соединение. Процесс *A* помещает данные в один конец канала, а процесс *B* извлекает их на другом. Именно для этого и предназначен транспортный сервис с установлением соединения — скрывать несовершенство сетевого сервиса, чтобы пользовательские процессы могли считать, что существует безошибочный поток битов, даже если процессы выполняются на разных машинах.

Кстати, транспортный уровень может также предоставлять ненадежный (дейтаграммный) сервис, но о нем сказать почти нечего (разве что «это дейтаграммы»), поэтому мы в данной главе сконцентрируемся на транспортном сервисе с установлением соединения. Тем не менее существуют приложения, например клиент-серверные вычислительные системы и потоковое мультимедиа, основанные на транспортных сервисах без установления соединения, поэтому ниже мы еще упомянем их.

Второе различие между сетевым и транспортным сервисами состоит в том, для кого они предназначены. Сетевой сервис используется только транспортными подсистемами. Мало кто пишет свои собственные транспортные подсистемы, и поэтому пользователи и программы почти не встречаются с голым сетевым сервисом. Базовые операции транспортного сервиса, напротив, используются многими программами, а следовательно, и программистами. Поэтому транспортный сервис должен быть удобным и простым в употреблении.

Чтобы получить представление о транспортном сервисе, рассмотрим пять базовых операций, перечисленных в табл. 6.1. Этот транспортный интерфейс сильно упрощен, но он дает представление о назначении транспортного интерфейса с установлением соединения. Он позволяет прикладным программам устанавливать, использовать и освобождать соединения, чего вполне достаточно для многих приложений.

Чтобы понять, как могут быть использованы эти операции, рассмотрим приложение, состоящее из сервера и нескольких удаленных клиентов. Вначале сервер выполняет операцию `LISTEN` — обычно для этого вызывается библиотечная процедура, которая обращается к системе. В результате сервер блокируется, пока клиент не обратится к нему. Когда клиент хочет поговорить с сервером, он выполняет операцию `CONNECT`. Транспортная подсистема выполняет эту операцию, блокируя обратившегося к ней клиента и посылая пакет серверу. Поле данных пакета содержит сообщение транспортного уровня, адресованное транспортной подсистеме сервера.

Следует сказать пару слов о терминологии. За неимением лучшего термина, для сообщений, посылаемых одной транспортной подсистемой другой транспортной подсистеме, нам придется использовать термин **сегмент (segment)**. Это обозначение используется в TCP, UDP и других интернет-протоколах. В более старых протоколах применяется несколько неуклюжее название **TPDU (Transport Protocol Data Unit — модуль данных транспортного протокола)**. Сейчас оно практически не используется, однако вы можете встретить его в старых статьях и книгах.

Таблица 6.1. Базовые операции простого транспортного сервиса

Базовая операция	Посланный сегмент	Значение
LISTEN (ОЖИДАТЬ)	(нет)	Блокировать сервер, пока какой-либо процесс не попытается соединиться
CONNECT (СОЕДИНИТЬ)	CONNECTION REQUEST (ЗАПРОС СОЕДИНЕНИЯ)	Активно пытаться установить соединение
SEND (ПОСЛАТЬ)	ДАННЫЕ	Послать информацию
RECEIVE (ПОЛУЧИТЬ)	(нет)	Блокировать сервер, пока не прибудут данные
DISCONNECT (РАЗЪЕДИНИТЬ)	DISCONNECTION REQUEST (ЗАПРОС РАЗЪЕДИНЕНИЯ)	Прервать соединение

Сегменты, которыми обменивается транспортный уровень, помещаются в пакеты (которыми обменивается сетевой уровень). Эти пакеты, в свою очередь, содержатся в кадрах, которыми обменивается канальный уровень. Получив кадр, процесс канального уровня обрабатывает заголовок кадра и, если адрес назначения совпадает с местом доставки, передает содержимое поля полезной нагрузки кадра наверх сетевой подсистеме. Точно так же сетевая подсистема обрабатывает заголовок пакета и передает содержимое поля полезной нагрузки пакета наверх транспортной подсистеме. Эта вложенность проиллюстрирована на рис. 6.2.

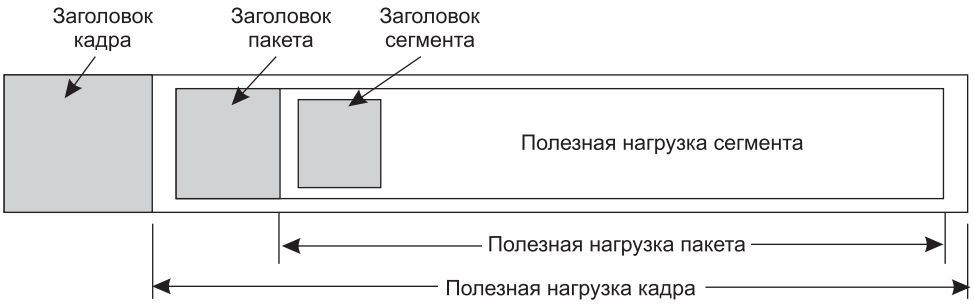


Рис. 6.2. Вложенность сегментов, пакетов и кадров

Итак, вернемся к нашему примеру общения клиента и сервера. В результате запроса клиента `CONNECT` серверу посылается сегмент, содержащий `CONNECTION REQUEST` (запрос

соединения). Когда он прибывает, транспортная подсистема проверяет, заблокирован ли сервер операцией LISTEN (то есть заинтересован ли сервер в обработке запросов). Затем она разблокирует сервер и посылает обратно клиенту сегмент CONNECTION ACCEPTED (соединение принято). Получив этот сегмент, клиент разблокируется, после чего соединение считается установленным.

Теперь клиент и сервер могут обмениваться данными с помощью базовых операций SEND и RECEIVE. В простейшем случае каждая из сторон может использовать блокирующую операцию RECEIVE для перехода в режим ожидания сегмента, посылаемого противоположной стороной при помощи операции SEND. Когда сегмент прибывает, получатель разблокируется. Затем он может обработать полученный сегмент и послать ответ. Такая схема прекрасно работает, пока обе стороны помнят, чей черед посылать, а чей — принимать.

Обратите внимание на то, что на транспортном уровне даже простая однонаправленная пересылка данных оказывается сложнее, чем на сетевом уровне. Каждый посланный пакет данных будет, в конце концов, подтвержден. Пакеты, содержащие управляющие сегменты, также подтверждаются, явно или неявно. Эти подтверждения управляются транспортными подсистемами при помощи протокола сетевого уровня и не видны пользователям транспортного уровня. Аналогично транспортные подсистемы должны беспокоиться о таймерах и повторных передачах. Все эти механизмы не видны пользователям транспортного уровня, для которых соединение представляется надежным битовым каналом. Один пользователь помещает в канал биты, которые волшебным образом появляются на другом конце канала в том же порядке. Эта способность скрывать сложность от пользователей свидетельствует о том, что многоуровневые протоколы являются довольно мощным инструментом.

Когда соединение больше не требуется, оно должно быть разорвано, чтобы можно было освободить место в таблицах двух транспортных подсистем. Разъединение существует в двух вариантах: симметричном и асимметричном. В асимметричном варианте любой потребитель транспортных услуг может вызвать примитив DISCONNECT, в результате чего удаленной транспортной сущности будет послан управляющий сегмент DISCONNECTION REQUEST (запрос разъединения). После получения сегмента удаленной транспортной подсистемой соединение разрывается.

В симметричном варианте каждое направление закрывается отдельно, независимо от другого. Когда одна сторона выполняет операцию DISCONNECT, это означает, что у нее более нет данных для передачи, но что она все еще готова принимать данные от своего партнера. В этой схеме соединение разрывается, когда обе стороны выполняют операцию DISCONNECT.

Диаграмма состояний для установления и разрыва соединения показана на рис. 6.3. Каждый переход вызывается каким-то событием — операцией, выполненной локальным пользователем транспортного сервиса, или входящим пакетом. Для простоты мы будем считать, что каждый сегмент подтверждается отдельно. Мы также предполагаем, что используется модель симметричного разъединения, в которой клиент делает первый ход. Обратите внимание на наивность этой модели. Позднее, когда мы будем говорить о TCP, мы рассмотрим более реалистичные модели.

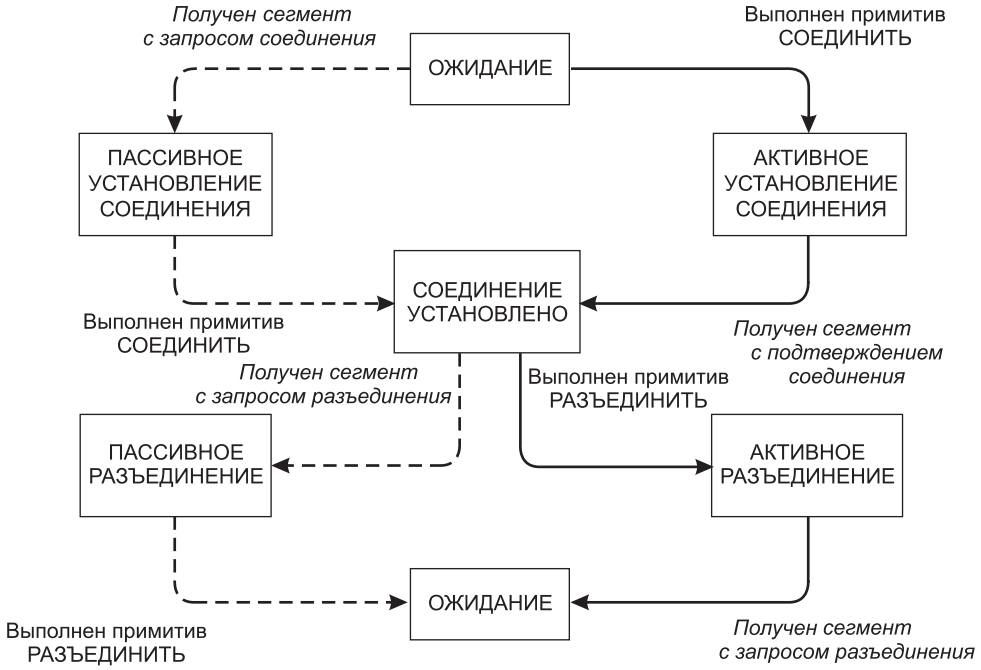


Рис. 6.3. Диаграмма состояний для простой схемы управления соединениями. Переходы, обозначенные курсивом, вызываются прибытием пакетов. Сплошными линиями показана последовательность состояний клиента. Пунктирными линиями показана последовательность состояний сервера

6.1.3. Сокеты Беркли

Теперь рассмотрим другой набор базовых операций транспортного уровня — базовые операции сокетов (иногда называемых гнездами), используемые для протокола **TCP** (**Transmission Control Protocol — протокол управления передачей**). Впервые сокеты стали применяться в 1983 году в операционной системе Berkeley UNIX 4.2BSD. Очень скоро они стали популярными и сейчас широко используются для интернет-программирования в большинстве операционных систем, особенно UNIX; кроме того, существует специальный API, предназначенный для программирования сокетов в системе Windows — «winsock».

Эти базовые операции приведены в табл. 6.2. Модель сокетов во многом подобна рассмотренной выше модели, но обладает большей гибкостью и предоставляет больше возможностей. Сегменты, соответствующие этой модели, будут рассматриваться ниже в этой главе.

Первые четыре базовых операции списка выполняются серверами в таком же порядке. Операция `SOCKET` создает новый сокет и выделяет для него место в таблице транспортной подсистемы. Параметры вызова указывают используемый формат адресов, тип требуемого сервиса (например, надежный поток байтов) и протокол. В случае

успеха операция SOCKET возвращает обычный описатель файла, используемый при вызове следующих операций, подобно тому, как процедура OPEN работает для файла.

Таблица 6.2. Базовые операции сокетов для TCP

Базовая операция	Значение
SOCKET (СОКЕТ)	Создать новый сокет (гнездо связи)
BIND (СВЯЗАТЬ)	Связать локальный адрес с сокетом
LISTEN (ОЖИДАТЬ)	Объявить о желании принять соединение; указать размер очереди
ACCEPT (ПРИНЯТЬ)	Пассивно установить входящее соединение
CONNECT (СОЕДИНИТЬ)	Активно пытаться установить соединение
SEND (ПОСЛАТЬ)	Послать данные по соединению
RECEIVE (ПОЛУЧИТЬ)	Получить данные у соединения
CLOSE (ЗАКРЫТЬ)	Разорвать соединение

У только что созданного сокета нет сетевых адресов. Они назначаются с помощью операции BIND. После того как сервер привязывает адрес к сокету, с ним могут связаться удаленные клиенты. Вызов SOCKET не создает адрес напрямую, так как некоторые процессы придают адресам большое значение (например, они использовали один и тот же адрес годами, и этот адрес всем известен), тогда как другим процессам это не важно.

Следом идет вызов LISTEN, который выделяет место для очереди входящих соединений на случай, если несколько клиентов попытаются соединиться одновременно. В отличие от операции LISTEN в нашем первом примере, операция LISTEN в модели сокетов не является блокирующим вызовом.

Чтобы заблокировать ожидание входящих соединений, сервер выполняет операцию ACCEPT. Получив сегмент с запросом соединения, транспортная подсистема создает новый сокет с теми же свойствами, что и у исходного сокета, и возвращает описатель файла для него. При этом сервер может разветвить процесс или поток, чтобы обработать соединение для нового сокета и вернуться к ожиданию следующего соединения для оригинального сокета.

Теперь посмотрим на этот процесс со стороны клиента. В этом случае также сначала с помощью операции SOCKET должен быть создан сокет, но операция BIND здесь не требуется, так как используемый адрес не имеет значения для сервера. CONNECT блокирует вызывающего и инициирует активный процесс соединения. Когда этот процесс завершается (то есть когда соответствующий сегмент, посланный сервером, получен), процесс клиента разблокируется, и соединение считается установленным. После этого обе стороны могут использовать SEND и RECIEVE для передачи и получения данных по полнодуплексному соединению. Могут также применяться стандартные UNIX-вызовы READ и WRITE, если нет нужды в использовании специальных свойств SEND и RECIEVE.

В модели сокетов используется симметричный разрыв соединения. Соединение разрывается, когда обе стороны выполняют операцию CLOSE.

Получив широкое распространение, сокеты де-факто стали стандартом абстрагирования транспортных сервисов для приложений. Часто сокет-API используется вместе с TCP-протоколом для предоставления сервиса с установлением соединения, который называется **надежным потоком байтов** (на деле это надежный битовый канал, о котором мы говорили выше). Этот API может сочетаться и с другими протоколами, но во всех случаях результат должен быть одинаковым для потребителя транспортных услуг.

Преимущество сокет-API состоит в том, что приложение может использовать его и для других транспортных сервисов. К примеру, с помощью сокетов можно реализовать транспортный сервис без установления соединения. В таком случае операция `CONNECT` будет задавать адрес удаленного узла, а `SEND` и `RECEIVE` — отправлять и получать дейтаграммы. (Иногда используется расширенный набор вызовов — например, операции `SENDTO` и `RECEIVEFROM`, позволяющие приложению не ограничиваться одним транспортным узлом.) Иногда сокеты используются с транспортными протоколами, в которых вместо байтового потока применяется поток сообщений и которые могут включать или не включать управление перегрузкой. К примеру, **DCCP (Datagram Congestion Control Protocol — дейтаграммный протокол с управлением перегрузкой)** является вариантом UDP, включающим управление перегрузкой (Rohler и др., 2006). Задачу выбора необходимого сервиса решают в первую очередь потребители транспортных услуг.

Тем не менее последнее слово в вопросе транспортных интерфейсов, скорее всего, останется не за сокетами. Довольно часто приложениям приходится работать с группой связанных потоков — так, например, браузер может одновременно запрашивать у сервера несколько объектов. В таком случае применение сокетов обычно означает, что для каждого объекта будет использоваться один поток, и в результате управление перегрузкой будет выполняться отдельно для каждого потока (а не для всей группы), что, безусловно, является далеко не оптимальным вариантом, поскольку управление набором потоков ложится на плечи приложения. Чтобы более эффективно обрабатывать группы связанных потоков и уменьшить роль приложения в этом процессе, были созданы новые протоколы и интерфейсы. В качестве примеров приведем **SCTP (Stream Control Transmission Protocol — протокол передачи с управлением потоками)**, описанный в RFC 4960, и **SST (Structured Stream Transport — иерархическая поточная транспортировка данных)** (Ford, 2007). Эти протоколы слегка изменяют сокет-API для удобства работы с группами потоков, обеспечивая ряд новых возможностей, таких как работа со смешанным трафиком (с установлением соединения и без) и даже поддержка множественных сетевых путей. О том, насколько они успешны, мы узнаем по прошествии некоторого времени.

6.1.4. Пример программирования сокета: файл-сервер для Интернета

Чтобы узнать, как выполняются настоящие вызовы для сокета, рассмотрим программу, демонстрирующую работу клиента и сервера, представленную в листинге 6.1. Имеется примитивный файл-сервер, работающий в Интернете и использующий его клиент.

У программы много ограничений (о которых еще будет сказано), но в принципе данный код, описывающий сервер, может быть скомпилирован и запущен на любой UNIX-системе, подключенной к Интернету. Код, описывающий клиента, может быть запущен с определенными параметрами. Это позволит ему получить любой файл, к которому у сервера есть доступ. Файл отображается на стандартном устройстве вывода, но, разумеется, может быть перенаправлен на диск или какому-либо процессу.

Листинг 6.1. Программы использования сокетов для клиента и сервера

```
/* На этой странице содержится клиентская программа, запрашивающая файл у серверной
программы, расположенной на следующей странице. */
/* Сервер в ответ на запрос высылает файл.*/

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345 /* По договоренности между клиентом и сервером */
#define BUF_SIZE 4096 /* Размер передаваемых блоков */

int main(int argc, char *argv)
{
  int c,s,bytes;
  char buf[BUF_SIZE];   /*буфер для входящего файла */
  struct hostent *h;    /*информация о сервере */
  struct sockaddr_in channel; /*хранит IP=адрес */

  if (argc!=3) fatal("Для запуска введите: client имя_сервера имя_файла");
  h = gethostbyname(argv[1]);     /* поиск IP-адреса хоста */
  if(!h) fatal("Ошибка выполнения gethostbyname")
  s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (s<0) fatal("Сокет");
  memset(&channel, 0, sizeof(channel));
  channel.sin_family=AF_INET;
  memcpy(&channel.sin_addr.s_addr,h>h_addr,h>h_length);
  channel.sin_port=htons(SERVER_PORT);

  c = connect(s,(struct sockaddr *) &channel, sizeof(channel));
  if (c<0) fatal("Ошибка соединения");

  /* Соединение установлено. Посылается имя файла с нулевым байтом на конце */
  write*s, argv[2], strlen(argv[2])+1);

  /* Получить файл, записать на стандартное устройство вывода */
  while (1) {
        bytes = read(s, buf, BUF_SIZE);  /* Читать из сокета */
        if (bytes <= 0) exit(0);         /* Проверка конца файла */
        write(1, buf, bytes);            /* Записать на стандартное устройство вывода */
  }
}
fatal(char *string)
{
  printf("%s\n", string);
```

```
  exit(1);
}
/* Код программы для сервера */
#include <sys/types.h>
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345  /* По договоренности между клиентом и сервером */
#define BUF_SIZE 4096      /* Размер передаваемых блоков */
#define QUEUE_SIZE 10
int main(int argc, char *argv[]);
{
  int s, b, l, fd, sa, bytes, on = 1;
  char buf[BUF_SIZE];            /* буфер для исходящего файла */
  struct sockaddr_in channel;    /* содержит IP-адрес */

  /* Создать структуру адреса для привязки к сокету */
  memset(&channel, 0, sizeof(channel));    /* Обнуление channel */
  channel.sin_family = AF_INET;
  channel.sin_addr.s_addr = htonl(INADDR_ANY);
  channel.sin_port = htons(SERVER_PORT);

  /* Пассивный режим. Ожидание соединения */
  s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); /* создать сокет */
  if (s<0) fatal("ошибка сокета");
  setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));
  b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
  if (b<0) fatal("Ошибка связывания");

  l = listen(s, QUEUE_SIZE);    /* Определение размера очереди */
  if (l<0) fatal("Ошибка ожидания");

  /* Теперь сокет установлен и привязан. Ожидание и обработка соединения */
  while (1) {
        sa = accept(s, 0, 0);   /* Блокировка в ожидании запроса соединения*/
        if (sa<0) fatal("Ошибка доступа");

        read(sa, buf, BUF_SIZE);        /* считать имя файла из сокета */

        /* Получить и вернуть файл */
        fd = open(buf, O_RDONLY);       /* Открыть файл для отсылки */
        if (fd < 0) fatal("Ошибка открытия файла");

        while (1) {
              bytes = read(fd, buf, BUF_SIZE); /* Читать из файла */
              if (bytes <= 0) break;           /* Проверка конца файла */
              write(sa, buf, bytes);           /* Записать байты в сокет */
        }
        close(fd);                             /* Закрыть файл */
        close(sa);                             /* Разорвать соединение */
  }
}
```

Рассмотрим сперва ту часть программы, которая описывает сервер. Она начинается с включения некоторых стандартных заголовков, последние три из которых содержат основные структуры и определения, связанные с Интернетом. Затем SERVER_PORT определяется как 12345. Значение выбрано случайным образом. Любое число от 1024 до 65535 подойдет с не меньшим успехом, если только оно не используется каким-либо другим процессом; порты с номерами 1023 и ниже зарезервированы для привилегированных пользователей.

В последующих двух строках определяются две необходимые серверу константы. Первая из них задает размер блока данных для передачи файлов (в байтах). Вторая определяет максимальное количество незавершенных соединений, после установки которых новые соединения будут отвергаться.

После объявления локальных переменных начинается сама программа сервера. Вначале она инициализирует структуру данных, которая будет содержать IP-адрес сервера. Эта структура будет связана с серверным сокетом. Вызов memset полностью обнуляет структуру данных. Последующие три присваивания заполняют три поля этой структуры. Последнее из них содержит порт сервера. Функции htonl и htons занимаются преобразованием значений в стандартный формат, что позволяет программе нормально выполняться как на машинах с представлением числовых разрядов little-endian (например, Intel x86), так и с представлением big-endian (например, SPARC). Детали их семантики здесь роли не играют.

После этого сервером создается и проверяется на ошибки (определяется по $s < 0$) сокет. В окончательной версии программы сообщение об ошибке может быть чуть более понятным. Вызов setsockopt нужен для того, чтобы порт мог использоваться несколько раз, а сервер — бесконечно, обрабатывая запрос за запросом. Теперь IP-адрес привязывается к сокету и выполняется проверка успешного завершения вызова bind. Конечным этапом инициализации является вызов listen, свидетельствующий о готовности сервера к приему входящих вызовов и сообщающий системе о том, что нужно ставить в очередь до QUEUE_SIZE вызовов, пока сервер обрабатывает текущий вызов. При заполнении очереди прибытие новых запросов спокойно игнорируется.

В этом месте начинается основной цикл программы, который никогда не покидается. Его можно остановить только извне. Вызов accept блокирует сервер на то время, пока клиент пытается установить соединение. Если вызов завершается успешно, accept возвращает дескриптор (описатель) сокета, который можно использовать для чтения и записи, аналогично тому, как файловые дескрипторы (описатели) могут применяться для чтения и записи в каналы. Однако, в отличие от однонаправленных каналов, сокеты двунаправлены, поэтому для чтения (и записи) данных из соединения можно использовать sa (принятый сокет). Файловые дескрипторы канала могут использоваться для чтения или записи, но не для того и другого одновременно.

После установления соединения сервер считывает имя файла. Если оно пока недоступно, сервер блокируется, ожидая его. Получив имя файла, сервер открывает файл и входит в цикл, который читает блоки данных из файла и записывает их в сокет. Это продолжается до тех пор, пока не будут скопированы все запрошенные данные. Затем файл закрывается, соединение разрывается и начинается ожидание нового вызова. Данный цикл повторяется бесконечно.

Теперь рассмотрим часть кода, описывающую клиента. Чтобы понять, как работает программа, необходимо вначале разобраться, как она запускается. Если она называется `client`, ее типичный вызов будет выглядеть так:

```
client flits.csvu.nl /usr/tom/filename >f
```

Этот вызов сработает только в том случае, если сервер расположен по адресу *flits.cs.vu.nl*, файл `/usr/tom/filename` существует и у сервера есть доступ по чтению для этого файла. Если вызов произведен успешно, файл передается по Интернету и записывается на место `f`, после чего клиентская программа заканчивает свою работу. Поскольку серверная программа продолжает работать, клиент может быть запущен снова с новыми запросами на получение файлов.

Клиентская программа начинается с подключения файлов и объявлений. Работа начинается с проверки корректности числа аргументов (`argc` = 3 означает, что в строке запуска содержалось имя программы и два аргумента). Обратите внимание на то, что `argv[1]` содержит имя сервера (например, *flits.cs.vu.nl*) и переводится в IP-адрес с помощью `gethostbyname`. Для поиска имени функция использует DNS. Мы будем изучать технологию DNS в главе 7.

Затем создается и инициализируется сокет, после чего клиент пытается установить TCP-соединение с сервером посредством `connect`. Если сервер включен, работает на указанной машине, соединен с `SERVER_PORT` и либо простаивает, либо имеет достаточно места в очереди `listen` (очереди ожидания), то соединение с клиентом будет рано или поздно установлено. По данному соединению клиент передает имя файла, записывая его в сокет. Количество отправленных байтов на единицу превышает требуемое для передачи имени, поскольку нужен еще нулевой байт-ограничитель, с помощью которого сервер может понять, где кончается имя файла.

Теперь клиентская программа входит в цикл, читает файл блок за блоком из сокета и копирует на стандартное устройство вывода. По окончании этого процесса она просто завершается.

Процедура `fatal` выводит сообщение об ошибке и завершается. Серверу также требуется эта процедура, и она пропущена в листинге только из соображений экономии места. Поскольку программы клиента и сервера компилируются отдельно и в обычной ситуации запускаются на разных машинах, код процедуры `fatal` не может быть разделяемым.

Эти две программы (как и другие материалы, связанные с этой книгой) можно найти на веб-сайте книги по адресу *http://www.prenhall.com/tanenbaum*.

Кстати говоря, такой сервер построен далеко не по последнему слову техники. Осуществляемая проверка ошибок минимальна, а сообщения об ошибках реализованы весьма посредственно. Система будет обладать низкой производительностью, поскольку все запросы обрабатываются только последовательно (используется один поток запросов). Понятно, что ни о какой защите информации здесь говорить не приходится, а применение аскетичных системных вызовов UNIX — это не лучшее решение для достижения независимости от платформы. Делаются некоторые некорректные с технической точки зрения предположения. Например, о том, что имя файла всегда поместится в буфер и будет передано без ошибок. Несмотря на эти недостатки, с помо-

щью данной программы можно организовать полноценный работающий файл-сервер для Интернета. Более подробную информацию можно найти в (Donahoo и Calvert, 2008, 2009).

6.2. Элементы транспортных протоколов

Транспортные услуги реализуются **транспортным протоколом**, используемым между двумя транспортными подсистемами. В некоторых отношениях транспортные протоколы напоминают протоколы канального уровня, подробно изучавшиеся в главе 3. И те и другие протоколы, наряду с другими вопросами, занимаются обработкой ошибок, управлением очередями и потоками.

Однако у них имеется и много существенных различий, обусловленных различиями условий, в которых работают эти протоколы, как показано на рис. 6.4. На канальном уровне два маршрутизатора общаются напрямую по физическому каналу (проводному или беспроводному), тогда как на транспортном уровне физический канал заменен целой сетью. Это отличие оказывает важное влияние на протоколы.

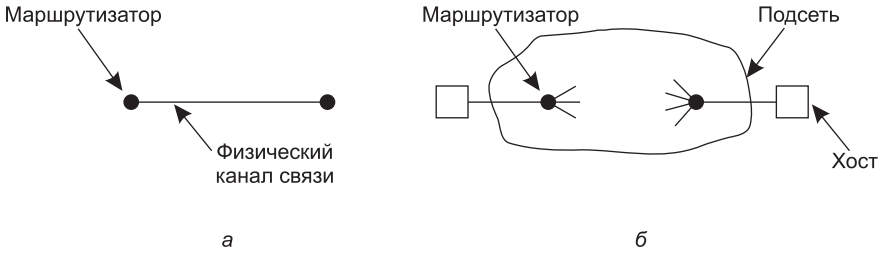


Рис. 6.4. Окружение: *а* — канального уровня; *б* — транспортного уровня

Во-первых, при двухточечном соединении по проводу или оптоволоконной линии маршрутизатору обычно не требуется указывать, с каким маршрутизатором он хочет поговорить, — каждая выходная линия ведет к определенному маршрутизатору. На транспортном уровне требуется явно указывать адрес получателя.

Во-вторых, процесс установки соединения по проводу (рис. 6.4, *а*) прост: противоположная сторона всегда присутствует (если только она не вышла из строя). В любом случае, работы не очень много. Даже в случае беспроводного соединения ситуация отличается не сильно. Простой отправки сообщения достаточно, чтобы оно дошло до всех адресов назначения. Если подтверждение о получении не приходит (вследствие ошибок), сообщение может быть отправлено повторно. На транспортном уровне начальная установка соединения, как будет показано ниже, происходит достаточно сложно.

Еще одно весьма досадное различие между канальным и транспортным уровнями состоит в том, что сеть потенциально обладает возможностями хранения информации. Когда маршрутизатор посылает пакет по каналу связи, он может прибыть или потеряться, но он не может побродить где-то какое-то время, спрятаться в отдаленном уголке земного шара, а затем внезапно появиться, прибыв в место назначения после пакетов, отправленных гораздо позже него. Если сеть использует дейтаграммы, кото-

рые независимо маршрутизируются внутри нее, то всегда есть ненулевая вероятность того, что пакет пройдет по какому-то странному пути и прибудет на место назначения позже, чем нужно, и в неправильном порядке; возможно также, что при этом будут получены копии пакета. Последствия способности сети задерживать и копировать пакеты иногда могут быть катастрофичными и требуют применения специальных протоколов, обеспечивающих правильную передачу данных.

Последнее различие между канальным и транспортным уровнями является скорее количественным, чем качественным. Буферизация и управление потоком необходимы на обоих уровнях, но наличие на транспортном уровне большого изменяющегося количества соединений с пропускной способностью, колеблющейся вследствие конкуренции соединений, может потребовать принципиально другого подхода, отличного от использовавшегося на канальном уровне. Некоторые протоколы, упоминавшиеся в главе 3, выделяют фиксированное количество буферов для каждой линии, так что, когда прибывает кадр, всегда имеется свободный буфер. На транспортном уровне из-за большого количества управляемых соединений и колебаний пропускной способности для каждого из них идея выделения нескольких буферов каждому соединению выглядит не столь привлекательно. В следующих разделах мы изучим эти и другие важные вопросы.

6.2.1. Адресация

Когда один прикладной процесс желает установить соединение с другим прикладным процессом, он должен указать, с кем именно он хочет связаться. (У не требующего соединений транспортного сервиса проблемы такие же: кому следует посылать каждое сообщение?) Применяемый обычно метод состоит в определении транспортных адресов, к которым процессы могут посылать запросы на установку соединения. В Интернете такие конечные точки называются **портами**. Мы будем пользоваться нейтральным термином **TSAP (Transport Service Access Point — точка доступа к услугам транспортного уровня)** для обозначения конечных точек на транспортном уровне. Аналогичные конечные точки сетевого уровня (то есть адреса сетевого уровня) называются **NSAP (Network Service Access Point — точка доступа к сетевым услугам)**. Примерами NSAP являются IP-адреса.

Рисунок 6.5 иллюстрирует взаимоотношения между NSAP, TSAP и транспортным соединением. Прикладные процессы, как клиента, так и сервера, могут связываться с локальной TSAP для установки соединения с удаленной TSAP. Такие соединения проходят через NSAP на каждом хосте, как показано на рисунке. TSAP нужны для того, чтобы различать конечные точки, совместно использующие NSAP, в сетях, где у каждого компьютера есть своя NSAP.

Возможный сценарий для транспортного соединения выглядит следующим образом:

1. Процесс почтового сервера подсоединяется к точке доступа TSAP 1522 на хосте 2 и ожидает входящего вызова. Вопрос о том, как процесс соединяется с TSAP, лежит за пределами сетевой модели и целиком зависит от локальной операционной системы. Например, может вызываться примитив, подобный `LISTEN`.

2. Прикладной процесс хоста 1 желает отправить почтовое сообщение, и поэтому он подсоединяется к TSAP 1208 и обращается к сети с запросом CONNECT, указывая TSAP 1208 на хосте 1 в качестве адреса отправителя и TSAP 1522 на хосте 2 в качестве адреса получателя. Это действие в результате приводит к установке транспортного соединения между прикладным процессом и сервером.
3. Прикладной процесс отправляет почтовое сообщение.
4. Почтовый сервер отвечает, что сообщение будет доставлено.
5. Транспортное соединение разрывается.

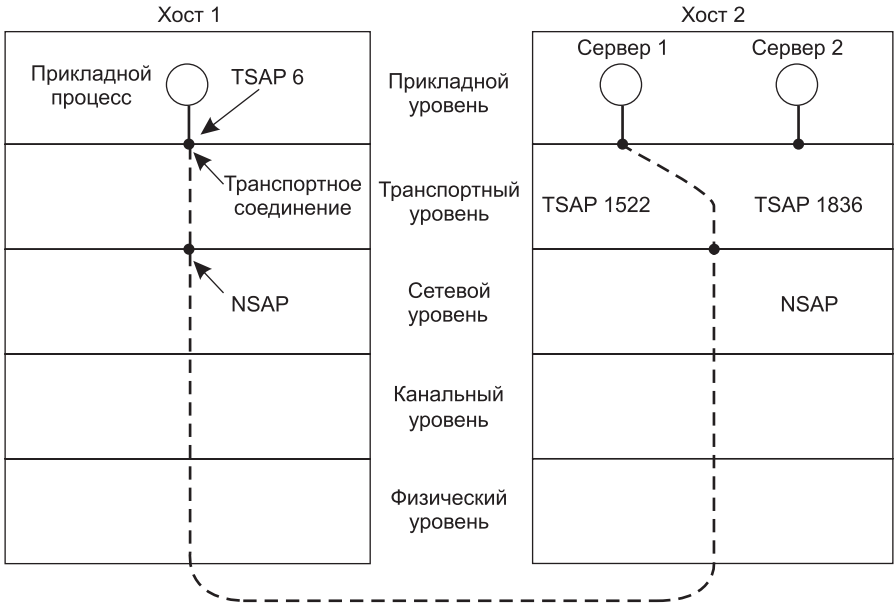


Рис. 6.5. Точки доступа к услугам транспортного и сетевого уровня и транспортные соединения

Обратите внимание на то, что на хосте 2 могут располагаться и другие серверы, соединенные со своими TSAP и ожидающие входящих запросов на соединение, приходящих с той же NSAP.

Нарисованная выше картинка всем хороша, но мы обошли стороной один маленький вопрос: как пользовательский процесс хоста 1 узнает, что почтовый сервер соединен с TSAP 1522? Возможно, почтовый сервер подключается к TSAP 1522 в течение долгих лет, и постепенно об этом узнают все пользователи сети. В этом случае службы имеют постоянные TSAP-адреса, хранящиеся в файлах, расположенных в известных местах. Так, например, в /etc/services UNIX-систем перечисляются серверы, за которыми жестко закреплены определенные порты, с указанием этих портов — в частности, там указано, что почтовый сервер использует TCP порт 25.

Хотя постоянные TSAP-адреса могут хорошо подходить для небольшого количества никогда не меняющихся ключевых служб (например, таких как веб-сервер), в общем случае пользовательские процессы часто хотят пообщаться с другими пользо-

вательскими процессами, TSAP-адреса которых заранее не известны или существуют только в течение короткого времени.

Чтобы справиться с этой ситуацией, может использоваться другая схема. В этой модели используется специальный процесс, называющийся **сопоставителем портов (portmapper)**. Чтобы найти TSAP-адрес, соответствующий данному имени службы, например «BitTorrent», пользователь устанавливает соединение с сопоставителем портов (TSAP-адрес которого всем известен). Затем пользователь посылает сообщение с указанием названия нужной ему услуги, и сопоставитель портов сообщает ему TSAP-адрес этой службы. После этого пользователь разрывает соединение с сопоставителем портов и устанавливает новое соединение с нужной ему службой.

В этой модели, когда создается новая служба, она должна зарегистрироваться на сопоставителе портов, сообщив ему название услуги (обычно строка ASCII) и TSAP-адрес. Сопоставитель портов сохраняет полученную информацию в своей базе данных, чтобы иметь возможность отвечать на будущие запросы.

Функция сопоставителя портов аналогична работе оператора телефонной справочной службы — он преобразует имена в номера. Как и в телефонной системе, важно, чтобы TSAP-адрес сопоставителя портов (или обрабатывающего сервера в протоколе начального соединения) был действительно хорошо известен. Если вы не знаете номера телефонной справочной, вы не сможете позвонить оператору. Если вы полагаете, что номер справочной является очевидным, попытайтесь угадать его, находясь в другой стране.

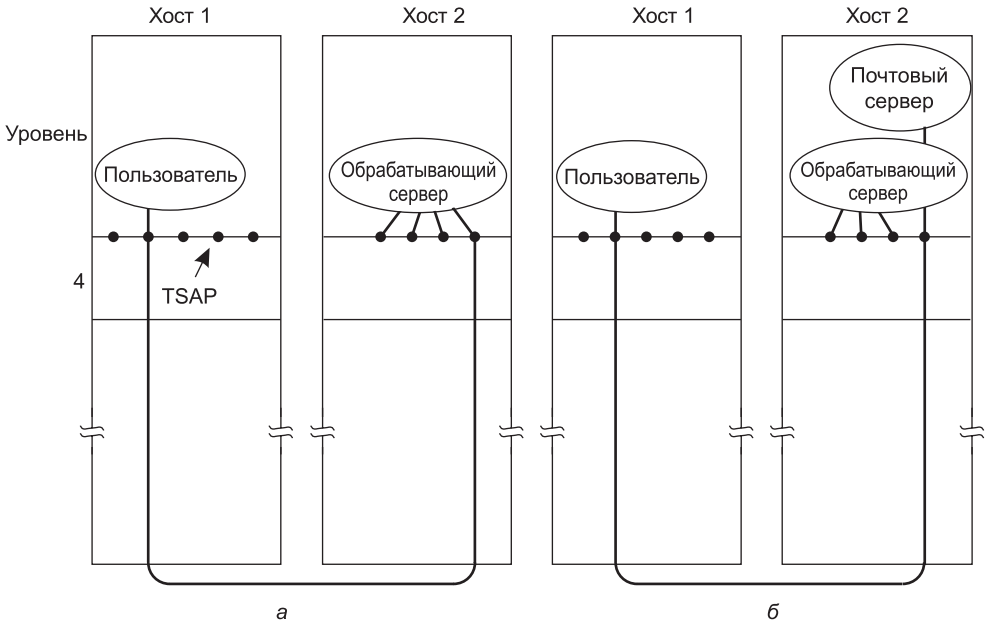


Рис. 6.6. Пользовательский процесс хоста 1 устанавливает соединение с почтовым сервером хоста 2 через обрабатывающий сервер

Так как многие из серверных процессов, существующих на конкретной машине, используются редко, слишком расточительным делом оказывается поддержка всех их

в активном состоянии с постоянными TSAP-адресами. Альтернативный вариант показан в упрощенном виде на рис. 6.6. Он называется **протоколом начального соединения (initial connection protocol)**. Вместо того чтобы назначать всем возможным серверам хорошо известные TSAP-адреса, каждая машина, желающая предоставлять услуги удаленным пользователям, обзаводится специальным **обрабатывающим сервером (process server)**, действующим как прокси (посредник) для менее активно используемых серверов. В UNIX-системах такой сервер называется *inetd*. Он прослушивает одновременно несколько портов, ожидая запроса на соединение. Потенциальные пользователи начинают с того, что посылают запрос CONNECT, указывая TSAP-адрес нужной им службы. Если никакой сервер их не ждет, они получают соединение с обрабатывающим сервером, как показано на рис. 6.6, *а*.

Получив запрос, обрабатывающий сервер порождает подпроцесс запрошенного сервера, позволяя ему унаследовать существующее соединение с пользователем. Новый сервер выполняет требуемую работу, в то время как обрабатывающий сервер возвращается к ожиданию новых запросов, как показано на рис. 6.6, *б*. Этот метод работает только в тех случаях, когда серверы могут создаваться по требованию.

6.2.2. Установка соединения

Установка соединения, хотя и просто звучит, неожиданно оказывается весьма непростым делом. На первый взгляд, должно быть достаточно одной транспортной подсистеме послать адресату сегмент с запросом соединения CONNECTION REQUEST и услышать в ответ CONNECTION ACCEPTED (соединение принято). Неприятность заключается в том, что сеть может потерять, задержать, повредить или дублировать пакеты.

Представьте себе сеть настолько перегруженную, что подтверждения практически никогда не доходят вовремя, каждый пакет опаздывает и пересылается повторно по два-три раза. Предположим, что сеть основана на дейтаграммах и что каждый пакет следует по своему маршруту. Некоторые пакеты могут застрять в давке и прийти с большим опозданием в тот момент, когда отправитель будет думать, что они утеряны.

Самый кошмарный сценарий выглядит следующим образом. Пользователь устанавливает соединение с банком и посылает сообщение с требованием банку перевести крупную сумму денег на счет не совсем надежного человека. К несчастью, пакеты решают прогуляться по замысловатому маршруту, посетив самые отдаленные уголки сети. Тем временем отправитель вынужден выполнить повторную передачу пакетов. На этот раз они идут по кратчайшему пути и доставляются быстро, в результате чего отправитель разрывает соединение.

Происходит очередная неудача: первая порция пакетов выходит из укрытия и добирается до адресата в нужном порядке, предлагая банку установить новое соединение и (снова) выполнить перевод денег. У банка нет способа определить, что это дубликаты. Он решает, что это вторая независимая транзакция, и еще раз переводит деньги.

Такой сценарий может показаться маловероятным или даже неправдоподобным, но идея состоит вот в чем: протоколы должны работать корректно во всех случаях. Самые часто встречающиеся ситуации должны быть реализованы с максимальной эффективностью, требующейся для высокой производительности сети, однако протокол должен уметь справляться и с редкими сценариями, не приводя к сбоям. В противном

случае сеть будет ненадежной и может неожиданно сломаться, даже не сообщив об ошибке.

В оставшейся части этого раздела мы будем изучать проблему задержавшихся дубликатов, уделяя особое внимание алгоритмам, устанавливающим соединение надежным образом. Основная проблема заключается в том, что задержавшиеся дубликаты распознаются как новые пакеты. Избежать копирования и задержки пакетов мы не можем. Однако если это происходит, дублированные пакеты должны отвергаться и не обрабатываться как новые.

Эту проблему можно попытаться решить несколькими способами, ни один из которых, на самом деле, не является удовлетворительным. Так, например, можно использовать одноразовые транспортные адреса. При таком подходе каждый раз, когда требуется транспортный адрес, генерируется новый адрес. Когда соединение разрывается, этот адрес уничтожается. В таком случае задержавшиеся дубликаты пакетов не смогут найти транспортный адрес и, следовательно, перестанут представлять угрозу. Однако при этом установление соединения с процессом окажется более трудной задачей.

Другая возможность состоит в том, что каждому соединению присваивается уникальный идентификатор (то есть последовательный номер, который возрастает на единицу для каждого установленного соединения), выбираемый инициатором соединения и помещаемый в каждый сегмент, включая тот, который содержит запрос на соединение. После разрыва каждого соединения каждая транспортная подсистема может обновить таблицу, в которой хранятся устаревшие соединения в виде пар (одноранговая транспортная подсистема, идентификатор соединения). Для каждого приходящего запроса соединения может быть проверено, не хранится ли уже его идентификатор в таблице (он мог остаться там со времен разорванного ранее соединения).

К сожалению, у этой схемы есть существенный изъян: требуется, чтобы каждая транспортная подсистема хранила неопределенно долго некоторое количество информации об истории соединений, причем эту информацию должен хранить как отправитель, так и получатель. Иначе, если машина выйдет из строя и потеряет свою память, она уже не сможет определить, какие соединения уже использовались, а какие нет.

Вместо этого можно применить другой подход, который существенно упростит проблему. Можно разработать механизм, уничтожающий устаревшие заблудившиеся пакеты и не позволяющий им существовать в сети бесконечно долго. При таком ограничении проблема станет более управляемой.

Время жизни пакета может быть ограничено до известного максимума с помощью одного из следующих методов:

1. Проектирование сети с ограничениями.
2. Помещение в каждый пакет счетчика транзитных участков.
3. Помещение в каждый пакет временного штампа.

К первому способу относятся все методы, предотвращающие зацикливание пакетов, в комбинации с ограничением задержки, включая перегрузки по самому длинному возможному пути. Осуществить эту идею достаточно трудно, учитывая, что интерсеть может охватывать как один город, так и весь мир. Второй метод заключается

в начальной установке счетчика на определенное значение и уменьшении на единицу этого значения на каждом маршрутизаторе. Сетевой протокол передачи данных просто игнорирует все пакеты, значение счетчика которых дошло до нуля. Третий метод состоит в том, что в каждый пакет помещается время его создания, а маршрутизаторы договариваются игнорировать все пакеты старше определенного времени. Для последнего метода требуется синхронизация часов маршрутизаторов, что само по себе является нетривиальной задачей. На практике возраст пакета можно довольно точно вычислять с помощью счетчика транзитных участков.

На практике нужно гарантировать не только то, что пакет мертв, но и что все его подтверждения также мертвы. Поэтому вводится период T, который в несколько раз превышает максимальное время жизни пакета. Для каждой сети максимальное время жизни пакета — это константа, выбранная с запасом; для сети Интернет она составляет 120 с. На какое число умножается максимальное время жизни пакета, зависит от протокола, и это влияет только на длительность интервала времени T. Если подождать в течение интервала времени T с момента отправки пакета, то можно быть уверенным, что все его следы уничтожены и что пакет не возникнет вдруг как гром среди ясного неба.

При ограниченном времени жизни пакетов можно разработать надежный и практичный способ отвергать задержавшиеся дублированные сегменты. Автором описанного ниже метода является Томлинсон (Tomlinson, 1975); позднее он был улучшен Саншайном (Sunshine) и Далалом (Dalal). Его варианты широко применяются на практике, и одним из примеров применения является TCP.

Основная идея метода заключается в том, что отправитель присваивает сегментам последовательные номера, которые не будут повторно использоваться в течение последующих T секунд. Размер такого номера складывается из этого периода, T, а также скорости пакетов в секунду. Таким образом, в любой момент времени может отправляться только один пакет с данным номером. Копии этого пакета все равно могут появиться, и получатель должен их удалить. Однако теперь невозможна такая ситуация, при которой задержавшийся дубликат пакета принимается получателем вместо нового пакета с тем же порядковым номером.

Чтобы обойти проблему потери машиной памяти предыдущих состояний (при выходе ее из строя), можно сделать так, чтобы транспортная подсистема оставалась неактивной в течение первых T секунд после восстановления. В таком случае все старые сегменты исчезнут, и отправитель сможет начать процесс заново, используя любой последовательный номер. Недостаток этой идеи состоит в том, что в крупных интерсетях период времени T может быть достаточно большим.

Вместо этого Томлинсон предложил снабдить каждый хост часами. Часы разных хостов синхронизировать не обязательно. Предполагалось, что часы представляют собой двоичный счетчик, увеличивающийся через равные интервалы времени. Кроме того, число разрядов счетчика должно равняться числу битов в последовательных номерах (или превосходить его). Последнее и самое важное предположение состоит в том, что часы продолжают идти, даже если хост зависает.

При установке соединения младшие k бит часов используются в качестве k-битного начального порядкового номера. Таким образом, в отличие от протоколов, описанных в главе 3, каждое соединение начинает нумерацию своих сегментов с разных чисел.

Диапазон этих номеров должен быть достаточно большим, чтобы к тому моменту, когда порядковые номера сделают полный круг, старые сегменты с такими же номерами уже давно исчезли. Линейная зависимость порядковых номеров от времени показана на рис. 6.7. Запретная зона показывает, в какой момент времени определенные порядковые номера сегментов являются недействительными. При отправке сегмента с порядковым номером из этой зоны он может задержаться и сыграть роль другого пакета с таким же номером, который будет отправлен позже. К примеру, если хост выходит из строя и возобновляет работу в момент времени 70 с, он будет использовать начальные порядковые номера на основе показаний часов; хост не начинает отсчет с наименьшего порядкового номера в запретной зоне.

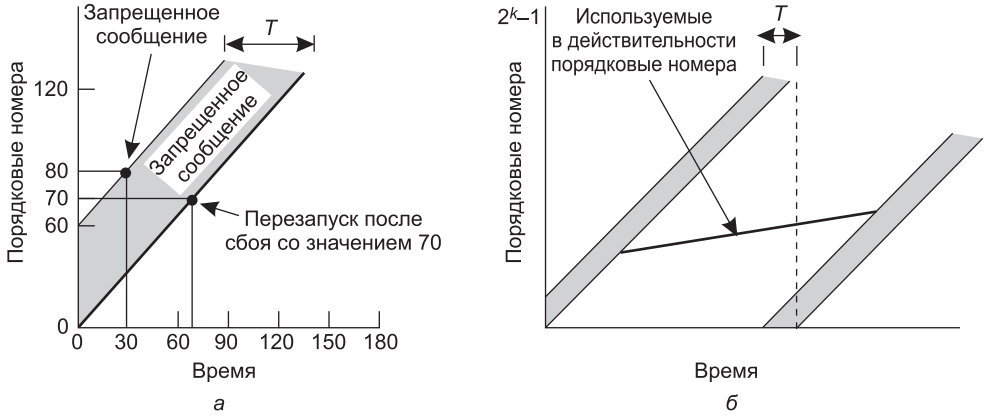


Рис. 6.7. Сегменты не могут заходить в запретную зону (*а*); проблема ресинхронизации (*б*)

Как только обе транспортные подсистемы договариваются о начальном порядковом номере, для управления потоком данных может применяться любой протокол скользящего окна. Такой протокол правильно найдет и удалит дубликаты пакетов, когда они уже будут приняты. В действительности график порядковых номеров (показанный жирной линией) не прямой, а ступенчатый, так как показания часов увеличиваются дискретно. Впрочем, для простоты мы проигнорируем эту деталь.

Чтобы порядковые номера пакетов не попадали в запретную зону, необходимо обратить внимание на следующее.

Неприятности у протокола могут возникнуть по двум причинам. Если хост посылает слишком быстро и слишком много данных, кривая используемых в действительности порядковых номеров может оказаться круче линии зависимости начальных номеров от времени, в результате чего порядковый номер попадет в запретную зону. Чтобы этого не произошло, скорость передачи данных в каждом открытом соединении должна быть ограничена одним сегментом за единицу времени. Кроме того, это означает, что транспортная подсистема после восстановления, прежде чем открывать новое соединение, должна подождать, пока изменят свое состояние часы, чтобы один и тот же номер не использовался дважды. Следовательно, интервал изменения состояния часов должен быть коротким (1 мкс или меньше). Но также часы не должны идти слишком быстро (относительно порядковых номеров). Если скорость

часов равна C, а пространство порядковых номеров имеет размер S, условие $S/C > T$ является обязательным, чтобы порядковые номера не сделали полный круг слишком быстро.

В запретную зону можно попасть не только снизу, передавая данные слишком быстро. Как видно из рис. 6.7, *б*, при любой скорости передачи данных, меньшей скорости часов, кривая используемых в действительности порядковых номеров попадет в запретную зону слева, когда порядковые номера пройдут по кругу. Чем круче наклон этой кривой, тем дольше придется ждать этого события. Чтобы избежать такой ситуации, можно ограничить скорость продвижения порядковых номеров для соединения (или время жизни соединения).

Использующий показания часов метод решает проблему невозможности различения опаздывающих дубликатов сегментов и новых сегментов. Однако в таком случае при установлении соединений могут возникнуть проблемы. Поскольку обычно получатель не помнит порядковые номера для различных соединений, он не может узнать, является ли сегмент CONNECTION REQUEST, содержащий какой-либо начальный порядковый номер, дубликатом одного из предыдущих соединений. Для текущего соединения такой проблемы не возникает, так как протокол скользящего окна знает текущий порядковый номер.

Для разрешения этой специфической проблемы Томлинсон (1975) предложил **тройное рукопожатие (three-way handshake)**. Этот протокол установления соединения предполагает, что одна из сторон проверяет, является ли соединение все еще действующим. Нормальная процедура установления соединения показана на рис. 6.8, *а*. Хост 1 инициирует установление, выбирая порядковый номер x, и посылает сегмент CONNECTION REQUEST, содержащий этот начальный порядковый номер, хосту 2. Хост 2 отвечает сегментом ACK, подтверждая x и объявляя свой начальный порядковый номер y. Наконец, хост 1 подтверждает выбранный хостом 2 начальный порядковый номер в первом посылаемом им информационном сегменте.

Рассмотрим теперь работу «тройного рукопожатия» в присутствии задержавшегося дубликата управляющего сегмента. На рис. 6.8, *б* первый сегмент представляет собой задержавшийся дубликат сегмента CONNECTION REQUEST от старого соединения. Этот сегмент прибывает на хост 2 тайком от хоста 1. Хост 2 реагирует на этот сегмент отправкой хосту 1 сегмента ACK, таким образом, прося хост 1 подтвердить, что тот действительно пытался установить новое соединение. Когда хост 1 отказывается это сделать, хост 2 понимает, что он был обманут задержавшимся дубликатом, и прерывает соединение. Таким образом, задержавшийся дубликат не причиняет вреда.

При наихудшем сценарии оба сегмента — CONNECTION REQUEST и ACK — блуждают по подсети. Этот случай показан на рис. 6.8, *в*. Как и в предыдущем примере, хост 2 получает задержавшийся сегмент CONNECTION REQUEST и отвечает на него. В этом месте следует обратить внимание на то, что хост 2 предложил использовать y в качестве начального порядкового номера для трафика от хоста 2 к хосту 1, хорошо зная, что сегментов, содержащих порядковый номер y, или их подтверждений в данный момент в сети нет. Когда хост 2 получает второй задержавшийся сегмент, он понимает, что это дубликат, так как в этом модуле подтверждается не y, а z. Здесь важно понять, что не существует такой комбинации сегментов, которая заставила бы протокол ошибиться и случайно установить соединение, когда оно никому не нужно.

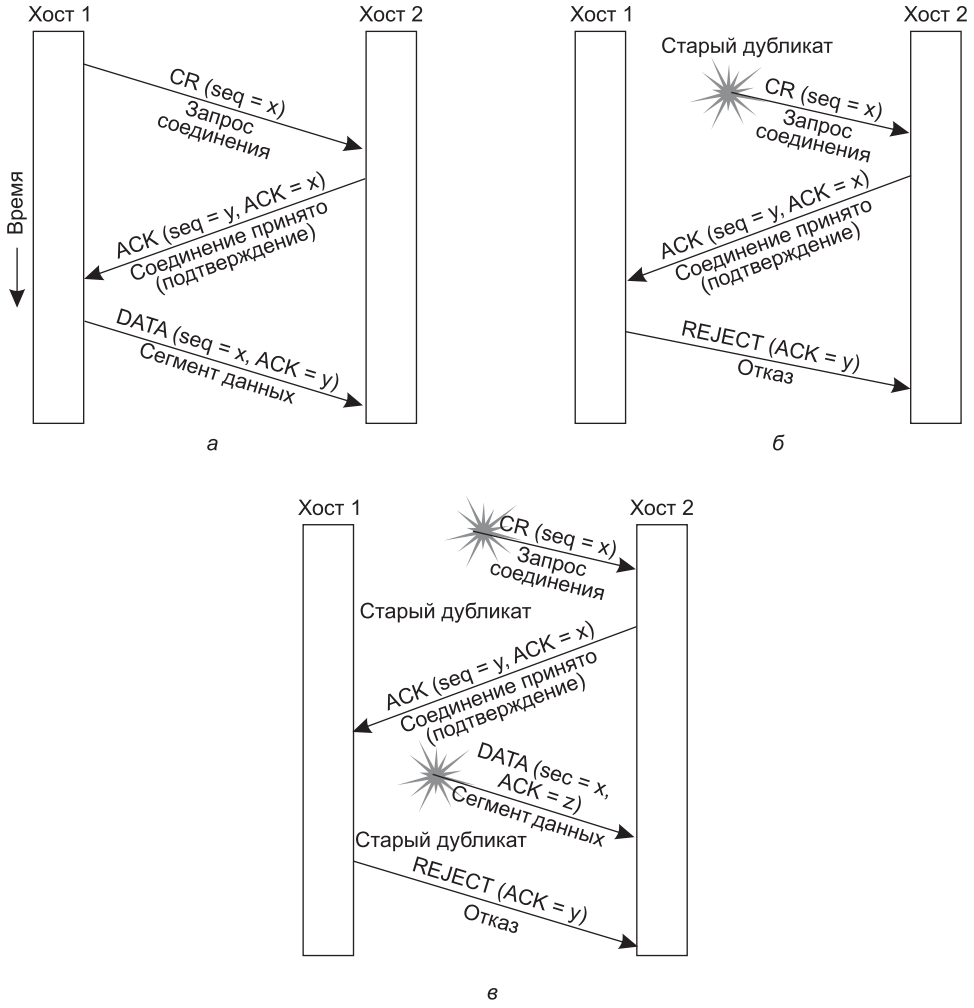


Рис. 6.8. Три сценария установки соединения с помощью «тройного рукопожатия» (CR означает CONNECTION REQUEST): *а* — нормальная работа; *б* — появление старого дубликата CR; *в* — дубликат сегмента CR и дубликат сегмента ACK

TCP использует «тройное рукопожатие» для установки соединения. Внутри соединения к 32-битному порядковому номеру добавляется метка времени, чтобы он не мог использоваться повторно в течение максимального времени жизни пакета, даже если скорость соединения составляет несколько гигабит в секунду. Этот механизм был добавлен в TCP для решения проблем, возникших при использовании быстрых линий. Он описан в RFC 1323 и называется **PAWS (Protection Against Wrapped Sequence numbers — детектирование повторного использования порядковых номеров)**. Для работы с несколькими соединениями, имеющими начальные порядковые номера, до появления PAWS протокол TCP применял метод, использующий показания часов (см. выше). Однако этот метод оказался неэффективным с точки зрения безопасности.

Благодаря использованию часов злоумышленники могли легко угадать следующий начальный порядковый номер, и таким образом обманув схему «тройного рукопожатия», инициировать ложное соединение. Поэтому на практике используются псевдослучайные начальные порядковые номера. Но для таких порядковых номеров, со стороны кажущихся абсолютно случайными, все-таки важно, чтобы они не повторялись в течение определенного промежутка времени. Иначе задержавшиеся дубликаты могут вызвать серьезные неполадки в работе сети.

6.2.3. Разрыв соединения

Разорвать соединение проще, чем установить. Тем не менее здесь также имеются подводные камни. Как уже было сказано, существует два стиля разрыва соединения: асимметричный и симметричный. Асимметричный разрыв связи соответствует принципу работы телефонной системы: когда одна из сторон вешает трубку, связь прерывается. При симметричном разрыве соединение рассматривается в виде двух отдельных однонаправленных связей, и требуется раздельное завершение каждого соединения.

Асимметричный разрыв связи является внезапным и может привести к потере данных. Рассмотрим сценарий, показанный на рис. 6.9. После установки соединения хост 1 посылает сегмент, который успешно добирается до хоста 2. Затем хост 1 посылает другой сегмент. К несчастью, хост 2 посылает `DISCONNECTION REQUEST` (запрос разъединения — DR) прежде, чем прибывает второй сегмент. В результате соединение разрывается, а данные теряются.

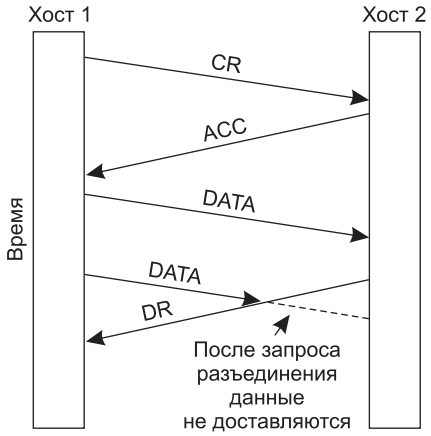


Рис. 6.9. Внезапное разъединение с потерей данных

Очевидно, требуется более сложный протокол, позволяющий избежать потери данных. Один из способов состоит в использовании симметричного разъединения, при котором каждое направление разъединяется независимо от другого. В этом случае хост может продолжать получать данные даже после того, как сам послал запрос разъединения.

Симметричное разъединение хорошо подходит для тех случаев, когда у каждой стороны есть фиксированное количество данных для передачи, и каждая сторона точно

знает, когда эти данные заканчиваются. В других случаях определить, что работа закончена и соединение может быть прервано, не так просто. Можно представить себе протокол, в котором хост 1 говорит: «Я закончил. Вы тоже закончили?» Если хост 2 отвечает: «Я тоже закончил. До свидания», соединение можно безо всякого риска разъединять.

К сожалению, этот протокол работает не всегда. Существует знаменитая проблема, называемая **проблемой двух армий**. Представьте, что армия белых расположилась в долине, как показано на рис. 6.10. На возвышенностях по обеим сторонам долины расположились две армии синих. Белая армия больше, чем любая из армий синих, но вместе синие превосходят белых. Если любая из армий синих атакует белых в одиночку, она потерпит поражение, но если синие сумеют атаковать белых одновременно, они могут победить.

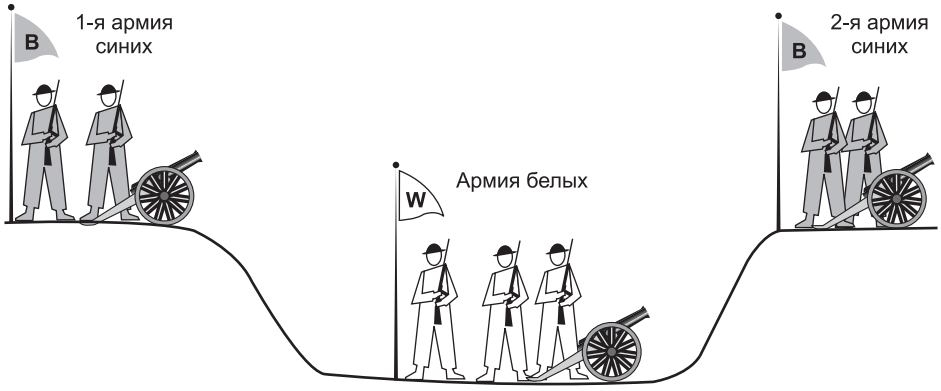


Рис. 6.10. Проблема двух армий

Синие армии хотели бы синхронизировать свое выступление. Однако единственный способ связи заключается в отправке вестового пешком по долине, где он может быть схвачен, а донесение потеряно (то есть приходится пользоваться ненадежным каналом). Спрашивается: существует ли протокол, позволяющий армиям синих победить?

Предположим, командир первой армии синих посылает следующее сообщение: «Я предлагаю атаковать 29 марта, на рассвете. Сообщите ваше мнение». Теперь предположим, что сообщение успешно доставляется и что командир 2-й армии синих соглашается, а его ответ успешно доставляется обратно в 1-ю армию синих. Состоится ли атака? Вероятно, нет, так как командир 2-й армии не уверен, что его ответ получен. Если нет, то 1-я армия синих не будет атаковать, и было бы глупо с его стороны в одиночку ввязываться в сражение.

Теперь улучшим протокол с помощью «тройного рукопожатия». Инициатор оригинального предложения должен подтвердить ответ. Но и в этом случае останется неясным, было ли доставлено последнее сообщение. Протокол четырехкратного рукопожатия здесь также не поможет.

В действительности, можно доказать, что протокола, решающего данную проблему, не существует. Предположим, что такой протокол все же существует. В этом случае по-

следнее сообщение протокола либо является важным, либо нет. Если оно не является важным, удалим его (а также все остальные несущественные сообщения), пока не останется протокол, в котором все сообщения являются существенными. Что произойдет, если последнее сообщение не дойдет до адресата? Мы только что сказали, что сообщение является важным, поэтому если оно потеряется, атака не состоится. Поскольку отправитель последнего сообщения никогда не сможет быть уверен в его получении, он не станет рисковать. Другая синяя армия это знает и также воздержится от атаки.

Чтобы увидеть, какое отношение проблема двух армий имеет к разрыву соединения, просто замените слово «атаковать» на «разъединить». Если ни одна сторона не готова разорвать соединение до тех пор, пока она не уверена, что другая сторона также готова к этому, то разъединение не произойдет никогда.

На практике, чтобы справиться с этой ситуацией, нужно отказаться от идеи договоренности и переложить ответственность на потребителей транспортных услуг, так чтобы каждая сторона приняла независимое решение о том, когда будет выполнена операция. Такую проблему решить проще. На рис. 6.11 показаны четыре сценария разъединения, использующих «тройное рукопожатие». Хотя этот протокол и не безошибочен, обычно он работает успешно.

На рис. 6.11, *а* показан нормальный случай, в котором один из пользователей посылает запрос разъединения DR (DISCONNECTION REQUEST), чтобы инициировать разрыв соединения. Когда запрос прибывает, получатель посылает обратно также запрос разъединения DR и включает таймер на случай, если запрос потеряется. Когда запрос прибывает, первый отправитель посылает в ответ на него сегмент с подтверждением ACK и разрывает соединение. Наконец, когда прибывает ACK, получатель также разрывает соединение. Разрыв соединения означает, что транспортная подсистема удаляет информацию об этом соединении из своей таблицы открытых соединений и сигнализирует о разрыве соединения владельцу соединения (потребителю транспортного сервиса). Эта процедура отличается от применения пользователем операции `DISCONNECT`.

Если последний сегмент с подтверждением теряется (рис. 6.11, *б*), ситуацию спасает таймер. Когда время истекает, соединение разрывается в любом случае.

Теперь рассмотрим случай потери второго запроса разъединения DR. Пользователь, инициировавший разъединение, не получит ожидаемого ответа, у него истечет время ожидания и он начнет все сначала. На рис. 6.11, *в* показано, как это происходит в случае, если все последующие запросы и подтверждения успешно доходят до адресатов.

Последний сценарий (рис. 6.11, *г*) аналогичен предыдущему с одной лишь разницей: в этом случае предполагается, что все повторные попытки передать запрос разъединения DR также терпят неудачу, поскольку все сегменты теряются. После *N* повторных попыток отправитель, наконец, сдается и разрывает соединение. Тем временем у получателя также истекает время, и он тоже разрывает соединение.

Хотя такого протокола обычно бывает вполне достаточно, теоретически он может ошибиться, если потеряются начальный запрос разъединения DR и все *N* повторных попыток. Отправитель сдается и разрывает соединение, тогда как другая сторона ничего не знает о попытках разорвать связь и сохраняет активность. В результате получается полуоткрытое соединение.

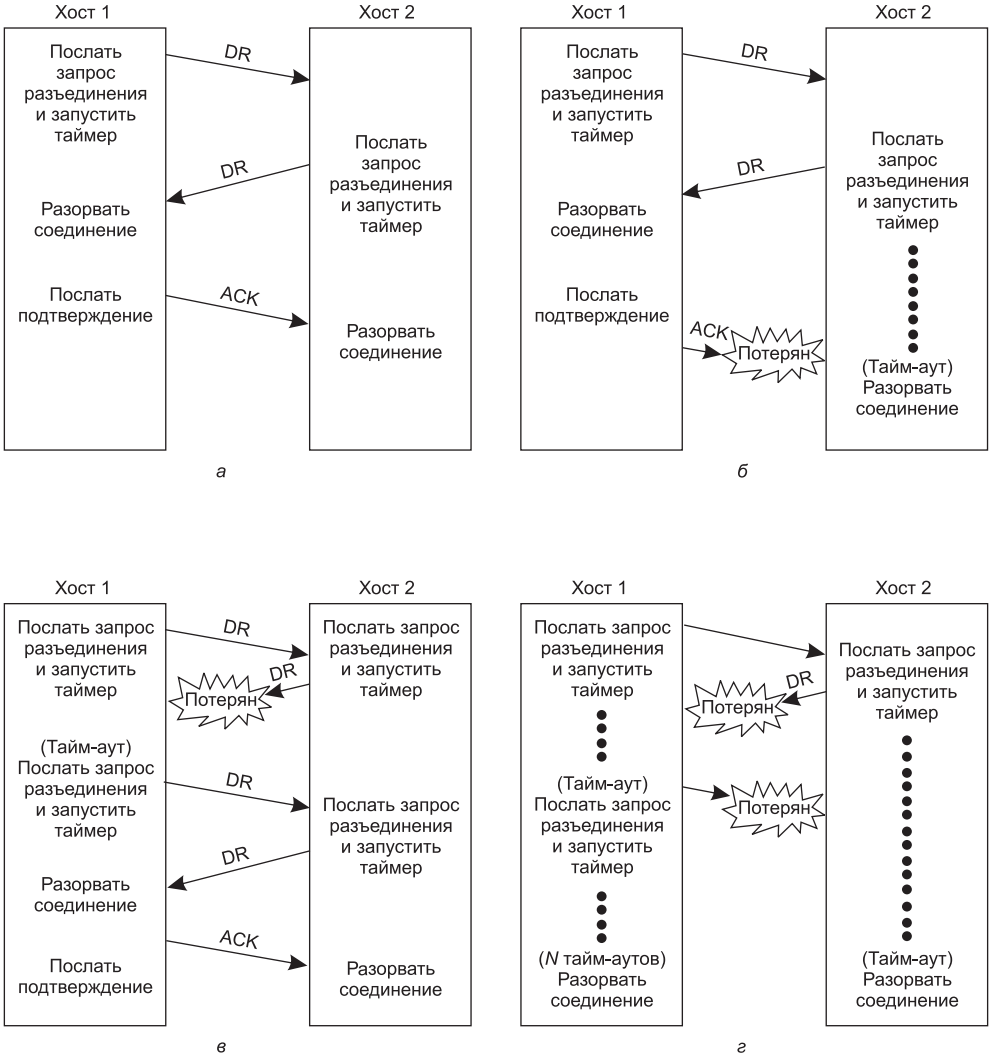


Рис. 6.11. Четыре сценария разрыва соединения: *а* — нормальный случай «тройного рукопожатия»; *б* — потеряно последнее подтверждение; *в* — потерян ответ; *г* — потерян ответ и последующие запросы

Этой ситуации можно было бы избежать, если не позволять отправителю сдаваться после N повторных попыток, а заставить его продолжать попытки, пока не будет получен ответ. Однако, если другой стороне будет разрешено разрывать связь по таймеру, тогда отправитель действительно будет вечно повторять попытки, так как ответа он не получит никогда. Если же получающей стороне также не разрешать разрывать соединение по таймеру, тогда протокол зависнет в ситуации, изображенной на рис. 6.11, *г*.

Чтобы удалять полуоткрытые соединения, можно применять правило, гласящее, что если по соединению в течение определенного времени не прибывает ни одного

сегмента, соединение автоматически разрывается. Таким образом, если одна сторона отсоединится, другая обнаружит отсутствие активности и также отсоединится. Это правило также работает для тех случаев, когда соединение разрывается не по инициативе одной из сторон, а по причине невозможности передачи пакетов между этими хостами в сети. Для реализации этого правила каждая сторона должна управлять таймером, перезапускаемым после передачи каждого сегмента. Если этот таймер срабатывает, посылается пустой сегмент — лишь для того, чтобы другая сторона не повесила трубку. С другой стороны, если применяется правило автоматического разъединения и теряется слишком большое количество сегментов подряд, то соединение автоматически разрывается сначала одной стороной, а затем и другой.

На этом мы заканчиваем обсуждение данного вопроса, но теперь должно быть ясно, что разорвать соединение без потери данных не так просто, как это кажется на первый взгляд. Из всего сказанного выше можно сделать вывод, что потребитель транспортных услуг должен принимать участие в решении вопроса о разрыве соединения — транспортная подсистема не может сама с этим справиться. Обратите внимание на то, что хотя TCP обычно использует симметричный разрыв связи (при этом каждая сторона независимо прерывает свое соединение, отправляя пакет FIN после окончания передачи данных), веб-серверы часто передают клиентам специальный пакет RST, сообщающий о мгновенном разрыве соединения — что больше похоже на асимметричный разрыв. Это возможно только благодаря тому, что веб-сервер знаком с процедурой обмена данными. Сначала он получает запрос от клиента (единственное, что отправляет клиент), а затем отсылает клиенту ответ. Таким образом, после отправки этого ответа обмен данными завершен: каждая сторона передала другой то, что требовалось. Поэтому сервер может резко разорвать соединение, отправив клиенту предупреждение. Получив это предупреждение, клиент сразу же разорвет соединение. Если предупреждение не дойдет до клиента, он через какое-то время поймет, что сервер с ним больше не общается, и также разорвет соединение. В любом случае данные будут успешно переданы.

6.2.4. Контроль ошибок и управление потоком данных

Изучив процессы установления и разрыва соединения, рассмотрим, как происходит управление соединением во время его использования. Одними из ключевых проблем являются контроль ошибок и управление потоком данных. Контроль ошибок отвечает за то, чтобы данные передавались с необходимой степенью надежности — как правило, это означает, что данные должны доставляться без ошибок. Управление потоком данных состоит в согласовании скорости передатчика и приемника.

Оба этих вопроса мы уже обсуждали, когда говорили о канальном уровне. На транспортном уровне используются те же механизмы, которые описаны в главе 3. Вкратце они выглядят так.

1. Фрейм содержит код с обнаружением ошибок (например, CRC-код или контрольную сумму), с помощью которого проверяется, правильно ли была доставлена информация.

2. Фрейм содержит идентифицирующий порядковый номер и передается отправителем до тех пор, пока не придет подтверждение об успешной доставке. Это называется **ARQ (Automatic Repeat reQuest — автоматический запрос повторной передачи)**.
3. Число кадров, передаваемых отправителем в любой момент времени, обычно ограничено: передача приостанавливается, если подтверждения приходят недостаточно быстро. Если этот максимум равен одному пакету, протокол называется **протоколом с остановкой и ожиданием подтверждения (stop-and-wait)**. Окна большего размера позволяют использовать конвейерную обработку, а также улучшить производительность при работе с длинными и быстрыми линиями.
4. Протокол **скользящего окна (sliding window)** сочетает в себе все эти возможности, а также поддерживает двунаправленную передачу данных.

Если эти механизмы применяются к кадрам на канальном уровне, то возникает логичный вопрос: как это можно перенести на сегменты транспортного уровня? На практике оказывается, что канальный и транспортный уровни во многом копируют друг друга. Но хотя к ним и применимы одинаковые механизмы, существуют различия в их функционировании и качестве.

Чтобы проиллюстрировать различие в функционировании, обратимся к обнаружению ошибок. Контрольная сумма канального уровня защищает кадр, пока он передается по одному каналу. Контрольная сумма транспортного уровня защищает сегмент, пока он передается по всему пути. Это сквозная проверка, отличная от проверки в каждом канале. Существуют примеры повреждения пакетов даже внутри маршрутизаторов (Saltzer и др., 1984): контрольные суммы канального уровня обеспечивали защиту пакетов, пока они передвигались по каналу, но не тогда, когда они были внутри маршрутизатора. В результате мы имеем дело с некорректной доставкой, хотя проверка на каждом канале не выявила ошибок.

Этот и другие примеры позволили ученым (Slatzer и др.) сформулировать **«сквозной» принцип (end-to-end argument)**. Согласно этому принципу, сквозная проверка, выполняемая на транспортном уровне, является необходимой для корректной передачи данных; проверка, выполняемая на канальном уровне, не является необходимой, но позволяет существенно улучшить производительность (так как иначе поврежденный пакет будет все равно проходить весь путь, что приведет к лишней нагрузке на сеть).

Чтобы продемонстрировать различие в качестве, рассмотрим повторную передачу данных и протокол скользящего окна. Большинство беспроводных каналов, в отличие от спутниковых, позволяют отправлять только один кадр в единицу времени. Это значит, что произведение пропускной способности и времени задержки для данного канала достаточно мало и внутри канала может едва ли поместиться целый кадр. Поэтому здесь для лучшей производительности следует использовать окно маленького размера. К примеру, стандарт 802.11 использует протокол с остановкой и ожиданием подтверждения, выполняя передачу и повторные передачи одного кадра до тех пор, пока не придет подтверждение о его получении, и только после этого переходя к другому кадру. Если бы размер окна был больше одного кадра, это бы не улучшило производительность, а только усложнило процесс передачи. В проводных и оптоволоконных линиях связи, таких как (коммутируемые) Ethernet и магистрали интернет-

провайдеров, частота появления ошибок достаточно мала, что позволяет отказаться от повторных передач на канальном уровне, так как небольшое количество потерянных кадров может быть восстановлено с помощью повторных сквозных передач.

С другой стороны, для многих TCP-соединений произведение пропускной способности и времени задержки составляет гораздо больше, чем один сегмент. Рассмотрим соединение, которое передает данные по всей территории США со скоростью 1 Мбит/с, а время, за которое пакет доходит до получателя и обратно, составляет 100 мс. Даже при таком медленном соединении 200 Кбит информации будут храниться на получателе в течение того времени, которое требуется для отправки сегмента и получения подтверждения. В таких случаях следует использовать скользящее окно большого размера. Протокол с остановкой и ожиданием подтверждения серьезно повредит производительности. В нашем примере он ограничил бы передачу одним сегментом в 200 мс (или 5 сегментами в секунду) независимо от реальной скорости сети.

Поскольку транспортные протоколы обычно используют скользящие окна большего размера, мы обсудим процесс буферизации более подробно. Так как у хоста может быть несколько соединений, каждое из которых обрабатывается отдельно, для скользящих окон ему может потребоваться большое количество буферного пространства. Буферы должны использоваться как отправителем, так и получателем. Отправителю буфер нужен для хранения всех переданных сегментов, для которых еще не пришло подтверждение о прибытии, на случай если эти сегменты будут потеряны и их придется отправить повторно.

Зная, что отправитель выполняет буферизацию, получатель может использовать свои буферы по своему усмотрению. Например, получатель может содержать единый буферный накопитель, используемый всеми соединениями. Когда приходит сегмент, предпринимается попытка динамически выделить ему новый буфер. Если это удается, то сегмент принимается, в противном случае он отвергается. Поскольку отправитель готов к тому, чтобы передавать потерянные сегменты повторно, игнорирование сегментов получателем не наносит существенного вреда, хотя и расходует некоторые ресурсы. Отправитель просто повторяет попытки до тех пор, пока не получит подтверждения.

Выбор компромиссного решения между буферированием у отправителя и у получателя зависит от типа трафика соединения. Если трафик импульсный, небольшой мощности, как, например, трафик интерактивного терминала, лучше не выделять никаких буферов, а получать их динамически на обоих концах, полагаясь на буферизацию со стороны отправителя (на случай, если сегменты будут случайно удалены). С другой стороны, при передаче файла будет лучше, если получатель выделит целое окно буферов, чтобы данные могли передаваться с максимальной скоростью. Такую стратегию использует TCP.

Тем не менее открытым остается вопрос об организации набора буферов. Если большинство сегментов имеют примерно одинаковые размеры, естественно организовать буферы в виде массива буферов равной величины, каждый из которых может вместить один сегмент, как показано на рис. 6.12, *а*. Однако если сегменты сильно отличаются по размеру, от коротких запросов на загрузку веб-страниц до крупных пакетов при одноранговой передаче файлов, массив из буферов фиксированного размера окажется неудобным. Если размер буфера выбирать равным наибольшему возможному сегменту, то при хранении небольших сегментов память будет расходо-

ваться неэффективно. Если же сделать размер буфера меньшим, тогда для хранения большого сегмента потребуется несколько буферов с сопутствующими сложностями.

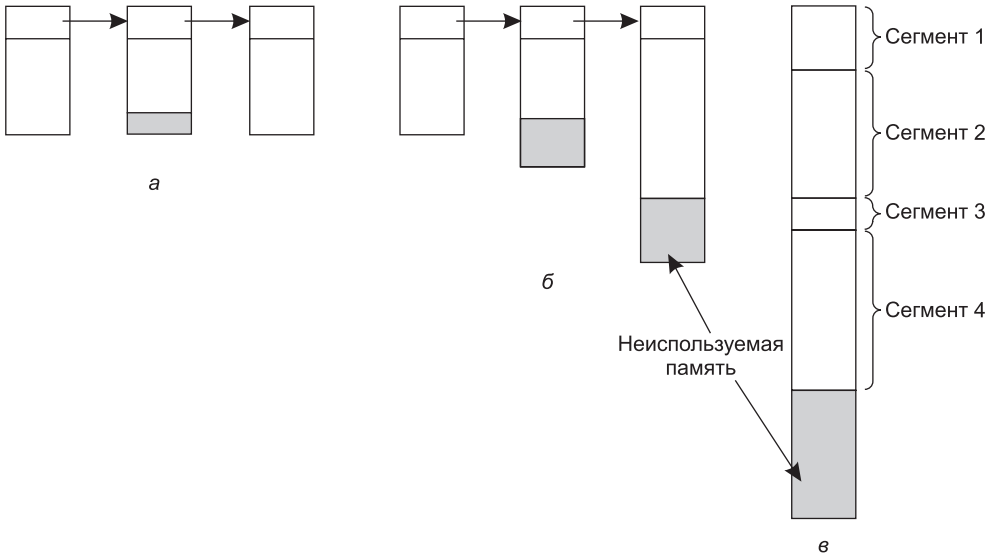


Рис. 6.12. Организация набора буферов: *а* — цепочка буферов фиксированного размера; *б* — цепочка буферов переменного размера; *в* — один большой циклический буфер для одного соединения

Другой метод решения указанной проблемы состоит в использовании буферов переменного размера, как показано на рис. 6.12, *б*. Преимущество этого метода заключается в более оптимальном использовании памяти, но платой за это является усложненное управление буферами. Третий вариант состоит в выделении соединению единого большого циклического буфера, как показано на рис. 6.12, *в*. Эта простая и изящная схема работает независимо от размера сегментов, однако она хорошо использует память, только если все соединения сильно нагружены.

При открытии и закрытии соединений и при изменении формы трафика отправитель и получатель должны динамически изменять выделенные буферы. Следовательно, транспортный протокол должен позволять отправителю посылать запросы на выделение буфера на другом конце. Буферы могут выделяться для каждого соединения или коллективно на все соединения между двумя хостами. В качестве альтернативы запросам буферов, получатель, зная состояние своих буферов, но не зная, какой трафик ему будет предложен, может сообщить отправителю, что он зарезервировал для него *X* буферов. При увеличении числа открытых соединений может потребоваться уменьшить количество или размеры выделенных буферов. Протокол должен предоставлять такую возможность.

В отличие от протоколов скользящего окна, описанных в главе 3, для реализации динамического выделения буферов следует отделить буферизацию от подтверждений. Динамическое выделение буферов означает, на самом деле, использование окна переменного размера. Вначале отправитель, основываясь на своих потребностях,

запрашивает определенное количество буферов. Получатель выделяет столько, сколько может. При отправлении каждого сегмента отправитель должен уменьшать на единицу число буферов, а когда это число достигнет нуля, он должен остановиться. Получатель отправляет обратно на попутных сегментах отдельно подтверждения и информацию об имеющихся у него свободных буферах. Эта схема используется в TCP; при этом информация о буферах хранится в поле заголовка *Window size*.

На рис. 6.13 показан пример управления динамическим окном в дейтаграммной сети с 4-битными порядковыми номерами. В этом примере данные передаются в виде сегментов от хоста *A* к хосту *B*, а подтверждения и запросы на предоставление буферов идут в обратном направлении (также в виде сегментов). Вначале хост *A* запрашивает 8 буферов, но ему выделяется только 4. Затем он посылает три сегмента, из которых последний теряется. На шаге 6 хост *A* получает подтверждение получения посланных им сегментов 0 и 1, в результате чего хост *A* может освободить буферы и послать еще три сегмента (с порядковыми номерами 2, 3 и 4). Хост *A* знает, что сегмент номер 2 он уже посылал, поэтому он думает, что может послать сегменты 3 и 4, что он и делает. На этом шаге он блокируется, так как его счетчик буферов достиг нуля и ждет предоставления новых буферов. На шаге 9 наступает тайм-аут хоста *A*, так как он до сих пор не получил подтверждения для сегмента 2. Этот сегмент посылается еще раз. В строке 10 хост *B* подтверждает получение всех сегментов, включая 4-й, но отказывается предоставлять буферы хосту *A*. Такая ситуация невозможна в протоколах с фиксированным размером окна, описанных в главе 3. Следующий сегмент, посланный хостом *B*, разрешает хосту *A* передать еще один сегмент. Это произойдет тогда, когда у *B* появится свободное буферное пространство — скорее всего, потому, что потребитель транспортных услуг принял больше данных.

	A	Сообщение	B	Комментарии
1	→	< request 8 buffers>	→	*A* хочет 8 буферов
2	←	<ack = 15, buf = 4>	←	*B* позволяет переслать только сообщения 0–3
3	→	<seq = 0, data = m0>	→	У *A* теперь осталось 3 буфера
4	→	<seq = 1, data = m1>	→	У *A* теперь осталось 2 буфера
5	→	<seq = 2, data = m2>	• • •	Сообщение потерялось, но *A* думает, что у него остался 1 буфер
6	←	<ack = 1, buf = 3>	←	*B* подтверждает получение сегментов 0 и 1, разрешает передать со 2-го по 4-й
7	→	<seq = 3, data = m3>	→	У *A* остался 1 буфер
8	→	<seq = 4, data = m4>	→	У *A* осталось 0 буферов, и он должен остановиться
9	→	<seq = 2, data = m2>	→	У *A* истекло время ожидания, и он передает еще раз
10	←	<ack = 4, buf = 0>	←	Все сегменты подтверждены, и он должен остановиться
11	←	<ack = 4, buf = 1>	←	Теперь *A* может послать сегмент 5
12	←	<ack = 4, buf = 2>	←	*B* где-то нашел новый буфер
13	→	<seq = 5, data = m5>	→	У *A* остался 1 буфер
14	→	<seq = 6, data = m6>	→	*A* снова блокирован
15	←	<ack = 6, buf = 0>	←	*A* все еще блокирован
16	• • •	<ack = 6, buf = 4>	←	Потенциальный тупик

Рис. 6.13. Динамическое выделение буферов. Стрелками показано направление передачи. Многоточие (...) означает потерянный сегмент

Проблемы при такой схеме выделения буферов в дейтаграммных сетях могут возникнуть при потере управляющего сегмента — что действительно может произойти. Взгляните на строку 16. Хост *B* выделил хосту *A* дополнительные буферы, но сообщение об этом было потеряно. Вот так неожиданность! Поскольку получение управляющих сегментов не подтверждается и, следовательно, управляющие сегменты не посылаются повторно по тайм-ауту, хост *A* теперь оказался блокированным всерьез и надолго. Для предотвращения такой тупиковой ситуации каждый хост должен периодически посылать управляющий сегмент, содержащий подтверждение и состояние буферов для каждого соединения. Это позволит, в конце концов, выбраться из тупика.

До сих пор мы по умолчанию предполагали, что единственное ограничение, накладываемое на скорость передачи данных, состоит в количестве свободного буферного пространства у получателя. Однако часто это бывает не так. По мере колоссального снижения цен (некогда очень высоких) на микросхемы памяти и винчестеры становится возможным оборудовать хосты таким количеством памяти, что проблема нехватки буферов будет возникать очень редко, если вообще будет возникать, даже если соединение охватывает крупные территории. Конечно же, необходимо, чтобы выбранный размер буфера был достаточно большим; это требование не всегда выполнялось в случае TCP (Zhang и др., 2002).

Если размер буферов перестанет ограничивать максимальный поток, возникнет другое узкое место: пропускная способность сети. Если максимальная скорость обмена кадрами между соседними маршрутизаторами будет *x* кадров в секунду и между двумя хостами имеется *k* непересекающихся путей, то, сколько бы ни было буферов у обоих хостов, они не смогут пересылать друг другу больше, чем *kx* сегментов в секунду. И если отправитель будет передавать с большей скоростью, то сеть окажется перегруженной.

Требуется механизм, который мог бы ограничивать передачу данных со стороны отправителя и основывался не столько на емкости буферов получателя, сколько на пропускной способности сети. В 1975 году Белснес (Belsnes) предложил использовать для управления потоком данных схему скользящего окна, в которой отправитель динамически приводит размер окна в соответствие с пропускной способностью сети. Таким образом, скользящее окно позволяет одновременно реализовать и управление потоком, и контроль перегрузки. Если сеть может обработать *c* сегментов в секунду, а время цикла (включая передачу, распространение, ожидание в очередях, обработку получателем и возврат подтверждения) равно *r*, тогда размер окна отправителя должен быть равен *cr*. При таком размере окна отправитель работает, максимально используя канал. Любое уменьшение производительности сети приведет к его блокировке. Так как пропускная способность сети меняется с течением времени, размер окна должен настраиваться довольно часто, чтобы можно было отслеживать изменения пропускной способности. Как будет показано ниже, в TCP используется похожая схема.

6.2.5. Мультиплексирование

Объединение нескольких разговоров в одном соединении, виртуальном канале и по одной физической линии играет важную роль в нескольких уровнях сетевой архитектуры. Потребность в подобном уплотнении возникает в ряде случаев и на транспортном

уровне. Например, если у хоста имеется только один сетевой адрес, он используется всеми соединениями транспортного уровня. Нужен какой-то способ, с помощью которого можно было бы различать, какому процессу следует передать входящий сегмент. Такая ситуация, называемая **мультиплексированием**, показана на рис. 6.14, *а*. На рисунке четыре различных соединения транспортного уровня используют одно сетевое соединение (например, один IP-адрес) с удаленным хостом.

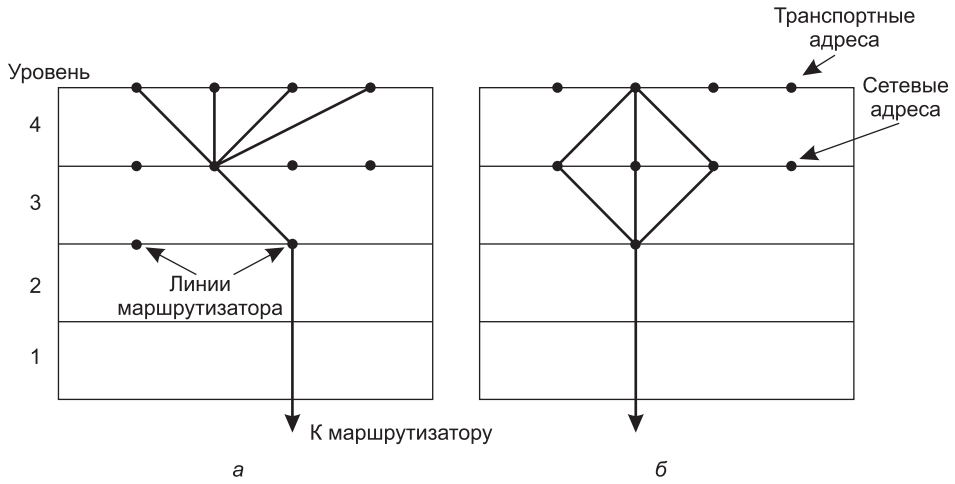


Рис. 6.14. Мультиплексирование: *а* — прямое; *б* — обратное мультиплексирование

Уплотнение может играть важную роль на транспортном уровне и по другой причине. Предположим, например, что хост может использовать несколько различных сетевых путей. Если пользователю требуется большая пропускная способность или большая надежность, нежели может предоставить один сетевой путь, то можно попробовать решить эту проблему путем открытия соединения, распределяющего трафик между путями, используя их поочередно, как показано на рис. 6.14, *б*. Такой метод называется **обратным мультиплексированием**. При k открытых сетевых соединениях эффективная пропускная способность может увеличиться в k раз. Примером обратного мультиплексирования является **SCTP (Stream Control Transmission Protocol — протокол передачи с управлением потоками)**, позволяющий устанавливать соединение с множественными сетевыми интерфейсами. TCP, наоборот, использует отдельный сокет. Обратное мультиплексирование используется и на канальном уровне — при этом несколько медленных каналов связи объединяются в один, работающий гораздо быстрее.

6.2.6. Восстановление после сбоев

Если хосты и маршрутизаторы подвержены сбоям или соединения длятся достаточно долго (например, при загрузке мультимедиа или программного обеспечения), вопрос восстановления после сбоев становится особенно актуальным. Если транспортная подсистема целиком помещается в хостах, восстановление после отказов сети и марш-

рутизаторов не вызывает затруднений. Транспортные подсистемы постоянно ожидают потери сегментов и знают, как с этим бороться: для этого выполняются повторные передачи.

Более серьезную проблему представляет восстановление после сбоя хоста. В частности, для клиентов может быть желательной возможность продолжать работу после отказа и быстрой перезагрузки сервера. Чтобы пояснить, в чем тут сложность, предположим, что один хост — клиент — посылает длинный файл другому хосту — файловому серверу — при помощи простого протокола с остановкой и ожиданием подтверждения. Транспортный уровень сервера просто передает приходящие сегменты один за другим пользователю транспортного уровня. Получив половину файла, сервер сбрасывается и перезагружается, после чего все его таблицы переинициализируются, поэтому он уже не помнит, что с ним было раньше.

Пытаясь восстановить предыдущее состояние, сервер может разослать широковещательный сегмент всем хостам, объявляя им, что он только что перезагрузился, и прося своих клиентов сообщить ему о состоянии всех открытых соединений. Каждый клиент может находиться в одном из двух состояний: один неподтвержденный сегмент (состояние *S1*) или ни одного неподтвержденного сегмента (состояние *S0*). Этой информации клиенту должно быть достаточно, чтобы решить, передавать ему повторно последний сегмент или нет.

На первый взгляд, здесь все очевидно: узнав о перезапуске сервера, клиент должен передать повторно последний неподтвержденный сегмент. То есть повторная передача требуется тогда и только тогда, когда клиент находится в состоянии *S1*. Однако при более детальном рассмотрении оказывается, что все не так просто, как мы наивно предположили. Рассмотрим, например, ситуацию, в которой транспортная подсистема сервера сначала посылает подтверждение, а уже затем передает пакет прикладному процессу. Запись сегмента в выходной поток и отправка подтверждения являются двумя различными неделимыми событиями, которые не могут быть выполнены одновременно. Если сбой произойдет после отправки подтверждения, но до того как выполнена запись, клиент получит подтверждение, а при получении объявления о перезапуске сервера окажется в состоянии *S0*. Таким образом, клиент не станет передавать сегмент повторно, так как будет считать, что сегмент уже получен, что в конечном итоге приведет к отсутствию сегмента.

В этом месте вы, должно быть, подумаете: «А что если поменять местами последовательность действий, выполняемых транспортной подсистемой сервера, чтобы сначала осуществлялась запись, а потом высылалось подтверждение?» Представим, что запись сделана, но сбой произошел до отправки подтверждения. В этом случае клиент окажется в состоянии *S1* и поэтому передаст сегмент повторно, и мы получим дубликат сегмента в выходном потоке.

Таким образом, независимо от того, как запрограммированы клиент и сервер, всегда могут быть ситуации, в которых протокол не сможет правильно восстановиться. Сервер можно запрограммировать двумя способами: так, чтобы он сначала передавал подтверждение, или так, чтобы сначала записывал сегмент. Клиент может быть запрограммирован одним из четырех способов: всегда передавать повторно последний сегмент, никогда не передавать повторно последний сегмент, передавать повторно сегмент только в состоянии *S0* и передавать повторно сегмент только в состоянии *S1*.

Таким образом, получаем восемь комбинаций, но как будет показано, для каждой комбинации имеется набор событий, заставляющий протокол ошибиться.

На сервере могут происходить три события: отправка подтверждения (*A*), запись сегмента в выходной процесс (*W*) и сбой (*C*). Они могут произойти в виде шести возможных последовательностей: *AC*(*W*), *AWC*, *C*(*AW*), *C*(*WA*), *WAC* и *WC*(*A*), где скобки означают, что после события *C* событие *A* или *B* может и не произойти (то есть уж сломался, так сломался). На рис. 6.15 показаны все восемь комбинаций стратегий сервера и клиента, каждая со своими последовательностями событий. Обратите внимание, что для каждой комбинации существует последовательность событий, приводящая к ошибке протокола. Например, если клиент всегда передает повторно неподтвержденный сегмент, событие *AWC* приведет к появлению неопознанного дубликата, хотя при двух других последовательностях событий протокол будет работать правильно.

Стратегия, используемая получающим хостом

Стратегия, используемая передающим хостом	Сначала подтверждение, потом запись			Сначала запись, потом подтверждение		
	AC(*W*)	*AWC*	*C*(*AW*)	*C*(*WA*)	*WAC*	*WC*(*A*)
Всегда повторять передачу	OK	DUP	OK	OK	DUP	DUP
Никогда не повторять передачу	LOST	OK	LOST	LOST	OK	OK
Повторять передачу в *S*0	OK	DUP	LOST	LOST	DUP	OK
Повторять передачу в *S*1	LOST	OK	OK	OK	OK	DUP

OK = Протокол работает корректно
DUP = Протокол формирует дубликат сообщения
LOST = Протокол теряет сообщение

Рис. 6.15. Различные комбинации стратегий сервера и клиента

Усложнение протокола не помогает. Даже если клиент и сервер обменяются несколькими сегментами, прежде чем сервер попытается записать полученный пакет, так что клиент будет точно знать, что происходит на сервере, у него нет возможности определить, когда произошел сбой на сервере: до или после записи. Отсюда следует неизбежный вывод: при жестком условии отсутствия одновременных событий — это значит, что отдельные события происходят одно за другим, а не одновременно — невозможно сделать отказ и восстановление хоста прозрачными для более высоких уровней.

В более общем виде это может быть сформулировано следующим образом: восстановление от сбоя уровня *N* может быть осуществлено только уровнем *N* + 1 и только при условии, что на более высоком уровне сохраняется информация о процессе, достаточная для восстановления его прежнего состояния. Это согласуется с приведенным выше утверждением о том, что транспортный уровень может обеспечить восстановление от сбоя на сетевом уровне, если каждая сторона соединения отслеживает свое текущее состояние.

Эта проблема подводит нас к вопросу о значении так называемого сквозного подтверждения. В принципе, транспортный протокол является сквозным, а не цепным, как более низкие уровни. Теперь рассмотрим случай обращения пользователя к удаленной базе данных. Предположим, что удаленная транспортная подсистема запрограммирована сначала передавать сегмент вышестоящему уровню, а затем отправлять подтверждение. Даже в этом случае получение подтверждения машиной пользователя не означает, что удаленный хост успел обновить базу данных. Настоящее сквозное подтверждение, получение которого означает, что работа была сделана, и, соответственно, отсутствие которого означает обратное, вероятно, невозможно. Более подробно этот вопрос обсуждается в (Saltzer и др., 1984).

6.3. Контроль перегрузки

Если транспортные подсистемы нескольких машин будут отправлять в сеть слишком много пакетов с большой скоростью, сеть окажется перегруженной, и производительность резко снизится, что приведет к задержке и потере пакетов. Контроль перегрузки, направленный на борьбу с такими ситуациями, требует совместной работы сетевого и транспортного уровней. Так как перегрузки происходят на маршрутизаторах, их обнаружением занимается сетевой уровень. Однако в конечном итоге причиной перегрузки является трафик, переданный транспортным уровнем в сеть. Поэтому единственный эффективный способ контролировать перегрузки состоит в более медленной передаче пакетов транспортными протоколами.

В главе 5 мы говорили о механизмах контроля перегрузки на сетевом уровне. В этом разделе мы расскажем о другой части этого процесса — механизмах контроля перегрузки на транспортном уровне. После обсуждения основных задач контроля перегрузки мы перейдем к описанию методов, позволяющих хостам регулировать скорость передачи пакетов в сеть. Контроль перегрузки в сети Интернет опирается во многом на транспортный уровень; для этого в TCP и другие протоколы встроены специальные алгоритмы.

6.3.1. Выделение требуемой пропускной способности

Перед тем как перейти к описанию методов регулирования трафика, необходимо понять, чего мы хотим от алгоритма контроля перегрузки. То есть мы должны определить, какое состояние сети должен поддерживать такой алгоритм. В его задачи входит не только предотвращение перегрузок: он должен правильно распределять пропускную способность между транспортными подсистемами, работающими в сети. Правильное распределение пропускной способности должно обеспечивать сети хорошую производительность (так как при этом сеть должна работать с использованием всей доступной мощности без перегрузок), следовать принципу равноправия конкурирующих транспортных подсистем и быстро отслеживать изменения в запросах на выделение ресурсов. Мы рассмотрим каждый из этих критериев отдельно.

Эффективность и мощность

Эффективное распределение пропускной способности между транспортными подсистемами должно использовать все возможности сети. Однако это не значит, что в случае канала со скоростью 100 Мбит/с каждая из пяти транспортных подсистем должна получить по 20 Мбит/с. Для хорошей производительности им необходимо выделить меньшую мощность. Причина в том, что трафик часто бывает неравномерным. В разделе 5.3 мы определили **эффективную пропускную способность** (goodput, скорость, с которой полезные пакеты прибывают к получателю) как функцию нагрузки на сеть. Эта кривая и соответствующая ей кривая задержки (как функция нагрузки) приведены на рис. 6.16.

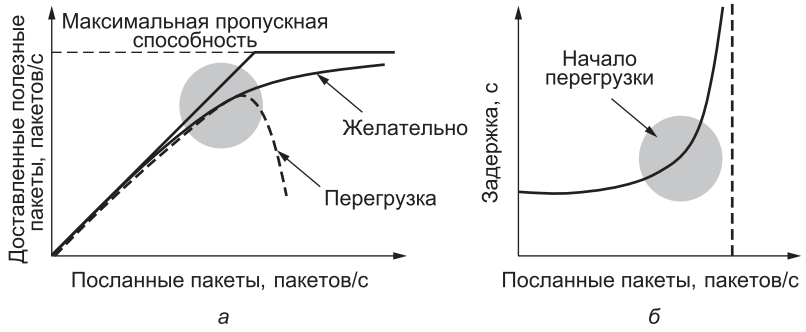


Рис. 6.16. Эффективная пропускная способность (*а*); задержка как функции нагрузки (*б*)

Сначала с ростом нагрузки на сеть эффективная пропускная способность увеличивается с постоянной скоростью (рис. 6.16, *а*), но когда значение нагрузки приближается к значению пропускной способности, рост эффективной пропускной способности замедляется. Этот спад необходим для того, чтобы предотвратить переполнение буферов сети и потерю данных в случае всплесков трафика. Если транспортный протокол выполняет повторную передачу задерживающихся, но не потерянных пакетов, может произойти отказ сети из-за перегрузки. В таком случае отправители продолжают передавать все больше и больше пакетов, а пользы от этого все меньше и меньше.

График задержки пакетов приведен на рис. 6.16, *б*. Сначала задержка является постоянной и соответствует задержке распространения в сети. Когда значение нагрузки приближается к значению пропускной способности, задержка возрастает, причем сначала медленно, а затем все быстрее и быстрее. Это происходит опять же из-за всплесков трафика, которые возрастают при высокой нагрузке. В действительности задержка не может уходить в бесконечность, если только модель не предполагает использование бесконечных буферов. Вместо этого пакеты будут теряться при переполнении буферов.

Для достижения хорошей эффективной пропускной способности и задержки производительность должна начать снижаться в момент возникновения перегрузки. Логично, что для достижения лучшей производительности сети можно выделять пропускную способность до тех пор, пока задержка не пойдет резко вверх. Эта точка находится ниже пропускной способности сети. Чтобы ее определить, Кляйнрок (Klein-

rock) в 1979 году предложил ввести метрику **мощность (power)**, которая вычисляется по формуле:

$$\text{мощность} = \text{нагрузка}/\text{задержка}.$$

Пока задержка достаточно мала и почти постоянна, мощность растет с ростом нагрузки на сеть; при резком повышении задержки она достигает максимума и резко снижается. Нагрузка, при которой мощность достигает максимума, и является наиболее эффективной нагрузкой, которую транспортная подсистема может передавать в сеть.

Равнодоступность по максиминному критерию

Пока мы еще не говорили о том, как распределять пропускную способность между несколькими отправителями. Кажется, что ответ очень прост: можно разделить пропускную способность между ними поровну. Однако на самом деле этот вопрос требует некоторых уточнений.

Во-первых, зададимся вопросом: какое отношение эта проблема имеет к контролю нагрузки? В конце концов, если сеть предоставляет отправителю некоторое количество пропускной способности, он должен просто использовать то, что у него есть. Однако на практике сети часто не резервируют строго ограниченное количество пропускной способности для потока или соединения. Конечно, для некоторых потоков они могут это делать (если есть поддержка качества обслуживания), но, как правило, многие соединения стремятся использовать максимум доступной пропускной способности; также возможен вариант, при котором сеть выделяет ресурсы для совместного использования группой соединений. К примеру, дифференцированное обслуживание IETF разделяет трафик на два класса, и соединения конкурируют друг с другом в пределах каждого из этих классов. Часто все соединения одного IP-маршрутизатора используют общую пропускную способность. В таких случаях распределение пропускной способности между конкурирующими соединениями должно осуществляться за счет механизмов контроля нагрузки.

Во-вторых, не совсем ясно, что такое равноправие потоков в сети. Если N потоков используют один канал, то решение довольно просто: каждому из них выделяется $1/N$ пропускной способности (в случае импульсного трафика эта величина по соображениям эффективности будет немного меньше). Но что если потоки используют разные, но пересекающиеся пути? К примеру, если один поток проходит по трем каналам, а остальные потоки — по одному? Очевидно, что поток, проходящий по трем каналам, потребляет больше сетевых ресурсов. Поэтому в некотором смысле справедливее было бы выделить ему меньше пропускной способности, чем остальным потокам (проходящим по одному каналу). Кроме того, неплохо было бы обеспечить возможность добавления новых одноканальных потоков за счет уменьшения пропускной способности трехканального. Как показывает этот пример, между равноправием и эффективностью всегда существуют противоречия.

Однако здесь мы будем использовать такое определение равноправия, в котором оно не зависит от длины сетевого пути. Даже в такой простой модели разделение пропускной способности поровну между соединениями является достаточно сложной задачей, так как разные соединения будут использовать разные пути, а эти пути, в свою

очередь, будут обладать разной пропускной способностью. Это может привести к тому, что в нисходящем канале какого-то из потоков возникнет узкое место, и такой поток получит меньший «кусок» восходящего канала, чем другие. В таком случае уменьшение пропускной способности, выделяемой другим потокам, лишь замедлит их работу, но не исправит ситуацию с «застрявшим» потоком.

Часто в сетях удобнее использовать форму равноправия, которая называется **равнодоступностью по максиминному критерию**. Это значит, что увеличение пропускной способности одного потока невозможно без уменьшения пропускной способности какого-либо другого потока с меньшей или равной пропускной способностью. Иными словами, от увеличения пропускной способности потока страдают менее «обеспеченные» потоки.

Рассмотрим пример. На рис. 6.17 показано распределение пропускной способности по максиминному критерию в сети с четырьмя потоками A, B, C и D. Все каналы, соединяющие маршрутизаторы, обладают одинаковой пропускной способностью, принятой за 1 (хотя на практике это обычно не так). На пропускную способность канала, расположенного слева внизу (между маршрутизаторами R4 и R5), претендуют три потока. Поэтому каждый из них получает по 1/3. Оставшийся поток, A, конкурирует с B на участке от R2 до R3. Поскольку для B уже выделена 1/3 пропускной способности канала, A получает оставшееся количество, 2/3. Как видно из рисунка, на остальных каналах часть пропускной способности не используется. Но эти ресурсы нельзя отдать какому-либо потоку, не снизив пропускную способность другого, более медленного потока. К примеру, если на участке от R2 до R3 выделить потоку B больше пропускной способности, то потоку A достанется меньше. Это логично, поскольку на данный момент у A пропускной способности больше. Но для того чтобы обеспечить бóльшую пропускную способность для потока B, придется снизить пропускную способность C или D (или обоих), и тогда его (или их) пропускная способность станет меньше пропускной способности B. Таким образом, данное распределение соответствует максиминному критерию.

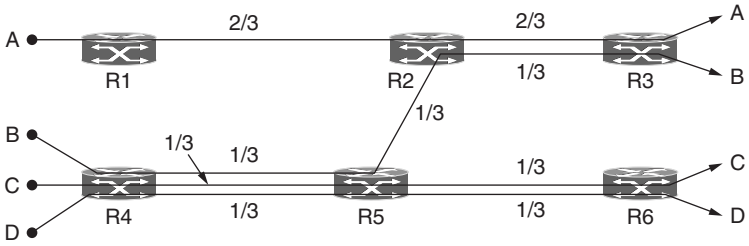


Рис. 6.17. Распределение пропускной способности по максиминному критерию для четырех потоков

Распределение пропускной способности по максиминному критерию можно вычислить на основе данных обо всей сети. Чтобы наглядно это себе представить, предположим, что скорость всех потоков медленно возрастает, начиная от 0. Как только у одного из потоков возникает узкое место, его скорость перестает расти. Далее скорость остальных потоков продолжает увеличиваться (при этом доступная пропускная способность делится поровну), пока каждый из них не дойдет до своего узкого места.

В-третьих, не очень понятно, на каком уровне рассматривать равноправие. Сеть может обеспечивать равноправие потоков на уровне соединений, соединений между парой хостов или всех соединений одного хоста. Эту проблему мы обсуждали, когда говорили о взвешенном справедливом обслуживании (см. раздел 5.4): выяснилось, что с каждым из этих вариантов связаны определенные трудности. К примеру, если считать равноправными все хосты, активно работающий сервер будет получать не больше ресурсов, чем обычный мобильный телефон; если же равноправными считать все соединения, хостам будет выгодно открывать дополнительные новые соединения. Поскольку на этот вопрос нет однозначного ответа, равноправие часто рассматривается применительно к соединениям, однако такое равноправие совсем не обязано быть точным. В первую очередь важно, чтобы ни одно соединение не осталось без пропускной способности. На самом деле, TCP позволяет открывать множественные соединения, что вызывает более жесткую конкуренцию. Эта тактика подходит для приложений, особенно требовательных к возможностям сети. Например, ее использует BitTorrent для одноранговой совместного использования файлов.

Сходимость

Последний критерий состоит в том, что алгоритм контроля нагрузки должен быстро сходиться к справедливому и эффективному распределению пропускной способности. При обсуждении выбора рабочего режима мы исходили из того, что сетевая среда является статической. Однако на практике соединения то появляются, то исчезают, а пропускная способность, требующаяся каждому отдельному соединению, изменяется со временем — так, например, пользователь может долго просматривать веб-страницу, а затем вдруг начать загрузку большого видеофайла.

Такое непостоянство требований к сети приводит к тому, что идеальный рабочий режим постоянно меняется. Хороший алгоритм контроля нагрузки должен быстро сходиться к идеальному рабочему режиму и следить за его изменением с течением времени. Если алгоритм будет сходиться слишком медленно, он не будет успевать за изменениями рабочего режима. Если алгоритм будет нестабильным, в некоторых случаях он не будет сходиться к нужной точке или будет долго колебаться вокруг нее.

На рис. 6.18 показан пример распределения пропускной способности, которая изменяется со временем и быстро сходится. Изначально поток 1 получает всю пропускную способность. Через секунду начинает работать поток 2. Ему тоже нужна пропускная способность. Поэтому распределение быстро меняется, так чтобы оба потока получали по 1/2. Еще через три секунды появляется третий поток. Но он использует только 20 % пропускной способности — меньше, чем то, что ему полагается исходя из идеи равноправия (1/3). Потоки 1 и 2 быстро реагируют на изменение ситуации, и каждый из них получает 40 %. В момент времени 9 с второй поток отключается, а третий продолжает работать без изменений. Поток 1 быстро занимает 80 % пропускной способности. В каждый момент времени суммарная пропускная способность приблизительно равна 100 %, то есть возможности сети используются полностью; при этом все конкурирующие потоки оказываются в равных условиях (но не используют больше пропускной способности, чем им нужно).

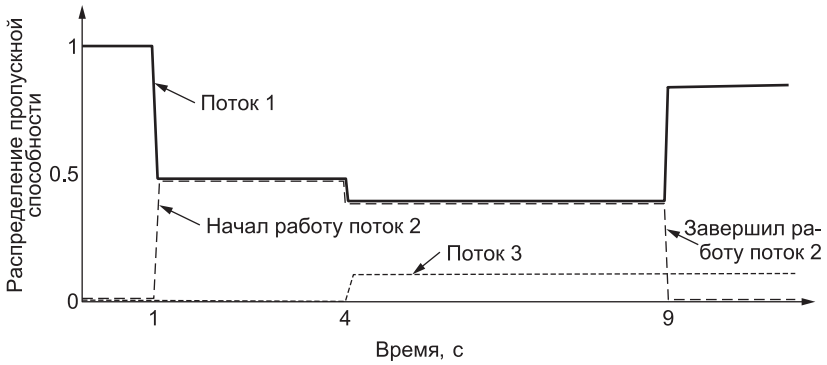


Рис. 6.18. Изменение распределения пропускной способности с течением времени

6.3.2. Регулирование скорости отправки

Теперь самое время перейти к основному вопросу: как можно регулировать скорость отправки, чтобы получить необходимую пропускную способность? Скорость отправки может зависеть от двух факторов. Первый из них — управление потоком данных, которое требуется, если буфер получателя обладает недостаточной емкостью. Второй фактор — перегрузка, которую необходимо учитывать при недостаточной пропускной способности сети. На рис. 6.19 эта проблема проиллюстрирована на примере водопровода. На рис. 6.19, *а* мы видим толстую трубу, ведущую к получателю с небольшой емкостью. Это тот случай, когда ограничительным фактором является управление потоком данных. До тех пор пока отправитель не посылает воды больше, чем может поместиться в ведро, вода не будет проливаться. На рис. 6.29, *б* ограничительным фактором является не емкость ведра, а пропускная способность сети. Если из крана в воронку вода будет литься слишком быстро, то уровень воды в воронке начнет подниматься, и, в конце концов, часть воды может перелиться через край воронки.

Отправителю эти примеры могут показаться похожими, так как в обоих случаях результатом слишком быстрой передачи данных является потеря пакетов. Но поскольку эти ситуации происходят по разным причинам, они требуют разных решений. Об одном из возможных решений — управлении потоком данных с помощью окна переменного размера — мы уже говорили. Теперь мы рассмотрим другое решение, связанное с управлением перегрузкой. Поскольку на практике может произойти любая из этих проблем, транспортный протокол должен включать реализацию обоих решений.

То, как транспортный протокол должен регулировать скорость отправки, зависит от формы обратной связи, использующейся в сети. Разные сетевые уровни могут использовать обратную связь разных типов: явную и неявную, точную и неточную.

Пример явной точной обратной связи — ситуация, когда маршрутизаторы сообщают источникам свою допустимую скорость отправки. Так работает, к примеру, XCP (eXplicit Congestion Protocol) (Katabi и др., 2002). Явная и неточная схема — использование ECN (Explicit Congestion Notification, явное уведомление о насыщении) с TCP. В этом случае маршрутизаторы специальным образом маркируют пакеты, страдающие от перегрузки, и таким образом сообщают отправителю, что ему следует

уменьшить скорость передачи данных, не указывая при этом, насколько сильно ее нужно уменьшить.

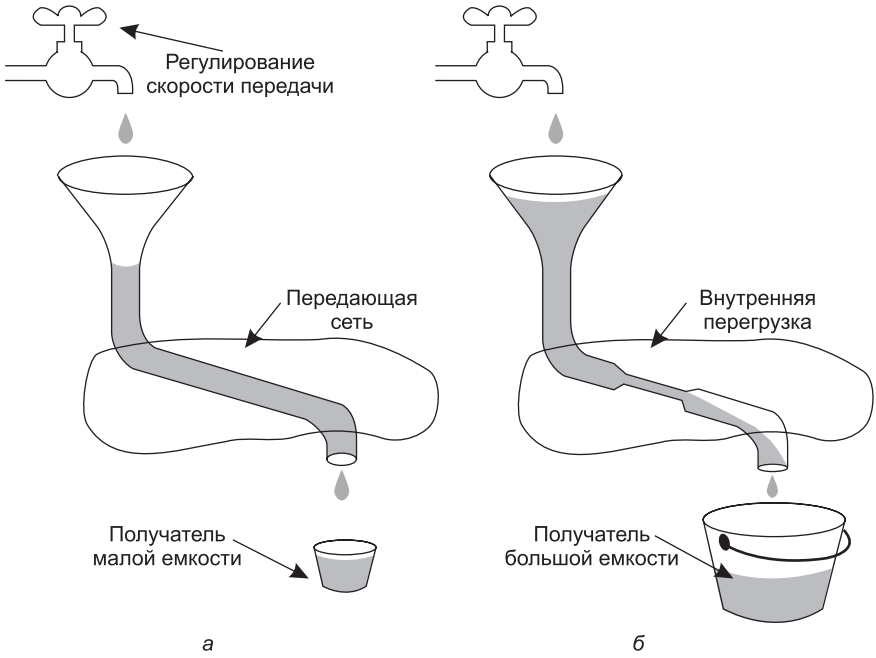


Рис. 6.19. Быстрая сеть и получатель малой емкости (*а*); медленная сеть и получатель большой емкости (*б*)

В других схемах явный сигнал отсутствует. FAST TCP измеряет круговую задержку и использует ее в качестве параметра для контроля перегрузки (Wei и др., 2006). Наконец, метод контроля перегрузки, наиболее распространенный в современном Интернете, использует TCP и маршрутизаторы с отбрасыванием конца очереди или маршрутизаторы со случайным ранним обнаружением перегрузки. Показателем перегрузки сети в нем является число потерянных пакетов. Существует множество вариантов этого вида TCP — в частности, CUBIC TCP, использующийся в системе Linux (На и др., 2008). Возможны также комбинации нескольких методов. К примеру, Windows включает Compound TCP (составной TCP), в котором показателями насыщения являются и круговая задержка, и число потерянных пакетов (Tan и др., 2006). Эти схемы показаны в табл. 6.3.

Получив явный и точный сигнал, транспортная подсистема может установить скорость отправки пакетов в соответствии с новым рабочим режимом. К примеру, если XCP сообщает отправителям рекомендуемую скорость, они могут просто использовать эту скорость. Но в остальных случаях транспортные подсистемы вынуждены действовать наугад. В отсутствие сигнала о насыщении отправители должны увеличивать скорость отправки, а при наличии такого сигнала — уменьшать. За увеличение и уменьшение скорости отправки отвечает **закон управления (control law)**. Выбор такого закона оказывает решающее влияние на производительность.

Таблица 6.3. Сигналы некоторых протоколов контроля насыщения

Протокол	Сигнал	Явный?	Точный?
XCP	Скорость, которую следует использовать	Да	Да
TCP с ECN	Предупреждение о перегрузке	Да	Нет
FAST TCP	Сквозная задержка	Нет	Да
Compound TCP	Потеря пакетов и сквозная задержка	Нет	Да
CUBIC TCP	Потеря пакетов	Нет	Нет
TCP	Потеря пакетов	Нет	Нет

Рассматривая бинарный признак насыщения, Chiu и Jain (1989) пришли к выводу, что закон управления **AIMD (Additive Increase Multiplicative Decrease — аддитивное увеличение мультипликативное уменьшение)** позволяет эффективно вычислять рабочий режим с учетом идеи равнодоступности. Чтобы это показать, они привели простой наглядный пример, в котором два соединения соперничают друг с другом за пропускную способность общего канала. На рис. 6.20 пропускная способность, выделяемая пользователю 1, располагается на оси X, а пропускная способность, выделяемая пользователю 2, — на оси Y. При справедливом распределении оба пользователя получают одинаковое количество пропускной способности. Таким распределениям соответствует прямая равноправия, изображенная пунктиром. Когда суммарная пропускная способность, выделяемая всем пользователям, составляет 100 % (то есть равна пропускной способности канала), распределение является эффективным. Эффективные распределения располагаются на прямой эффективности. Когда суммарная пропускная способность пересекает эту прямую, оба пользователя получают сигнал о насыщении. Точка пересечения этих прямых соответствует оптимальному рабочему режиму, при котором пользователи получают одинаковое количество пропускной способности и используется вся пропускная способность сети.

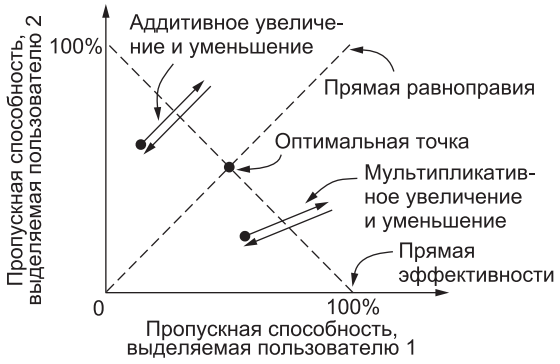


Рис. 6.20. Аддитивное и мультипликативное регулирование пропускной способности

Пусть изначально пользователю 1 и пользователю 2 выделено некоторое количество пропускной способности. Посмотрим, что произойдет, если оба пользователя начнут аддитивно увеличивать свою пропускную способность. К примеру, они могут

повышать скорость отправки пакетов на 1 Мбит/с каждую секунду. В результате рабочий режим пересечет прямую эффективности, и оба пользователя получат сигнал о насыщении сети. В этот момент им придется снизить используемую пропускную способность. Однако аддитивное уменьшение попросту приведет к колебаниям значений вдоль аддитивной прямой (см. рис. 6.20). В результате рабочий режим будет оставаться близким к эффективному, однако он не обязательно будет справедливым.

Рассмотрим аналогичную ситуацию, в которой оба пользователя увеличивают свою пропускную способность мультипликативно до тех пор, пока не поступит сигнал о насыщении сети. Например, они могут каждую секунду повышать скорость отправки пакетов на 10 %. Если после получения сигнала о насыщении пользователи будут уменьшать пропускную способность мультипликативно, рабочий режим будет колебаться вдоль мультипликативной прямой (см. рис. 6.20). Наклон мультипликативной прямой отличается от наклона аддитивной прямой. (Мультипликативная прямая проходит через начало координат, а аддитивная имеет наклон 45°.) Однако это ничего нам не дает. Ни в том, ни в другом случае оптимальная скорость отправки не будет достигнута.

Теперь рассмотрим другую ситуацию. Предположим, что пользователи аддитивно увеличивают пропускную способность, а после получения сигнала о насыщении начинают мультипликативно ее уменьшать. Такой метод называется законом управления AIMD (рис. 6.21). Оказывается, что в результате таких преобразований рабочий режим сходится к оптимальной точке, в которой распределение пропускной способности является одновременно справедливым и эффективным, причем это происходит независимо от начальной точки. Поэтому AIMD широко применяется на практике. При обратном варианте — мультипликативном увеличении и аддитивном уменьшении — рабочий режим будет расходиться от оптимальной точки.

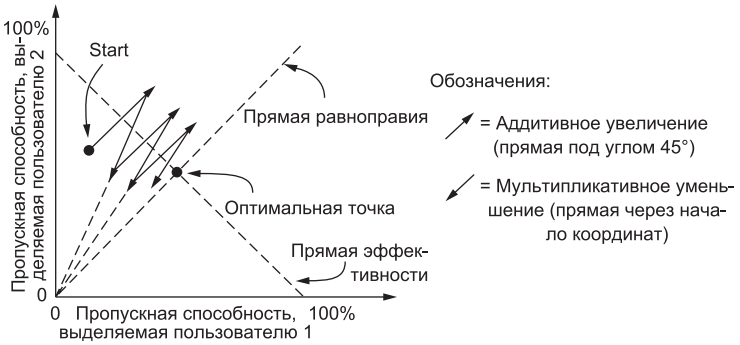


Рис. 6.21. Закон управления AIMD

Протокол TCP использует закон управления AIMD не только по этой причине, но и по соображениям стабильности (поскольку перейти в состояние перегрузки гораздо проще, чем из него выйти, политика увеличения должна быть мягкой, а уменьшения — агрессивной). Однако в результате распределение пропускной способности является не совсем справедливым. Дело в том, что размер окна задается исходя из круговой задержки, индивидуальной для каждого соединения. Поэтому соединения с ближними

хостами получают большую пропускную способность, чем соединения с удаленными хостами (при прочих равных условиях).

В разделе 6.5 мы подробно поговорим о том, как с помощью закона управления AIMD в TCP осуществляется регулировка скорости отправки пакетов и контроль перегрузки. Эта задача гораздо сложнее, чем может показаться. Проблемы возникают из-за того, что значения скорости измеряются через определенный интервал времени, а трафик, как правило, бывает импульсным. На практике вместо скорости отправки пакетов часто задается размер скользящего окна. Такая стратегия используется и в TCP. Если размер окна равен *W*, а круговая задержка — *RTT*, то эквивалентная скорость равна *W*/*RTT*. Это удобно, поскольку управление потоком данных также основано на регулировке размера окна. Кроме того, такой метод обладает важным преимуществом: если подтверждения о доставке пакетов перестанут приходить, через время *RTT* скорость отправки уменьшится.

Наконец, иногда в одной сети используются разные транспортные протоколы. Что произойдет, если эти протоколы будут одновременно пытаться контролировать перегрузку с помощью разных законов управления? Ответ очевиден: распределение пропускной способности не будет справедливым. Поскольку самой распространенной формой контроля перегрузки в сети Интернет является TCP, новые транспортные протоколы настоятельно рекомендуется разрабатывать так, чтобы при совместной работе с TCP выполнялось условие справедливого распределения ресурсов. Использовавшиеся ранее протоколы потокового вещания не соответствовали этому требованию, существенно уменьшая пропускную способность TCP. В результате стала развиваться идея дружественного к TCP контроля перегрузки, при котором транспортные протоколы TCP и не-TCP могут работать вместе, не мешая друг другу (Floyd и др., 2000).

6.3.3. Проблемы беспроводного соединения

Транспортные протоколы, включающие контроль перегрузки (такие как TCP), не должны зависеть от используемой сети и устройства канального уровня. В теории это звучит хорошо, но на практике в случае беспроводных сетей возникают определенные проблемы. Главная из них заключается в том, что во многих протоколах (в частности, в TCP) сигналом перегрузки является потеря пакетов. Но в беспроводных сетях из-за ошибок передачи потеря пакетов происходит постоянно.

Если протокол использует закон управления AIMD, большая пропускная способность требует минимальной потери пакетов. Исследования Padhye и др. (1998) показали, что пропускная способность растет обратно пропорционально квадратному корню из скорости потери пакетов. На практике это означает, что скорость потери пакетов для быстрых TCP-соединений очень мала; в среднем этот показатель составляет 1 %, а при значении 10 % соединение перестает работать. Однако для беспроводных сетей, таких как локальная сеть 802.11, вполне обычными являются значения скорости потери кадров порядка 10 % и выше. Если этого не учитывать, схемы контроля перегрузки, использующие в качестве сигнала насыщения потерю пакетов, попросту «задушат» беспроводные соединения, крайне ограничив их скорость.

Чтобы такой алгоритм контроля перегрузки работал правильно, он должен учитывать только те потери пакетов, которые произошли из-за недостатка пропускной

способности, а не из-за ошибок передачи. Одно из возможных решений — скрыть потерю данных с помощью их повторной передачи по беспроводным каналам. К примеру, в 802.11 для доставки каждого кадра используется протокол с остановкой и ожиданием подтверждения: передача кадра повторяется несколько раз, и только после этого информация о потере пакета поступает на более высокий уровень. В большинстве случаев пакеты доставляются на место назначения, несмотря на ошибки передачи (о которых более высокие уровни ничего не знают).

На рис. 6.22 показан путь по проводному и беспроводному каналу, для которого используется эта стратегия. Здесь следует обратить внимание на два аспекта. Во-первых, отправитель не знает о том, что путь содержит беспроводной канал, так как все, что он видит, — это проводной канал, к которому он непосредственно подсоединен. В сети Интернет пути являются гетерогенными, однако не существует универсального способа, позволяющего отправителю определить, из каких каналов состоит путь. Это осложняет проблему контроля перегрузки, поскольку использовать разные протоколы для проводных и беспроводных каналов — очень непростая задача.

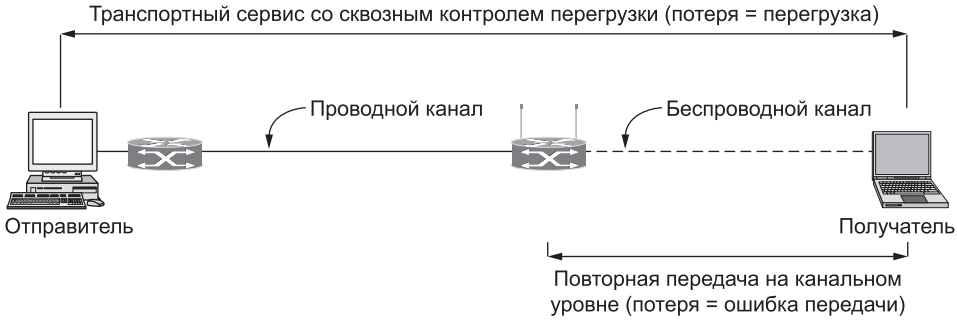


Рис. 6.22. Контроль перегрузки для пути, содержащего беспроводной канал

Второй аспект — своего рода загадка. Рисунок иллюстрирует два механизма, основанных на данных о потере пакетов: повторную передачу кадра на канальном уровне и контроль перегрузки на транспортном уровне. Загадка в том, как эти два механизма могут сосуществовать. Ведь потеря пакетов должна приводить в действие только один из механизмов, так как это либо ошибка передачи данных, либо сигнал о перегрузке — но не и то и другое одновременно. Если же начинают работать оба механизма (то есть происходит и повторная передача кадра, и уменьшение скорости передачи), мы возвращаемся к нашей первоначальной проблеме: схема работает слишком медленно для беспроводных каналов. Попробуйте немного поразмышлять над этим вопросом и понять, можете ли вы на него ответить.

Решение загадки заключается в том, что эти механизмы работают в разных временных масштабах. В беспроводных сетях, таких как 802.11, повторные передачи канального уровня происходят через промежутки времени порядка нескольких микросекунд или миллисекунд. Таймеры транспортных протоколов, следящие за потерей пакетов, срабатывают раз в несколько миллисекунд или секунд. Благодаря разнице в три порядка беспроводные каналы успевают обнаружить потерю кадров и решить проблему путем их повторной передачи задолго до того, как об этом узнает транспортная подсистема.

Такая стратегия необходима для нормальной работы большинства транспортных протоколов с большинством беспроводных каналов. Однако существуют ситуации, для которых это решение не подходит. Некоторые беспроводные каналы — например, спутниковые — обладают большой круговой задержкой. В таких случаях требуются другие методы, позволяющие скрыть потерю пакетов, — такие, как **FEC (Forward Error Correction — упреждающая коррекция ошибок)**; или же транспортный протокол должен использовать сигналы без потерь для контроля перегрузки.

Вторая проблема, связанная с контролем перегрузки в беспроводных каналах, — переменная пропускная способность. Это значит, что пропускная способность беспроводного канала меняется со временем, причем иногда скачкообразно, при перемещении узлов и изменении соотношения сигнал/шум; в проводных каналах пропускная способность является постоянной. В результате транспортный протокол вынужден адаптироваться к изменениям пропускной способности беспроводного канала. В противном случае либо произойдет перегрузка сети, либо мощность канала будет использоваться не полностью.

Одно из возможных решений — просто не задумываться об этом. Алгоритмы контроля перегрузки должны и так хорошо работать в случае добавления новых пользователей или изменения скоростей отправки пакетов. Несмотря на то что пропускная способность проводного канала является фиксированной, для каждого отдельного пользователя меняющееся поведение других пользователей выглядит как колебание доступной пропускной способности. Поэтому для пути, содержащего беспроводной канал 802.11, можно просто использовать протокол TCP и добиться при этом хорошей производительности.

Однако при высокой нестабильности беспроводного канала транспортные протоколы, разработанные для проводных соединений, могут не успевать следить за изменениями, в результате чего производительность существенно снижается. В таком случае требуется специальный транспортный протокол, разработанный для беспроводных каналов. Особый интерес представляет разработка такого протокола для беспроводной ячеистой сети с множественными пересекающимися беспроводными каналами и мобильными маршрутизаторами и, конечно же, с большой потерей пакетов. Исследования в этой области продолжаются. Пример транспортного протокола для беспроводных каналов можно найти в книге (Li и др., 2009).

6.4. Транспортные протоколы Интернета: UDP

В Интернете нашли применение два основных протокола транспортного уровня, один из которых требует установления соединения, другой — нет. Эти протоколы дополняют друг друга. Протоколом без установления соединения является UDP. Он не делает практически ничего, кроме отправки пакетов между приложениями, позволяя последним надстраивать свои собственные протоколы. TCP, напротив, является протоколом с установлением соединения. В его задачи входит практически все. Он устанавливает соединения и обеспечивает надежность сети, выполняя повторную передачу данных, а также осуществляет управление потоком данных и контроль перегрузки — и все это от лица приложений, которые его используют.

В следующих разделах мы изучим UDP и TCP. Так как UDP проще, его мы изучим первым. Мы также рассмотрим два практических применения UDP. Поскольку протокол транспортного уровня UDP обычно запускается в операционной системе, а протоколы, использующие UDP, — в пространстве пользователя, эти применения вполне можно считать приложениями. Однако методы, которые в них используются, оказываются полезными для многих приложений и их лучше считать частью транспортного сервиса. Поэтому о них мы тоже поговорим.

6.4.1. Основы UDP

Среди набора протоколов Интернета есть транспортный протокол без установления соединения, **UDP (User Datagram Protocol — протокол передачи дейтаграмм пользователя)**. UDP позволяет приложениям отправлять инкапсулированные IP-дейтаграммы без установления соединений. UDP описан в RFC 768.

С помощью протокола UDP передаются сегменты, состоящие из 8-байтного заголовка, за которым следует поле полезной нагрузки. Заголовок показан на рис. 6.23. Два номера портов служат для идентификации сокетов внутри отправляющей и принимающей машин. Когда прибывает пакет UDP, содержимое его поля полезной нагрузки передается процессу, связанному с портом назначения. Это связывание происходит при выполнении базовой операции типа `BIND`. Это было продемонстрировано в листинге 6.1 применительно к TCP (в UDP процесс связывания происходит точно так же). Представьте себе, что порты — это почтовые ящики, арендуемые приложениями. Мы поговорим о них подробнее при обсуждении TCP, который тоже использует порты. В сущности, весь смысл использования UDP вместо обычного IP заключается как раз в указании портов источника и приемника. Без этих двух полей на транспортном уровне невозможно было бы определить действие, которое следует производить с каждым входящим пакетом. В соответствии с полями портов производится доставка вложенных сегментов соответствующим приложениям.

Рис. 6.23. Заголовок UDP

Информация о порте источника требуется, прежде всего, при создании ответа, пересылаемого отправителю. Копируя значения поля *Порт источника* из входящего сегмента в поле *Порт назначения* исходящего сегмента, процесс, посылающий ответ, может указать, какому именно процессу на противоположной стороне он предназначается.

Поле *Длина UDP* состоит из заголовка и данных. Минимальная длина равна длине заголовка, то есть 8 байтам. Максимальная длина равна 65 515 байтам — это меньше, чем максимальное число, записывающееся с помощью 16 бит, из-за ограничений на размер IP-пакета.

Необязательное поле *Контрольная сумма* служит для повышения надежности. Оно содержит контрольную сумму заголовка, данных и псевдозаголовка. При выполнении вычислений поле *Контрольная сумма* устанавливается равным нулю, а поле данных дополняется нулевым байтом, если его длина представляет собой нечетное число. Алгоритм вычисления контрольной суммы просто складывает все 16-разрядные слова в дополнительном коде, а затем вычисляет дополнение для всей суммы. В результате, когда получатель считает контрольную сумму всего сегмента, включая поле *Контрольная сумма*, результат должен быть равен 0. Если она не подсчитывается, ее значение равно 0 (настоящая нулевая контрольная сумма кодируется всеми единицами). Однако отключать функцию подсчета контрольной суммы глупо, за исключением одного случая — когда нужна высокая производительность (например, при передаче оцифрованной речи).

В случае IPv4 псевдозаголовок будет выглядеть так, как показано на рис. 6.24. Он содержит 32-разрядные IP-адреса отправителя и получателя, номер протокола для UDP (17) и счетчик байтов для UDP-сегмента (включая заголовок). Включение псевдозаголовка в контрольную сумму UDP помогает обнаружить неверно доставленные пакеты, хотя это нарушает иерархию протоколов, так как IP-адреса в нем принадлежат IP-уровню, а не UDP-уровню. В TCP для контрольной суммы используется такой же псевдозаголовок.

Рис. 6.24. Псевдозаголовок, включаемый в контрольную сумму UDP

Наверное, стоит прямо сказать о том, чего UDP *не* делает. Итак, UDP не занимается управлением потоком данных, контролем перегрузки, повторной передачей после приема испорченного сегмента. Все это перекладывается на пользовательские процессы. Что же он делает? UDP предоставляет интерфейс для IP путем демультиплексирования нескольких процессов с использованием портов и необязательного сквозного обнаружения ошибок. Это все, что он делает.

Для процессов, которым хочется управлять потоком, контролировать ошибки и временные интервалы, протокол UDP — это как раз то, что доктор прописал. Одной из областей, где это особенно полезно, является область клиент-серверных приложений. Зачастую клиент посылает короткий запрос серверу и надеется получить короткий ответ. Если запрос или ответ теряется, клиент по прошествии определенного временного интервала может попытаться еще раз. Это позволяет не только упростить код, но и уменьшить требуемое количество сообщений по сравнению с протоколами, которым требуется начальная настройка (такими, как TCP).

DNS (Domain Name System — служба имен доменов) — это приложение, которое использует UDP именно так, как описано выше. Мы изучим его в главе 7. В двух словах, если программе нужно найти IP-адрес по имени хоста, например *www.cs.berkeley.edu*, она может послать UDP-пакет с этим именем на сервер DNS. Сервер в ответ на запрос посылает UDP-пакет с IP-адресом хоста. Никакой предварительной настройки не требуется, как не требуется и разрыва соединения после завершения задачи. По сети просто передаются два сообщения.

6.4.2. Вызов удаленной процедуры

В определенном смысле процессы отправки сообщения на удаленный хост и получения ответа очень похожи на вызов функции в языке программирования. В обоих случаях необходимо передать один или несколько параметров, и вы получаете результат. Это соображение навело разработчиков на мысль о том, что можно попробовать организовать запросно-ответное взаимодействие по сети, выполняемое в форме вызовов процедур. Такое решение позволяет упростить и сделать более привычной разработку сетевых приложений. Например, представьте себе процедуру с именем `get_IP_address(имя_хоста)`, работающую посредством отправки UDP-пакетов на сервер DNS, ожидания ответа и отправки повторного запроса в случае наступления таймаута (если одна из сторон работает недостаточно быстро). Таким образом, все детали, связанные с особенностями сетевых технологий, скрыты от программиста.

Ключевая работа в этой области написана в 1984 году Бирреллом (Birrell) и Нельсоном (Nelson). По сути дела, было предложено разрешить программам вызывать процедуры, расположенные на удаленных хостах. Когда процесс на машине 1 вызывает процедуру, находящуюся на машине 2, вызывающий процесс машины 1 блокируется, и выполняется вызванная процедура на машине 2. Информация от вызывающего процесса может передаваться в виде параметров и приходить обратно в виде результата процедуры. Передача сообщений по сети скрыта от программиста приложения. Такая технология известна под названием **RPC (Remote Procedure Call — удаленный вызов процедур)** и стала основой многих сетевых приложений. Традиционно вызывающая процедура считается клиентом, а вызываемая — сервером. Мы и здесь будем называть их так же.

Идея RPC состоит в том, чтобы сделать вызов удаленной процедуры максимально похожим на локальный вызов. В простейшем случае для вызова удаленной процедуры клиентская программа должна быть связана с маленькой библиотечной процедурой, называемой **клиентской заглушкой (client stub)**, которая отображает серверную процедуру в пространство адресов клиента. Аналогично сервер должен быть связан с процедурой, называемой **серверной заглушкой (server stub)**. Эти процедуры скрывают тот факт, что вызов клиентом серверной процедуры осуществляется не локально.

Реальные шаги, выполняемые при удаленном вызове процедуры, показаны на рис. 6.25. Шаг 1 заключается в вызове клиентом клиентской заглушки. Это локальный вызов процедуры, параметры которой самым обычным образом помещаются в стек. Шаг 2 состоит в упаковке параметров клиентской заглушки в сообщение и в осуществлении системного вызова для отправки этого сообщения. Упаковка параметров называется **маршалингом (marshaling)**. На шаге 3 операционная система передает

сообщение с клиентской машины на сервер. Шаг 4 заключается в том, что операционная система передает входящий пакет серверной заглушке. Последняя на пятом шаге вызывает серверную процедуру с распакованными параметрами. При ответе выполняются те же самые шаги, но передача происходит в обратном направлении.

Важнее всего здесь то, что клиентская процедура, написанная пользователем, выполняет обычный (то есть локальный) вызов клиентской заглушки, имеющей то же имя, что и серверная процедура. Поскольку клиентская процедура и клиентская заглушка существуют в одном и том же адресном пространстве, параметры передаются обычным образом. Аналогично серверная процедура вызывается процедурой, находящейся в том же адресном пространстве, с ожидаемыми параметрами. С точки зрения серверной процедуры не происходит ничего необычного. Таким образом, вместо ввода/вывода с помощью сокетов сетевая коммуникация осуществляется обычным вызовом процедуры.

Несмотря на элегантность концепции RPC, в ней есть определенные подводные камни. Речь идет, прежде всего, об использовании указателей в качестве параметров. В обычной ситуации передача указателя процедуре не представляет никаких сложностей. Вызываемая процедура может использовать указатель так же, как и вызывающая, поскольку они обе существуют в одном и том же виртуальном адресном пространстве. При удаленном вызове процедуры передача указателей невозможна, потому что адресные пространства клиента и сервера отличаются.

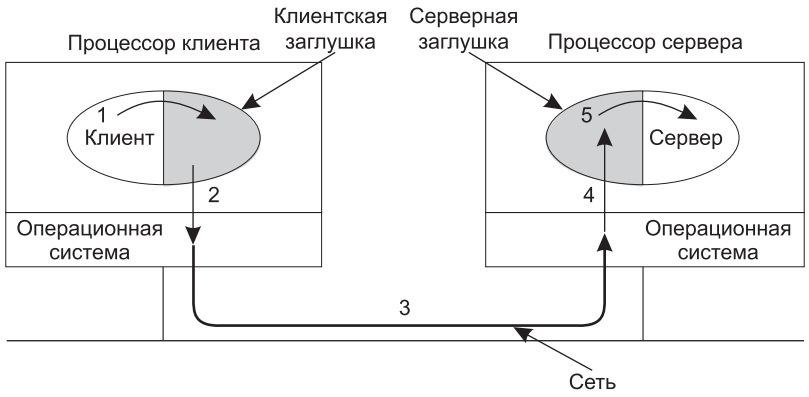


Рис. 6.25. Этапы выполнения удаленного вызова процедуры. Заглушки затенены

Иногда с помощью некоторых уловок все же удается передавать указатели. Допустим, первым параметром является указатель на целое число *k*. Клиентская заглушка может выполнить маршалинг *k* и передать его серверу. Серверная заглушка создаст указатель на полученную переменную *k* и передаст его серверной процедуре. Именно этого та и ожидала. Когда серверная процедура возвращает управление серверной заглушке, последняя отправляет *k* обратно клиенту, где обновленное значение этой переменной записывается вместо старого (если оно было изменено сервером). В принципе, стандартная последовательность действий, выполняемая при передаче параметра по ссылке, заменилась прямой и обратной передачей копии параметра. Увы, этот трюк не всегда удается применить — в частности, нельзя это сделать, если указатель ссылается

на граф или иную сложную структуру данных. Как мы увидим далее, по этой причине на параметры удаленно вызываемых процедур должны быть наложены определенные ограничения.

Вторая проблема заключается в том, что в языках со слабой типизацией данных (например, в C) можно совершенно законно написать процедуру, которая подсчитывает скалярное произведение двух векторов (массивов), не указывая их размеры. Каждая из этих структур в качестве ограничителя имеет какое-то значение, известное только вызывающей и вызываемой процедурам. При этих обстоятельствах клиентская заглушка не способна запаковать параметры: нет никакой возможности определить их размеры.

Третья проблема заключается в том, что не всегда можно распознать типы параметров по спецификации или по самому коду. В качестве примера можно привести процедуру `printf`, у которой может быть любое число параметров (не меньше одного), и они могут представлять собой смесь различных целочисленных (int, short, long), символьных, строковых, вещественных с плавающей запятой различной длины и других типов. Задача удаленного вызова процедуры `printf` может оказаться практически невыполнимой из-за такой своеобразной толерантности языка C. Тем не менее нет правила, говорящего, что удаленный вызов процедур возможен только в том случае, если используется не C (C++), — это подорвало бы репутацию метода RPC среди программистов.

Четвертая проблема связана с применением глобальных переменных. В нормальной ситуации вызывающая и вызываемая процедуры могут общаться друг с другом посредством глобальных переменных (кроме общения с помощью параметров). Но если вызываемая процедура переедет на удаленную машину, программа, использующая глобальные переменные, не сможет работать, потому что глобальные переменные больше не смогут служить в качестве разделяемого ресурса.

Эти проблемы не означают, что метод удаленного вызова процедур безнадежен. На самом деле, он широко используется, просто нужны некоторые ограничения для его нормальной практической работы.

С точки зрения протоколов транспортного уровня UDP является хорошей основой для реализации RPC. В простейшем случае запросы и ответы можно отправлять в одном UDP-пакете, а обработка может выполняться очень быстро. Однако для реализации этой идеи потребуются и другие механизмы. На случай потери запроса или ответа клиенту необходим таймер, отсчитывающий время до повторной отправки пакета. Обратите внимание на то, что ответ служит неявным подтверждением запроса, поэтому запрос не требует отдельного подтверждения. Иногда параметры или результаты могут оказаться больше максимального размера UDP-пакета, поэтому для отправки больших сообщений также требуется специальный протокол. Если множественные запросы и ответы могут пересекаться (как в случае параллельного программирования), соответствие ответа запросу должно быть указано с помощью специальной метки.

Проблема более высокого уровня связана с тем, что операция может не быть идемпотентной (то есть не может повторяться без риска сбоя). В простом случае мы имеем дело с идемпотентными операциями, такими как DNS-запросы и ответы. Клиент может повторно без риска передавать такие пакеты сколько угодно раз, до тех пор пока не придет ответ. Не важно, в чем причина: либо пакет не дошел до сервера,

либо был потерян ответ. В любом случае ответ, когда он придет, будет одним и тем же (если, конечно, за это время не обновится база данных DNS). Однако не все операции идемпотентны — например, потому, что они могут включать побочные действия наподобие инкрементирования счетчика. RPC для таких операций требует более сложной семантики: при вызове процедуры она не должна выполняться несколько раз. В таких случаях может понадобиться установка TCP-соединения и отправки запроса по TCP, а не по UDP.

6.4.3. Транспортные протоколы реального масштаба времени

Клиент-серверный удаленный вызов процедур — это та область, в которой UDP применяется очень широко. Еще одной такой областью являются мультимедийные приложения реального времени. В частности, с ростом популярности интернет-радио, интернет-телефонии, музыки и видео по заказу, видеоконференций и других мультимедийных приложений стало понятно, что все они пытаются заново изобрести велосипед, используя более или менее одинаковые транспортные протоколы реального времени. Вскоре пришли к мысли о том, что было бы здорово иметь один общий транспортный протокол для мультимедийных приложений.

Так появился на свет протокол **RTP (Real-Time Transport Protocol — транспортный протокол реального масштаба времени)**. Он описан в RFC 3550 и ныне широко используется для мультимедийных приложений. Мы рассмотрим два аспекта транспортировки данных в реальном времени. Первый из них — протокол RTP, использующийся для пакетной передачи аудио и видео. Второй аспект — обработка данных, необходимая для воспроизведения (в основном со стороны получателя). Эти функции входят в стек протоколов, изображенный на рис. 6.26.

RTP обычно работает поверх UDP (в операционной системе). Делается это так. Мультимедийное приложение может состоять из нескольких аудио-, видео-, текстовых и некоторых других потоков. Они прописываются в библиотеке RTP, которая, как и само приложение, находится в пользовательском пространстве. Библиотека уплотняет потоки и помещает их в пакеты RTP, которые, в свою очередь, отправляются в сокет. На другом конце сокета (в операционной системе) генерируются UDP-пакеты, в которые помещаются RTP-пакеты. Они передаются протоколу IP для отправки по каналу (например, Ethernet). На хосте-получателе происходит обратный процесс. В конечном итоге мультимедийное приложение получает данные из RTP-библиотеки. Оно отвечает за воспроизведение мультимедиа. Стек протоколов для описанной ситуации показан на рис. 6.26, *а*. Система вложенных пакетов показана на рис. 6.26, *б*.

В результате оказывается непросто определить, к какому уровню относится RTP. Так как протокол работает в пользовательском пространстве и связан с прикладной программой, он, несомненно, выглядит, как прикладной протокол. С другой стороны, он является общим, независимым от приложения протоколом, который просто предоставляет транспортные услуги, не более. С этой точки зрения он может показаться транспортным протоколом. Наверное, лучше всего будет определить его таким образом: RTP — это транспортный протокол, который случайно оказался на прикладном уровне, и именно поэтому мы говорим о нем в этой главе.

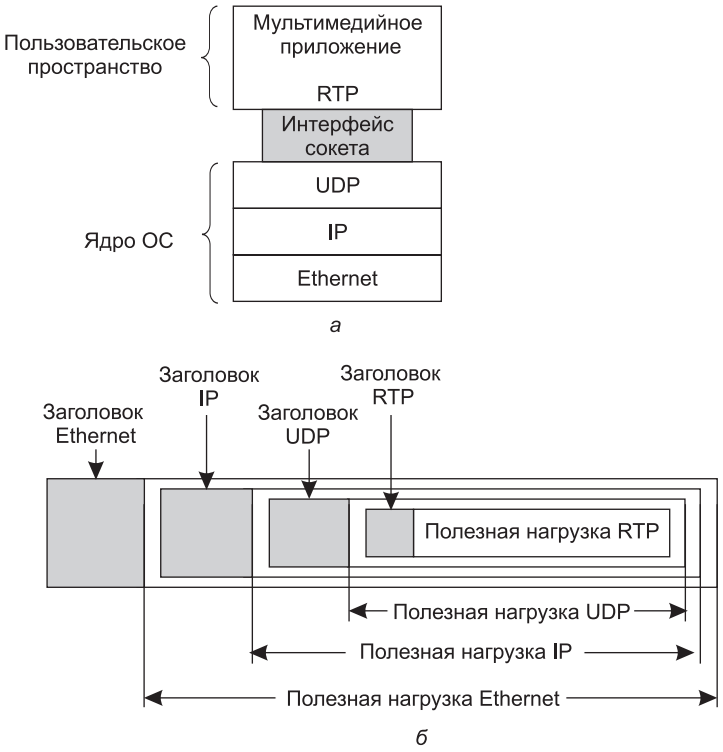


Рис. 6.26. RTR: *а* — положение RTP в стеке протоколов; *б* — вложение пакетов

RTP — транспортный протокол реального масштаба времени

Основной функцией RTP является уплотнение нескольких потоков реального масштаба времени в единый поток пакетов UDP. Поток UDP можно направлять либо по одному, либо сразу по нескольким адресам. Поскольку RTP использует обычный UDP, его пакеты не обрабатываются маршрутизаторами каким-либо особым образом, если только не включены свойства доставки с качеством обслуживания уровня IP. В частности, нет никаких гарантий касательно доставки, и при этом пакеты могут теряться, задерживаться, повреждаться и т. д.

Формат RTP обладает рядом особенностей, которые помогают получателям обрабатывать мультимедийные данные. Каждый пакет, посылаемый с потоком RTP, имеет номер, на единицу превышающий номер своего предшественника. Такой способ нумерации позволяет получателю определить пропажу пакетов. Если обнаруживается исчезновение какого-либо пакета, то выбор оптимального для получателя решения остается за приложением. Если пакеты содержат видеоданные, возможен пропуск потерянных кадров. В случае аудиоданных можно путем интерполяции аппроксимировать пропущенное значение. Повторная передача в данном случае не является хорошим решением, поскольку это займет много времени, и повторно переданный пакет окажется уже никому не нужным. Поэтому в протоколе RTP не предусмотрено никаких подтверждений и механизмов запроса повторной передачи.

В поле полезной нагрузки RTP может содержаться несколько элементов данных, которые могут иметь формат, соответствующий отправившему их приложению. Межсетевое взаимодействие обеспечивается в протоколе RTP за счет определения нескольких профилей (например, отдельных аудиопотоков), каждому из которых может сопоставляться несколько форматов кодирования. Скажем, аудиопоток может кодироваться при помощи PCM (8-битными символами с полосой 8 кГц), дельта-кодирования, кодирования с предсказанием, GSM-кодирования, MP3-кодирования и т. д. В RTP имеется специальное поле заголовка, в котором источник может указать метод кодирования, однако далее источник никак не влияет на процесс кодирования.

Еще одна функция, необходимая приложениям реального времени, — это метки времени. Идея состоит в том, чтобы позволить источнику связать метку времени с первым элементом каждого пакета. Метки времени ставятся относительно момента начала передачи потока, поэтому важны только интервалы *между* метками. Абсолютные значения, по сути дела, никакой роли не играют. Вскоре мы узнаем, что такой механизм позволяет приемнику буферизировать небольшое количество данных и проигрывать каждый отрезок спустя правильное число миллисекунд после начала потока, независимо от того, когда, на самом деле, приходит пакет, содержащий данный отрезок.

За счет этого не только снижается эффект колебания сетевой задержки, но и появляется возможность синхронизации между собой нескольких потоков. Например, в цифровом телевидении может быть видеопоток и два аудиопотока. Второй аудиопоток обычно нужен либо для обеспечения стереозвучания, либо для дублированной на иностранный язык звуковой дорожки фильма. У каждого потока свой физический источник, однако с помощью временных отметок, генерируемых единым таймером, эти потоки при воспроизведении можно синхронизовать даже в том случае, если при передаче и/или получении произошли ошибки.

Заголовок RTP показан на рис. 6.27. Он состоит из трех 32-разрядных слов и некоторых возможных расширений. Первое слово содержит поле *Версия*, которое в настоящий момент уже имеет значение 2. Будем надеяться, что текущая версия окажется окончательной или хотя бы предпоследней, поскольку в идентифицирующем ее двухбитном поле осталось место только для одного нового номера (впрочем, код 3 может обозначать, что настоящий номер версии содержится в поле расширения).

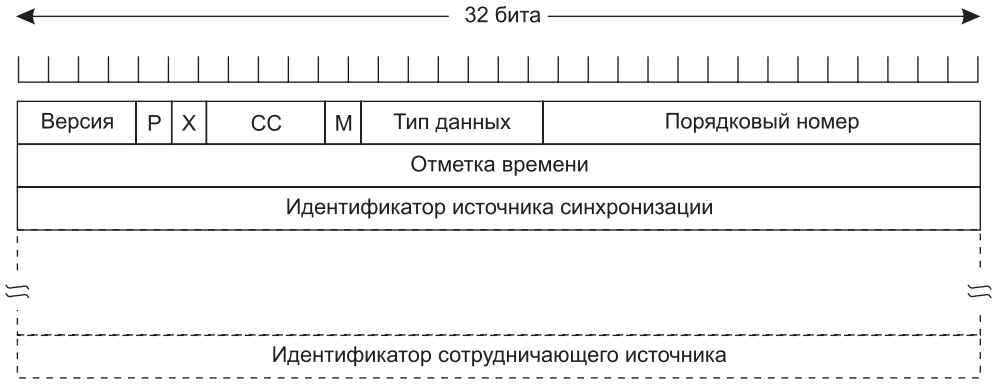


Рис. 6.27. Заголовок RTP

Бит *P* указывает на то, что размер пакета сделан кратным 4 байтам за счет байтов заполнения. При этом в последнем байте заполнения содержится общее число байтов заполнения. Бит *X* говорит о том, что присутствует расширенный заголовок. Формат и назначение расширенного заголовка не определяются. Обязательным для него является только то, что первое слово расширения должно содержать общую длину расширения. Это запасная возможность для разнообразных непредсказуемых будущих требований.

Поле *CC* говорит о том, сколько сотрудничающих источников формируют поток. Их число может колебаться от 0 до 15 (см. ниже). Бит *M* — это маркер, связанный с конкретным приложением. Он может использоваться для обозначения начала видеокадра, начала слова в аудиоканале или еще для чего-нибудь, важного и понятного для приложения. Поле *Тип данных* содержит информацию об использующемся алгоритме кодирования (например, несжатое 8-битное аудио, MP3 и т. д.). Поскольку такое поле есть в каждом пакете, метод кодирования может изменяться прямо во время передачи потока. *Порядковый номер* — это просто счетчик, который инкрементируется в каждом пакете RTP. Он используется для определения потерявшихся пакетов.

Метка времени генерируется источником потока и служит для записи момента создания первого слова пакета. Отметки времени помогают снизить эффект временных колебаний, или джиттера, на приемнике за счет того, что момент воспроизведения делается независимым от времени прибытия пакета. *Идентификатор источника синхронизации* позволяет определить, какому потоку принадлежит пакет. Это и есть применяемый метод мультиплексирования и демультиплексирования нескольких потоков данных, следующих в виде единого потока UDP-пакетов. Наконец, *Идентификаторы сотрудничающих источников*, если таковые имеются, используются, когда конечный поток формируется несколькими источниками. В этом случае микширующее устройство является источником синхронизации, а в полях идентификаторов источников перечисляются смешиваемые потоки.

RTCP — управляющий транспортный протокол реального времени

У протокола RTP есть небольшой родственный протокол под названием **RTCP (Real-Time Transport Control Protocol — управляющий транспортный протокол реального времени)**. Этот протокол определен в RFC 3550 вместе с RTP. Он занимается поддержкой обратной связи, синхронизацией, обеспечением пользовательского интерфейса, однако не занимается передачей каких-либо медиаданных.

Первая его функция может использоваться для обратной связи по задержкам, колебаниям времени задержки или джиттеру, пропускной способности, перегрузке и другим свойствам сети, о которых сообщается источникам. Полученная информация может приниматься во внимание кодировщиком для увеличения скорости передачи данных (что приведет к улучшению качества), когда это позволяет делать состояние сети, или уменьшения скорости при возникновении в сети каких-либо проблем. Постоянная обратная связь обеспечивает динамическую настройку алгоритмов кодирования на обеспечение наилучшего качества при текущих обстоятельствах. Например, пропускная способность при передаче потока может как увеличиваться, так и умень-

шаться, и в соответствии с этим могут изменяться методы кодирования — скажем, MP3 может заменяться 8-битным PCM или дельта-кодированием. Поле *Тип данных* сообщает приемнику о том, какой алгоритм кодирования применяется для данного пакета, что позволяет изменять их по требованию при передаче потока.

Проблема обратной связи заключается в том, что сообщения RTCP рассылаются всем участникам. В таком случае, если группа большая, то пропускная способность, используемая RTCP, резко возрастет. Чтобы этого не произошло, отправители RTCP снижают скорость отправки своих сообщений так, чтобы все вместе они потребляли, скажем, 5 % пропускной способности, использующейся для передачи медиаданных. А для этого каждый участник должен знать эту пропускную способность (эту информацию он может получить от отправителя) и общее число участников (которое он вычисляет на основании сообщений RTCP).

RTCP также обеспечивает межпотоковую синхронизацию. Проблема состоит в том, что разные потоки могут использовать разные таймеры с разной степенью разрешения и разными скоростями дрейфа. RTCP помогает решить эти проблемы и синхронизировать потоки с разными параметрами.

Наконец, RTCP позволяет именовать различные источники (например, с помощью обычного ASCII-текста). Эта информация может отображаться на приемнике, позволяя определить источник текущего потока.

Более подробную информацию о протоколе RTP можно найти в (Perkins, 2002).

Буферизация и контроль джиттера при воспроизведении

Когда информация попадает на приемник, необходимо в правильный момент времени начать ее воспроизведение. В общем случае этот момент времени не совпадает со временем прибытия RTP-пакета, поскольку время прохождения через сеть для разных пакетов немного отличается. Даже если отправитель будет передавать их в сеть через одинаковые промежутки времени, они все равно будут приходить к получателю с разным интервалом. Такое варьирование времени задержки называется **джиттером (jitter)**. Если медиаданные будут проигрываться сразу же по прибытии, даже небольшой джиттер может вызвать серьезные помехи: изображение может быть прерывистым, а звук — неразборчивым.

Решение этой проблемы состоит в **буферизации** пакетов на приемнике перед их воспроизведением. В примере, изображенном на рис. 6.28, мы видим поток пакетов, прибывающих на место назначения с достаточно большим джиттером. Пакет 1 был передан с сервера в момент времени $t = 0$ с и прибыл к клиенту в момент времени $t = 1$ с. Пакет 2 задерживается и прибывает через 2 с. По прибытии пакеты помещаются в буфер на машине клиента.

В момент времени $t = 10$ с начинается воспроизведение. В буфере уже находятся пакеты с 1 по 6, поэтому их можно доставать оттуда через равные промежутки времени и тем самым обеспечить равномерное воспроизведение. В общем случае равные промежутки времени использовать не обязательно, так как метки времени RTP содержат информацию о том, когда следует начать воспроизведение.

К сожалению, пакет 8 задерживается настолько, что не успевает прибыть к тому моменту, когда он должен быть воспроизведен. В таком случае возможны два вари-

анта. Проигрыватель может пропустить восьмой пакет и перейти к воспроизведению следующих пакетов. Или же он может остановить воспроизведение до тех пор, пока не прибудет восьмой пакет, и тем самым создать неприятную паузу в музыке или фильме. Если приложение работает в режиме реального времени (например, при звонке через Интернет), то пакет, скорее всего, будет пропущен. Такие приложения не очень хорошо работают в режиме ожидания. В приложениях, работающих с потоковым мультимедиа, проигрыватель может на время остановить воспроизведение. Чтобы улучшить ситуацию, можно начать воспроизведение еще позже и использовать буфер большего размера. Проигрыватели потокового аудио и видео часто используют буферы размером около 10 с, чтобы все пакеты (кроме тех, которые были потеряны) успели прийти вовремя. Приложения, работающие в реальном времени (например, видеоконференции) и требующие быстрой ответной реакции, используют маленькие буферы.

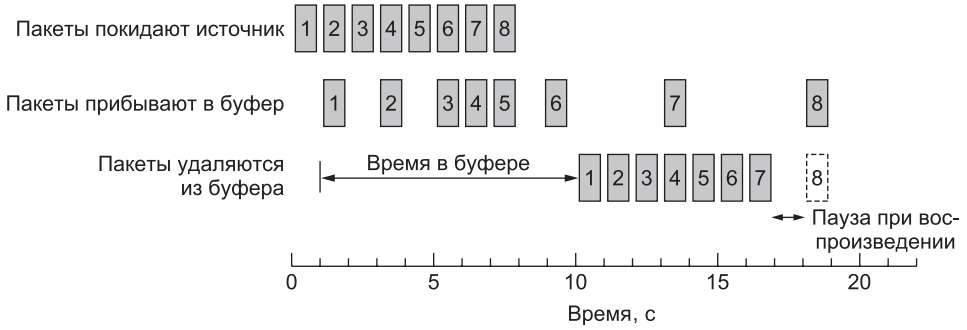


Рис. 6.28. Выравнивание выходного потока с помощью буферизации пакетов

При равномерном воспроизведении ключевую роль играет **задержка воспроизведения (playback point)**, то есть время, которое получатель должен ждать медиаданных, прежде чем начать их воспроизведение. Этот параметр зависит от джиттера. Различие между соединением с маленьким и большим джиттером показано на рис. 6.29. Средняя задержка может отличаться не сильно, однако при большом джиттере может потребоваться задержка воспроизведения, захватывающая 99 % пакетов — гораздо больше, чем при маленьком джиттере.

Чтобы правильно подобрать задержку воспроизведения, приложение может измерять джиттер, вычисляя разницу между значениями меток времени RTP и временем прибытия. Эта разница (плюс некоторая константа) и составляет задержку каждого отдельного пакета. Однако некоторые факторы, такие как конкурирующий трафик и изменение маршрутов, могут стать причиной изменения времени задержки. Приложения должны учитывать это и при необходимости менять задержку воспроизведения. Но при неправильной реализации такое изменение может вызвать сильные помехи. Одно из возможных решений для аудио состоит в том, чтобы корректировать задержку воспроизведения между **сегментами речи (talkspurts)**, то есть во время пауз. Разница между короткой паузой и чуть более длинной вряд ли будет заметна. Чтобы это было возможным, RTP позволяет приложениям отмечать начало нового сегмента речи с помощью маркера *M*.

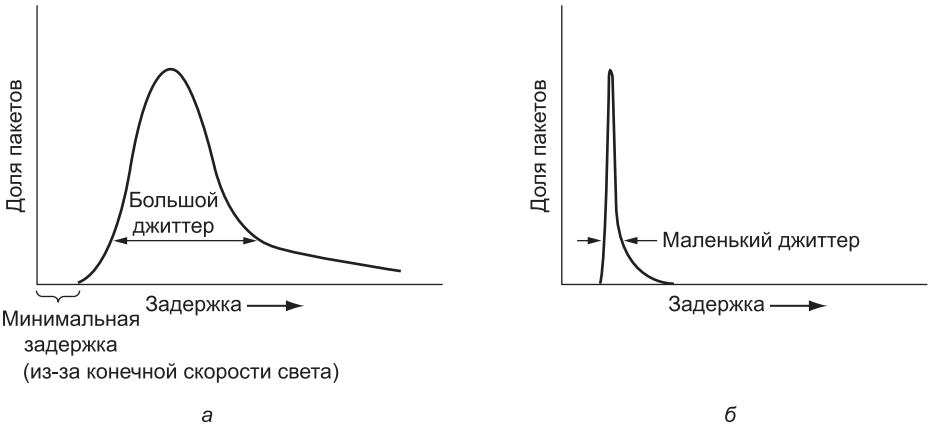


Рис. 6.29. Джиттер: *а* — большой; *б* — маленький

Ситуация, в которой медиаданные не проигрываются слишком долго, представляет серьезную проблему для приложений, работающих в реальном времени. Не существует способа уменьшить задержку распространения, если уже и так используется кратчайший путь. Задержку воспроизведения можно снизить за счет увеличения доли пакетов, опаздывающих к началу проигрывания. Если это недопустимо, остается последний вариант: уменьшить джиттер, запросив более высокое качество обслуживания — например, дифференцированный сервис с приоритетной доставкой. Иными словами, для этого требуется сеть с лучшими параметрами.

6.5. Транспортные протоколы Интернета: TCP

UDP является простым протоколом и имеет очень важную область применения. В первую очередь, это клиент-серверные взаимодействия и мультимедиа. Тем не менее большинству интернет-приложений требуется надежная, последовательная передача. UDP не удовлетворяет этим требованиям, поэтому необходим иной протокол. Такой протокол называется TCP, и он является рабочей лошадкой Интернета. Ниже мы рассмотрим его детально.

6.5.1. Основы TCP

Протокол **TCP (Transmission Control Protocol — протокол управления передачей)** был специально разработан для обеспечения надежного сквозного байтового потока по ненадежной интерсети. Объединенная сеть отличается от отдельной сети тем, что ее различные участки могут обладать сильно различающейся топологией, пропускной способностью, значениями времени задержки, размерами пакетов и другими параметрами. При разработке TCP основное внимание уделялось способности протокола адаптироваться к свойствам объединенной сети и отказоустойчивости при возникновении различных проблем.

Протокол TCP был описан в RFC 793 в сентябре 1981 года. Со временем он был во многом усовершенствован, были исправлены различные ошибки и неточности. На сегодняшний день существует множество других RFC, являющихся дополнениями к RFC 793 (что позволяет судить о степени распространения этого протокола): уточнения и исправления описаны в RFC 1122, расширения для высокой производительности — в RFC 1323, выборочные подтверждения — в RFC 2018, контроль перегрузки — в RFC 2581, использование полей заголовка для качества обслуживания — в RFC 2873, усовершенствованные таймеры повторной передачи — в RFC 2988, явные уведомления о перегрузке — в RFC 3168. Поскольку это далеко не полный список, для удобной работы со всеми этими RFC был создан специальный указатель (конечно же, в виде еще одного RFC-документа) — RFC 4614.

Каждая машина, поддерживающая протокол TCP, обладает транспортной подсистемой TCP, являющейся либо библиотечной процедурой, либо пользовательским процессом, либо (чаще всего) частью ядра системы. В любом случае, транспортная подсистема управляет TCP-потоками и интерфейсом с IP-уровнем. TCP-подсистема принимает от локальных процессов пользовательские потоки данных, разбивает их на куски, не превосходящие 64 Кбайт (на практике это число обычно равно 1460 байтам данных, что позволяет поместить их в один кадр Ethernet с заголовками IP и TCP), и посылает их в виде отдельных IP-дейтаграмм. Когда IP-дейтаграммы с TCP-данными прибывают на машину, они передаются TCP-подсистеме, которая восстанавливает исходный байтовый поток. Для простоты мы иногда будем употреблять «TCP» для обозначения транспортной подсистемы TCP (части программного обеспечения) или протокола TCP (набора правил). Из контекста будет понятно, что имеется в виду. Например, в выражении «Пользователь передает данные TCP» подразумевается, естественно, транспортная подсистема TCP.

Уровень IP не гарантирует правильной доставки дейтаграмм и не накладывает ограничений на скорость их отправки. Именно TCP приходится выбирать правильную скорость отправки (следуя целям эффективного использования пропускной способности и предотвращения перегрузок), следить за истекшими интервалами ожидания и в случае необходимости заниматься повторной передачей дейтаграмм, не дошедших до места назначения. Бывает, что дейтаграммы прибывают в неправильном порядке. Восстанавливать сообщения из таких дейтаграмм обязан также TCP. Таким образом, протокол TCP призван обеспечить хорошую производительность и надежность, о которой мечтают многие приложения и которая не предоставляется протоколом IP.

6.5.2. Модель сервиса TCP

В основе сервиса TCP лежат так называемые сокеты (гнезда или конечные точки), создаваемые как отправителем, так и получателем. Они обсуждались в разделе 6.1.3. У каждого сокета есть номер (адрес), состоящий из IP-адреса хоста и 16-битного номера, локального по отношению к хосту, называемого **портом**. Портом в TCP называют TSAP-адрес. Для обращения к сервису TCP между сокетом одной машины и сокетом другой машины должно быть явно установлено соединение. Базовые операции сокетов приведены в табл. 6.2.

Один сокет может использоваться одновременно для нескольких соединений. Другими словами, два и более соединения могут оканчиваться одним сокетом. Соединения различаются по идентификаторам сокетов на обоих концах — (*socket1*, *socket2*). Номера виртуальных каналов или другие идентификаторы не используются.

Номера портов со значениями ниже 1024 зарезервированы стандартными сервисами и доступны только привилегированным пользователям (например, `root` в UNIX-системах). Они называются **известными портами (well-known ports)**. Например, любой процесс, желающий удаленно загрузить почту с хоста, может связаться с портом 143 хоста-адресата и обратиться, таким образом, к его IMAP-демону. Список известных портов приведен на сайте *www.iana.org*. Таких портов на данный момент более 700. Некоторые из них включены в табл. 6.4.

Порты с номерами от 1024 до 49151 можно зарегистрировать через IANA для непривилегированных пользователей, однако приложения могут выбирать свои собственные порты (что они, как правило, и делают). К примеру, приложение BittTorrent для однорангового совместного доступа к файлам (неофициально) использует порты 6881–6887, однако другие порты также возможны.

Можно было бы, конечно, связать FTP-демона с портом 21 еще во время загрузки, тогда же связать демона SSH с портом 22 и т. д. Однако, если бы мы так сделали, мы бы только зря заняли память информацией о демонах, которые, на самом деле, большую часть времени простаивают. Вместо этого обычно пользуются услугами одного демона, называемого в UNIX **inetd (Internet daemon)**, который связывается с несколькими портами и ожидает первое входящее соединение. Когда оно происходит, *inetd* создает новый процесс, для которого вызывается подходящий демон, обрабатывающий запрос. Таким образом, постоянно активен только *inetd*, остальные вызываются только тогда, когда для них есть работа. Inetd имеет специальный конфигурационный файл, из которого он может узнать о назначении портов. Это значит, что системный администратор может настроить систему таким образом, чтобы с самыми загруженными портами (например, 80) были связаны постоянные демоны, а с остальными — *inetd*.

Таблица 6.4. Некоторые зарезервированные порты

Порт	Протокол	Использование
20, 21	FTP	Передача файлов
22	SSH	Дистанционный вход в систему, замена Telnet
25	SMTP	Электронная почта
80	HTTP	Всемирная паутина (World Wide Web)
110	POP-3	Удаленный доступ к электронной почте
143	IMAP	Удаленный доступ к электронной почте
443	HTTPS	Защита от угроз (HTPP через SSL/TLS)
543	RTSP	Контроль воспроизведения мультимедиа
631	IPP	Коллективное использование принтера

Все TCP-соединения являются полнодуплексными и двухточечными. Полный дуплекс означает, что трафик может следовать одновременно в противоположные

стороны. Двухточечное соединение подразумевает, что у него имеются ровно две конечные точки. Широковещание и многоадресная рассылка протоколом TCP не поддерживаются.

TCP-соединение представляет собой байтовый поток, а не поток сообщений. Границы между сообщениями не сохраняются. Например, если отправляющий процесс записывает в TCP-поток четыре 512-байтовых порции данных, эти данные могут быть доставлены получающему процессу в виде четырех 512-байтовых порций, двух 1024-байтовых порций, одной 2048-байтовой порции (рис. 6.30) или как-нибудь еще. Нет способа, которым получатель смог бы определить, каким образом записывались данные.

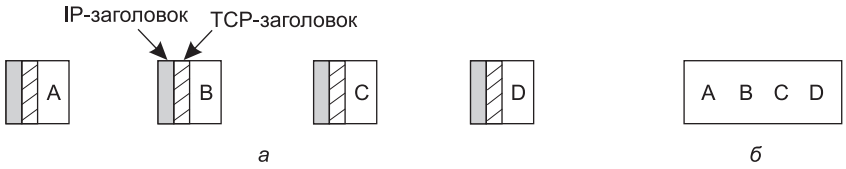


Рис. 6.30. Четыре 512-байтовых сегмента, посланные как отдельные IP-дейтаграммы (*а*); 2048 байт данных, доставленные приложению с помощью одного вызова процедуры READ (*б*)

Файлы в системе UNIX также обладают этим свойством. Программа, читающая файл, не может определить, как был записан этот файл: поблочно, побайтно или сразу целиком. Как и в случае с файлами системы UNIX, **TCP-программы не имеют представления о назначении байтов** и не интересуются этим. Байт для них — просто байт.

Получив данные от приложения, протокол TCP может послать их сразу или поместить в буфер, чтобы послать большую порцию данных, по своему усмотрению. Однако иногда приложению бывает необходимо, чтобы данные были посланы немедленно. Допустим, например, что пользователь интерактивной игры хочет отправить поток обновлений. Важно, чтобы обновления передавались сразу же, а не сохранялись в буфере до появления других обновлений. Чтобы можно было вынудить передачу данных, в TCP существует флаг PUSH (протолкнуть). Изначально предполагалось, что с помощью этого флага приложения будут сообщать TCP о том, что не нужно задерживать передачу пакета. Однако приложения не могут устанавливать такие флаги при отправке данных. Вместо этого в различных операционных системах существуют специальные параметры, позволяющие ускорить передачу данных (например, `TCP_NONDELAY` в Windows и Linux).

Для тех, кто интересуется доисторическими методами Интернета, мы расскажем об интересной особенности службы TCP, которая все еще входит в состав протокола, но используется редко. Речь пойдет о **срочных данных (urgent data)**. Когда часть данных обладает высоким приоритетом, то есть должна обрабатываться сразу же, — например, если пользователь, взаимодействующий с программой в интерактивном режиме, нажимает Ctrl-C, чтобы прервать начавшийся удаленный процесс, — посылающее приложение может поместить в выходной поток данных управляющую информацию и передать ее TCP-службе вместе с флагом `URGENT` (срочно). Этот флаг заставляет TCP-подсистему прекратить накопление данных и без промедления передать в сеть все, что у нее есть для данного соединения.

Когда срочные данные прибывают по назначению, получающее приложение прерывается (то есть в терминологии UNIX «получает сигнал»), после чего оно может считать данные из входного потока и найти среди них срочные. Конец срочных данных маркируется, так что приложение может распознать, где они заканчиваются. Начало срочных данных не маркируется. Приложение должно само догадаться.

Такая схема представляет собой грубый сигнальный механизм, оставляя все прочее приложению. Хотя теоретически использование срочных данных выглядит целесообразным, в первое время после своего появления эта схема не была хорошо реализована и поэтому быстро вышла из употребления. Сейчас использовать ее не рекомендуется из-за трудностей реализации, так что приложения вынуждены прибегать к своим собственным системам сигналов. Возможно, в последующих транспортных протоколах эта идея будет реализована лучше.

6.5.3. Протокол TCP

В данном разделе будет рассмотрен протокол TCP в общих чертах. В следующем разделе мы обсудим заголовок протокола, поле за полем.

Ключевым свойством TCP, определяющим всю структуру протокола, является то, что в TCP-соединении у каждого байта есть свой 32-разрядный порядковый номер. В первые годы существования Интернета базовая скорость передачи данных между маршрутизаторами по выделенным линиям составляла 56 Кбит/с. Хосту, постоянно выдающему данные с максимальной скоростью, потребовалось бы больше недели на то, чтобы порядковые номера совершили полный круг. При нынешних скоростях порядковые номера могут кончиться очень быстро, об этом еще будет сказано ниже. Отдельные 32-разрядные порядковые номера используются для указания позиции скользящего окна в одном направлении и подтверждений в обратном направлении, о чем также будет сказано ниже.

Отправляющая и принимающая TCP-подсистемы обмениваются данными в виде сегментов. **Сегмент TCP** состоит из фиксированного 20-байтового заголовка (плюс необязательная часть), за которым могут следовать байты данных. Размер сегментов определяется программным обеспечением TCP. Оно может объединять в один сегмент данные, полученные в результате нескольких операций записи, или, наоборот, распределять результат одной записи между несколькими сегментами. Размер сегментов ограничен двумя пределами. Во-первых, каждый сегмент, включая TCP-заголовок, должен помещаться в 65 515-байтное поле полезной нагрузки IP-пакета. Во-вторых, в каждом канале есть **максимальный размер передаваемого блока** (**MTU**, **Maximum Transfer Unit**). На отправителе и получателе каждый сегмент должен помещаться в MTU, чтобы он мог передаваться и приниматься в отдельном пакете, не разделенном на фрагменты. На практике максимальный размер передаваемого блока составляет обычно 1500 байт (что соответствует размеру поля полезной нагрузки Ethernet), и таким образом определяется верхний предел размера сегмента.

Тем не менее, если IP-пакет, содержащий TCP-сегменты, проходит по пути со слишком низким MTU, его фрагментация возможна. Но в таком случае снижается производительность, а также возникают другие проблемы (Kent и Mogul, 1987). Вместо этого реализации TCP выполняют обнаружение MTU маршрута. При этом

используется метод, описанный в RFC 1191, — о нем мы говорили в разделе 5.5.5. Этот метод вычисляет минимальное значение MTU по всем каналам пути, используя сообщения об ошибках ICMP. На основе этого значения TCP выбирает размер сегмента, позволяющий избежать фрагментации.

Основным протоколом, используемым TCP-подсистемами, является протокол скользящего окна с динамическим размером окна. При передаче сегмента отправитель включает таймер. Когда сегмент прибывает в пункт назначения, принимающая TCP-подсистема посылает обратно сегмент (с данными, если есть, что посылать, иначе — без данных) с номером подтверждения, равным порядковому номеру следующего ожидаемого сегмента, и новым размером окна. Если время ожидания подтверждения истекает, отправитель посылает сегмент еще раз.

Хотя этот протокол кажется простым, в нем имеется несколько деталей, которые следует рассмотреть подробнее. Сегменты могут приходить в неверном порядке. Так, например, возможна ситуация, в которой байты с 3072 по 4095 уже прибыли, но подтверждение для них не может быть выслано, так как байты с 2048 по 3071 еще не получены. К тому же сегменты могут задерживаться в сети так долго, что у отправителя истечет время ожидания, и он передаст их снова. Переданный повторно сегмент может включать в себя уже другие диапазоны фрагментов, поэтому потребуется очень аккуратное администрирование для определения номеров байтов, которые уже были приняты корректно. Тем не менее, поскольку каждый байт в потоке единственным образом определяется по своему сдвигу, эта задача оказывается реальной.

Протокол TCP должен уметь справляться с этими проблемами и решать их эффективно. На оптимизацию производительности TCP-потоков было потрачено много сил. В следующем разделе мы обсудим несколько алгоритмов, используемых в различных реализациях протокола TCP.

6.5.4. Заголовок TCP-сегмента

На рис. 6.31 показана структура заголовка TCP-сегмента. Каждый сегмент начинается с 20-байтного заголовка фиксированного формата. За ним могут следовать дополнительные поля (параметры). После дополнительных полей может располагаться до 65 535 – 20 – 20 = 65 495 байт данных, где первые 20 байт — это IP-заголовок, а вторые — TCP-заголовок. Сегменты могут и не содержать данных. Такие сегменты часто применяются для передачи подтверждений и управляющих сообщений.

Рассмотрим TCP-заголовок поле за полем. Поля *Порт получателя* и *Порт отправителя* являются идентификаторами локальных конечных точек соединения. TCP-порт вместе с IP-адресом хоста образуют уникальный 48-битный идентификатор конечной точки. Пара таких идентификаторов, относящихся к источнику и приемнику, однозначно определяет соединение. Этот идентификатор соединения называется **кортежем из пяти компонентов (5 tuple)**, так как он включает пять информационных составляющих: протокол (TCP), IP-адрес источника, порт источника, IP-адрес получателя и порт получателя.

Поля *Порядковый номер* и *Номер подтверждения* выполняют свою обычную функцию. Обратите внимание: поле *Номер подтверждения* относится к следующему по порядку ожидаемому байту, а не к последнему полученному. Это **накопительное**

подтверждение (cumulative acknowledgement), так как один номер объединяет в себе информацию обо всех полученных данных. Сфера его применения не выходит за рамки потерянных данных. Оба поля 32-разрядные, так как в TCP-потоке нумеруется каждый байт данных.

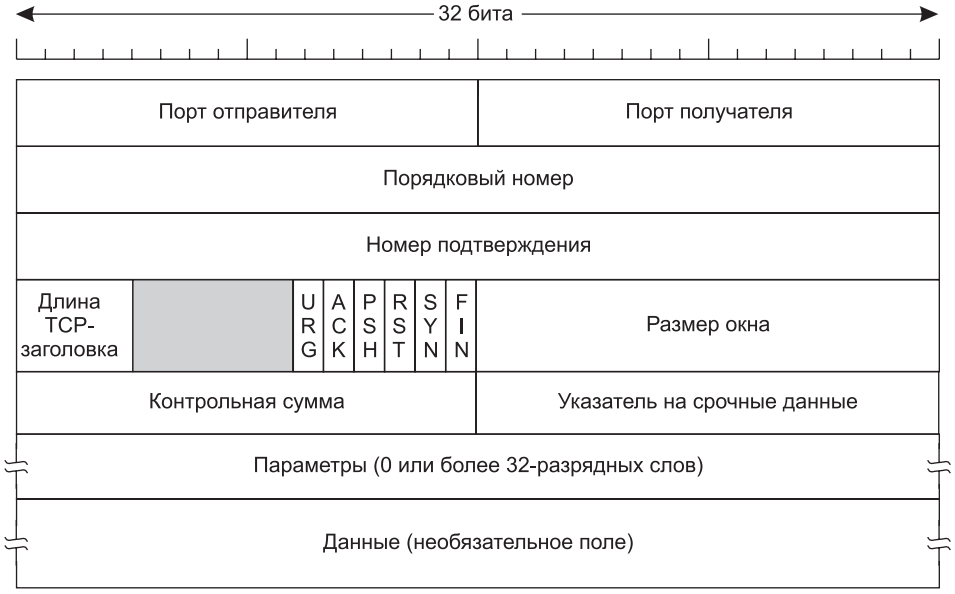


Рис. 6.31. Заголовок TCP

Поле *Длина TCP-заголовка* содержит размер TCP-заголовка, выраженный в 32-разрядных словах. Эта информация необходима, так как поле *Параметры*, а вместе с ним и весь заголовок может быть переменной длины. По сути, это поле указывает смещение от начала сегмента до поля данных, измеренное в 32-битных словах. Это то же самое, что длина заголовка.

Следом идет неиспользуемое 4-битное поле. Тот факт, что эти биты не используются уже в течение 30 лет (а востребованными оказываются только 2 бита изначально 6-битного поля), является свидетельством того, насколько хорошо продуман дизайн TCP. Иначе протоколы использовали бы эти биты, чтобы справиться с недостатками TCP.

Затем следуют восемь 1-битовых флагов. *CWR* и *ECE* сообщают о перегрузках сети в случае, если используется явное уведомление о насыщении (см. RFC 3168). Когда TCP-приемник узнает, что сеть перегружена, он с помощью флага *ECE* передает TCP-отправителю сигнал *ECN-эхо*, предлагая ему уменьшить скорость отправки. После того как TCP-приемник уменьшил скорость отправки, он сообщает об этом TCP-приемнику с помощью флага *CWR* с сигналом *Окно насыщения уменьшено*, после чего приемник перестает передавать сигнал *ECN-эхо*. Подробнее о роли явных уведомлений о перегрузке в контроле перегрузки TCP мы поговорим в разделе 6.5.10.

Бит *URG* устанавливается в 1 в случае использования поля *Указатель на срочные данные*, содержащего смещение в байтах от текущего порядкового номера байта до места расположения срочных данных. Таким образом, в протоколе TCP реализуются

прерывающие сообщения. Как уже упоминалось, этот метод позволяет отправителю передать получателю сигнал, не вовлекая в это TCP; он используется редко.

Если бит *ACK* установлен в 1, значит, поле *Номер подтверждения* содержит осмысленные данные. Для большинства пакетов это так. В противном случае данный сегмент не содержит подтверждения, и поле *Номер подтверждения* просто игнорируется.

Бит *PSH* является, по сути, PUSH-флагом, с помощью которого отправитель просит получателя доставить данные приложению сразу по получении пакета, а не хранить их в буфере, пока тот не наполнится. (Получатель может заниматься буферизацией для достижения большей эффективности.)

Бит *RST* используется для внезапного сброса состояния соединения, которое из-за сбоя хоста или по другой причине попало в тупиковую ситуацию. Кроме того, он используется для отказа от неверного сегмента или от попытки создать соединение. Если вы получили сегмент с установленным битом *RST*, это означает наличие какой-то проблемы.

Бит *SYN* применяется для установки соединения. У запроса соединения бит *SYN* = 1, а бит *ACK* = 0, это означает, что поле подтверждения не используется. Но в ответе на этот запрос содержится подтверждение, поэтому значения этих битов в нем равны: *SYN* = 1, *ACK* = 1. Таким образом, бит *SYN* используется для обозначения как сегментов CONNECTION REQUEST, так и CONNECTION ACCEPTED, а бит *ACK* — чтобы отличать их друг от друга.

Бит *FIN* используется для разрыва соединения. Он указывает на то, что у отправителя больше нет данных для передачи. Однако, даже закрыв соединение, процесс может продолжать получать данные в течение неопределенного времени. У сегментов с битами *FIN* и *SYN* есть порядковые номера, что гарантирует правильный порядок их выполнения.

Управление потоком в протоколе TCP осуществляется при помощи скользящего окна переменного размера. Поле *Размер окна* сообщает, сколько байт может быть послано после байта, получившего подтверждение. Значение поля *Размер окна* может быть равно нулю, это означает, что все байты вплоть до *Номер подтверждения*–1 получены, но получатель еще не обработал эти данные, и поэтому остальные байты он пока принять не может. Разрешение на дальнейшую передачу может быть получено путем отправки сегмента с таким же значением поля *Номер подтверждения* и ненулевым значением поля *Размер окна*.

В главе 3 мы обсуждали протоколы, в которых подтверждения приема кадров были связаны с разрешениями на продолжение передачи. Эта связь была следствием жестко закрепленного размера скользящего окна в этих протоколах. В TCP подтверждения отделены от разрешений на передачу данных. В сущности, приемник может сказать: «Я получил байты вплоть до *k*-го, но я сейчас не хочу продолжать прием данных». Такое разделение (выражающееся в скользящем окне *переменного размера*) придает протоколу дополнительную гибкость. Ниже мы обсудим этот аспект более детально.

Поле *Контрольная сумма* служит для повышения надежности. Как и в случае UDP, оно содержит контрольную сумму заголовка, данных и псевдозаголовка. Но в отличие от UDP псевдозаголовок этого поля содержит номер протокола TCP (6), а контрольная сумма является обязательной. Для более подробной информации см. раздел 6.4.1.

Поле *Параметры* предоставляет дополнительные возможности, не покрываемые стандартным заголовком. Существует множество параметров и некоторые из них

широко используются. Они имеют разную длину, кратную 32 битам (лишнее место заполняется нулями), и могут доходить до отметки в 40 байт — максимального размера заголовка TCP. При установлении соединения факультативные поля могут использоваться для того, чтобы договориться с противоположной стороной или просто сообщить ей о характеристиках этого соединения. Существуют поля, сохраняющиеся в течение всего времени жизни соединения. Все факультативные поля имеют формат Тип-Длина-Значение.

С помощью одного из таких полей каждый хост может указать **максимальный размер сегмента (MSS, Maximum Segment Size)**, который он может принять. Чем больше размер используемых сегментов, тем выше эффективность, так как при этом снижается удельный вес накладных расходов в виде 20-байтных заголовков, однако не все хосты способны принимать очень большие сегменты. Хосты могут сообщить друг другу максимальный размер сегмента во время установки соединения. По умолчанию этот размер равен 536 байтам. Все хосты обязаны принимать TCP-сегменты размером 536 + 20 = 556 байт. Для каждого из направлений может быть установлен свой максимальный размер сегмента.

Для линий с большой скоростью передачи и/или большой задержкой окно размером в 64 Кбайт, соответствующее 16-битному полю, оказывается слишком маленьким. Так, для линии OC-12 (приблизительно 600 Мбит/с) полное окно может быть передано в линию менее чем за 1 мс. Если значение времени распространения сигнала в оба конца составляет 50 мс (что типично для трансконтинентального оптического кабеля), 98 % времени отправитель будет заниматься ожиданием подтверждения. Больший размер окна мог бы улучшить эффективность. Параметр **масштаб окна** позволяет двум хостам договориться о масштабе окна при установке соединения. Это число позволяет обеим сторонам сдвигать поле *Размер окна* до 14 разрядов влево, обеспечивая расширение размера окна до 230 байт (1 Гбайт). Большинство реализаций протокола TCP поддерживают эту возможность.

Для **меток времени**, передаваемых от отправителя к получателю и обратно, существует одноименный параметр. Если во время настройки соединения этот параметр решено было использовать, он добавляется в каждый пакет. Он позволяет получить данные о круговой задержке, которые, в свою очередь, используются для определения потери пакетов. Также он используется в качестве логического расширения 32-битного порядкового номера. При быстром соединении порядковые номера могут проходить полный круг очень быстро, и в результате новые и старые данные будет невозможно отличить. **Детектирование повторного использования порядковых номеров (PAWS, Protection Against Wrapped Sequence numbers)** удаляет сегменты со старыми метками времени, позволяя избежать этой проблемы. Наконец, с помощью **выборочного подтверждения (SACK, Selective ACKnowledgement)** получатель может сообщать отправителю диапазоны порядковых номеров доставленных пакетов. Этот параметр является дополнением к *Номеру подтверждения* и используется в случаях, когда после потери пакета данные все равно были доставлены (возможно, в виде копии). Новые данные не отражены в поле заголовка *Номер подтверждения*, так как оно содержит только следующий по порядку ожидаемый байт. Если использовать выборочное подтверждение, отправитель всегда будет знать, какие данные есть у получателя, и сможет выполнять повторную передачу только тогда, когда это действительно нужно. Выбо-

рочное подтверждение описано в RFC 2108 и RFC 2883. В последнее время эта схема используется все чаще. О ее применении при контроле перегрузки мы поговорим в разделе 6.5.10.

6.5.5. Установка TCP-соединения

В протоколе TCP соединения устанавливаются с помощью «тройного рукопожатия». Чтобы установить соединение, одна сторона (например, сервер) пассивно ожидает входящего соединения, выполняя базовые операции LISTEN и ACCEPT, либо указывая конкретный источник, либо не указывая его.

Другая сторона (например, клиент) выполняет операцию CONNECT, указывая IP-адрес и порт, с которым она хочет установить соединение, максимальный размер TCP-сегмента и, по желанию, некоторые данные пользователя (например, пароль). Примитив CONNECT посылает TCP-сегмент с установленным битом *SYN* и сброшенным битом *ACK* и ждет ответа.

Когда этот сегмент прибывает в пункт назначения, TCP-подсистема проверяет, выполнил ли какой-нибудь процесс операцию LISTEN, указав в качестве параметра тот же порт, который содержится в поле *Порт получателя*. Если такого процесса нет, она отвечает отправкой сегмента с установленным битом *RST* для отказа от соединения.

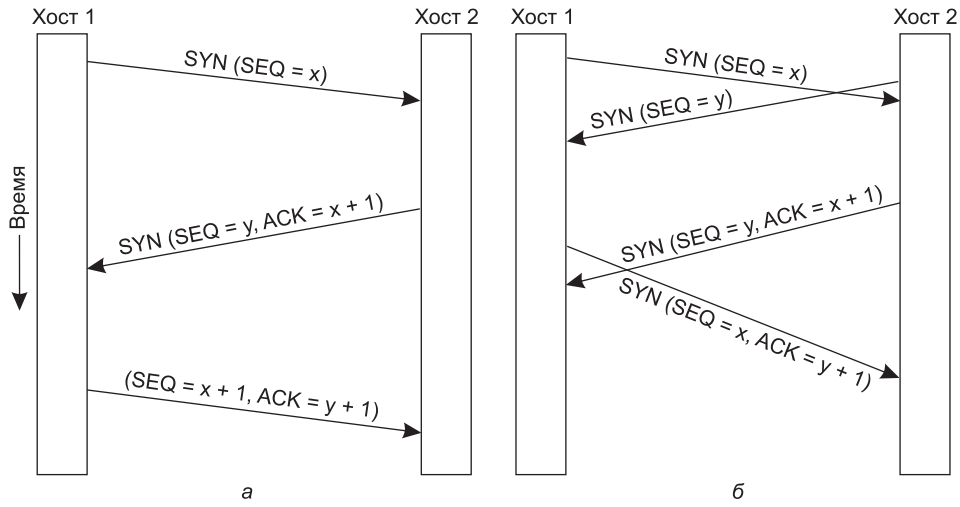


Рис. 6.32. Установка TCP-соединения в нормальном случае (*а*); одновременная установка соединения обеими сторонами (*б*)

Если какой-либо процесс прослушивает указанный порт, то входящий TCP-сегмент передается этому процессу. Последний может принять соединение или отказаться от него. Если процесс принимает соединение, он отсылает в ответ подтверждение. Последовательность TCP-сегментов, посылаемых в нормальном случае, показана на рис. 6.32, *а*. Обратите внимание на то, что сегмент с установленным битом *SYN* занимает 1 байт пространства порядковых номеров, что позволяет избежать неоднозначности в их подтверждениях.

Если два хоста одновременно попытаются установить соединение друг с другом, то последовательность происходящих при этом событий будет соответствовать рис. 6.32, *б*. В результате будет установлено только одно соединение, а не два, так как пара конечных точек однозначно определяет соединение. То есть если оба соединения пытаются идентифицировать себя с помощью пары (x, y), делается всего одна табличная запись для (x, y).

Если вы помните, начальное значение порядкового номера, выбранное каждым отдельным хостом, должно медленно меняться, а не равняться константе (например, нулю). Как мы уже говорили в разделе 6.2.2, это правило обеспечивает защиту от задержавшихся копий пакетов. Изначально эта схема была реализована с помощью таймера, изменяющего свое состояние каждые 4 мкс.

Однако проблема реализации схемы «тройного рукопожатия» состоит в том, что слушающий процесс должен помнить свой порядковый номер до тех пор, пока он не отправит свой собственный *SYN*-сегмент. Это значит, что злонамеренный отправитель может блокировать ресурсы хоста, отправляя на него поток *SYN*-сегментов и не разрывая соединение. Такие атаки называются **затоплением SYN-сегментами (SYN flood)**. В 1990-е годы из-за них многие веб-серверы оказались парализованными.

Один из методов, позволяющих обезопасить себя от таких атак, называется **SYN cookies**. Вместо того чтобы запоминать порядковый номер, хост генерирует криптографическое значение номера, записывает его в исходящий сегмент и забывает. Если «тройное рукопожатие» завершается, этот порядковый номер (увеличенный на 1) вернется на хост. Хост может повторно сгенерировать правильный порядковый номер, вычислив значение той же криптографической функции, при условии, что известны входные данные (это может быть IP-адрес и порт другого хоста, а также какое-то секретное значение). С помощью этой процедуры хост может проверять правильность подтвержденного порядкового номера без необходимости отдельно запоминать этот номер. Одна из тонкостей этого метода состоит в том, что он не работает с дополнительными полями (параметрами) TCP. Поэтому SYN cookies можно использовать только в случае затопления SYN-сегментами. Но в целом это очень интересный прием. Более подробно см. RFC 4987 и (Lemon, 2002).

6.5.6. Разрыв соединения TCP

Хотя TCP-соединения являются полнодуплексными, чтобы понять, как происходит их разъединение, лучше считать их парами симплексных соединений. Каждое симплексное соединение разрывается независимо от своего напарника. Чтобы разорвать соединение, любая из сторон может послать TCP-сегмент с установленным в единицу битом *FIN*, это означает, что у него больше нет данных для передачи. Когда этот TCP-сегмент получает подтверждение, это направление передачи закрывается. Тем не менее данные могут продолжать передаваться неопределенно долго в противоположном направлении. Соединение разрывается, когда оба направления закрываются. Обычно для разрыва соединения требуются четыре TCP-сегмента: по одному с битом *FIN* и по одному с битом *ACK* в каждом направлении. Первый бит *ACK* и второй бит *FIN* могут также содержаться в одном TCP-сегменте, что уменьшит количество сегментов до трех.

Как при телефонном разговоре, когда оба участника могут одновременно попрощаться и повесить трубки, оба конца TCP-соединения могут послать *FIN*-сегменты в одно и то же время. Они оба получают обычные подтверждения, и соединение закрывается. По сути, между одновременным и последовательным разъединением нет никакой разницы.

Чтобы избежать проблемы двух армий (см. раздел 6.2.3), используются таймеры. Если ответ на посланный *FIN*-сегмент не приходит в течение двух максимальных интервалов времени жизни пакета, отправитель *FIN*-сегмента разрывает соединение. Другая сторона, в конце концов, заметит, что ей никто не отвечает, и также разорвет соединение. Хотя такое решение и не идеально, но, учитывая недостижимость идеала, приходится пользоваться тем, что есть. На практике проблемы возникают довольно редко.

6.5.7. Модель управления TCP-соединением

Этапы, необходимые для установки и разрыва соединения, могут быть представлены в виде конечного автомата, 11 состояний которого перечислены в табл. 6.5. В каждом из этих состояний могут происходить разрешенные и запрещенные события. В ответ на какое-либо разрешенное событие может осуществляться определенное действие. При возникновении запрещенных событий сообщается об ошибке.

Таблица 6.5. Состояния конечного автомата, управляющего TCP-соединением

Состояние	Описание
CLOSED	Закрыто. Соединение не является активным и не находится в процессе установления
LISTEN	Ожидание. Сервер ожидает входящего запроса
SYN RCVD	Прибыл запрос соединения. Ожидание подтверждения
SYN SENT	Запрос соединения послан. Приложение начало открывать соединение
ESTABLISHED	Установлено. Нормальное состояние передачи данных
FIN WAIT 1	Приложение сообщило, что ему больше нечего передавать
FIN WAIT 2	Другая сторона согласна разорвать соединение
TIME WAIT	Ожидание, пока в сети не исчезнут все пакеты
CLOSING	Обе стороны попытались одновременно закрыть соединение
CLOSE WAIT	Другая сторона инициировала разъединение
LAST ACK	Ожидание, пока в сети не исчезнут все пакеты

Каждое соединение начинается в состоянии *CLOSED* (закрытое). Оно может покинуть это состояние, предпринимая либо активную (`CONNECT`), либо пассивную (`LISTEN`) попытку открыть соединение. Если противоположная сторона осуществляет противоположные действия, соединение устанавливается и переходит в состояние *ESTABLISHED*. Инициатором разрыва соединения может выступить любая сторона. По завершении процесса разъединения соединение возвращается в состояние *CLOSED*.

Конечный автомат показан на рис. 6.33. Типичный случай клиента, активно соединяющегося с пассивным сервером, показан жирными линиями — сплошными для клиента и пунктирными для сервера. Тонкие линии обозначают необычные последовательности событий. Каждая линия на рис. 6.33 маркирована парой *событие/действие*. Событие может представлять собой либо обращение пользователя к системной процедуре (`CONNECT`, `LISTEN`, `SEND` или `CLOSE`), либо прибытие сегмента (*SYN*, *FIN*, *ACK* или *RST*), либо, в одном случае, окончание периода ожидания, равного двойному времени жизни пакетов. Действие может состоять в отправке управляющего сегмента (*SYN*, *FIN* или *RST*). Впрочем, может не предприниматься никакого действия, что обозначается прочерком. В скобках приводятся комментарии.

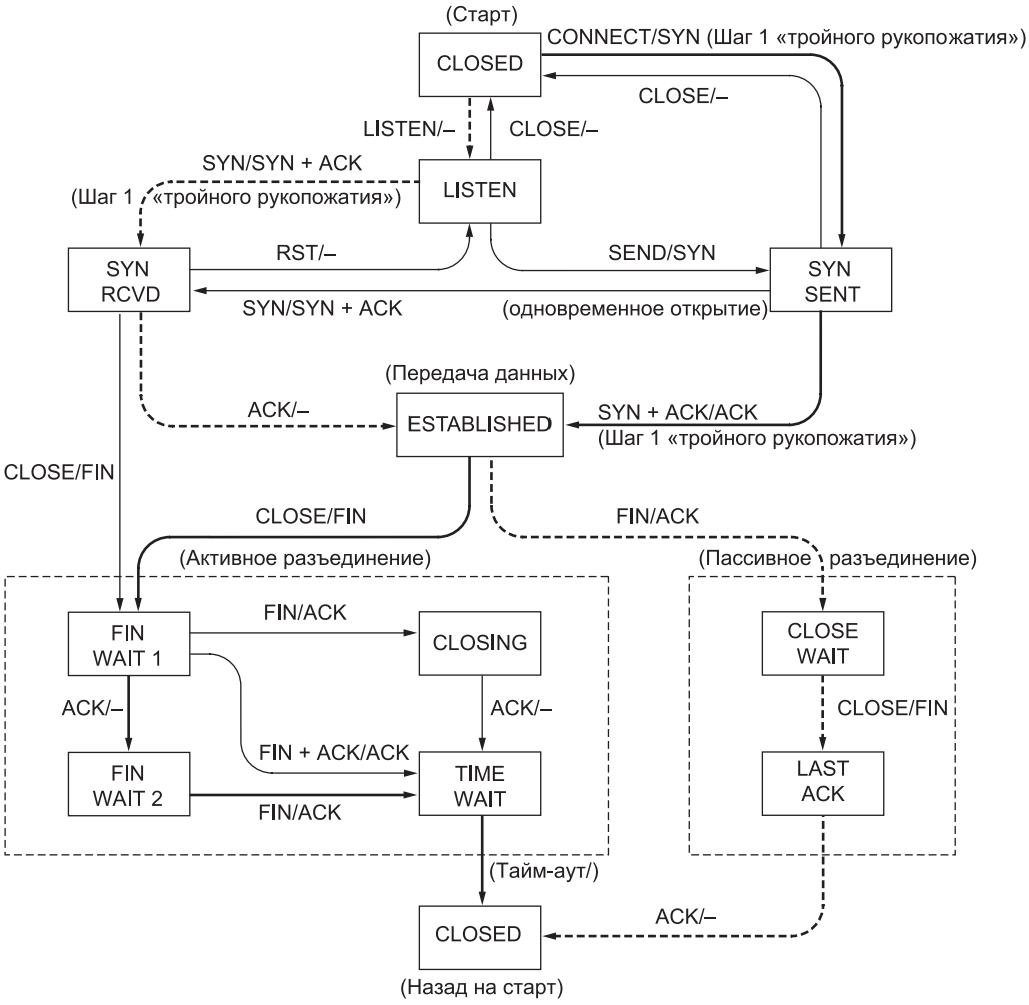


Рис. 6.33. Конечный автомат TCP-соединения. Жирная сплошная линия показывает нормальный путь клиента. Жирным пунктиром показан нормальный путь сервера. Тонкими линиями обозначены необычные события

Диаграмму легче всего понять, если сначала проследовать по пути клиента (сплошная жирная линия), а затем — по пути сервера (жирный пунктир). Когда приложение на машине клиента вызывает операцию CONNECT, локальная TCP-подсистема создает запись соединения, помечает его состояние как *SYN SENT* и посылает *SYN*-сегмент. Примечательно, что несколько приложений одновременно могут открыть несколько соединений, поэтому свое состояние, хранящееся в записи соединения, имеется у каждого отдельного соединения. Когда прибывает сегмент *SYN + ACK*, TCP-подсистема посылает последний *ACK*-сегмент «тройного рукопожатия» и переключается в состояние *ESTABLISHED*. В этом состоянии можно пересылать и получать данные.

Когда у приложения заканчиваются данные для передачи, оно выполняет операцию CLOSE, заставляющую локальную TCP-подсистему послать *FIN*-сегмент и ждать ответного *ACK*-сегмента (пунктирный прямоугольник с пометкой «активное разъединение»). Когда прибывает подтверждение, происходит переход в состояние *FIN WAIT 2*, и одно направление соединения закрывается. Когда приходит встречный *FIN*-сегмент, в ответ на него также высылается подтверждение, и второе направление соединения также закрывается. Теперь обе стороны соединения закрыты, но TCP-подсистема выжидает в течение времени, равного удвоенному максимальному времени жизни пакета, чтобы можно было гарантировать, что все пакеты этого соединения больше не перемещаются по сети даже в том случае, если подтверждение было потеряно. Когда этот период ожидания истекает, TCP-подсистема удаляет запись о соединении.

Рассмотрим теперь управление соединением с точки зрения сервера. Сервер выполняет операцию LISTEN и переходит в режим ожидания запросов соединения. Когда приходит *SYN*-сегмент, в ответ на него высылается подтверждение, после чего сервер переходит в состояние *SYN RCVD* (запрос соединения получен). Когда в ответ на *SYN*-подтверждение сервера от клиента приходит *ACK*-сегмент, процедура «тройного рукопожатия» завершается, и сервер переходит в состояние *ESTABLISHED*. Теперь можно пересылать данные.

По окончании передачи данных клиент выполняет операцию CLOSE, в результате чего на сервер прибывает *FIN*-сегмент (пунктирный прямоугольник, обозначенный как пассивное разъединение). Теперь сервер выполняет операцию CLOSE, а *FIN*-сегмент посылается клиенту. Когда от клиента прибывает подтверждение, сервер разрывает соединение и удаляет запись о нем.

6.5.8. Скользящее окно TCP

Как уже было сказано выше, управление окном в TCP решает проблемы подтверждения корректной доставки сегментов и выделения буферов на приемнике. Например, предположим, что у получателя есть 4096-байтовый буфер, как показано на рис. 6.34. Если отправитель передает 2048-байтовый сегмент, который успешно принимается получателем, то получатель подтверждает его получение. Однако при этом у получателя остается всего лишь 2048 байт свободного буферного пространства (пока приложение не заберет сколько-нибудь данных из буфера), о чем он и сообщает отправителю, указывая соответствующий размер окна (2048) и номер следующего ожидаемого байта.

Необходимо исправить все обозначения «1К» на «1 Кбайт», «2К» на «2 Кбайт», «4К» на «4 Кбайт», а «3К» на «2 Кбайт» (для соответствия оригиналу).

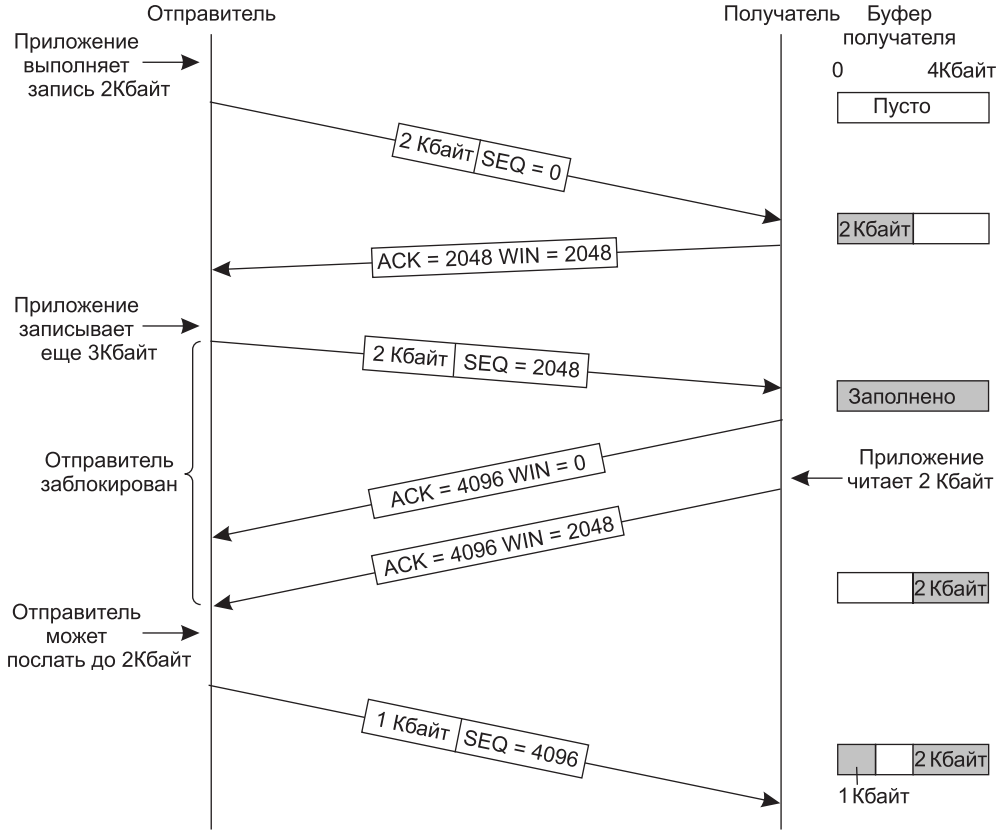


Рис. 6.34. Управление окном в TCP

После этого отправитель посылает еще 2048 байт, получение которых подтверждается, но размер окна объявляется равным 0. Отправитель должен прекратить передачу до тех пор, пока получающий хост не освободит место в буфере и не увеличит размер окна.

При нулевом размере окна отправитель не может посылать сегменты, за исключением двух случаев. Во-первых, разрешается посылать срочные данные, например, чтобы пользователь мог уничтожить процесс, выполняющийся на удаленной машине. Во-вторых, отправитель может послать 1-байтовый сегмент, прося получателя повторить информацию о размере окна и ожидаемом следующем байте. Такой пакет называется **пробным сегментом (window probe)**. Стандарт TCP явно предусматривает эту возможность для предотвращения тупиковых ситуаций в случае потери объявления о размере окна.

Отправители не обязаны передавать данные сразу как только они приходят от приложения. Также никто не требует от получателей посылать подтверждения как можно скорее. Например, на рис. 6.34 TCP-подсистема, получив от приложения первые 2 Кбайт данных и зная, что размер окна равен 4 Кбайт, была бы совершенно права, если бы просто сохранила полученные данные в буфере до тех пор, пока не прибудут

еще 2 Кбайт данных, чтобы передать сразу сегмент с 4 Кбайтами полезной нагрузки. Эта свобода действий может использоваться для улучшения производительности.

Рассмотрим соединение (к примеру, telnet или SSH) с удаленным терминалом, реагирующим на каждое нажатие клавиши. В худшем случае, когда символ прибывает к передающей TCP-подсистеме, она создает 21-байтовый TCP-сегмент и передает его IP-уровню, который, в свою очередь, посылает 41-байтовую IP-дейтаграмму. На принимающей стороне TCP-подсистема немедленно отвечает 40-байтовым подтверждением (20 байт TCP-заголовка и 20 байт IP-заголовка). Затем, когда удаленный терминал прочитает этот байт из буфера, TCP-подсистема пошлет обновленную информацию о размере буфера, передвинув окно на 1 байт вправо. Размер этого пакета также составляет 40 байт. Наконец, когда удаленный терминал обработает этот символ, он отправляет обратно эхо, передаваемое 41-байтовым пакетом. Итого для каждого введенного с клавиатуры символа пересылается четыре пакета общим размером 162 байта. В условиях дефицита пропускной способности линий этот метод работы нежелателен.

Для улучшения ситуации многие реализации TCP используют **отложенные подтверждения (delayed acknowledgements)**. Идея этого метода состоит в том, чтобы задерживать подтверждения и обновления размера окна на время до 500 мс в надежде получить дополнительные данные, вместе с которыми можно будет отправить подтверждение одним пакетом. Если терминал успеет выдать эхо в течение 500 мс, удаленной стороне нужно будет выслать только один 41-байтовый пакет, таким образом, нагрузка на сеть снизится вдвое.

Хотя отложенные подтверждения и снижают нагрузку на сеть, тем не менее эффективность использования сети отправителем, передающим множество маленьких пакетов (к примеру, 41-байтовые пакеты с 1 байтом реальных данных), продолжает оставаться невысокой. Метод, позволяющий повысить эффективность, известен как **алгоритм Нагля (Nagle's algorithm)** (Nagle, 1984). Предложение Нагля звучит довольно просто: если данные поступают отправителю маленькими порциями, отправитель просто передает первый фрагмент, а остальные помещает в буфер, пока не будет получено подтверждение приема первого фрагмента. После этого можно переслать все накопленные в буфере данные в виде одного TCP-сегмента и снова начать буферизацию до получения подтверждения о доставке следующего сегмента. Таким образом, в каждый момент времени может передаваться только один маленький пакет. Если за время прохождения пакета в оба конца приложение отправляет много порций данных, алгоритм Нагля объединяет несколько таких порций в один сегмент, и, таким образом, нагрузка на сеть существенно снижается. Кроме того, согласно этому алгоритму новый пакет должен быть отправлен, если объем данных в буфере превышает максимальный размер сегмента.

Алгоритм Нагля широко применяется различными реализациями протокола TCP, однако иногда бывают ситуации, в которых его лучше отключить. В частности, интерактивным играм через Интернет обычно требуется быстрый поток маленьких пакетов с обновлениями. Если буферизировать эти данные для пакетной пересылки, игра будет работать неправильно, и пользователи будут недовольны. Более тонкая проблема заключается в том, что иногда при задержке подтверждений использование алгоритма Нагля может приводить к временным тупиковым ситуациям: получатель ждет данных, к которым можно присоединить подтверждение, а отправитель ждет

подтверждения, без которого не будут переданы новые данные. Из-за этого, в частности, может задерживаться загрузка веб-страниц. На случай таких проблем существует возможность отключения алгоритма Нагля (параметр *TCP_NODELAY*). Подробнее об этих и других решениях см. Mogul и Minshall (2001).

Еще одна проблема, способная значительно снизить производительность протокола TCP, известна под именем **синдрома глупого окна (silly window syndrome)** (Clark, 1982). Суть проблемы состоит в том, что данные пересылаются TCP-подсистемой крупными блоками, но принимающая сторона интерактивного приложения считывает их посимвольно. Чтобы ситуация стала понятнее, рассмотрим рис. 6.35. Начальное состояние таково: TCP-буфер приемной стороны полон (то есть размер его окна равен 0), и отправителю это известно. Затем интерактивное приложение читает один символ из TCP-потока. Принимающая TCP-подсистема радостно сообщает отправителю, что размер окна увеличился и что он теперь может послать 1 байт. Отправитель повинуется и посылает 1 байт. Буфер снова оказывается полон, о чем получатель и извещает, посылая подтверждение для 1-байтового сегмента с нулевым размером окна. И так может продолжаться вечно.

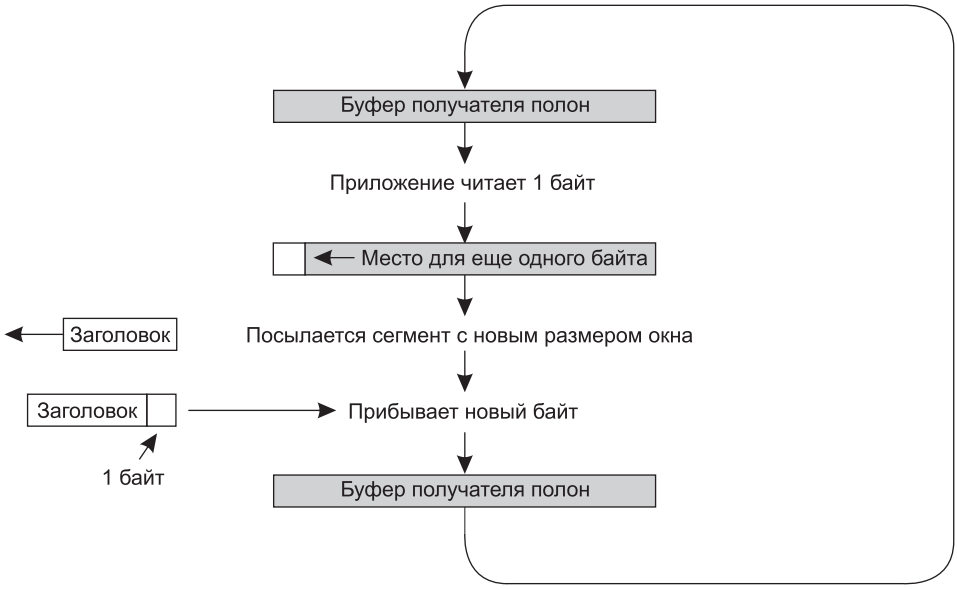


Рис. 6.35. Синдром глупого окна

Дэвид Кларк (David Clark) предложил запретить принимающей стороне отправлять информацию об однобайтовом размере окна. Вместо этого получатель должен подождать, пока в буфере не накопится значительное количество свободного места. В частности, получатель не должен отправлять сведения о новом размере окна до тех пор, пока он не сможет принять сегмент максимального размера, который он объявлял при установке соединения, или его буфер не освободился хотя бы наполовину. Кроме того, увеличению эффективности отправки может способствовать сам отправитель, отказываясь от отправки слишком маленьких сегментов. Вместо этого он должен

подождать, пока размер окна не станет достаточно большим для того, чтобы можно было послать полный сегмент или, по меньшей мере, равный половине размера буфера получателя.

В задаче избавления от синдрома глупого окна алгоритм Нагля и решение Кларка дополняют друг друга. Нагль пытался решить проблему приложения, предоставляющего данные TCP-подсистеме посимвольно. Кларк старался разрешить проблему приложения, посимвольно получающего данные у TCP. Оба решения хороши и могут работать одновременно. Суть их состоит в том, чтобы не посылать и не просить передавать данные слишком малыми порциями.

Принимающая TCP-подсистема может пойти еще дальше в деле повышения производительности, просто обновляя информацию о размере окна большими порциями. Как и отправляющая TCP-подсистема, она также может буферизировать данные и блокировать запрос на чтение READ, поступающий от приложения, до тех пор, пока у нее не накопится большого объема данных. Таким образом, снижается количество обращений к TCP-подсистеме (и вместе с ними накладные расходы). Конечно, такой подход увеличивает время ожидания ответа, но для неинтерактивных приложений, например, при передаче файла, сокращение времени, затраченного на всю операцию, значительно важнее увеличения времени ожидания ответа на отдельные запросы.

Еще одна проблема получателя состоит в том, что сегменты могут приходить в неправильном порядке. Они могут храниться в буфере до тех пор, пока их нельзя будет передать приложению в правильном порядке. В принципе нет ничего плохого в том, чтобы отвергать пакеты, прибывшие не в свою очередь, ведь они все равно будут повторно переданы отправителем. Однако такая схема, очевидно, работает неэффективно.

Подтверждение может быть выслано, только если все данные вплоть до подтверждаемого байта получены. Это называется **накопительным подтверждением**. Если до получателя доходят сегменты 0, 1, 2, 4, 5, 6 и 7, он может подтвердить получение данных вплоть до последнего байта сегмента 2. Когда у отправителя истечет время ожидания, он передаст сегмент 3 еще раз. Если к моменту прибытия сегмента 3 получатель сохранит в буфере сегменты с 4 по 7, он сможет подтвердить получение всех байтов, вплоть до последнего байта сегмента 7.

6.5.9. Управление таймерами в TCP

В протоколе TCP используется много различных таймеров (по крайней мере, такова концепция). Наиболее важным из них является **таймер повторной передачи (RTO, Retransmission TimeOut)**. Когда посылается сегмент, запускается таймер повторной передачи. Если подтверждение получения сегмента прибывает раньше, чем истекает период таймера, таймер останавливается. Если же, наоборот, период ожидания истечет раньше, чем прибудет подтверждение, сегмент передается еще раз (а таймер запускается снова). Соответственно возникает вопрос: каким должен быть интервал времени ожидания?

На транспортном уровне эта проблема оказывается значительно сложнее, чем в протоколах канального уровня, таких как 802.11. В последнем случае величина ожидаемой задержки измеряется в микросекундах, и ее довольно легко предсказать (ее

разброс невелик), поэтому таймер можно установить на момент времени чуть позднее ожидаемого прибытия подтверждения (рис. 6.36, *а*). Поскольку на канальном уровне подтверждения редко запаздывают на больший срок (благодаря тому, что нет заторов), отсутствие подтверждения в течение установленного временного интервала с большой вероятностью означает потерю кадра или подтверждения.

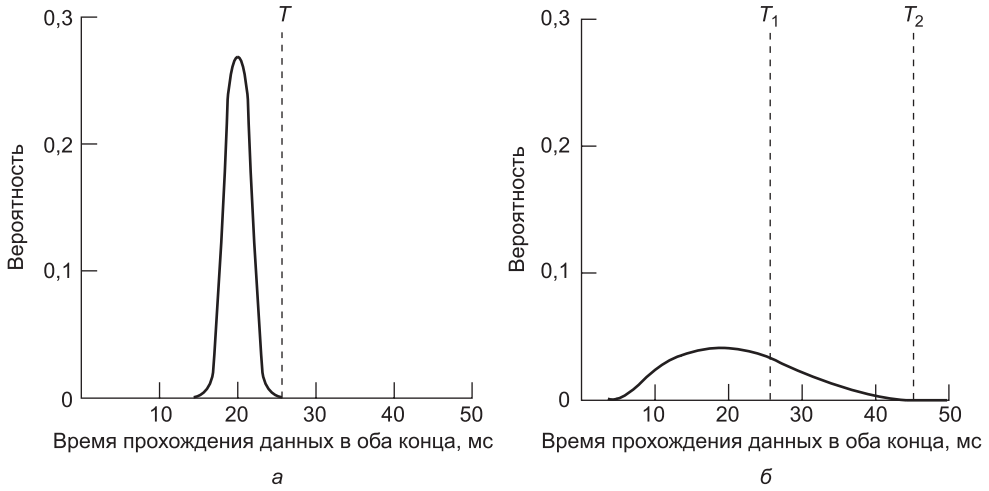


Рис. 6.36. Плотность вероятности времени прибытия подтверждения: *а* — на канальном уровне; *б* — для TCP

Протокол TCP вынужден работать в совершенно иных условиях. Функция распределения плотности вероятности для времени прибытия подтверждения на этом уровне выглядит значительно более полого (рис. 6.36, *б*), а вариативность достаточно велика. Поэтому предсказать, сколько потребуется времени для прохождения данных от отправителя до получателя и обратно, весьма непросто. Если выбрать величину интервала ожидания слишком малой (например, T_1 на рис. 6.36, *б*), возникнут излишние повторные передачи, забивающие Интернет бесполезными пакетами. Если же установить значение этого интервала слишком большим (T_2), то из-за увеличения времени ожидания в случае потери пакета пострадает производительность. Более того, среднее значение и величина дисперсии времени прибытия подтверждений может изменяться всего за несколько секунд при возникновении и устранении перегрузки.

Решение заключается в использовании динамического алгоритма, постоянно изменяющего величину периода ожидания, основываясь на измерениях производительности сети. Алгоритм, применяемый в TCP, разработан Джекобсоном (Jacobson) в 1988 году и работает следующим образом. Для каждого соединения в протоколе TCP предусмотрена переменная *SRTT* (**Smoothed Round-Trip Time** — **усредненная круговая задержка**), в которой хранится текущее наилучшее ожидаемое время получения подтверждения для данного соединения. При передаче сегмента запускается таймер, который замеряет время, требуемое для получения подтверждения, а также запускает повторную передачу, если подтверждение не приходит в срок. Если подтверждение успевает вернуться, прежде чем истечет период ожидания, TCP-подсистема замеряет

время, потребовавшееся для его получения (*R*). Затем значение переменной *SRTT* обновляется по следующей формуле:

$$SRTT = \alpha SRTT + (1 - \alpha)R,$$

где α — весовой коэффициент, определяющий, насколько быстро забываются старые значения. Обычно он равен 7/8. Эта формула — **взвешенное скользящее среднее (EWMA, Exponentially Weighed Moving Average)** или фильтр низких частот, с помощью которого можно удалять шум.

Даже при известном значении переменной *SRTT* выбор периода ожидания подтверждения оказывается задачей нетривиальной. В первых реализациях TCP это значение вычислялось как 2*xRTT*, однако экспериментально было показано, что постоянный множитель является слишком негибким и плохо учитывает ситуации, при которых разброс значений времени прибытия подтверждения увеличивается. В частности, модели очередей случайного (то есть пуассоновского) трафика показывают, что когда нагрузка приближается к пропускной способности, задержка и ее разброс увеличиваются. В результате может сработать таймер повторной передачи, после чего будет отправлена копия пакета, хотя оригинальный пакет все еще будет в сети. Как правило, такие ситуации возникают именно при высокой нагрузке — и это не самое лучшее время для того, чтобы отправлять в сеть лишние пакеты.

Чтобы решить эту проблему, Джекобсон предложил сделать интервал времени ожидания чувствительным к отклонению круговой задержки и к усредненной круговой задержке. Для этого потребовалась еще одна сглаженная переменная **RTTVAR (Round-Trip Time Variation, отклонение круговой задержки)**, которая вычисляется по формуле:

$$RTTVAR = \beta RTTVAR + (1 - \beta)|SRTT - R|.$$

Как и в предыдущем случае, это взвешенное скользящее среднее. Как правило, $\beta = 3/4$. Значение интервала ожидания, *RTO*, устанавливается по формуле:

$$RTO = SRTT + 4 \times RTTVAR.$$

Выбор множителя 4 является произвольным, однако умножение целого числа на 4 может быть выполнено одной командой сдвига, а относительное количество излишних повторных передач при таком значении времени ожидания не превысит одного процента. Обратите внимание на то, что *RTTVAR* является не стандартным, а средним отклонением, но на практике их значения оказываются достаточно близкими. В работе Джекобсона приводится множество способов вычисления значений интервала ожидания с помощью только целочисленного сложения, вычитания и сдвига. Хотя для современных хостов такая экономия не представляет интереса, ее использования требует идея повсеместной применимости TCP: он должен работать как на суперкомпьютерах, так и на небольших устройствах. Для RFID-чипов его пока еще никто не реализовал, но вдруг? Кто знает, может быть.

Более подробные сведения о том, как вычислять это значение интервала ожидания, а также начальные значения переменных, можно найти в RFC 2988. Для таймера повторной передачи минимальное значение также устанавливается равным 1 с, независимо от предварительной оценки. Это значение нужно для того, чтобы избежать лишних повторных передач, основанных на измерениях (Allman и Paxson, 1999).

При получении данных для вычисления круговой задержки, R, возникает вопрос, что делать при повторной передаче сегмента. Когда наконец прибывает подтверждение для такого сегмента, непонятно, относится это подтверждение к первой передаче пакета или же к последней. Неверная догадка может серьезно снизить значение интервала ожидания. Эта проблема была обнаружена радиолюбителем Филом Карном (Phil Karn). Его интересовал вопрос передачи TCP/IP-пакетов с помощью коротковолновой любительской радиосвязи, известной своей ненадежностью. Предложение Ф. Карна было очень простым: не обновлять оценки для переданных повторно пакетов. Кроме того, при каждой повторной передаче время ожидания можно удваивать до тех пор, пока сегменты не пройдут с первой попытки. Это исправление получило название **алгоритма Карна (Karn's algorithm)** (Karn и Partridge, 1987). Он применяется в большинстве реализаций протокола TCP.

В протоколе TCP используется не только таймер повторной передачи. Еще один применяемый в этом протоколе таймер называется **таймером настойчивости (persistence timer)**. Он предназначен для предотвращения следующей тупиковой ситуации. Получатель посылает подтверждение, в котором указывает окно нулевого размера, давая тем самым отправителю команду подождать. Через некоторое время получатель посылает пакет с новым размером окна, но этот пакет теряется. Теперь обе стороны ожидают действий противоположной стороны. Когда срабатывает таймер настойчивости, отправитель посылает получателю пакет с вопросом, не изменилось ли текущее состояние. В ответ получатель сообщает текущий размер окна. Если он все еще равен нулю, таймер настойчивости запускается снова, и весь цикл повторяется. Если же окно увеличилось, отправитель может передавать данные.

В некоторых реализациях протокола используется третий таймер, называемый **таймером проверки активности (keepalive timer)**. Когда соединение не используется в течение долгого времени, срабатывает таймер проверки активности, заставляя одну сторону проверить, есть ли еще кто живой на том конце соединения. Если проверяющая сторона не получает ответа, соединение разрывается. Эта особенность протокола довольно противоречива, поскольку она приносит дополнительные накладные расходы и может разорвать вполне жизнеспособное соединение из-за кратковременной потери связи.

Последний таймер, используемый в каждом TCP-соединении, — это таймер, запускаемый в состоянии *TIME WAIT* конечного автомата при закрытии соединения. Он отсчитывает двойное время жизни пакета, чтобы гарантировать, что после закрытия соединения в сети не останутся созданные им пакеты.

6.5.10. Контроль перегрузки в TCP

Мы оставили себе на закуску одну из ключевых функций TCP: контроль перегрузки. Когда в какую-либо сеть поступает больше данных, чем она способна обработать, в сети образуются заторы. Интернет в этом смысле не является исключением. Когда сетевой уровень узнает о том, что на маршрутизаторах скопились длинные очереди, он пытается справиться с этой ситуацией, пусть даже простым удалением пакетов. В задачи транспортного уровня входит обратная связь с сетевым уровнем, что позволяет ему следить за перегрузкой и при необходимости снижать скорость отправки пакетов.

В Интернете TCP является незаменимым в процессе контроля перегрузки, так же как и при транспортировке данных.

Общие вопросы, касающиеся контроля перегрузки, мы уже обсуждали в разделе 6.3. Основная мысль заключалась в следующем: транспортный протокол, использующий закон управления **AIMD (Additive Increase Multiplicative Decrease, аддитивное увеличение мультипликативное уменьшение)** при получении двоичных сигналов сети о перегрузке, сходится к справедливому и эффективному распределению пропускной способности. Контроль перегрузки в TCP реализует этот подход с помощью окна, а в качестве сигнала используется потеря пакетов. В каждый момент времени в сети может находиться не более чем фиксированное число байт от каждого отправителя. Это число байт составляет размер **окна перегрузки**. Соответственно, скорость отправки равна размеру окна, деленному на круговую задержку соединения. Размер окна задается в соответствии с правилом AIMD.

Как вы наверняка помните, помимо окна перегрузки существует также окно управления потоком, определяющее число байт, которое получатель может поместить в буфер. Оба эти параметра играют важную роль: число байт, которое отправитель может передать в сеть, равно размеру меньшего из этих окон. Таким образом, эффективное окно — меньшее из того, что устраивает отправителя, и того, что устраивает получателя. Здесь необходимо участие обеих сторон. TCP остановит отправку данных, если одно из окон временно окажется заполненным. Если получатель говорит: «Посылайте 64 Кбайт», но отправитель знает, что если он пошлет более 32 Кбайт, то в сети образуется затор, он посылает все же 32 Кбайт. Если же отправитель знает, что сеть способна пропустить и большее количество данных, например 128 Кбайт, он передаст столько, сколько просит получатель (то есть 64 Кбайт). Окно управления потоком было описано ранее, поэтому в дальнейшем мы будем говорить только об окне перегрузки.

Современная схема контроля перегрузки была реализована в TCP во многом благодаря стараниям Ван Джекобсона (Van Jacobson, 1988). Это поистине захватывающая история. Начиная с 1986 года рост популярности Интернета привел к возникновению ситуаций, которые позже стали называть **отказом сети из-за перегрузки (congestion collapse)**, — длительных периодов, во время которых эффективная пропускная способность резко падала (более чем в 100 раз) из-за перегрузки сети. Джекобсон (и многие другие) решил разобраться в ситуации и придумать конструктивное решение.

В результате Джекобсону удалось реализовать решение высокого уровня, которое состояло в использовании метода AIMD для выбора окна перегрузки. Особенно интересно то, что при всей сложности контроля перегрузки в TCP он смог добавить его в уже существующий протокол, не изменив ни одного формата сообщений. Благодаря этому новое решение можно было сразу применять на практике. Сначала Джекобсон заметил, что потеря пакетов является надежным сигналом перегрузки, даже несмотря на то, что эта информация и приходит с небольшим опозданием (уже после возникновения перегрузки). В конце концов, трудно представить себе маршрутизатор, который не удаляет пакеты при перегрузке. И в дальнейшем это вряд ли изменится. Даже когда буферная память будет исчисляться терабайтами, скорость сетей, скорее всего, также возрастет до нескольких терабит в секунду.

Здесь есть одна тонкость. Дело в том, что использование потери пакетов в качестве сигнала перегрузки предполагает, что ошибки передачи происходят сравнительно

редко. В случае беспроводных сетей (например, 802.11) это не так, поэтому в них используются свои собственные механизмы повторной передачи данных на канальном уровне. Из-за особенностей повторной передачи в беспроводных сетях потеря пакетов на сетевом уровне, вызванная ошибками передачи, обычно не учитывается. В сетях, использующих провода и оптоволоконные линии, частота ошибок по битам, как правило, низкая.

Все алгоритмы TCP для Интернета основаны на том предположении, что пакеты теряются из-за перегрузок. Поэтому они внимательно следят за тайм-аутами и пытаются обнаружить любые признаки проблемы подобно тому, как шахтеры следят за своими канарейками. Чтобы узнавать о потере пакетов вовремя и с высокой точностью, необходим хороший таймер повторной передачи. Мы уже говорили о том, как таймеры повторной передачи в TCP учитывают среднее значение и отклонение круговой задержки. Усовершенствование таких таймеров за счет учета отклонений стало важным шагом в работе Джекобсона. Если время ожидания повторной передачи выбрано правильно, TCP-отправитель может следить за количеством исходящих байт, нагружающих сеть. Для этого ему необходимо просто сравнить порядковые номера переданных и подтвержденных пакетов.

Теперь наша задача выглядит вполне простой. Все, что нам нужно, — это следить за размером окна перегрузки (с помощью порядковых номеров и номеров подтверждений) и изменять его, следуя правилу AIMD. Но, как вы уже догадались, на самом деле все гораздо сложнее. Первая сложность заключается в том, что способ отправки пакетов в сеть (даже за короткие промежутки времени) должен зависеть от сетевого пути. Иначе возникнет затор. Для примера рассмотрим хост с окном насыщения 64 Кбайт, подключенный к коммутируемой сети Ethernet со скоростью 1 Гбит/с. Если хост отправит целое окно за один раз, всплеск трафика может пойти через медленную ADSL-линию (1 Мбит/с). Этот трафик, прошедший по гигабитной линии за половину миллисекунды, на целых полсекунды парализует работу медленной линии, полностью блокируя такие протоколы, как VoIP. Так может работать протокол, нацеленный на создание перегрузок, но не на борьбу с ними.

Однако отправка небольших порций пакетов может оказаться полезной. На рис. 6.37 показано, что произойдет, если хост-отправитель, подключенный к быстрой линии (1 Гбит/с), отправит небольшую порцию пакетов (4) получателю, находящемуся в медленной сети (1 Мбит/с), которая является узким местом пути или его самой медленной частью. Сначала эти 4 пакета будут перемещаться по сети с той скоростью, с которой они будут отправлены. Затем маршрутизатор поместит их в очередь, так как они будут прибывать по высокоскоростной линии быстрее, чем передаваться по медленной линии. Но эта очередь не будет длинной, поскольку число пакетов, отправленных за один раз, невелико. Обратите внимание на то, что на рисунке пакеты, проходящие по медленной линии, выглядят длиннее, так как для их отправки требуется больше времени, чем по быстрой линии.

Наконец, пакеты попадают на приемник, где подтверждается их получение. Время отправки подтверждения зависит от времени прибытия пакета по медленному каналу. Поэтому на обратном пути расстояние между пакетами будет больше, чем в самом начале, когда исходные пакеты перемещались по быстрому каналу. Оно не изменится на протяжении всего прохождения подтверждений через сеть и обратно.

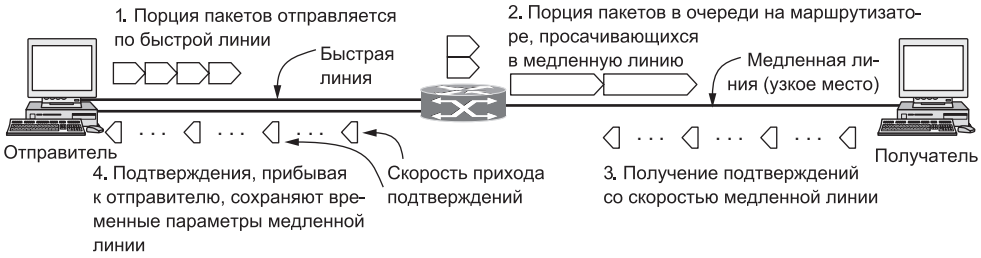


Рис. 6.37. Порция пакетов, переданная отправителем, и скорость прихода подтверждений

Здесь особенно важно следующее: подтверждения прибывают к отправителю примерно с той же скоростью, с которой пакеты могут передаваться по самому медленному каналу пути. Именно эта скорость и нужна отправителю. Если он будет передавать пакеты в сеть с такой скоростью, они будут перемещаться настолько быстро, насколько позволяет самый медленный канал, но зато не будут застревать в очередях на маршрутизаторах. Такая скорость называется **скоростью прихода подтверждений (ack clock)** и является неотъемлемой частью TCP. Этот параметр позволяет TCP выровнять трафик и избежать ненужных очередей на маршрутизаторах.

Вторая сложность состоит в том, что достижение хорошего рабочего режима по правилу AIMD в быстрых сетях потребует очень большого времени, если изначально выбрано маленькое окно перегрузки. Рассмотрим средний сетевой путь, позволяющий передавать трафик со скоростью 10 Мбит/с и круговой задержкой 100 мс. В данном случае удобно использовать окно перегрузки, равное произведению пропускной способности и времени задержки, то есть 1 Мбит или 100 пакетов по 1250 байт. Если изначально взять окно размером один пакет и увеличивать это значение на один пакет через интервал времени, равный круговой задержке, соединение начнет работать с нормальной скоростью только через 100 круговых задержек, то есть через 10 с. Это слишком долго. Теоретически мы могли бы начать с большего окна — скажем, размером 50 пакетов. Но для медленных линий это значение будет слишком большим. Тогда при отправке 50 пакетов за один раз возникнет затор — о таком сценарии мы говорили выше.

Решение, предложенное Джекобсоном, объединяет линейное и мультипликативное увеличение. При установлении соединения отправитель задает маленькое окно размером не более четырех сегментов. (Изначально начальный размер окна не превышал один сегмент, но впоследствии это значение на основании экспериментов было увеличено до четырех.) Подробнее об этом рассказывается в RFC 3390. Затем отправитель передает в сеть начальное окно. Получение пакетов подтвердится через время, равное круговой задержке. В каждом случае, когда подтверждение о получении сегмента приходит до срабатывания таймера повторной передачи, отправитель увеличивает окно перегрузки на длину одного сегмента (в байтах). Кроме того, если сегмент был получен, то в сети стало на один сегмент меньше. Поэтому в результате каждый подтвержденный сегмент позволяет отправить еще два сегмента. Таким образом, размер окна насыщения увеличивается вдвое через интервал времени, равный круговой задержке.

Этот алгоритм называется **медленным стартом (slow start)**, однако на самом деле он не медленный — он растет экспоненциально, — особенно в сравнении с предыду-

щим алгоритмом, который позволяет отправлять целое окно управления потоком за один раз. Медленный старт показан на рис. 6.38. Во время первой круговой задержки (RTT) отправитель передает в сеть один пакет (и приемник получает один пакет). Во время второй круговой задержки передается два пакета, во время третьей — четыре.

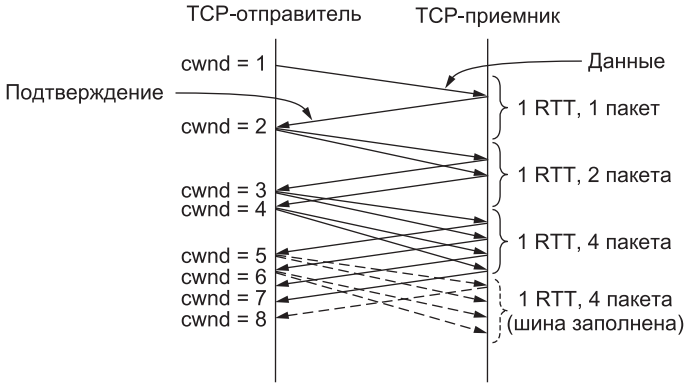


Рис. 6.38. Медленный старт с начальным окном перегрузки в один сегмент

Медленный старт хорошо работает для широкого диапазона значений скорости и круговой задержки. Чтобы регулировать скорость отправки в зависимости от сетевого пути, он использует скорость прихода подтверждений. Посмотрим на то, как подтверждения возвращаются от отправителя к получателю (рис. 6.38). Когда отправитель получает подтверждение, он увеличивает окно насыщения на единицу и сразу же передает в сеть два пакета. (Один пакет соответствует увеличению окна на единицу, а второй передается взамен пакета, прибывшего на место назначения и таким образом покинувшего сеть. В каждый момент времени число неподтвержденных пакетов определяется окном перегрузки.) Однако эти два пакета не обязательно придут на хост-получатель с таким же интервалом, с каким они были отправлены. Пусть, например, отправитель подключен к сети Ethernet мощностью 100 Мбит/с. На отправку каждого 1250-байтного пакета уходит 100 мкс. Поэтому интервал между пакетами может быть маленьким, от 100 мкс. Ситуация меняется, если путь следования пакетов проходит через ADSL-линию мощностью 1 Мбит/с. Теперь для отправки такого же пакета требуется 10 мс. Таким образом, минимальный интервал между пакетами возрастает, по меньшей мере, в 100 раз. Этот интервал так и останется большим, если только в какой-то момент пакеты не будут все вместе ожидать отправки в одном буфере.

На рис. 6.38 описанный эффект можно увидеть, если посмотреть на интервал прибытия пакетов на приемник. Этот интервал сохраняется при отправке подтверждений и, следовательно, при их получении отправителем. Если сетевой путь медленный, подтверждения приходят медленно (через время, равное круговой задержке). Если сетевой путь быстрый, подтверждения приходят быстро (опять же через время, равное круговой задержке). Отправитель должен просто учитывать скорость прихода подтверждений при отправке новых пакетов, — а это и делает алгоритм медленного старта.

Поскольку алгоритм медленного старта приводит к экспоненциальному росту, в какой-то момент (скорее рано, чем поздно) в сеть будет слишком быстро отправлено

слишком много пакетов. В результате на маршрутизаторах выстроятся очереди. Когда очереди переполняются, происходит потеря пакетов. В таком случае, когда подтверждение о доставке этого пакета не придет на хост-отправитель, сработает соответствующий таймер. На рис. 6.38 можно увидеть, в какой момент окно насыщения оказалось слишком большим. Через три круговых задержки в сети находится четыре пакета. Чтобы добраться до получателя, этим пакетам требуется время, равное целой круговой задержке. Это значит, что для данного соединения подходит окно перегрузки размером в четыре пакета. Но так как получение этих пакетов подтверждается, алгоритм медленного старта продолжает увеличивать окно перегрузки, и в результате через одну круговую задержку отправитель посылает в сеть восемь пакетов. Независимо от того, сколько пакетов отправлено, только четыре из них успевают дойти до места назначения за одну круговую задержку. Это значит, что сетевая шина заполнена. Дополнительные пакеты, попадая в сеть, будут застревать в очередях на маршрутизаторах, так как сеть не может достаточно быстро доставлять их получателю. Вскоре возникнут заторы и потери пакетов.

Чтобы контролировать медленный старт, отправитель должен хранить в памяти пороговое значение для каждого соединения. Оно называется **порогом медленного старта (slow start threshold)**. Изначально устанавливается произвольное высокое значение, не превышающее размер окна управления потоком, чтобы оно никак не ограничивало возможности соединения. Используя алгоритм медленного старта, TCP продолжает увеличивать окно перегрузки, пока не произойдет тайм-аут или размер окна перегрузки не превысит пороговое значение (или не заполнится окно получателя).

При обнаружении потери пакета (например, в ситуации тайм-аута) порог медленного старта устанавливается равным половине окна перегрузки, и весь процесс начинается заново. Идея в том, что если текущее окно перегрузки слишком велико, оно вызывает перегрузку, которую удается зафиксировать с опозданием. Вдвое меньший размер окна, при котором перегрузок не было, лучше подходит для окна перегрузки: пропускная способность пути используется довольно эффективно, а потерь данных не происходит. В нашем примере на рис. 6.38 увеличение окна до восьми пакетов может вызвать потерю данных, тогда как окно размером четыре пакета хорошо подходит. После этого размер окна перегрузки устанавливается равным начальному значению, и алгоритм медленного старта выполняется сначала.

При превышении порога медленного старта TCP переключается с медленного старта на аддитивное увеличение. В этом режиме окно перегрузки увеличивается на один сегмент через промежутки времени, равные круговой задержке. Как и в случае медленного старта, увеличение происходит по мере получения подтверждений о доставке, а не ровно на один сегмент на каждом круге. Пусть *cwnd* — окно перегрузки, а *MSS* — максимальный размер сегмента. Обычно увеличение окна производится с коэффициентом $(MSS \times MSS)/cwnd$ для каждого из $cwnd/MSS$ пакетов, доставка которых может быть подтверждена. Это увеличение не обязано быть быстрым. Общая идея состоит в том, чтобы TCP-соединение максимально долго работало с размером окна, близким к оптимальному, — не слишком маленьким, чтобы пропускная способность не была низкой, и не слишком большим, чтобы не возникало заторов.

Аддитивное увеличение показано на рис. 6.39. Ситуация та же, что и для медленного старта. В конце каждого круга окно перегрузки отправителя увеличивается так, что

в сеть может быть передан один дополнительный пакет. По сравнению с медленным стартом линейная скорость роста оказывается очень низкой. Для маленьких окон разница будет не очень существенной, но она станет ощутимой, если, к примеру, вам нужно увеличить окно на 100 сегментов.

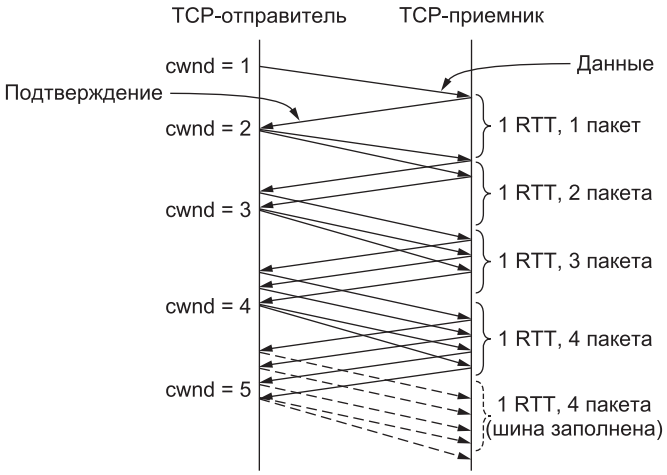


Рис. 6.39. Аддитивное увеличение при начальном окне размером один сегмент

Здесь кое-что можно улучшить, тем самым повысив производительность. Дело в том, что недостатком этой схемы является ожидание тайм-аута. Время ожидания подтверждения обычно бывает большим, так как его оценка должна быть заниженной. Если пакет будет потерян, приемник не отправит соответствующее подтверждение, так что номер подтверждения не изменится. Отправитель не сможет отправить в сеть новые пакеты, так как его окно все еще будет полным. В таком состоянии хост может пробыть довольно долго, пока не сработает таймер и не произойдет повторная передача пакета. В этот момент медленный старт начинается заново.

Для отправителя существует быстрый способ узнать, что один из его пакетов потерян. По мере того как пакеты, следующие за потерянным пакетом, прибывают на приемник, они инициируют отправку подтверждений, которые приходят к отправителю. Все они имеют один и тот же номер подтверждения и называются **дубликатами подтверждений (duplicate acknowledgements)**. Каждый раз, когда отправитель получает дубликат подтверждения, есть вероятность, что другой пакет уже пришел, а потерянный — нет.

Так как пакеты могут следовать разными путями, они могут приходить в неправильном порядке. В таком случае дубликаты подтверждений не будут означать потерю пакетов. Однако в Интернете такое случается достаточно редко. Если в результате прохождения пакетов по разным путям порядок пакетов все же нарушается, он нарушается не сильно. Поэтому в TCP условно считается, что три дубликата подтверждений сигнализируют о потере пакета. Также по номеру подтверждения можно установить, какой именно пакет потерян. Это следующий по порядку пакет. Его повторную передачу можно выполнить сразу, не дожидаясь, пока сработает таймер.

Этот эвристический метод получил название **быстрый повтор (fast retransmisson)**. Когда это происходит, порог медленного старта все равно устанавливается равным половине текущего окна перегрузки, как и в случае тайм-аута. Медленный старт можно начать заново, взяв окно размером в один сегмент. Новый пакет будет отправлен через время, за которое успеет прийти подтверждение для повторно переданного пакета, а также все остальные данные, переданные в сеть до обнаружения потери пакета.

На данный момент мы имеем алгоритм контроля перегрузки, работа которого проиллюстрирована на рис. 6.40. Эта версия называется TCP Tahoe в честь 4.2BSD Tahoe, выпущенного в 1988 году, где эти идеи были реализованы. Максимальный размер сегмента в данном примере равен 1 Кбайт. Сначала окно перегрузки было установлено равным 64 Кбайт, но затем произошел тайм-аут, и порог стал равным 32 Кбайт, а окно перегрузки — 1 Кбайт (передача 0). Затем размер окна перегрузки удваивается на каждом шаге, пока не достигает порога (32 Кбайт). Размер окна увеличивается каждый раз, когда приходит новое подтверждение, то есть не непрерывно, поэтому мы и имеем дискретный ступенчатый график. На каждом круге размер окна увеличивается на один сегмент.

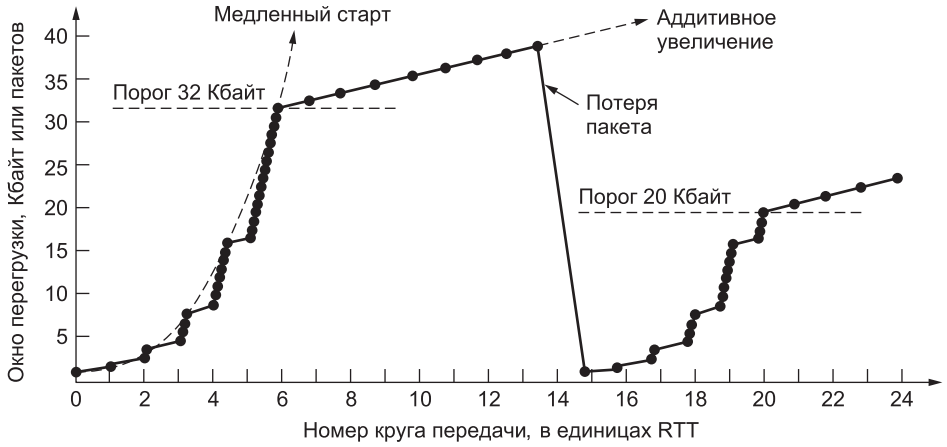


Рис. 6.40. Медленный старт и последующее аддитивное увеличение в TCP Tahoe

Передачи на круге 13 оказываются неудачными (как и положено), и одна из них заканчивается потерей пакета. Отправитель обнаруживает это после получения трех дубликатов подтверждений. После этого потерянный пакет передается повторно, а пороговое значение устанавливается равным половине текущего размера окна (40 Кбайт пополам, то есть 20 Кбайт), и опять происходит медленный старт. До запуска нового медленного старта с окном в один сегмент проходит еще один круг, за который все ранее переданные данные, включая копию потерянного пакета, успевают покинуть сеть. Окно перегрузки снова увеличивается в соответствии с алгоритмом медленного старта до тех пор, пока оно не дойдет до порогового значения в 20 Кбайт. После этого рост размера окна опять становится линейным. Так будет продолжаться до следующей потери пакета, которая будет обнаружена с помощью дубликатов подтверждений или после наступления тайм-аута (или же до заполнения окна получателя).

Версия TCP Tahoe (в которой, кстати, используются хорошие таймеры повторной передачи) реализует работающий алгоритм контроля перегрузки, решающий проблему отказа сети из-за перегрузки. Джекобсон придумал, как сделать его еще лучше. Во время быстрой повторной передачи соединение работает с окном слишком большого размера, но скорость прихода подтверждений продолжает учитываться. Каждый раз, когда приходит дубликат подтверждения, велика вероятность того, что еще один пакет покинул сеть. Таким образом можно считать общее число пакетов в сети и продолжать отправку нового пакета при получении каждого дополнительного дубликата подтверждения.

Эвристический метод, реализующий эту идею, получил название **быстрое восстановление (fast recovery)**. Это временный режим, позволяющий не останавливать учет скорости прихода подтверждений в тот момент, когда порогом медленного старта становится текущий размер окна или его половина (во время быстрой повторной передачи). Для этого считаются дубликаты подтверждений (включая те три, которые инициировали быструю повторную передачу) до тех пор, пока число пакетов в сети не снизится до нового порогового значения. На это уходит примерно половина круговой задержки. Начиная с этого момента для каждого полученного дубликата подтверждения отправитель может передавать в сеть новый пакет. Через один круг после быстрой повторной передачи получение потерянного пакета подтвердится. В этот момент дубликаты подтверждений перестанут приходить сплошным потоком, и алгоритм выйдет из режима быстрого восстановления. Окно перегрузки станет равным новому порогу медленного старта и начнет увеличиваться линейно.

В итоге этот метод позволяет избежать медленного старта в большинстве ситуаций, за исключением случаев установления нового соединения и возникновения таймаутов. Последнее может произойти, если теряется более чем один пакет, а быстрая повторная передача не помогает. Вместо того чтобы снова и снова начинать с медленного старта, окно перегрузки активного соединения перемещается по линиям аддитивного увеличения (на один сегмент за круг) и мультипликативного уменьшения (в два раза за круг), имеющим пилообразный вид. Это и есть правило AIMD, которое мы с самого начала хотели реализовать.

Такое пилообразное движение показано на рис. 6.41. Данный метод используется в TCP Reno, который назван в честь выпущенного в 1990 году 4.3BSD Reno. По сути, TCP Reno — это TCP Tahoe с быстрым восстановлением. После начального медленного старта окно насыщения увеличивается линейно, пока отправитель не обнаружит потерю пакета, получив нужное количество дубликатов подтверждения. Потерянный пакет передается повторно, и далее алгоритм работает в режиме быстрого восстановления, который дает возможность не останавливать учет скорости прихода подтверждений до тех пор, пока не прибудет подтверждение о доставке повторно переданного пакета. После этого окно перегрузки принимает значение, равное новому порогу медленного старта, а не 1. Это продолжается неопределенно долго. Почти все время размер окна перегрузки близок к оптимальному значению произведения пропускной способности и времени задержки.

Механизмы выбора размера окна, использующиеся в TCP Reno, более чем на два десятилетия стали основой контроля перегрузки в TCP. В течение этих лет механизмы претерпели ряд незначительных изменений — в частности, появились новые способы

выбора начального окна, были устранены неоднозначные ситуации. Усовершенствования коснулись и механизмов восстановления после потери двух или более пакетов. К примеру, версия TCP RenoNew использует номера частичных подтверждений, полученных после повторной передачи одного из потерянных пакетов, для восстановления другого потерянного пакета (Hoe, 1996) (см. RFC 3782). С середины 1990-х годов стали появляться варианты описанного выше алгоритма, основанные на других законах управления. К примеру, в системе Linux используется CUBIC TCP (Ha и др., 2008), а Windows включает вариант Compound TCP (Tan и др., 2006).

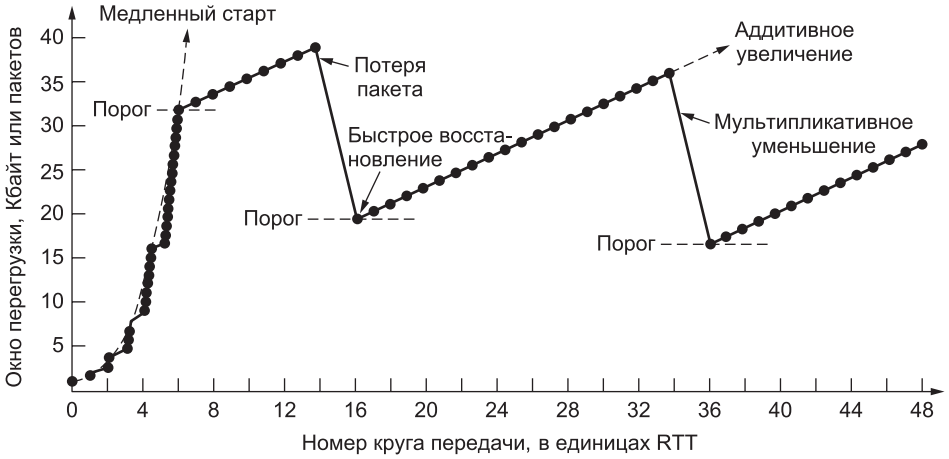


Рис. 6.41. Быстрое восстановление и пилообразный график для TCP Reno

Два более серьезных нововведения касаются реализаций TCP. Во-первых, сложность этого алгоритма заключается в том, что по дубликатам подтверждений необходимо определить, какие пакеты были потеряны, а какие — нет. Номер накопительного подтверждения не содержит такой информации. Простым решением стало использование **выборочных подтверждений** (**SACK**, **Selective ACKnowledgement**), в которых может содержаться до трех диапазонов успешно полученных байтов. Такая информация позволяет отправителю более точно определить, какие пакеты следует передать повторно, и следить за еще не доставленными пакетами.

При установлении соединения отправитель и получатель передают друг другу параметр *SACK permitted*, чтобы показать, что они могут работать с выборочными подтверждениями. Когда выборочные подтверждения включены, обмен данными происходит так, как показано на рис. 6.42. Получатель использует поле *Номер подтверждения* обычным способом — как накопительное подтверждение последнего по порядку полученного байта. Когда пакет 3 приходит к нему вне очереди (так как пакет 2 потерян), он отправляет *SACK option* для полученных данных вместе с накопительным подтверждением (дубликатом) для пакета 1. *SACK option* содержит информацию о диапазонах полученных байтов, которые располагаются после номера самого подтверждения. Первый диапазон — пакет, к которому относится дубликат подтверждения. Следующие диапазоны, если они есть, относятся к последующим блокам. Обычно используется не более трех диапазонов. К моменту прихода пакета 6 в выборочном

подтверждении используется уже два диапазона: первый указывает на получение пакетов 3 и 4, второй — на получение пакета 6 (вдобавок к пакетам, полученным до пакета 1). Основываясь на всех полученных *SACK option*, отправитель решает, какие пакеты следует передать повторно. В данном случае лучше всего взять пакеты 2 и 5.

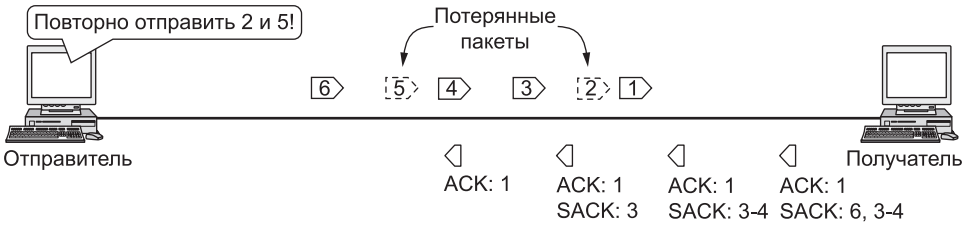


Рис. 6.42. Выборочные подтверждения

Выборочные подтверждения содержат рекомендательную информацию. На самом деле, обнаружение потери пакетов по дубликатам подтверждений и изменение окна перегрузки происходят так, как было описано выше. Тем не менее выборочные подтверждения упрощают процесс восстановления в ситуациях, когда несколько пакетов теряются примерно в одно и то же время, так как с их помощью отправитель узнает, какие пакеты не дошли до адресата. Выборочные подтверждения применяются далеко не везде. О них рассказывается в RFC 2883, а контроль управления TCP с использованием выборочных подтверждений описан в RFC 3517.

Второе усовершенствование состоит в использовании явных уведомлений о перегрузке (**ECN**, **Explicit Congestion Notification**) в качестве дополнительного сигнала о перегрузке. Явные уведомления о перегрузке представляют собой механизм IP-уровня, позволяющий сообщать хостам о перегрузке (см. раздел 5.3.4). С его помощью TCP-приемник может получать сигналы о насыщении от IP.

Явные уведомления о перегрузке включены для TCP-соединения, если при установке этого соединения отправитель и получатель сообщили друг другу о том, что они могут использовать такие уведомления, с помощью битов *ECE* и *CWR*. В таком случае в заголовке каждого пакета с TCP-сегментом указано, что этот пакет может передавать явные уведомления о перегрузке. Тогда при угрозе перегрузки маршрутизаторы, поддерживающие такие уведомления, будут помещать сигналы о перегрузке в пакеты, имеющие соответствующие флаги, вместо того чтобы удалять пакеты, когда перегрузка действительно возникнет.

Если какой-то из прибывших пакетов содержит явное уведомление о перегрузке, TCP-приемник узнает об этом и с помощью флага *ECE* (ECN-эхо) сообщает отправителю о перегрузке. Отправитель подтверждает получение этого сигнала с помощью флага *CWR* (Окно перегрузки уменьшено).

На такие сигналы отправитель реагирует точно так же, как и на потерю пакетов. Но теперь результат выглядит лучше: перегрузка обнаружена, хотя ни один пакет не пострадал. Явные уведомления о перегрузке описаны в RFC 3168. Так как им требуется поддержка как хостов, так и маршрутизаторов, в Интернете они не очень широко используются.

Полный список механизмов контроля перегрузки в TCP и их работа описаны в RFC 5681.

6.5.11. Будущее TCP

TCP — «рабочая лошадка» Интернета — используется многими приложениями; с течением времени разработчики видоизменяют и дополняют этот протокол, стараясь добиться высокой производительности в различных сетях. На сегодняшний день существует много версий, каждая из которых добавляет что-то новое к классическим алгоритмам, описанным здесь; особенно это касается контроля перегрузки и устойчивости к сетевым атакам. По всей вероятности, TCP будет развиваться параллельно с Интернетом. Здесь мы рассмотрим два частных вопроса.

Во-первых, протокол TCP не решает проблем транспортной семантики, актуальных для многих приложений. К примеру, некоторые приложения хотят, чтобы границы передаваемых ими сообщений или записей сохранялись. Другие приложения хотят работать с группами взаимосвязанных сообщений — как, например, веб-браузер, загружающий несколько объектов с одного сервера. Некоторым приложениям нужны дополнительные возможности управления сетевыми путями. TCP со своим стандартным интерфейсом сокетов не в состоянии выполнить эти требования. В результате каждое приложение вынуждено самостоятельно решать проблемы, которые не по силам TCP. Это привело к возникновению новых протоколов со слегка измененным интерфейсом. Среди них **SCTP (Stream Control Transmission Protocol, протокол передачи с управлением потоками)**, описанный в RFC 4960, и **SST (Structured Stream Transport, иерархическая поточная транспортировка данных)** (Ford, 2007). Но, как известно, всегда, когда кто-то предлагает изменить проверенный веками механизм, возникает два противоборствующих лагеря: «Пользователи хотят новых возможностей» и «Если ничего не сломано, не надо ничего чинить».

Второй вопрос касается контроля перегрузки. После такого подробного обсуждения различных механизмов и их усовершенствований этот вопрос мог показаться решенным. Это не так. Форма контроля перегрузки, описанная выше и достаточно популярная, использует в качестве сигнала о перегрузке потерю пакетов. При моделировании пропускной способности с помощью пилообразной схемы Padhye и его коллеги (1998) обнаружили, что при увеличении скорости частота потери пакетов должна резко снижаться. Чтобы добиться пропускной способности 1 Гбит/с при круговой задержке 100 мс и размере пакета 1500 байт, один пакет может теряться каждые 10 минут. Это соответствует частоте потери пакетов $2 \times 10 - 8$, а это очень мало. Потеря пакетов будет происходить слишком редко, так что этот параметр не может служить хорошим сигналом о перегрузке; число пакетов, потерянных по другой причине (например, ошибки при передаче происходят с частотой $10 - 7$), может легко превысить это значение, что приведет к снижению пропускной способности.

До сих пор это не представлялось актуальным, но по мере того как сети становятся все более быстрыми, разработчики возвращаются к вопросу контроля перегрузки. Один из возможных вариантов — использовать новый алгоритм, в котором потеря пакетов вообще не учитывается. Несколько примеров было приведено в разделе 6.2. Сигналом перегрузки может быть, например, круговая задержка, которая растет при перегрузках, как в FAST TCP (Wei и др., 2006). Возможны и другие подходы; какой из них лучше — покажет время.

6.6. Вопросы производительности

Вопросы производительности играют важную роль в компьютерных сетях. Когда сотни или тысячи компьютеров соединены вместе, их взаимодействие становится очень сложным и может привести к непредсказуемым последствиям. Часто эта сложность приводит к низкой производительности, причины которой довольно трудно определить. В следующих разделах мы рассмотрим многие вопросы, связанные с производительностью сетей, определим круг существующих проблем и обсудим методы их разрешения.

К сожалению, понимание производительности сетей — это скорее искусство, чем наука. Теоретическая база, допускающая хоть какое-то практическое применение, крайне скудна. Лучшее, что мы можем сделать, это представить несколько практических методов, полученных в результате долгих экспериментов, а также привести несколько реально действующих примеров. Мы намеренно отложили эту дискуссию до того момента, когда будет изучен транспортный уровень в сетях TCP, чтобы иметь возможность иллюстрировать некоторые места примерами из TCP.

В следующих разделах мы рассмотрим основные аспекты производительности сети.

1. Причины снижения производительности.
2. Измерение производительности сети.
3. Проектирование хостов для быстрых сетей.
4. Быстрая обработка сегментов.
5. Сжатие заголовка.
6. Протоколы для протяженных сетей с высокой пропускной способностью (long fat networks).

Эти аспекты позволяют рассмотреть производительность с точки зрения операций, выполняемых на хостах и при перемещении данных внутри сети, а также с точки зрения размера и быстродействия сети.

6.6.1. Причины снижения производительности компьютерных сетей

Некоторые виды снижения производительности вызваны временным отсутствием свободных ресурсов. Если на маршрутизатор вдруг прибывает трафик больше, чем он способен обработать, образуется затор, и производительность резко падает. Вопросы перегрузки подробно рассматривались в этой и предыдущей главах.

Производительность также снижается, если возникает структурный дисбаланс ресурсов. Например, если гигабитная линия связи присоединена к компьютеру с низкой производительностью, то несчастный центральный процессор не сможет достаточно быстро обрабатывать приходящие пакеты, что приведет к потере некоторых пакетов. Эти пакеты рано или поздно будут переданы повторно, что приведет к увеличению задержек, непроизводительному использованию пропускной способности и снижению общей производительности.

Перегрузка может также возникать синхронно. Например, если сегмент содержит неверный параметр (например, номер порта назначения), во многих случаях получатель заботливо пошлет обратно сообщение об ошибке. Теперь рассмотрим, что случится, если неверный сегмент будет разослан широковещательным способом 10 000 машинам. Каждая машина может послать обратно сообщение об ошибке. Образовавшийся в результате **широковещательный шторм (broadcast storm)** может надолго остановить нормальную работу сети. Протокол UDP страдал от подобной проблемы, пока протокол ICMP не был изменен так, чтобы хосты воздерживались от отправки сообщений об ошибке в ответ на широковещательные сегменты UDP. Беспроводным сетям особенно важно не отвечать на непроверенные широковещательные пакеты, поскольку их пропускная способность ограничена.

Второй пример синхронной перегрузки может быть вызван временным отключением электроэнергии. Когда питание снова включается, все машины одновременно начинают перезагружаться. Типичная последовательность загрузки может требовать обращения к какому-нибудь DHCP-серверу (серверу динамической конфигурации хоста), чтобы узнать свой истинный адрес, а затем к файловому серверу, чтобы получить копию операционной системы. Если сотни машин обратятся к серверу одновременно, он не сможет обслужить сразу всех.

Даже при отсутствии синхронной перегрузки и при достаточном количестве ресурсов производительность может снижаться из-за неверных системных настроек. Например, у машины может быть мощный процессор и много памяти, но недостаточно памяти выделено под буфер. В этом случае управление потоком данных замедлит получение сегментов и ограничит производительность. Такая проблема возникала для многих TCP-соединений, по мере того как Интернет становился более быстрым, а окно управления потоком было фиксированным (64 Кбайт).

Также на производительность могут повлиять неверно установленные значения таймеров ожидания. Когда посылается сегмент, обычно включается таймер, на случай если этот модуль потеряется. Выбор слишком короткого интервала ожидания подтверждения приведет к излишним повторным передачам сегментов. Если же интервал ожидания сделать слишком большим, это приведет к увеличению задержки в случае потери сегмента. К настраиваемым параметрам также относятся интервал ожидания попутного модуля данных для отправки подтверждения и количество попыток повторной передачи в случае отсутствия подтверждений.

Еще одна проблема, касающаяся приложений, работающих в реальном времени (например, воспроизводящих аудио и видео), — джиттер. Для хорошей производительности им недостаточно хорошей средней пропускной способности. Таким приложениям необходима низкая задержка. А для этого требуется эффективное распределение нагрузки на сеть, а также поддержка качества обслуживания на канальном и сетевом уровнях.

6.6.2. Измерение производительности сети

Когда качество работы сети оказывается не слишком хорошим, ее пользователи часто жалуются сетевым операторам, требуя усовершенствований. Чтобы улучшить производительность сети, операторы должны сначала точно определить, в чем суть

проблемы. Чтобы выяснить текущее состояние сети, операторы должны произвести измерения. В данном разделе мы рассмотрим вопрос измерения производительности сети. Приводимое ниже обсуждение основано на работе Могола (Mogul, 1993).

Измерения могут быть произведены разными способами и во многих местах (как физически, так и в стеке протоколов). Наиболее распространенный тип измерений представляет собой включение таймера при начале какой-либо активности и измерение продолжительности этой активности. Например, одним из ключевых измерений является измерение времени, необходимого для получения отправителем подтверждения в ответ на отправку сегмента. Другие измерения производятся при помощи счетчиков, в которых учитывается частота некоторых событий (например, количество потерянных сегментов). Наконец, часто измеряются такие количественные показатели, как число байтов, обработанных за определенный временной интервал.

Процедура измерения производительности сети и других параметров содержит множество подводных камней. Ниже мы перечислим некоторые из них. Следует тщательно избегать подобных ошибок при любых попытках измерить производительность сети.

Убедитесь, что выборка достаточно велика

Не следует ограничиваться единственным измерением какого-нибудь параметра — например, времени, необходимого для передачи одного сегмента. Повторите измерение, скажем, миллион раз и вычислите среднее значение. Начальные эффекты, например ситуации, когда после периода бездействия 802.16 NIC или кабельный модем получает зарезервированную пропускную способность, могут замедлить отправку первого сегмента, а помещение его в очередь увеличит разброс значений. Чем больше будет выборка, тем выше окажется точность оценки среднего значения и его среднеквадратичного отклонения. Погрешность может быть вычислена при помощи стандартных формул статистики.

Убедитесь, что выборка является репрезентативной

В идеале, следует повторить всю последовательность миллиона измерений параметров в различное время суток и в разные дни недели, чтобы заметить влияние различной загруженности системы на измеряемые параметры. Так, измерение перегрузки вряд ли принесет пользу, если эти измерения производить, когда перегрузки нет. Иногда результаты могут показаться на первый взгляд странными, как, например, наличие серьезных заторов в сети в 10, 11, 13 и 14 часов, но их отсутствие в полдень (когда все пользователи обедают).

В случае беспроводных сетей местоположение играет важную роль из-за распространения сигнала. Даже если узел, на котором выполняются измерения, находится рядом с беспроводным клиентом, он может не видеть некоторых пакетов, доступных клиенту, из-за различий в используемых антеннах. Измерения лучше проводить на хосте беспроводного клиента. Если это невозможно, необходимо выбрать несколько пунктов наблюдения и таким образом получить более полную картину (Mahajan и др., 2006).

Кэширование может сильно исказить ваши измерения

Если протоколы используют механизмы кэширования, многократное повторение измерений может дать неожиданные результаты. К примеру, обращение к веб-странице или просмотр имени DNS (чтобы найти IP-адрес) может в первый раз выполняться через сеть, а затем путем обращения к кэшу, при котором, естественно, пересылки пакетов не происходит. Результаты таких измерений будут абсолютно бесполезными (если только вы не хотите измерить производительность кэша).

Буферирование пакетов может производить аналогичный эффект. Одна популярная TCP/IP-программа измерения производительности славилась тем, что сообщала, что протокол UDP может достичь производительности, значительно превышающей максимально допустимую для данной физической линии. Как это происходило? Обращение к UDP обычно возвращает управление сразу, как только сообщение принимается ядром системы и добавляется в очередь на передачу. При достаточном размере буфера время выполнения 1000 обращений к UDP не означает, что за это время все данные были переданы в линию. Большая часть их все еще находится в буфере ядра.

Следует быть абсолютно уверенным в том, что вы понимаете, как происходит кэширование и буферизация данных.

Убедитесь, что во время ваших тестов не происходит ничего неожиданного

Если в то время, когда вы производите тестирование сети, какой-нибудь исследователь решит устроить в сети видеоконференцию, результаты могут отличаться от результатов, полученных в обычный день. Лучше всего запускать тесты на пустой системе, создавая всю нагрузку самому. Хотя и в этом случае можно ошибиться. Вы можете предполагать, что никто не пользуется сетью в 3 часа ночи, но может оказаться, что именно в это время программа автоматического резервного копирования начинает свою работу по архивации всех жестких дисков на магнитную ленту. Кроме того, именно в это время пользователи, находящиеся в другой временной зоне на другой стороне земного шара, могут создавать довольно сильный трафик, просматривая ваши замечательные веб-страницы.

Особые трудности возникают с беспроводными сетями, так как в этом случае часто бывает трудно отделить сигнал от помех, генерируемых различными источниками. Даже если рядом нет другой активной беспроводной сети, помехи могут возникнуть от того, что кто-то просто начнет готовить попкорн в микроволновой печи, и производительность 802.11 снизится. Поэтому рекомендуется следить за общей активностью сети, чтобы по крайней мере знать о возникновении непредвиденных ситуаций.

Используя часы с грубой шкалой, будьте внимательны

Компьютерные часы работают, добавляя единицу к некоему счетчику через равные интервалы времени. Например, миллисекундный таймер увеличивает на единицу значение счетчика раз в 1 мс. Применение такого таймера для измерения длитель-

ности события, занимающего менее 1 мс, возможно, но требует осторожности. Конечно, некоторые компьютеры используют более точные часы, но каким бы ни был шаг таймера, всегда найдется событие, которое происходит за меньшее время. Также стоит помнить, что точность часов не всегда совпадает с точностью возвращаемого ими значения.

Например, чтобы измерить время, необходимое для передачи одного сегмента, следует считывать показания системных часов (скажем, в миллисекундах) при входе и выходе из программы транспортного уровня. Если время, требуемое для передачи одного сегмента, равно 300 мкс, то измеряемая величина будет равна либо 0, либо 1 мс, что неверно. Тем не менее если повторить измерения миллион раз, сложить все значения и разделить на миллион, то полученное среднее значение будет отличаться от истинного значения менее чем на 1 мкс.

Будьте осторожны с экстраполяцией результатов

Предположим, вы что-нибудь измеряете (например, время ответа удаленного хоста), моделируя нагрузку сети от 0 (простой) до 0,4 (40 % максимальной пропускной способности). Пусть время ответа при отправке VoIP-пакета по сети 802.11 будет таким, как показано жирной линией на рис. 6.43. Может оказаться соблазнительным линейно экстраполировать полученную кривую (пунктир). Однако в действительности многие параметры в теории массового обслуживания содержат в качестве сомножителя $1/(1 - \rho)$, где ρ — нагрузка, поэтому истинная кривая зависимости будет больше походить на гиперболу, показанную штриховой линией, особенно при большой нагрузке. Таким образом, следует обращать внимание на влияние конкуренции, которое усиливается при большой нагрузке.

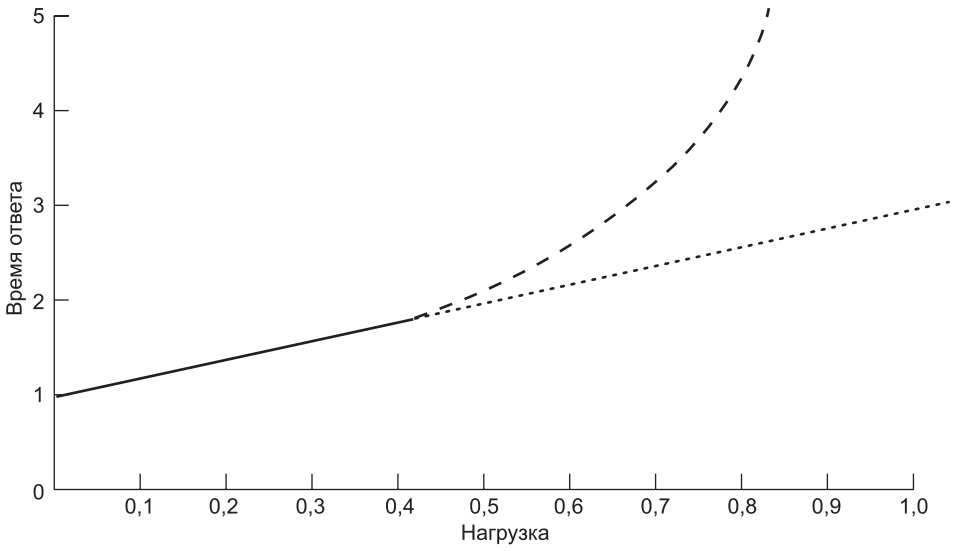


Рис. 6.43. Зависимость времени ответа от нагрузки

6.6.3. Проектирование хостов для быстрых сетей

Измерения и настройки часто позволяют значительно улучшить производительность сети, однако они никогда не заменят хорошо разработанного проекта. Плохо спроектированная сеть может быть усовершенствована только до определенного уровня. Для дальнейшего увеличения производительности ее потребуется переделать с нуля.

В данном разделе мы рассмотрим несколько эмпирических правил, относящихся к программной реализации сетевых протоколов на хостах. Опыт показывает, что это, как правило, и является ограничивающим фактором производительности в случаях, когда сеть сама по себе работает быстро. Так происходит по двум причинам. Во-первых, сетевые карты (NIC) и маршрутизаторы разрабатываются так, чтобы они могли работать со скоростью сети. То есть они могут обрабатывать пакеты с той скоростью, с которой пакеты поступают. Во-вторых, нас интересует производительность, которая доступна приложениям. А она вычисляется не исходя из мощности канала, а исходя из пропускной способности и задержки, которые являются результатом работы сетевого и транспортного уровней.

Уменьшение накладных расходов из-за программного обеспечения улучшает производительность за счет повышения пропускной способности и снижения задержки. Оно также позволяет снизить затраты энергии, необходимые для передачи данных по сети, что актуально для мобильных компьютеров. Большинство этих идей известны разработчикам сетей уже много лет. Впервые они были записаны в явном виде Моголом (Mogul, 1993). Наше повествование во многом пересекается с его книгой. Другим источником по этой же теме является (Metcalfe, 1993).

Скорость центрального процессора важнее скорости сети

Длительные эксперименты показали, что почти во всех сетях накладные расходы операционной системы и протокола составляют основное время задержки сетевой операции. Например, теоретически минимальное время удаленного вызова процедур (**RPC**, **Remote Procedure Call**) в сети Ethernet мощностью 1 Гбит/с составляет 1 мкс, что соответствует минимальному запросу (512 байт), на который приходит минимальный (512-байтовый) ответ. На практике значительным достижением считается хотя бы какое-нибудь снижение накладных расходов, возникающих за счет программного обеспечения при вызове удаленной процедуры. Что происходит редко.

Аналогично, при работе с гигабитной линией основная задача заключается в достаточно быстрой передаче битов из буфера пользователя в линию, а также в том, чтобы получатель смог обработать их с той скоростью, с которой они приходят. Удвоение производительности процессора и быстродействия памяти нередко может привести к почти удвоению пропускной способности. Удвоение же пропускной способности линии часто не дает никакого эффекта, если узким местом являются хосты.

Уменьшайте число пакетов, чтобы уменьшить накладные расходы

Каждый сегмент содержит определенное количество накладных расходов (например, заголовок) и данные (например, полезную нагрузку). Оба эти компонента требуют

пропускной способности. Также каждому из них требуется обработка (например, обработка заголовка и вычисление контрольной суммы). При отправке 1 млн байт побайтовые затраты времени процессора на обработку не зависят от размера сегмента. Однако при использовании 128-байтовых сегментов затраты на обработку заголовков будут в 32 раза выше, чем для сегментов размером 4 Кбайт. И эти затраты увеличиваются очень быстро.

Накладные расходы более низких уровней усиливают этот эффект. Каждый раз прибытие каждого пакета вызывает новое прерывание, если хост работает в активном режиме. В современных конвейерных процессорах каждое прерывание нарушает работу процессорного конвейера, снижает эффективность работы кэша, требует изменения контекста управления памятью, аннулирует таблицу предсказания переходов и требует сохранения в стеке значительного числа регистров процессора. Таким образом, уменьшение на n числа посылаемых сегментов дает снижение числа прерываний и накладных расходов в целом в n раз.

Можно было бы сказать, что компьютер, как и человек, плохо справляется с многозадачностью. В основном поэтому возникает желание отправлять MTU-пакеты максимального размера, позволяющего избежать фрагментации. Механизмы наподобие алгоритма Нагля и метода Кларка также являются попытками избежать отправки маленьких пакетов.

Минимизируйте число операций с данными

Самый простой способ реализовать многоуровневый стек протоколов — использовать один модуль для каждого уровня. К сожалению, это приводит к многократному копированию данных (или, по крайней мере, многократному доступу к ним). К примеру, после того как пакет принимается сетевой картой, он обычно копируется в системный буфер ядра. Оттуда он копируется в буфер сетевого уровня для обработки сетевого уровня, а затем — в буфер транспортного уровня для обработки транспортного уровня, и наконец, доставляется получающему приложению. Часто полученный пакет копируется три или четыре раза, прежде чем содержащийся в нем сегмент доставляется по назначению.

Такое копирование может сильно снизить производительность, так как операции с памятью часто оказываются в десятки раз медленнее, чем операции с использованием только внутренних регистров. К примеру, если 20 % инструкций действительно связано с обращением к памяти (то есть данных нет в кэше), — а это вполне вероятная цифра при обработке входящих пакетов, — среднее время выполнения инструкции снизится в 2,8 раз ($0{,}8 \times 1 + 0{,}2 \times 10$). Аппаратные улучшения здесь не помогут. Проблема состоит в слишком большом числе операций копирования, выполняемых операционной системой.

Разумная операционная система постарается уменьшить число операций копирования, объединяя процессы обработки на разных уровнях. К примеру, TCP и IP обычно работают вместе («TCP/IP»), поэтому, когда обработка переходит от сетевого уровня к транспортному, копировать полезную нагрузку пакета не нужно. Еще один популярный прием состоит в том, чтобы выполнять несколько операций за один обход. Например, контрольные суммы часто вычисляются во время копирования данных (когда это действительно необходимо), и новое значение добавляется в конец.

Минимизируйте количество переключений контекста

Переключения контекста (например, из режима ядра в режим пользователя) обладают рядом неприятных свойств, в этом они сходны с прерываниями. Самое неприятное — потеря содержимого кэша. Количество переключений контекста может быть снижено при помощи библиотечной процедуры, посылающей данные во внутренний буфер до тех пор, пока их не наберется достаточное количество. Аналогично, на получающей стороне небольшие сегменты следует собирать вместе и передавать пользователю за один раз, минимизируя количество переключений контекста.

В лучшем случае, прибывший пакет вызывает одно переключение контекста из текущего пользовательского процесса в режим ядра, а затем еще одно переключение контекста при передаче управления принимающему процессу и предоставлении ему прибывших данных. К сожалению, во многих операционных системах происходит еще одно переключение контекста. Например, если сетевой менеджер работает в виде отдельного процесса в пространстве пользователя, поступление пакета вызывает передачу управления от процесса пользователя ядру, затем от ядра сетевому менеджеру, затем снова ядру, и наконец, от ядра получающему процессу. Эта последовательность переключений контекста показана на рис. 6.44. Все переключения контекста при получении каждого пакета сильно расходуют время центрального процессора и существенно снижают производительность сети.

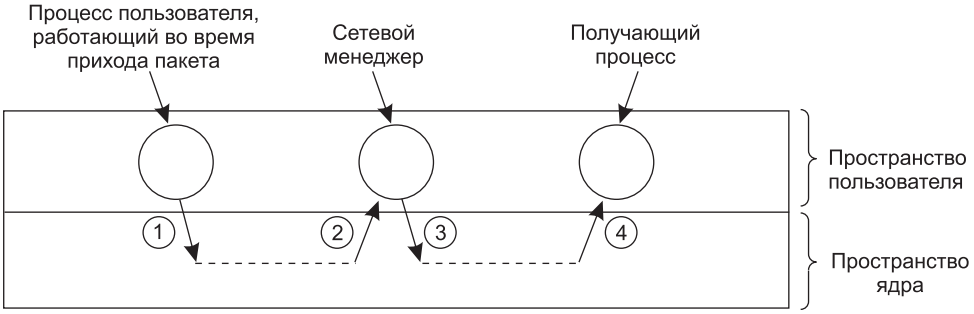


Рис. 6.44. Четыре переключения контекста для обработки одного пакета, в случае если сетевой менеджер находится в пространстве пользователя

Лучше избегать перегрузки, чем бороться с уже возникшей перегрузкой

Старая пословица, гласящая, что профилактика лучше лечения, справедлива и в деле борьбы с перегрузками в сетях. Когда в сети образуется затор, пакеты теряются, пропускная способность растрачивается впустую, увеличиваются задержки и т. п. Все эти издержки нежелательны. Процесс восстановления после перегрузки требует времени и терпения. Гораздо более эффективной стратегией является предотвращение перегрузки, напоминающее прививку от болезни — это несколько неприятно, зато избавляет от возможных больших неприятностей.

Избегайте тайм-аутов

Таймеры необходимы в сетях, но их следует применять умеренно и стремиться минимизировать количество тайм-аутов. Когда срабатывает таймер, обычно повторяется какое-либо действие. Если повтор действия необходим, так и следует поступить, однако повторение действия без особой необходимости является расточительным.

Чтобы избегать излишней работы, следует устанавливать период ожидания с небольшим запасом. Таймер, срабатывающий слишком поздно, несколько увеличивает задержку в (маловероятном) случае потери сегмента. Преждевременно срабатывающий таймер растрачивает попусту время процессора, пропускную способность и напрасно увеличивает нагрузку на, возможно, десятки маршрутизаторов.

6.6.4. Быстрая обработка сегментов

Теперь, когда мы поговорили об общих правилах, рассмотрим несколько методов, позволяющих ускорить обработку сегментов. Дополнительные сведения по этой теме см. в (Clark и др., 1989; Chase и др., 2001).

Накладные расходы по обработке сегментов состоят из двух компонентов: затрат по обработке заголовка и затрат по обработке каждого байта. Следует вести наступление сразу на обоих направлениях. Ключ к быстрой обработке сегментов лежит в выделении нормального случая (односторонней передачи данных) и отдельной обработке этого случая. Многие протоколы придают особое значение тому, что следует делать при возникновении нештатной ситуации (например, при потере пакета). Однако, чтобы протоколы работали быстро, разработчик должен пытаться минимизировать время обработки данных в нормальном режиме. Минимизация времени обработки при возникновении ошибок является лишь второстепенной задачей.

Хотя для перехода в состояние *ESTABLISHED* требуется передача последовательности специальных сегментов, но, как только это состояние достигнуто, обработка сегментов не вызывает затруднений, пока одна из сторон не начнет закрывать соединение. Начнем с рассмотрения посылающей стороны, находящейся в состоянии *ESTABLISHED*, когда должны отправляться данные. Для простоты мы предположим, что транспортная подсистема расположена в ядре, хотя те же самые идеи применимы и в случае, когда она представляет собой процесс, находящийся в пространстве пользователя, или набор библиотечных процедур в посылающем процессе. На рис. 6.45 отправляющий процесс эмулирует прерывание, выполняя базовую операцию `SEND`, и передает управление ядру. Прежде всего, транспортная подсистема проверяет, находится ли она в нормальном состоянии, то есть таком, когда установлено состояние *ESTABLISHED*, ни одна сторона не пытается закрыть соединение, посылается стандартный полный сегмент (без флага *URGENT*) и у получателя достаточный размер окна. Если все эти условия выполнены, то никаких дополнительных проверок не требуется, и алгоритм транспортной подсистемы может выбрать быстрый путь. В большинстве случаев именно так и происходит.

В нормальной ситуации заголовки соседних сегментов почти одинаковы. Чтобы использовать этот факт, транспортная подсистема сохраняет в своем буфере прототип заголовка. В начале быстрого пути он как можно быстрее, пословно, копируется

в буфер нового заголовка. Затем поверх перезаписываются все отличающиеся поля. Часто эти поля легко выводятся из переменных состояния — например, следующий порядковый номер сегмента. Затем указатель на заполненный заголовок сегмента и указатель на данные пользователя передаются сетевому уровню. Здесь может быть применена та же стратегия (на рис. 6.45 это не показано). Наконец, сетевой уровень передает полученный в результате пакет канальному уровню для отправки.

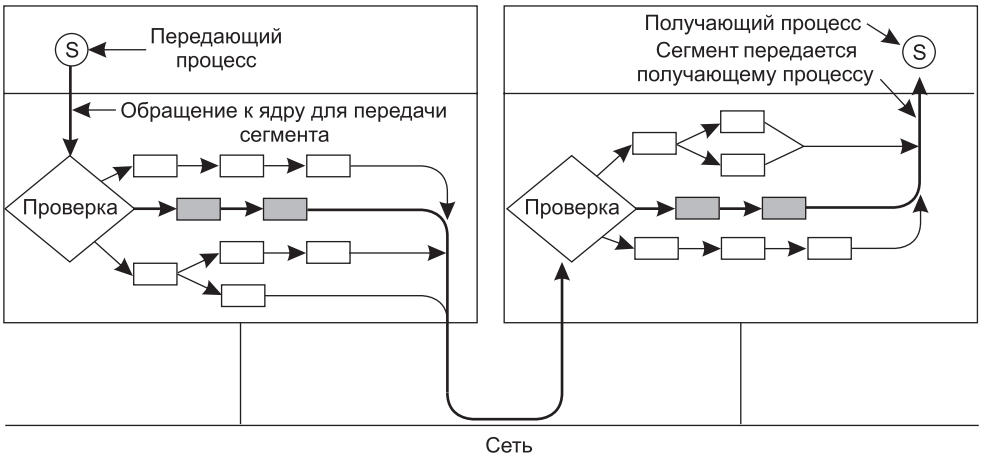


Рис. 6.45. Быстрый путь от отправителя до получателя показан жирной линией. Шаги обработки вдоль этого пути показаны затененными прямоугольниками

Чтобы увидеть, как работает этот принцип на практике, рассмотрим случай TCP/IP. На рис. 6.46, *а* изображен TCP-заголовок. Поля, одинаковые для заголовков последующих сегментов в однонаправленном потоке, затенены. Все, что нужно сделать передающей транспортной подсистеме, это скопировать пять слов заголовка-прототипа в выходной буфер, заполнить поле порядкового номера (скопировав одно слово), сосчитать контрольную сумму и увеличить на единицу значение переменной, хранящей текущий порядковый номер. Затем она может передать заголовок и данные специальной IP-процедуре, предназначенной для отправки стандартного, максимального сегмента. Затем IP-процедура копирует свой заголовок-прототип из пяти слов (рис. 6.46, *б*) в буфер, заполняет поле *Идентификатор* и вычисляет контрольную сумму заголовка. Теперь пакет готов к передаче.

Рассмотрим теперь быстрый путь обработки пакета получающей стороной на рис. 6.45. Первый шаг состоит в нахождении для пришедшего сегмента записи соединения. В протоколе TCP запись соединения может храниться в хэш-таблице, ключом к которой является какая-нибудь простая функция двух IP-адресов и двух портов. После обнаружения записи соединения следует проверить адреса и номера портов, чтобы убедиться, что найдена верная запись.

Процесс поиска нужной записи можно дополнительно ускорить, если установить указатель на последнюю использовавшуюся запись и сначала проверить ее. Кларк с соавторами книги, вышедшей в 1989 году, исследовал этот вопрос и пришел к выводу, что в этом случае доля успешных обращений превысит 90 %.

а *б*

Рис. 6.46. TCP-заголовок (*а*); IP-заголовок (*б*). В обоих случаях затененные поля взяты у прототипа без изменений

Затем сегмент проверяется на адекватность: соединение в состоянии *ESTABLISHED*, ни одна сторона не пытается его разорвать, сегмент является полным, специальные флаги не установлены, и порядковый номер соответствует ожидающемуся. Программа, выполняющая всю эту проверку, состоит из довольно значительного количества инструкций. Если все эти условия удовлетворяются, вызывается специальная процедура быстрого пути TCP-уровня.

Процедура быстрого пути обновляет запись соединения и копирует данные пользователю. Во время копирования она одновременно подсчитывает контрольную сумму, что уменьшает количество циклов обработки данных. Если контрольная сумма верна, запись соединения обновляется и отправляется подтверждение. Метод, реализованный в виде отдельной процедуры, сначала производящей быструю проверку заголовка, чтобы убедиться, что заголовок именно такой, какой ожидается, называется **предсказанием заголовков (header prediction)**. Он применяется в большинстве реализаций протокола TCP. Использование этого метода оптимизации вместе с остальными, описанными в данном разделе, позволяет протоколу TCP достичь 90 % от скорости локального копирования из памяти в память, при условии, что сама сеть достаточно быстрая.

Две другие области, в которых возможны основные улучшения производительности, — это управление буферами и управление таймерами. Как уже было сказано, при управлении буферами следует пытаться избегать излишнего копирования. При управлении таймерами следует учитывать, что они почти никогда не срабатывают. Они предназначены для обработки нестандартного случая потерь сегментов, но большинство сегментов и их подтверждений прибывают успешно, и поэтому истечение периода ожидания является редким событием.

В программе таймеры обычно реализуются в виде связанного списка таймеров, отсортированного по времени срабатывания. Начальный элемент списка содержит счетчик, хранящий число минимальных интервалов времени (тиков — ticks), оставшихся до истечения периода ожидания. В каждом последующем элементе списка содержится счетчик, указывающий, через сколько тиков относительно предыдущего элемента списка истечет время ожидания данного таймера. Например, если таймеры сработают через 3, 10 и 12 тиков, счетчики списка будут содержать значения 3, 7 и 2 соответственно.

При каждом тике часов счетчик начального элемента списка уменьшается на единицу. Когда значение счетчика достигает нуля, обрабатывается связанное с этим

таймером событие, а начальным элементом списка становится следующий элемент. При такой схеме добавление и удаление таймера является операцией, требующей затрат ресурсов, при этом время выполнения операции пропорционально длине списка.

Если максимальное значение таймера ограничено и известно заранее, то может быть применен более эффективный метод. Здесь можно использовать массив, называющийся **циклическим расписанием** (рис. 6.47) **(timing wheel)**. Каждое гнездо соответствует одному тику. Текущее время, показанное на рисунке, — T = 4. Таймеры должны сработать через 3, 10 и 12 тиков. Если новый таймер должен сработать через 7 тиков, то все, что нужно сделать, — задать значение указателя в гнезде 11 на новый список таймеров. Аналогично, если таймер, установленный на время T + 10, должен быть отключен, нужно всего лишь обнулить запись в гнезде 14. Обратите внимание на то, что массив на рис. 6.47 не может поддерживать таймеры за пределами T + 15.

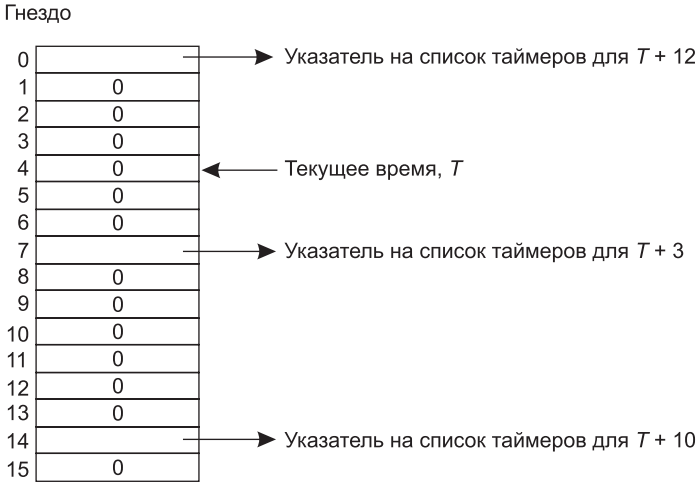


Рис. 6.47. Циклическое расписание

При тике часов указатель текущего времени перемещается по циклическому расписанию вперед на одно гнездо. Если гнездо, на которое он указывает, не нулевое, обрабатываются все таймеры списка, на который указывает гнездо. Многочисленные варианты основной идеи обсуждаются в (Varghese и Lauck, 1989).

6.6.5. Сжатие заголовков

Мы уже слишком долго говорим о быстрых сетях. Есть и другие темы. Давайте перейдем к вопросу о производительности в беспроводных сетях и сетях других типов, где пропускная способность ограничена. Если уменьшить издержки, возникающие за счет программного обеспечения, мобильные компьютеры будут работать эффективнее, но это не поможет улучшить производительность в тех случаях, когда узким местом являются сетевые каналы.

Чтобы эффективно использовать пропускную способность, заголовок протокола и полезная нагрузка должны занимать как можно меньше бит. Для полезной нагрузки

этого можно достичь с помощью компактного кодирования информации — например, изображения лучше передавать в формате JPEG, а не в растровых форматах, или же в форматах документов со сжатием, таких как PDF. Также для этого необходимы механизмы кэширования прикладного уровня (такие как веб-кэш), уменьшающие число передач.

А что известно про заголовки протоколов? В беспроводных сетях на канальном уровне заголовки обычно занимают мало места, так как они изначально рассчитаны на низкую пропускную способность. Так, в 802.16 вместо длинных адресов используются короткие идентификаторы соединения. Однако протоколы более высоких уровней (такие как IP, TCP и UDP) универсальны для всех типов сетей, поэтому в них не используются компактные заголовки. На самом деле потоковая обработка, уменьшающая накладные расходы на программное обеспечение, часто приводит к еще большему увеличению длины заголовка (например, в IPv6 используются более разреженные заголовки, чем в IPv4).

Заголовки более высоких уровней могут сильно влиять на производительность. Рассмотрим, к примеру, процесс передачи VoIP-данных через IP, UDP и RTP. Этим протоколам требуется заголовок длиной 40 байт (20 байт для IPv4, 8 — для UDP, 12 — для RTP). В случае IPv6 ситуация еще хуже: 60 байт, включая 40-байтный заголовок IPv6. Эти заголовки могут становиться еще длиннее, потребляя более половины пропускной способности.

Чтобы уменьшить пропускную способность, потребляемую заголовками высоких уровней, используется **сжатие заголовков (header compression)**. При этом используются не универсальные методы, а специально разработанные схемы. Дело в том, что по отдельности заголовки сжимаются плохо (поскольку они и так короткие), а для декомпрессии требуется, чтобы начальные данные пришли на место назначения. Последнее может не выполняться из-за потери пакетов.

Сжатие заголовков можно реализовать эффективно, если учитывать формат протокола. Одна из первых схем была разработана Ван Джекобсоном (1990) и применялась для сжатия TCP/IP-заголовков при передаче по медленным последовательным каналам. Она позволяет уменьшить обычный 40-байтовый TCP/IP-заголовок в среднем до 3 байт. Если посмотреть на рис. 6.46, можно догадаться, на чем основан этот метод. Дело в том, что многие поля заголовка не меняются от пакета к пакету. К примеру, совсем не обязательно передавать одно и то же значение времени жизни пакета или один и тот же номер TCP-порта с каждым пакетом. Эти поля можно пропустить при передаче пакета и заполнить при получении.

Остальные поля меняются по предсказуемому сценарию. К примеру, если потерь пакетов не происходит, порядковые номера TCP увеличиваются по мере отправки данных. Следовательно, получатель может предсказать наиболее вероятное значение. Номер необходимо указать только в том случае, если он отличается от ожидаемого. Но даже в таком случае можно передавать разность между данным номером и предыдущим, как в случае с номером подтверждения, который увеличивается, когда новые данные приходят в обратном направлении.

С помощью сжатия заголовков можно добиться того, чтобы в протоколах высоких уровней заголовки были простыми, а при передаче пакета по каналам с низкой пропускной способностью использовалось компактное кодирование. **Надежное**

сжатие заголовков (ROHC, Robust Header Compression) — современный вариант сжатия заголовков, который определен в RFC 5795 как фреймворк. В соответствии с замыслом разработчиков, он не реагирует на потерю пакетов в беспроводных каналах. Для каждого набора протоколов — например, IP/UDP/RTP — существует отдельный профиль. Сжатые заголовки передаются с помощью отсылки к контексту, то есть фактически к соединению; значения полей заголовков можно легко предсказать для пакетов данного соединения, но не для пакетов других соединений. При нормальной работе ROHC размер заголовков для IP/UDP/RTP снижается от 40 байт до 1–3 байт.

Хотя основная цель компрессии заголовков — уменьшение потребляемой пропускной способности, с помощью этого механизма можно также снизить задержку. Задержка складывается из задержки распространения, фиксированной для данного сетевого пути, и задержки передачи, которая зависит от пропускной способности и объема передаваемых данных. Например, канал мощностью 1 Мбит/с отправляет 1 бит за 1 мкс. Если мы имеем дело с передачей медиаданных по беспроводной сети, такой канал считается достаточно медленным, поэтому задержка передачи составляет существенную часть общей задержки, и стабильно низкая задержка крайне важна для качества обслуживания.

Если использовать сжатие заголовков, объем передаваемых данных уменьшится, и вместе с ним уменьшится задержка передачи. Того же эффекта можно добиться, отправляя пакеты меньшего размера. Так мы фактически обменяем хорошие показатели накладных расходов, возникающих за счет программного обеспечения, на хорошие показатели задержки передачи. Стоит отметить, что еще одним источником задержки является время ожидания в очереди для доступа к беспроводному каналу. Этот фактор может оказаться важным, так как беспроводные сети обычно сильно загружены. В таком случае беспроводной канал должен включать механизмы качества обслуживания, обеспечивающие низкую задержку пакетам реального времени. Одного сжатия заголовков будет недостаточно.

6.6.6. Протоколы для протяженных сетей с высокой пропускной способностью

Появление гигабитных сетей, передающих данные на большие расстояния, приходится на начало 90-х годов. Их также называют протяженными сетями с **высокой пропускной способностью (long fat networks)**. Сначала к ним пытались применить старые протоколы, но при этом сразу же возникло множество проблем. В данном разделе мы обсудим некоторые из этих проблем, переходя к более высоким скоростям и задержкам сетевых протоколов.

Первая проблема заключается в том, что многие протоколы используют 32-разрядные порядковые номера. В прежние годы, когда типичная скорость выделенных линий между маршрутизаторами равнялась 56 Кбит/с, хосту, который постоянно выдавал бы данные в сеть на полной скорости, потребовалось бы больше недели на то, чтобы у него закончились порядковые номера. С точки зрения разработчиков TCP 232 считалось неплохим приближением к бесконечности, поскольку вероятность блуждания пакетов по сети в течение недели практически равна нулю. В Ethernet со

скоростью 10 Мбит/с критическое время снизилось с одной недели до 57 минут. Это, конечно, гораздо меньше, но даже с таким интервалом еще можно иметь дело. Когда же Ethernet выдает в Интернет данные со скоростью 1 Гбит/с, порядковые номера закончатся примерно через 34 с. Это уже никуда не годится, поскольку максимальное время жизни пакета в Интернете равно 120 с. Внезапно оказалось, что 232 совершенно не подходит в качестве значения, приближенного к бесконечности, поскольку стало очевидно, что отправителю, посылающему достаточно много пакетов, придется повторять их порядковые номера в то время, как старые пакеты все еще будут блуждать по сети.

Проблема состоит в том, что многие разработчики протоколов просто предполагали, что время цикла пространства порядковых номеров значительно превосходит максимальное время жизни пакетов. Соответственно, в этих протоколах даже не рассматривалась сама возможность того, что старые пакеты еще могут находиться где-то в сети, когда порядковые номера опишут полный круг. В гигабитных сетях это предположение оказалось неверным. К счастью, оказалось возможным расширить эффективный порядковый номер, взяв в качестве старших битов метку времени, которая добавляется в TCP-заголовок каждого пакета в качестве дополнительного параметра. Этот механизм называется детектированием повторного использования порядковых номеров (**PAWS**, **Protection Against Wrapped Sequence numbers**). Он описан в RFC 1323.

Вторая проблема состоит в необходимости существенного увеличения окна управления потоком. Рассмотрим, например, процесс отправки 64 Кбайт данных из Сан-Диего в Бостон при размере буфера получателя в 64 Кбайт. Пусть скорость передачи данных в линии составляет 1 Гбит/с, а время прохождения сигнала в одну сторону, ограниченное скоростью света в стекле оптоволокна, равно 20 мс. Вначале (t = 0), как показано на рис. 6.48, *а*, канал пуст. Только 500 мкс спустя все сегменты попадут в канал (рис. 6.48, *б*). Первый сегмент оказывается где-то в окрестностях Броли, все еще в Южной Калифорнии. Тем не менее передатчик уже должен остановиться, пока он не получит в ответ новую информацию об окне.

Через 20 мс первый сегмент, как показано на рис. 6.48, *в*, достигнет Бостона, и в ответ будет передано подтверждение. Наконец, через 40 мс после начала операции первое подтверждение возвращается к отправителю, после чего передается следующая порция данных. Поскольку линия передачи использовалась всего 0,5 мс из 40 мс, эффективность ее использования составит около 1,25 %. Такая ситуация является типичной для работы старых протоколов по гигабитным линиям.

При анализе производительности сетей полезно обращать внимание на **произведение пропускной способности и времени задержки (bandwidth-delay product)**. Пропускная способность канала (в битах в секунду) умножается на время прохождения сигнала в оба конца (в секундах). В результате получается емкость канала в битах.

В примере на рис. 6.48 произведение пропускной способности и времени задержки равно 40 млн бит. Другими словами, отправитель к моменту получения ответа успеет переслать 40 млн бит, если будет передавать с максимальной скоростью. Столько бит потребуется, чтобы заполнить канал в обоих направлениях. Таким образом, порция данных в полмиллиона бит составляет всего 1,25 % емкости канала, что и выражается в 1,25 % эффективности использования канала.

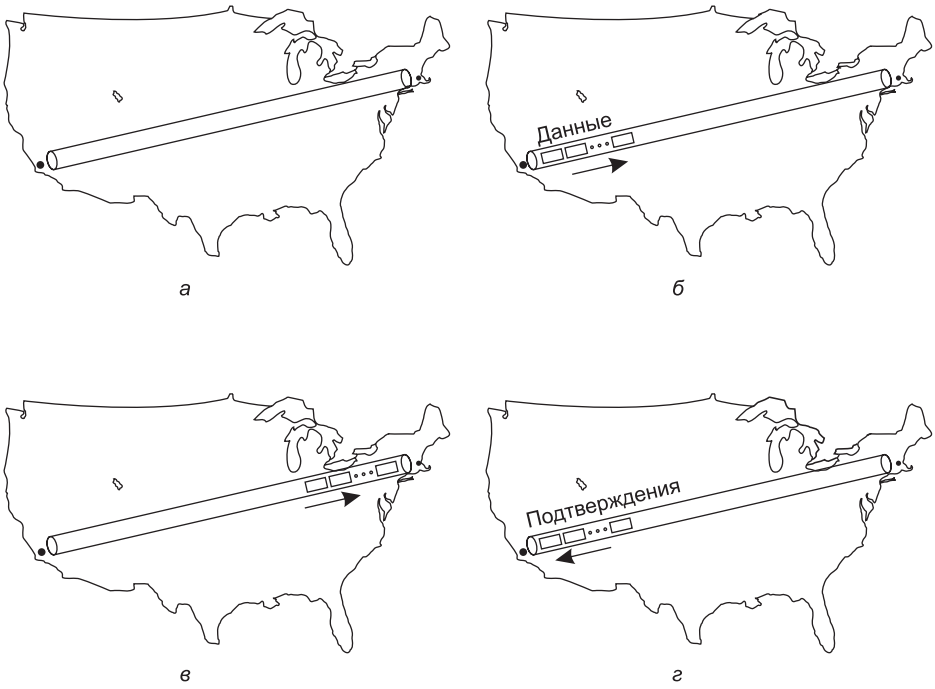


Рис. 6.48. Передача половины мегабита из Сан-Диего в Бостон: *а* — в момент времени t = 0; *б* — через 500 мкс; *в* — через 20 мс; *г* — через 40 мс

Отсюда следует, что для эффективного использования канала размер окна получателя должен быть, по меньшей мере, равен произведению пропускной способности и времени задержки, а лучше превосходить его, так как получатель может сразу и не ответить. Для трансконтинентальной гигабитной линии каждому соединению потребуется, по меньшей мере, по 5 Мбайт.

Третья проблема, связанная с предыдущей, состоит в том, что простые схемы повторной передачи, такие как протокол с возвращением на N блоков, плохо работают в линиях с большим значением произведения пропускной способности на задержку. Рассмотрим трансконтинентальную линию, работающую со скоростью 1 Гбит/с. Время прохождения сигнала в оба конца равно 40 мс. За это время отправитель успевает передать 5 Мбайт. Если обнаруживается ошибка, потребуется 40 мс, чтобы оповестить об этом отправителя. При использовании протокола с возвращением на N блоков отправителю потребуется повторять передачу не только поврежденного пакета, но также и до 5 Мбайт пакетов, переданных после поврежденного. Очевидно, что этот протокол использует ресурсы очень неэффективно. Поэтому необходимы более сложные механизмы, такие как метод выборочного повтора.

Суть четвертой проблемы состоит в том, что гигабитные линии принципиально отличаются от мегабитных — в длинных гигабитных линиях главным ограничивающим фактором является не пропускная способность, а задержка. На рис. 6.49 изображена зависимость времени, требующегося для передачи файла размером 1 Мбит по линии длиной в 4000 км, от скорости передачи. На скоростях до 1 Мбит/с время передачи

в основном зависит от скорости передачи данных. При скорости 1 Гбит/с задержка в 40 мс значительно превосходит 1 мс (время, требующееся для передачи битов по оптоволоконному кабелю). Дальнейшее увеличение скорости вообще не оказывает на время передачи файла почти никакого действия.

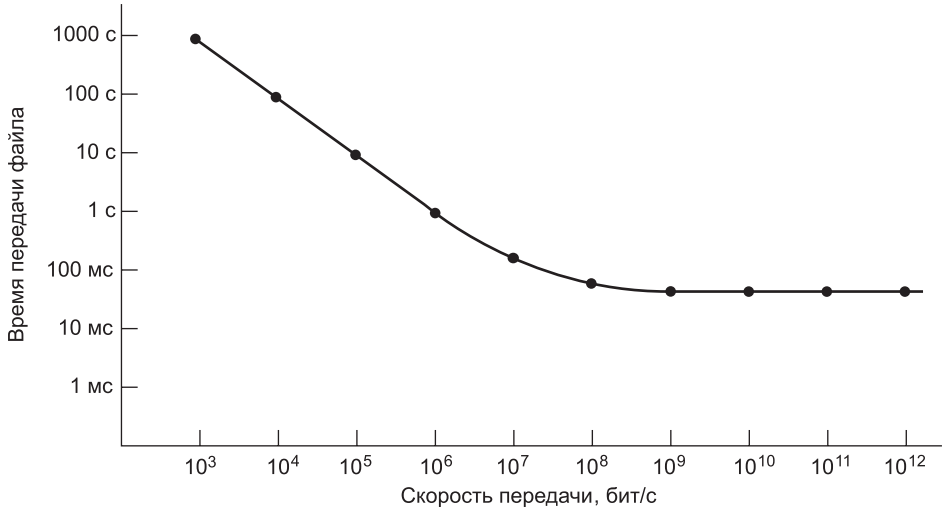


Рис. 6.49. Время передачи и подтверждения файла размером 1 Мбит по линии длиной 4000 км

Изображенная на рис. 6.49 зависимость демонстрирует ограничения сетевых протоколов. Она показывает, что протоколы с ожиданием подтверждений, такие как удаленный вызов процедур (**RPC**, **Remote Procedure Call**), имеют врожденное ограничение на производительность. Это ограничение связано со скоростью света. Никакие технологические прорывы в области оптики здесь не помогут (хотя могли бы помочь новые законы физики). Если в периоды времени, пока хост ждет ответа, гигабитная линия будет простаивать, она будет ничем не лучше простой мегабитной линии, а только дороже.

Пятая проблема заключается в том, что скорости передачи данных увеличиваются значительно быстрее, чем скорости обработки данных. (Обращение к разработчикам компьютеров: идите и побейте разработчиков средств связи! Мы рассчитываем на вас.) В 70-е годы объединенная сеть ARPANET работала на скорости 56 Кбит/с и состояла из компьютеров с производительностью около 1 MIPS (1 млн инструкций в секунду). Сравните эти цифры с цифрами для современных компьютеров, имеющих производительность 1000 MIPS и обменивающихся пакетами по гигабитной линии. Число инструкций на один байт уменьшилось более, чем в десять раз. Точные цифры зависят от времени и конкретной ситуации, но в итоге можно сказать следующее: у протокола осталось меньше времени на обработку пакетов, поэтому протоколы должны стать проще.

Перейдем теперь к рассмотрению методов решения всего этого набора проблем. Главный принцип, который каждый разработчик высокоскоростных сетей должен выучить наизусть, звучит так:

Проектируя, стремись увеличить скорость, а не оптимизировать пропускную способность.

При разработке старых протоколов обычно ставилась задача минимизировать количество битов, переданных в линию, часто за счет использования полей малого размера и упаковывания нескольких полей вместе в один байт или слово. В настоящее время экономить пропускную способность смысла нет: ее более чем достаточно. Проблема заключается в обработке протоколами пакетов, поэтому при разработке новых протоколов минимизировать нужно именно время обработки. Разработчики IPv6, к счастью, хорошо понимают это.

Конечно, заманчивы попытки ускорить работу сетей, создав быстрые сетевые интерфейсы на аппаратном уровне. Сложность такого подхода состоит в том, что если только протокол не является чрезвычайно простым, аппаратный интерфейс представляет собой дополнительную плату с отдельным процессором и своей программой. Чтобы сетевой сопроцессор не был столь же дорогим, как и центральный процессор, он часто представляет собой более медленную микросхему. В результате такого подхода быстрый центральный процессор ждет, пока медленный сопроцессор выполнит свою работу. Было бы неверным полагать, что у центрального процессора на время ожидания обязательно найдется другая работа. Более того, для общения двух процессоров общего назначения потребуются тщательно продуманные протоколы, обеспечивающие их корректную синхронизацию. Как правило, наилучший метод заключается в создании простых протоколов, чтобы всю работу мог выполнять центральный процессор.

Крайне важен в гигабитных сетях формат пакета. В заголовке должно быть как можно меньше полей — это позволит снизить время его обработки. Сами поля должны быть достаточно большими, чтобы нести в себе как можно больше полезной (служебной) информации. Кроме того, они не должны пересекать границы слов, тогда их будет проще обработать. В данном контексте под «достаточно большим» подразумевается такой размер, который исключает возникновение проблем зацикливания порядковых номеров при нахождении в сети старых пакетов, учитывает отсутствие у получателя возможности объявить реально доступный размер окна из-за слишком малого служебного поля размера окна и т. д.

Максимальный размер поля данных должен быть достаточно большим, чтобы снизить накладные расходы, возникающие за счет программного обеспечения, и обеспечить возможность эффективной работы сети. Размер пакета 1500 байт слишком мал для гигабитных сетей, поэтому гигабитная сеть Ethernet позволяет передавать сверхдлинные кадры размером до 9 Кбайт, а IPv6 поддерживает джамбограммы размером более 64 Кбайт.

Рассмотрим теперь вопрос обратной связи в высокоскоростных протоколах. Из-за относительно длинного цикла задержки желателен отказ от обратной связи: на оповещение отправителя уходит слишком много времени. Один пример обратной связи — управление скоростью передачи с помощью протокола скользящего окна. Чтобы избежать долгих задержек, связанных с передачей получателем информации о состоянии окна отправителю, лучше использовать протокол, основанный на скорости. В таком протоколе отправитель может посылать не быстрее, чем с некоторой скоростью, с которой получатель заранее согласился.

Второй пример обратной связи — алгоритм медленного старта, разработанный Джекобсоном. Этот алгоритм проводит многочисленные пробы, пытаясь определить пропускную способность сети. В высокоскоростных сетях на проведение пяти-шести проб для определения ответной реакции сети тратится огромное количество сетевых ресурсов. Более эффективная схема состоит в резервировании ресурсов отправителем, получателем и сетью в момент установки соединения. Кроме того, заблаговременное резервирование ресурсов позволяет несколько снизить эффект неравномерности доставки (джиттер). Короче говоря, повышение скоростей передачи в сетях неумолимо заставляет разработчиков выбирать ориентированную на соединение (или близкую к этому) схему работы сети.

Еще одно полезное свойство протокола — возможность посылать нормальное количество данных вместе с запросом соединения. При этом можно сэкономить время одного запроса и ответа.

6.7. Сети, устойчивые к задержкам

Последний раздел мы посвятим новому виду транспорта, который, возможно, когда-нибудь станет важной частью Интернета. TCP и большинство других транспортных протоколов исходят из предположения о том, что между отправителем и получателем постоянно существует рабочий путь; в противном случае протокол дает сбой, и пакеты не доставляются. В некоторых сетях сквозной путь часто отсутствует. В качестве примера можно привести космическую сеть, в которой спутники **LEO (Low-Earth Orbit, низкая околоземная орбита)** попеременно находятся в зоне и вне зоны досягаемости наземных станций. Каждый конкретный спутник может установить связь с данной наземной станцией только в определенное время, а два спутника никогда не могут установить связь друг с другом даже через наземную станцию, так как один из них всегда находится вне зоны досягаемости этой станции. Другие примеры касаются подводных лодок, автобусов, мобильных телефонов и других устройств, оснащенных компьютерами, для которых связь является непостоянной из-за перемещений или экстремальных условий.

Тем не менее передача данных возможна и в сетях с непостоянной связью: эти данные могут задерживаться на узлах до тех пор, пока не появится рабочее соединение. Такой метод называется **коммутацией сообщений**. Сети, сконструированные по такому принципу, называются сетями, **устойчивыми к задержкам (DTN, Delay-Tolerant Network)**, или **распадоустойчивыми сетями (Disruption-Tolerant Network, DTN)**.

Работа над DTN началась в 2002 году, когда комиссией IETF была создана специальная исследовательская группа. Необходимость создания сетей, устойчивых к задержкам, возникла в неожиданном месте: при попытках отправлять пакеты в космосе. Космические сети вынуждены иметь дело с непостоянной связью и очень длинными задержками. Кевин Фолл заметил, что идеи, применимые к таким межпланетным интерсетям, вполне подойдут и для земных сетей, в которых непостоянная связь между узлами является обычным делом (Fall, 2003). Эта модель является обобщением Интернета, в котором при коммуникации возможны задержки и временное хранение данных.

Передача данных скорее похожа на доставку почтовой системой или по электронной почте, чем на коммутацию пакетов на маршрутизаторах[1].

С 2002 года архитектура DTN была доработана, и сфера применения модели DTN расширилась. Представьте себе, к примеру, терабайтные массивы данных, полученных в результате экспериментов, из СМИ или с помощью веб-сервисов, которые необходимо передать в центры сбора данных, расположенные по всему миру. Операторам удобнее отправлять такой трафик во внепиковое время, чтобы использовать пропускную способность, за которую они заплатили, но которая не используется, и они готовы к определенным задержкам. Это напоминает резервное копирование, которое выполняется по ночам, когда другие приложения используют сеть не так активно. Если мы имеем дело с глобальными сервисами, проблема состоит в том, что в разных уголках мира внепиковое время разное. Периоды внепиковой пропускной способности могут почти не пересекаться, если центры сбора данных находятся, к примеру, в Перте и Бостоне (когда в одном из этих городов день, в другом — ночь).

Однако модели DTN предусматривают возможность хранения данных и большие задержки. С помощью такой модели массив данных можно сначала передать из Бостона в Амстердам во внепиковое время (так как время в них различается всего на 6 часов). Далее данные будут храниться в Амстердаме до того момента, когда их можно будет передать в Перт, используя внепиковую пропускную способность. В работе (Laoutaris и др., 2009) было установлено, что такая модель способна обеспечить хорошую пропускную способность при незначительных затратах, и эта пропускная способность часто вдвое превышает показатели обычной сквозной модели.

Далее мы рассмотрим архитектуру и протоколы DTN, разработанные IETF.

6.7.1. Архитектура DTN

Модель DTN предлагает отказаться от одного из предположений, на которых основан современный Интернет. Оно звучит так: в течение всего сеанса связи существует сквозной путь между отправителем и получателем. Когда это не так, обычные Интернет-протоколы не работают. Сети, устойчивые к задержкам, обходят проблему отсутствия сквозного пути с помощью архитектуры, основанной на коммутации сообщений (рис. 6.50). Кроме того, они приспособлены к передаче данных по каналам с низкой надежностью и большими задержками. Эта архитектура определена в RFC 4838.

В терминологии DTN сообщение называется **посылкой**. Узлы DTN оснащены запоминающими устройствами — как правило, с постоянной памятью (диски, флэш-память и т. д.). В них посылки хранятся до тех пор, пока нужный канал не активизируется; затем происходит отправка посылок. Каналы работают с перерывами. На рис. 6.50

[1] Следует заметить, что компьютерные сети, основанные на методе коммутации сообщений (и, исходя из данного автором определения, — являющиеся DTN), были известны и ранее. В качестве примера можно привести международную любительскую компьютерную сеть FidoNet (с середины 1980-х годов) и построенные на аналогичном программном обеспечении другие так называемые FTN (Fidonet technology network). Можно найти и иные примеры, поскольку сама идея коммутации сообщений восходит к обычной «бумажной» почтовой связи. А привлечение (или даже независимая повторная разработка) известных методов (возможно, в некотором, ранее не встречавшемся, сочетании) для решения новой проблемы — весьма распространенная инженерная практика. — *Примеч. ред.*

изображено пять непостоянных каналов, которые в данный момент не работают, и два активных канала. Активный канал называется **контактом**. На рис. 6.50 также изображены две посылки, которые хранятся в узлах DTN, ожидая нужного контакта. По такой схеме пакеты передаются от источника в пункт назначения.

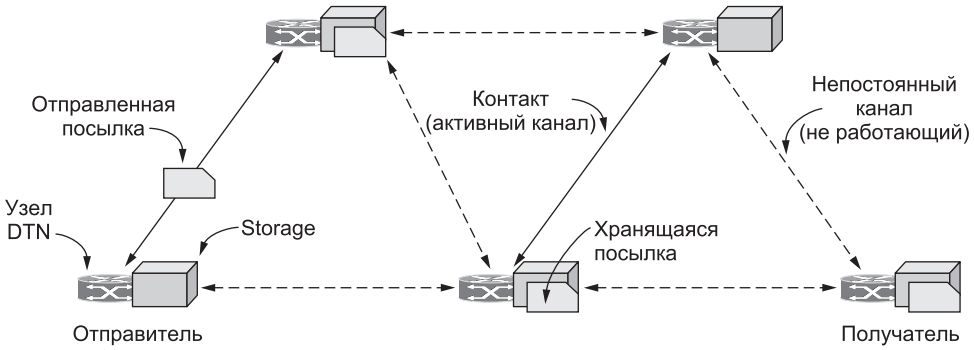


Рис. 6.50. Архитектура DTN

Такая схема очень похожа на то, что происходит с пакетами на маршрутизаторах. Однако здесь есть качественные отличия. На маршрутизаторах в Интернете ожидание в очереди длится несколько миллисекунд, в худшем случае — секунд. В узлах DTN посылки могут храниться часами — до тех пор, пока автобус не прибудет в город, самолет не приземлится, узел сенсорной сети не накопит солнечную энергию, необходимую для его работы, спящий компьютер не проснется и т. д. Эти примеры иллюстрируют и второе отличие: узлы могут перемещаться (вместе с автобусом или самолетом) вместе с хранящимися в них посылками, и это может играть ключевую роль в доставке данных; маршрутизаторы Интернета двигаться не могут. Чтобы описать весь процесс перемещения посылок, иногда используют термин «получение—перенос—отправка» («store—carry—forward»).

Для примера рассмотрим ситуацию, изображенную на рис. 6.51. Так протоколы DTN впервые использовались в космосе (Wood и др., 2008). Источник посылок — один из спутников LEO Международной системы мониторинга стихийных бедствий, который делает снимки Земли. Изображения должны приходить на пункт сбора данных. Но спутник имеет непостоянную связь с тремя наземными станциями. Двигаясь по орбите, он связывается ними по очереди. Таким образом, спутник, наземные станции и пункт сбора данных являются узлами DTN. По каждому контакту посылка (или часть посылки) передается наземной станции. Затем посылки доставляются на пункт сбора данных по транзитной наземной сети. На этом передача завершается.

В этом примере основное преимущество архитектуры DTN состоит в том, что она прекрасно подходит для ситуации, когда спутнику требуется хранить изображения в памяти, так как в момент получения изображения связь отсутствует. Помимо этого у DTN есть еще два преимущества. Во-первых, один контакт может быть слишком коротким, чтобы отправить все изображения. Эта проблема решается легко: данные можно распределить между контактами с тремя наземными станциями. Во-вторых, канал между спутником и наземной станцией работает независимо от канала, соединяющего станцию и наземную сеть. Это значит, что скорость обмена данными

между спутником и станцией не будет ограничена даже при наличии медленных каналов в наземной сети. Данные могут передаваться на максимальной скорости. Посылка будет храниться на станции до тех пор, пока ее не удастся передать на пункт сбора данных.

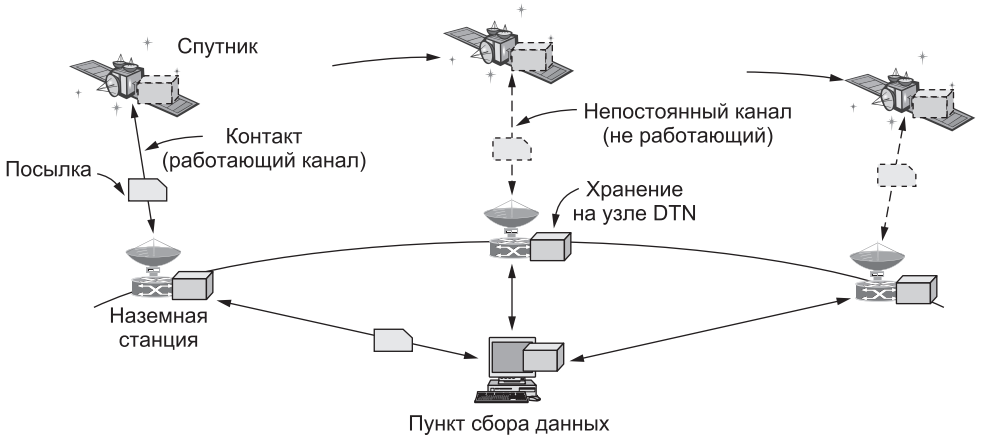


Рис. 6.51. Использование DTN в космосе

В описании архитектуры DTN не рассматривается важный вопрос: как находить хорошие маршруты через узлы DTN. Хорошие маршруты зависят от архитектуры, которая описывает, когда следует отправлять данные и по каким направлениям (контактам). О некоторых контактах можно узнать заранее. Так, в нашем космическом примере заранее известно движение небесных тел. В эксперименте по использованию DTN в космосе заранее было известно время связи, а также то, что контакт с каждой наземной станцией длится от 5 до 14 минут и что пропускная способность нисходящей линии составляет 8,134 Мбит/с. С помощью этих сведений можно планировать передачу посылок с изображениями.

В других случаях контакты можно предсказать с меньшей долей вероятности. Примеры таких ситуаций — автобусы, которые связываются друг с другом регулярно (по расписанию), но все же с некоторыми отклонениями; сети провайдеров, в которых внепиковое время и доступное количество пропускной способности можно предсказать по данным, полученным ранее. Другой крайностью являются ситуации, в которых контакты являются эпизодическими и в произвольные моменты. Например, так происходит при передаче данных от пользователя к пользователю с помощью мобильных телефонов, которая зависит от того, какие пользователи выйдут на связь друг с другом в течение дня. Когда контакты непредсказуемы, одна из возможных стратегий — отправлять копии посылки по разным путям в надежде на то, что одна из копий дойдет до места назначения до окончания времени ее жизни.

6.7.2. Протокол Bundle

Чтобы лучше понять, как работает DTN, мы рассмотрим протоколы IETF. DTN — это развивающийся тип сетей. В экспериментальных реализациях DTN используются

самые разные протоколы. Для этого не обязательно использовать именно протоколы IETF. Но на их примере нам будет удобнее рассмотреть наиболее важные вопросы.

Стек протокола DTN показан на рис. 6.52. Основной протокол — это **протокол Bundle**; он описан в RFC 5050. Он принимает сообщения от приложения и передает их в виде одной или нескольких посылок с помощью операций получения—переноса—отправки принимающему узлу DTN. Как видно из рис. 6.52, он работает над уровнем TCP/IP. Иными словами, TCP/IP может использоваться в каждом из контактов для передачи посылок между узлами. Следовательно, возникает вопрос, к какому уровню относится протокол Bundle — транспортному или прикладному. Как и в случае с RTP, мы придерживаемся мнения, что, несмотря на более высокую позицию в иерархии, протокол Bundle предоставляет многим приложениям транспортные услуги, поэтому мы рассматриваем DTN в этой главе.

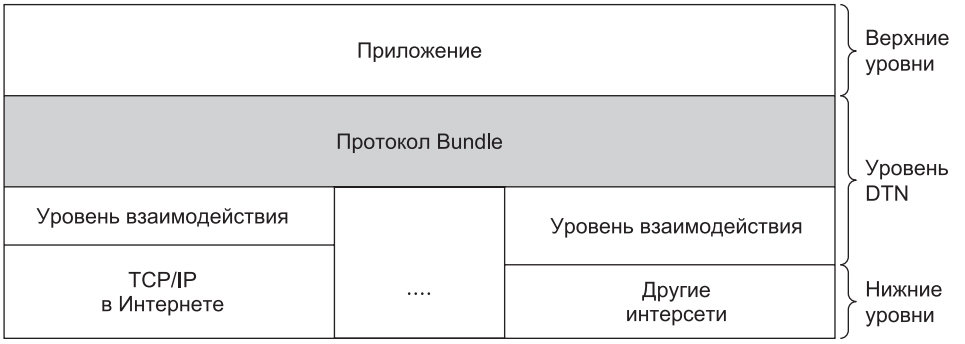


Рис. 6.52. Стек протоколов DTN

На рис. 6.52 также показано, что протокол Bundle может работать поверх других протоколов — например, UDP — или даже других интерсетей. К примеру, в космической сети каналы могут обладать большой задержкой. Круговая задержка между Землей и Марсом может составлять 20 минут (в зависимости от их взаимного расположения). Представьте себе, как здорово в таком канале будут работать подтверждения и повторные передачи, особенно для коротких сообщений! Вовсе не здорово. В такой ситуации необходим другой протокол, использующий коды с коррекцией ошибок. В сенсорных сетях, где ресурсы сильно ограничены, вместо TCP может использоваться более легковесный протокол.

Так как протокол Bundle является фиксированным, но в его задачи входит совместимость с различными транспортами, между сферами действия протоколов должен быть небольшой зазор. Эта идея привела к добавлению дополнительного **уровня взаимодействия (convergence layer)**, как показано на рис. 6.52. На самом деле это просто связующий уровень, обеспечивающий совместную работу протоколов. По определению для каждого транспорта более низкого уровня должен существовать отдельный уровень взаимодействия. Уровни взаимодействия, позволяющие подключать новые и существующие протоколы, обычно можно найти в стандартах.

Формат сообщений протокола Bundle приведен на рис. 6.53. По названиям полей можно догадаться, чем занимается этот протокол.

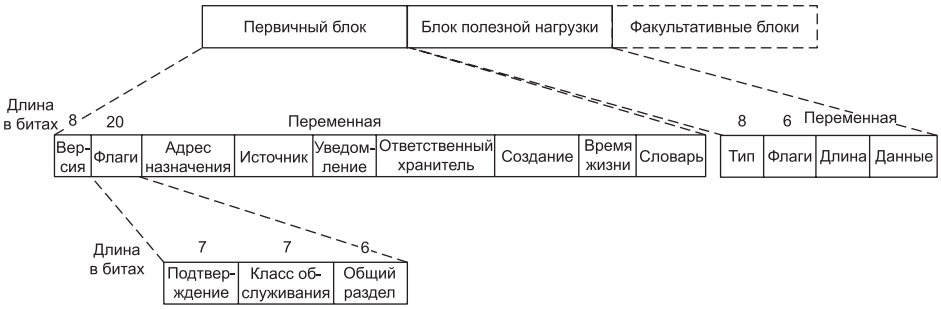


Рис. 6.53. Формат сообщений протокола Bundle

Каждое сообщение состоит из первичного блока, который можно считать заголовком, блока полезной нагрузки (для данных) и факультативных блоков (например, для параметров безопасности). Первичный блок начинается с поля *Версия* (на данный момент — 6), за которым следует поле *Флаги*. Помимо всего прочего, с помощью флагов указывается класс обслуживания (чтобы источник мог отметить посылку как высокоприоритетную или низкоприоритетную) и другие обрабатывающие запросы (например, должен ли получатель подтвердить доставку).

Далее следуют адреса. И здесь уже есть кое-что интересное. Помимо полей идентификаторов *Адрес назначения* и *Источник*, мы видим идентификатор *Ответственный хранитель*. Ответственный хранитель — это сторона, обязанная следить за тем, чтобы пакет был доставлен. В Интернете эта роль обычно возложена на источник, так как именно он выполняет повторную передачу, если данные не доходят до пункта назначения. Но в DTN узел-источник не всегда находится на связи и, следовательно, не всегда может узнать, доставлены ли данные. Для решения этой проблемы в DTN используется **процедура сдачи-приемки (custody transfer)**, при которой другой узел, расположенный ближе к получателю, принимает на себя ответственность за доставку данных. Например, если посылка временно хранится на самолете и будет передана позднее и в другом месте, самолет может стать ответственным хранителем этой посылки.

Второй интересный момент заключается в том, что эти идентификаторы *не являются* IP-адресами. Поскольку протокол Bundle работает с самыми разными транспортами и интерсетями, он использует свои собственные идентификаторы. Они больше похожи на имена высоких уровней, такие как URL веб-страниц, чем на адреса нижних уровней (IP). Такие идентификаторы дают сетям DTN возможности маршрутизации прикладного уровня, например доставки электронной почты или рассылки обновлений программного обеспечения.

Третий любопытный аспект — это то, как кодируются идентификаторы, так же как и идентификаторы в поле *Уведомление*, для диагностических сообщений. Все эти идентификаторы кодируются с помощью ссылок на поле *Словарь* переменной длины. Это позволяет использовать сжатие, когда узел ответственного хранителя или узел для диагностических сообщений совпадают с источником или адресом назначения. На самом деле разработчики формата сообщений стремились добиться как эффективности, так и возможности изменения длины поля, используя компактное представление

для полей переменной длины. Последнее играет важную роль в беспроводных сетях, а также в сетях с ограниченными ресурсами, таких как сенсорные сети.

Далее следует поле *Создание*, в котором хранится время создания посылки, а также порядковый номер отправителя; за ним располагается поле *Время жизни*, в котором указано время, когда посылка будет уже не нужна. Эти поля нужны, так как посылки могут храниться в узлах DTN очень долго, и поэтому в сети должен существовать механизм, позволяющий удалять устаревшие данные. В отличие от Интернета в данном случае часы на узлах должны быть слабо синхронизированы.

Первичный блок завершается полем *Словарь*. Далее идет блок полезной нагрузки. Он начинается с короткого поля *Тип*, в котором указано, что это полезная нагрузка, а за ним располагаются *Флаги*, в которых задаются параметры обработки. Далее следует поле *Данные*, перед которым располагается поле *Длина*. Наконец, за ними могут быть факультативные блоки — в частности, блок с параметрами безопасности.

Многие аспекты сетей, устойчивых к задержкам, продолжают обсуждаться в научных сообществах. Как мы уже говорили, хорошие стратегии маршрутизации зависят от природы контактов. Идея хранения данных внутри сети приводит к возникновению новых проблем. Теперь контроль перегрузки должен относить память в узлах DTN к другому типу ресурсов, и такие ресурсы тоже могут заканчиваться. Отсутствие сквозного соединения усугубляет проблему безопасности. Перед тем как принять на себя ответственность за доставку посылки, узел захочет проверить, что отправитель официально зарегистрирован в сети и что получателю нужна эта посылка. Решения этих проблем будут зависеть от типа DTN, ведь космические сети так сильно отличаются от сенсорных!

6.8. Резюме

Транспортный уровень — это ключ к пониманию многоуровневых протоколов. Он предоставляет различные сервисы, наиболее важным из которых является сквозной, надежный, ориентированный на соединение поток байтов от отправителя к получателю. Доступ к нему предоставляется при помощи сервисных примитивов, позволяющих устанавливать, использовать и разрывать соединения. Общепринятый интерфейс транспортного уровня обеспечивается сокетами Беркли.

Транспортные протоколы должны обладать способностью управлять соединением в ненадежных сетях. Установка соединения осложняется возможностью существования дубликатов пакетов, которые могут появляться в самый неподходящий момент. Для борьбы с этими дубликатами при установке соединения применяется алгоритм «тройного рукопожатия». Разрыв соединения проще установки и тем не менее далеко не тривиален из-за наличия проблемы двух армий.

Даже если сетевой уровень абсолютно надежен, у транспортного уровня полно работы. Он должен обрабатывать все служебные примитивы, управлять соединениями и таймерами, распределять пропускную способность и осуществлять контроль перегрузки, а также использовать скользящее окно переменного размера для управления потоком данных.

Контроль перегрузки должен справедливо распределять пропускную способность между конкурирующими потоками и следить за изменениями в использовании сети.

Закон управления AIMD позволяет получить эффективное и справедливое распределение.

Основными транспортными протоколами Интернета являются TCP и UDP. UDP — это протокол без установления соединения, который работает с IP-пакетами и занимается обеспечением мультиплексирования и демультиплексирования нескольких процессов с использованием единого IP-адреса. UDP может использоваться при клиент-серверных взаимодействиях, например при удаленном вызове процедур (RPC). Кроме того, на его основе можно создавать протоколы реального времени, такие как RTP.

Наиболее распространенным протоколом Интернета является TCP. Он обеспечивает надежный дуплексный поток байтов с контролем перегрузки. Он использует 20-байтный заголовок для всех сегментов. Оптимизации производительности протокола TCP было уделено много внимания. Для этого в нем применяются алгоритмы Нагля (Nagle), Кларка (Clark), Джекобсона (Jacobson), Карна (Karn) и др.

Производительность сети обычно в основном определяется протоколом и накладными расходами по обработке сегментов, причем с увеличением скорости передачи данных эта ситуация ухудшается. Протоколы должны разрабатываться таким образом, чтобы минимизировать количество сегментов и обтаваться работоспособными при больших значениях задержки передачи. В гигабитных сетях требуются простые протоколы и потоковая обработка.

Разработка сетей, устойчивых к задержкам, позволяет доставлять пакеты в сетях с непостоянной связностью узлов или с большой задержкой в каналах. Промежуточные узлы хранят, переносят и отправляют посылки данных, гарантируя их доставку, даже если ни в какой момент времени не существует действующего пути между отправителем и получателем.

Вопросы

1. В нашем примере транспортных примитивов, приведенных в табл. 6.1, `LISTEN` является блокирующим вызовом. Обязательно ли это? Если нет, объясните, как следует пользоваться неблокирующей базовой операцией. Какое преимущество это даст по сравнению со схемой, описанной в тексте?
2. Базовые операции транспортного сервиса предполагают, что во время установки соединения две конечные точки ведут себя несимметрично: один конец (сервер) выполняет операцию `LISTEN`, а другой конец (клиент) — операцию `CONNECT`. Но в одноранговых приложениях (например, в приложениях для совместного доступа к файлам, таких как BitTorrent) все конечные точки считаются одинаковыми. Среди них нет серверов и клиентов. Как такое приложение может работать, используя базовые операции транспортного сервиса?
3. В модели, лежащей в основе диаграммы состояний на рис. 6.3, предполагается, что пакеты могут теряться на сетевом уровне и поэтому должны подтверждаться индивидуально. Допустим, что сетевой уровень обеспечивает 100 % надежность доставки и никогда не теряет пакеты. Нужны ли какие-либо изменения в диаграмме состояний, показанной на рис. 6.3, и если да, то какие?
4. В обеих частях листинга 6.1 значение `SERVER_PORT` должно быть одинаковым у клиента и у сервера. Почему это так важно?
5. Посмотрите на пример с файловым сервером, приведенный в листинге 6.1. Может ли системный вызов клиента `connect()` закончиться неудачно по причине, отличной от переполнения очереди ожидания сервера? Считайте, что сеть идеальна.

6. Чтобы решить, следует ли все время поддерживать сервер в активном состоянии, или лучше запускать его по требованию с помощью обрабатывающего сервера, можно использовать такой критерий: частоту использования данного сервера. Можете ли вы придумать другой критерий?
7. Предположим, что используется управляемая часами схема генерирования начальных порядковых номеров с 15-разрядным счетчиком часов. Часы тикают раз в 100 мс, а максимальное время жизни пакета равно 60 с. Как часто должна производится ресинхронизация:

   1) в худшем случае?

   2) когда данные потребляют 240 порядковых номеров в минуту?
8. Почему максимальное время жизни пакета *T* должно быть достаточно большим, чтобы гарантировать, что не только пакет, но и его подтверждение исчезли?
9. Представьте, что для установки соединений вместо «тройного рукопожатия» использовалось бы «двойное» (то есть третье сообщение не требовалось). Возможны ли при этом тупиковые ситуации? Приведите пример или докажите, что тупиковых ситуаций нет.
10. Представьте себе обобщенную проблему *n* армий, в которой договоренность двух любых армий достаточна для победы. Существует ли протокол, позволяющий армиям синих выиграть?
11. Рассмотрим проблему восстановления от сбоев хостов (рис. 6.15). Если бы интервал между записью и отправкой подтверждения (или наоборот) можно было сделать относительно небольшим, какими были бы две лучшие стратегии отправителя и получателя, минимизирующие шансы ошибки протокола?
12. Представьте, что в сеть, изображенную на рис. 6.17, добавляется новый поток *E*, использующий путь через маршрутизаторы *R1*, *R2* и *R6*. Как изменится распределение пропускной способности по максиминному критерию для пяти потоков?
13. Обсудите преимущества и недостатки схемы кредитного протокола (получатель информирует отправителя, сколько блоков информации он может отправить) по сравнению с протоколами скользящего окна.
14. Существуют и другие политики, обеспечивающие равноправие при контроле перегрузки: аддитивное увеличение аддитивное уменьшение (AIAD), мультипликативное увеличение аддитивное уменьшение (MIAD), мультипликативное увеличение мультипликативное уменьшение (MIMD). Что вы можете сказать об их сходимости и стабильности?
15. Зачем нужен протокол UDP? Разве не достаточно было бы просто позволить пользовательским процессам посылать необработанные IP-пакеты?
16. Рассмотрите простой протокол прикладного уровня, построенный на основе UDP, который позволяет клиенту запрашивать файл с удаленного сервера, расположенного по общеизвестному адресу. Клиент вначале посылает запрос с именем файла, а сервер отвечает последовательностью информационных пакетов, содержащих различные части запрошенного файла. Для обеспечения надежности и доставки частей в правильном порядке клиент и сервер используют протокол с ожиданием. Какие проблемы могут возникнуть с таким протоколом, кроме очевидных проблем с производительностью? Обратите внимание на вероятность сбоя процессов.
17. Клиент посылает 128-байтный запрос на сервер, удаленный от него на 100 км, по оптоволокну со скоростью 1 Гбит/с. Какова эффективность линии во время выполнения удаленного вызова процедуры?
18. Рассмотрите снова ситуацию, описанную в предыдущем вопросе. Вычислите минимально возможное время ответа для данной линии со скоростью 1 Гбит/с и для линии со скоростью 1 Мбит/с. Какой вывод можно сделать, исходя из полученных значений?

19. Как в UDP, так и в TCP номера портов используются для идентификации принимающей подсистемы при доставке сообщения. Назовите две причины того, почему для этих протоколов были изобретены новые абстрактные идентификаторы (номера портов) и не использовались идентификаторы процессов, уже существовавшие на момент появления данных протоколов?
20. Некоторые реализации RPC позволяют клиенту выбрать между реализацией с помощью UDP и реализацией с помощью TCP. В каком случае клиент выберет первый вариант, а в каком — второй?
21. Рассмотрим две сети, *N1* и *N2*, с одинаковой средней задержкой при передаче пакетов от источника A получателю D. В сети *N1* задержка распределена равномерно с максимальным значением 10 с, а в сети *N2* 99 % пакетов имеют задержку менее одной секунды, но максимальная задержка может быть сколь угодно большой. Подумайте, как в таких ситуациях можно использовать RTP, если вы планируете передавать аудио/видео поток в режиме реального времени.
22. Каков суммарный размер минимального MTU протокола TCP, включая накладные расходы TCP и IP, но не включая накладные расходы канального уровня?
23. Фрагментация и дефрагментация дейтаграмм выполняется протоколом IP и невидима для протокола TCP. Означает ли это, что протокол TCP не должен беспокоиться о данных, приходящих в неверном порядке?
24. RTP используется для передачи звукозаписей, по качеству соответствующих компакт-дискам. При этом для передачи одного отсчета каждого из стереоканалов используется пара 16-битных слов, передающихся 44 100 раз в секунду. Сколько пакетов в секунду должен уметь передавать RTP?
25. Возможно ли поместить код RTP в ядро операционной системы наряду с UDP? Ответ поясните.
26. Процессу хоста 1 был назначен порт *p*, а процессу хоста 2 — порт *q*. Может ли быть одновременно несколько соединений между этими двумя портами?
27. На рис. 6.31 мы видели, что в дополнение к 32-разрядному полю *Номер подтверждение* в четвертом слове имеется бит *ACK*. Приносит ли он какую-либо пользу? Ответ поясните.
28. Максимальный размер полезной нагрузки TCP-сегмента может быть равен 65 495 байт. Почему было выбрано такое странное число?
29. Опишите два способа, которыми можно попасть в состояние *SYN RCVD* на рис. 6.33.
30. Рассмотрите эффект использования алгоритма медленного старта в линии со значением времени прохождения сигнала в оба конца, равным 10 мс, без перегрузок. Размер окна получателя 24 Кбайт, а максимальный размер сегмента равен 2 Кбайт. Через какое время может быть послано полное окно?
31. Предположим, окно перегрузки протокола TCP установлено на 18 Кбайт, когда происходит тайм-аут. Каким будет размер окна, если четыре последующих передачи будут успешными? Максимальный размер предполагается равным 1 Кбайт.
32. Текущее значение оценки времени прохождения сигнала в оба конца протокола TCP *RTT* равно 30 мс, а следующие подтверждения приходят через 26, 32 и 24 мс. Каково будет новое значение *RTT*? Используйте $\alpha = 0{,}9$.
33. TCP-машина посылает окна по 65 535 байт по гигабитному каналу, в котором время прохождения сигнала в один конец равно 10 мс. Какова максимальная достижимая пропускная способность канала? Чему равна эффективность использования линии?
34. Какова максимальная скорость, с которой хост может посылать в линию TCP-пакеты, содержащие 1500 байт полезной нагрузки, если максимальное время жизни пакета в сети

равно 120 с? Требуется, чтобы порядковые номера не зацикливались. При расчете учитывайте накладные расходы на TCP, IP и Ethernet. Предполагается, что кадры Ethernet могут посылаться непрерывно.

35. Чтобы исправить ограничения на адреса в IPv4, IETF приложила немало усилий и разработала IPv6, однако эта версия до сих пор внедряется неохотно. А к исправлению ограничений на адреса в TCP никто таких усилий не прилагает. Объясните, почему это так.

36. Чему равна максимальная скорость передачи данных для каждого соединения, если максимальный размер сегмента равен 128 байт, максимальное время жизни сегмента равно 30 с и используются 8-разрядные порядковые номера сегментов?

37. Предположим, вы измеряете время, необходимое для получения сегмента. Когда возникает прерывание, вы читаете показания системных часов в миллисекундах. После полной обработки сегмента вы снова читаете показания часов. В результате миллиона измерений вы получаете значения 0 мс 270 000 раз и 1 мс 730 000 раз. Какой вывод можно сделать на основании полученных результатов?

38. Центральный процессор выполняет 1000 млн инструкций в секунду (1000 MIPS). Данные могут копироваться 64-разрядными словами. На копирование каждого слова требуется 10 инструкций. Может ли такая система управлять гигабитной линией, если каждый приходящий пакет должен быть скопирован четырежды? Для простоты предположим, что все инструкции, даже обращения к памяти, выполняются с максимальной скоростью, 1000 MIPS.

39. Для решения проблемы повторного использования старых порядковых номеров пакетов, в то время как старые пакеты еще существуют, можно использовать 64-разрядные порядковые номера. Однако теоретически оптоволоконный кабель может обладать пропускной способностью до 75 Тбит/с. Каким следует выбрать максимальное время жизни пакетов, чтобы гарантировать отсутствие в сети будущего пакетов с одинаковыми номерами при скорости линий 75 Тбит/с и 64-разрядных порядковых номерах? Предполагается, что порядковый номер дается каждому байту, как в протоколе TCP.

40. В результате расчета выяснилось, что гигабитная линия отправляет хосту 80 000 пакетов в секунду, оставляя ему только 6250 инструкций на их обработку — половина производительности процессора отдается приложениям. При этом предполагалось, что размер пакета должен быть 1500 байт. Выполните подсчеты заново для пакетов ARPANET (128 байт). В обоих случаях предполагается, что в размер пакета включены все накладные расходы.

41. В гигабитной линии протяженности более 4000 км ограничивающим фактором является не пропускная способность, а время задержки. Рассмотрим региональную сеть со средней удаленностью отправителя от получателя 20 км. При какой скорости передачи данных время прохождения сигнала в оба конца будет равно времени передачи одного пакета размером 1 Кбайт?

42. Рассчитайте произведение пропускной способности на задержку в следующих сетях:

    1) T1 (1,5 Мбит/с);

    2) Ethernet (10 Мбит/с);

    3) T3 (45 Мбит/с);

    4) STS-3 (155 Мбит/с).

    Предполагается, что RTT = 100 мс. Не забудьте о том, что в заголовке TCP на размер окна отводится 16-разрядное поле. Как это замечание отразится на результатах вычислений?

43. Чему равно произведение пропускной способности на задержку для канала геостационарной спутниковой связи с пропускной способностью 50 Мбит/с? Если все пакеты имеют размер 1500 байт (включая накладные расходы), какого размера должно быть окно в пакетах?

44. Файловый сервер, моделируемый листингом 6.1, далек от совершенства. Можно внести некоторые улучшения. Проделайте следующие изменения:

    1) Пусть у клиента появится третий аргумент, указывающий байтовый диапазон.

    2) Добавьте флаг –w в программу клиента, который позволил бы записывать файл на сервер.

45. Почти все сетевые протоколы должны уметь работать с сообщениями. Если вы помните, протоколы передают сообщения путем добавления/отделения заголовков. Некоторые протоколы могут разбивать сообщение на несколько фрагментов, а потом восстанавливать его из этих фрагментов. Попробуйте разработать библиотеку управления сообщениями, реализовав поддержку создания нового сообщения, добавления/отделения заголовка, разбиения одного сообщения на два, объединения двух сообщений в одно и сохранения копии сообщения. Насколько это возможно, минимизируйте копирование данных из одного буфера в другой. Важно, чтобы эти операции работали только с указателями, не затрагивая данных, содержащихся в сообщении.

46. Разработайте и реализуйте систему сетевого общения (чат), рассчитанную на несколько групп пользователей. Координатор чата должен располагаться по общеизвестному сетевому адресу, использовать для связи с клиентами протокол UDP, настраивать чат-серверы перед каждой сессией общения и поддерживать каталог чат-сессий. На каждую сессию должен выделяться один обслуживающий сервер. Для связи с клиентами сервер должен использовать TCP. Клиентская программа должна позволять пользователям начинать разговор, присоединяться к уже ведущейся дискуссии и покидать сессию. Разработайте и реализуйте код координатора, сервера и клиента.

Глава 7
Прикладной уровень

Завершив изучение базовых сведений о компьютерных сетях, мы переходим к уровню, на котором расположены все приложения. Все уровни, находящиеся ниже прикладного, служат для обеспечения надежной доставки данных, но никаких полезных для пользователя действий не производят. В этой главе мы изучим некоторые реальные сетевые приложения.

Разумеется, даже прикладной уровень нуждается в обслуживающих протоколах, с помощью которых осуществляется функционирование приложений. Соответственно, прежде чем начать рассмотрение самих приложений, мы изучим один из таких протоколов. Речь идет о службе имен доменов, DNS, обеспечивающей присвоение имен в Интернете. Затем мы рассмотрим три реально действующих приложения: электронную почту, Всемирную паутину и, наконец, мультимедиа. Мы закончим эту главу рассказом о доставке контента, в том числе и в равноранговых (пиринговых — peer-to-peer или сокращенно p2p) сетях.

7.1. Служба имен доменов DNS

Хотя программы теоретически могут обращаться к веб-страницам, почтовым ящикам и другим ресурсам по сетевым адресам компьютеров (например, IP), на которых хранится данная информация, пользователям тяжело запоминать такие адреса. Кроме того, размещение веб-страницы компании по адресу *128.111.24.41* будет означать, что в случае переезда сервера компании на новую машину новый IP будет необходимо сообщить всем заинтересованным лицам. Для отделения имен машин от их адресов было решено использовать понятные имена высокого уровня. Поэтому обратиться к веб-серверу компании можно по адресу *www.cs.washington.edu*. Тем не менее, так как сеть сама по себе понимает только числовые адреса, нужен механизм преобразования имен в сетевые адреса. В следующих разделах мы изучим, как производится это отображение в Интернете.

Когда-то давно во времена сети ARPANET соответствие между текстовыми и числовыми адресами просто записывалось в файле *hosts.txt*, в котором перечислялись все имена компьютеров и их IP-адреса. Каждую ночь все хосты получали этот файл с сайта, на котором он хранился. В сети, состоящей из нескольких сотен больших машин, работающих под управлением системы с разделением времени, такой подход оправдывал себя.

Однако еще за долго до того, как к сети были подключены миллионы компьютеров, всем стало ясно, что этот способ не сможет работать вечно. Во-первых, размер файла рано или поздно стал бы слишком большим. Однако, что еще важнее, если управление именами хостов не осуществлять централизованно, неизбежно возникновение конфликтов имен. В то же время представить себе централизованное управление именами всех хостов гигантской международной сети довольно сложно. Для разрешения всех этих проблем в 1983 году и была разработана **служба имен доменов (DNS, Domain Name System)**. С тех пор она стала важнейшей частью Интернета.

Суть системы DNS заключается в иерархической схеме имен, основанной на доменах, и распределенной базе данных, реализующей эту схему имен. В первую очередь эта система используется для преобразования имен хостов в IP-адреса, но также может использоваться и в иных целях. Определение системы DNS дано в RFC 1034, 1035, 2181 и далее разработано во многих других.

В общих чертах система DNS применяется следующим образом. Для преобразования имени в IP-адрес прикладная программа обращается к библиотечной процедуре, называющейся **распознавателем (resolver)**, передавая ей имя в качестве параметра. Мы видели пример распознавателя *gethostbyname* в листинге 6.1. Распознаватель посылает запрос, содержащий имя, локальному DNS-серверу, который ищет имя и возвращает соответствующий IP-адрес распознавателю, который, в свою очередь, передает этот адрес вызвавшей его прикладной программе. Запрос и ответ передаются как UDP-пакеты. Имея IP-адрес, программа может установить TCP-соединение с адресатом или послать ему UDP-пакеты.

7.1.1. Пространство имен DNS

Управление большим и постоянно изменяющимся набором имен представляет собой нетривиальную задачу. В почтовой системе на письмах требуется указывать (явно или неявно) страну, штат или область, город, улицу, номер дома, квартиру и фамилию получателя. Благодаря использованию такой иерархической схемы не возникает путаницы между Марвином Андерсоном, живущим на Мейн-стрит в Уайт-Плейнс, штат Нью-Йорк, и Марвином Андерсоном с Мейн-стрит в Остине, штат Техас. Система DNS работает аналогично.

Для Интернета основа иерархии именования разработана организацией под названием **ICANN (Internet Corporation for Assigned Names and Numbers — интернет-корпорация по присвоению имен и адресов)**. ICANN была создана для этих целей в 1998 году, так как Интернет развился во всемирный экономический концерн. Интернет концептуально разделен на более чем 250 **доменов верхнего уровня (top-level domains)**. Доменами называют в Интернете множество хостов, объединенных в логическую группу. Каждый домен верхнего уровня подразделяется на поддомены (subdomains), которые, в свою очередь, также могут состоять из других доменов и т. д. Все эти домены можно рассматривать в виде дерева, показанного на рис. 7.1. Листьями дерева являются домены, не разделяющиеся на поддомены (но состоящие из хостов, конечно). Такой конечный домен может состоять из одного хоста или может представлять компанию и содержать в себе тысячи хостов.

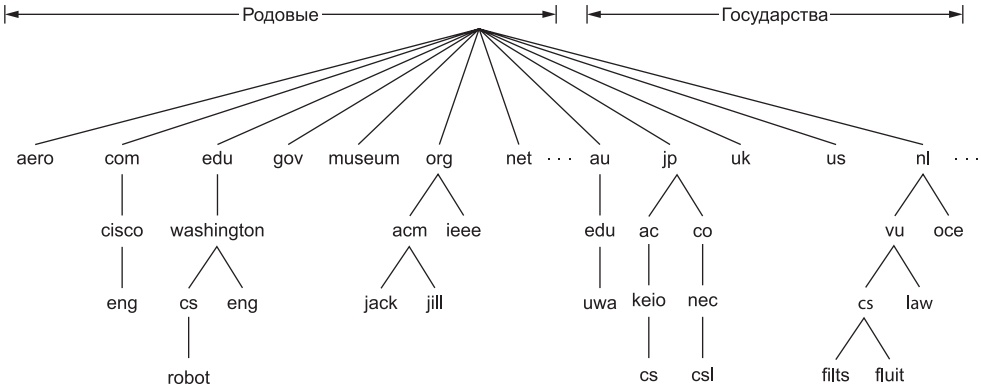


Рис. 7.1. Часть доменного пространства имен Интернета

Домены верхнего уровня разделяются на две группы: родовые домены и домены государств. К родовым относятся домены, перечисленные в табл 7.1, включая оригинальные домены, созданные 1980-х годах, и новые, введенные ICANN. В будущем будут добавляться новые базовые домены высшего уровня.

Таблица 7.1. Родовые домены верхнего уровня

Домен	Использование	Дата основания	Ограниченный?
com	Коммерческие цели	1985	Нет
edu	Образовательные учреждения	1985	Да
gov	Правительство	1985	Да
int	Международные организации	1985	Да
mil	Военные	1985	Да
net	Сетевые провайдеры	1985	Нет
org	Некоммерческие организации	1985	Нет
aero	Авиатранспорт	2001	Да
biz	Бизнес	2001	Нет
coop	Кооперативы	2001	Да
info	Информация	2002	Нет
museum	Музеи	2002	Да
name	Люди	2002	Нет
pro	Профессионалы	2002	Да
cat	Каталония	2005	Да
jobs	Занятость	2005	Да
mobi	Мобильные устройства	2005	Да
tel	Контактная информация	2005	Да
travel	Индустрия путешествий	2005	Да
xxx	Секс-индустрия	2010	Нет

За каждым государством в соответствии с международным стандартом ISO 3166 закреплен один домен государства. Интернационализированные доменные имена стран, в которых используется алфавит, отличный от латинского, были введены в 2010 году. Эти домены позволяют именовать хосты, используя арабские, кириллические, китайские и другие письменности.

Зарезервировать домен второго уровня, такой как *имя_компании.com*, просто. Домены высшего уровня управляются **регистраторами (registrars)**, назначенными ICANN. Для того чтобы получить имя, нужно просто обратиться к соответствующему регистратору (в данном случае *com*) и проверить, доступно ли желаемое имя и не является ли оно чьей-либо торговой маркой. Если все в порядке, заказчик регистрируется и за небольшую ежегодную абонентскую плату получает домен второго уровня.

Однако по мере коммерциализации и интернационализации Интернета появляется все больше спорных вопросов, особенно в отношении именования доменов. Эти споры захватывают и саму ICANN. Так, например, создание домена *xxx* заняло несколько лет и повлекло несколько судебных разбирательств. Размещение порно-контента на отдельном домене — это хорошо или плохо? (Некоторые и вовсе не хотели бы видеть в Интернете сайты с порнографическим содержанием, другие хотели бы разместить их все на один домен, чтобы было легче заблокировать их при помощи программ родительского контроля.) Некоторые домены являются самоорганизующимися, на других существуют ограничения и не всякий может получить имя (как показано в табл. 7.1). Но какие ограничения уместны? Взять хотя бы домен *pro*. Он предназначен для квалифицированных специалистов. Но кто является специалистом, а кто нет? Понятно, что доктора и адвокаты — это профессионалы, спору нет. А что делать со свободными фотографами, учителями музыки, заклинателями, водопроводчиками, парикмахерами, мусорщиками, татуировщиками, наемниками и проститутками? Имеют ли право квалифицированные представители всех этих и многих других профессий получать домены *pro*? Кто должен это определять?

Кроме того, на именах можно зарабатывать. Так страна Тувалу сдала в аренду права на свой домен *tv* за $50 млн благодаря тому, что код страны отлично подходит для рекламы телевизионных сайтов. Практически все общеупотребительные английские слова используются в качестве имен поддоменов *com*, вместе с наиболее частыми опечатками. Попробуйте набрать какое-нибудь слово, касающееся домашнего хозяйства, животных, растений, частей тела и т. д. У самой практики регистрации доменных имен с целью их дальнейшей продажи заинтересованной стороне даже есть название — **киберсквоттинг (cybersquatting)**. Многие компании, которые оказались не достаточно шустрыми в этом вопросе, обнаружили, что самые очевидные доменные имена уже заняты, когда началась эра Интернета и они попытались зарегистрироваться. В общем и целом, если не были нарушены права на товарный знак и не было предпринято мошеннических действий, в отношении имен работает правило «первым запросил — первым получил». Тем не менее политика в отношении разрешения споров по поводу имен все еще не до конца устоялась.

Имя каждого домена, подобно полному пути к файлу в файловой системе, состоит из пути от этого домена до (безымянной) вершины дерева. Компоненты пути разделяются точками. Так, домен технического отдела корпорации Cisco может выглядеть как *eng.cisco.com*, а не так, как это принято в стиле UNIX (*/com/cisco/eng*). Следует отметить, что из-за такой иерархической системы наименования *eng.cisco.com* не кон-

фликтует с потенциальным использованием имени *eng* в домене *eng.washington.edu*, где он может обозначать факультет английского языка Вашингтонского университета.

Имена доменов могут быть абсолютными и относительными. Абсолютное имя домена всегда оканчивается точкой (например, *eng.cisco.com.*), тогда как относительное имя — нет. Для того чтобы можно было единственным образом определить истинные значения относительных имен, они должны интерпретироваться в некотором контексте. В любом случае именованный домен означает определенный узел дерева и все узлы под ним.

Имена доменов нечувствительны к изменению регистра символов. Так, например, *edu*, *Edu* и *EDU* означают одно и то же. Длина имен компонентов может достигать 63 символов, а длина полного пути не должна превосходить 255 символов.

В принципе, новые домены могут добавляться в дерево с использованием как родового домена, так и домена, обозначающего страну. Например, *cs.wasington.edu* можно без проблем поместить в домен *us* под именем *cs.wasington.wa.us*. На практике, однако, почти все организации в США помещаются под родовыми доменами, тогда как почти все организации за пределами США располагаются под доменами своих государств. Не существует каких-либо правил, запрещающих регистрацию под несколькими доменами верхнего уровня. Большие компании зачастую именно так и поступают (например, *sony.com*, *sony.net* и *sony.nl*).

Каждый домен управляет распределением доменов, расположенных под ним. Например, в Японии домены *ac.jp* и *co.jp* соответствуют американским доменам *edu* и *com*. В Голландии подобное различие не используется, и все домены организаций помещаются прямо под доменом *nl*. В качестве примера приведем имена доменов факультетов вычислительной техники («компьютерных наук» — computer science) трех университетов.

1. *cs.wasington.edu* (Вашингтонский университет, США)
2. *cs.vu.nl* (университет Врийе, Нидерланды)
3. *cs.keio.ac.jp* (университет Кейо, Япония)

Для создания нового домена требуется разрешение домена, в который он будет включен. Например, если в Вашингтонском университете образовалась группа VLSI, которая хочет зарегистрировать домен *vlsi.cs.wasington.edu*, ей нужно разрешение от того, кто управляет доменом *cs.wasington.edu*. Аналогично, если создается новый университет, например, университет Северной—Южной Дакоты, он должен попросить менеджера домена *edu* присвоить их домену имя *unsd.edu* (если оно еще не занято). Таким образом, удается избежать конфликта имен, а каждый домен отслеживает состояние всех своих поддоменов. После того как домен создан и зарегистрирован, в нем могут создаваться поддомены, например *cs.unsd.edu*, для чего уже не требуется разрешения вышестоящих доменов.

Структура доменов отражает не физическое строение сети, а логическое разделение между организациями и их внутренними подразделениями. Так, если факультеты вычислительной техники и электротехники располагаются в одном здании и пользуются одной общей локальной сетью, они тем не менее могут иметь различные домены. И наоборот, если, скажем, факультет вычислительной техники располагается в двух различных корпусах университета с различными локальными сетями, логически все хосты обоих зданий обычно принадлежат к одному и тому же домену.

7.1.2. Записи ресурсов доменов

У каждого домена, независимо от того, является ли он одиноким хостом или доменом верхнего уровня, может быть набор ассоциированных с ним **записей ресурсов (resource records)**. Эти записи являются базой данных DNS. Для одинокого хоста запись ресурсов чаще всего представляет собой просто его IP-адрес, но существует также много других записей ресурсов. Когда распознаватель передает имя домена DNS-серверу, то, что он получает обратно, представляет собой записи ресурсов, ассоциированные с его именем. Таким образом, истинное назначение системы DNS заключается в преобразовании доменных имен в записи ресурсов.

Запись ресурса состоит из пяти частей. Хотя для эффективности они часто перекодируются в двоичную форму, в большинстве описаний записи ресурсов представлены в виде ASCII-текста, по одной строке на запись ресурса. Мы будем использовать следующий формат:

```
Domain_name    Time_to_live    Class    Type    Value
```

Поле *Domain_name* (имя домена) обозначает домен, к которому относится текущая запись. Обычно для каждого домена существует несколько записей ресурсов, и каждая копия базы данных хранит информацию о нескольких доменах. Поле имени домена является первичным ключом поиска, используемым для выполнения запросов. Порядок записей в базе данных значения не имеет. В ответ на запрос о домене возвращаются все удовлетворяющие запросу записи требуемого класса.

Поле *Time_to_live* (время жизни) указывает, насколько стабильно состояние записи. Редко меняющимся данным присваивается высокое значение этого поля, например, 86 400 (число секунд в сутках). Непостоянная информация помечается небольшим значением, например, 60 (1 минута). Мы вернемся к этому вопросу позднее, когда будем обсуждать кэширование.

Третьим полем каждой записи является поле *Class* (класс). Для информации Интернета значение этого поля всегда равно *IN*. Для прочей информации применяются другие коды, однако на практике они встречаются редко.

Поле *Type* (тип) означает тип DNS-записи. Их существует довольно много. Важные типы записей перечислены в табл. 7.2.

Запись SOA (**Start Of Authority — начальная точка полномочий**) сообщает имя первичного источника информации о зоне сервера имен (описанного ниже), адрес электронной почты его администратора, уникальный порядковый номер, различные флаги и тайм-ауты.

Самой важной является запись ***A*** (**Address — адрес**). Она содержит 32-разрядный IPv4-адрес интерфейса для хоста. У соответствующей записи **AAAA** (**«quad A» — «четыре A»**) есть 128-разрядный IPv6-адрес. У каждого хоста в Интернете должен быть по меньшей мере один IP-адрес, чтобы другие машины могли с ним общаться. На некоторых хостах может быть одновременно установлено несколько сетевых соединений. В этом случае им требуется по две или более записи типа *A* или *AAAA*. Соответственно, DNS может выдавать несколько адресов на одно имя.

Запись *MX* является стандартной. В ней указывается имя хоста, готового принимать почту для указанного домена. Дело в том, что не каждая машина может заниматься приемом почты. Если кто-нибудь хочет послать письмо на адрес, например *bill@*

microsoft.com, то отправляющему хосту нужно будет вначале найти почтовый сервер на *microsoft.com*. Запись *MX* может помочь в этих поисках.

Таблица 7.2. Основные типы записей ресурсов DNS

Тип	Смысл	Значение
SOA	Начальная запись зоны	Параметры для этой зоны
A	IPv4-адрес хоста	Целое число, 32 двоичных разряда
AAAA	IPv6-адрес хоста	Целое число, 128 двоичных разрядов
MX	Обмен почтой	Приоритет, с которым домен желает принимать электронную почту
NS	Сервер имен	Имя сервера для этого домена
CNAME	Каноническое имя	Имя домена
PTR	Указатель	Псевдоним IP-адреса
SPF	Правила отправки почты	Правила отправки почты, закодированные в текстовом виде
SRV	Сервис	Хост, предоставляющий данный сервис
TXT	Текст	Не интерпретируемый ASCII-текст

Еще один важный тип записи — это *NS*. Запись *NS* содержит информацию о сервере имени для домена или поддомена. Это хост, на котором содержится копия базы данных для домена. Он используется в процессе поиска имени, поэтому мы вкратце опишем этот процесс.

Записи *CNAME* позволяют создавать псевдонимы. Представим себе, что человек, знакомый в общих чертах с формированием имен в Интернете, хочет послать сообщение пользователю *paul* на отделении вычислительной техники Массачусетского технологического института (M.I.T.). Он может попытаться угадать нужный ему адрес, составив строку *paul@cs.mit.edu*. Однако этот адрес работать не будет, так как домен отделения вычислительной техники Массачусетского технологического института на самом деле называется *csail.mit.edu*. Таким образом, для удобства тех, кто этого не знает, M.I.T. может создать запись *CNAME*, позволяющую обращаться к нужному домену по обоим именам. Такая запись будет иметь следующий вид:

```
cs.mit.edu    86400    IN    CNAME    csail.mit.edu
```

Как и *CNAME*, запись *PTR* указывает на другое имя. Однако в отличие от записи *CNAME*, являющейся, по сути, макроопределением (то есть механизмом замены одной строки другой), *PTR* представляет собой обычный тип данных DNS, интерпретация которого зависит от контекста. На практике запись *PTR* почти всегда используется для ассоциации имени с IP-адресом, что позволяет по IP-адресу находить имя соответствующей машины. Это называется **обратным поиском (reverse lookups)**.

Запись *SRV* — это новый тип, позволяющий определять хост для искомого сервиса в домене. Например, веб-сервер для *cs.wasington.edu* может быть определен как *cockatoo.cs.wasington.edu*. Данная запись является расширенным вариантом записи *MX*, которая выполняет ту же задачу в рамках почтовых сервисов.

SPF — также новый тип записи. Он позволяет домену закодировать информацию о том, какие машины будут отсылать с него письма в остальную часть Интернета. Это помогает принимающим машинам проверять, допустима ли данная почта. Если почта приходит с машины, которая называется *dodgy*, а доменные записи говорят о том, что почта с домена будет отсылаться только машиной под названием *smtp*, велики шансы того, что данные сообщения являются спамом.

Последние в списке, *TXT*-записи изначально предназначались для того, чтобы позволить доменам идентифицировать себя произвольным образом. Сегодня с их помощью обычно кодируется информация, предназначенная для считывания машиной, обычно это *SPF*-информация.

Наконец, последнее поле записи ресурса — это поле *Value* (значение) — может быть числом, именем домена или текстовой ASCII-строкой. Смысл поля зависит от типа записи. Краткое описание поля Value для каждого из основных типов записей дано в табл. 7.2.

Пример информации, хранящейся в базе данных DNS домена, приведен в листинге 7.1. В нем показана часть (гипотетической) базы данных домена *cs.vu.nl*, представленного также в виде узла дерева доменов на рис. 7.1. В базе данных содержится семь типов записей ресурсов.

Листинг 7.1. Часть возможной базы данных домена cs.vu.nl

```
; Официальная информация для cs.vu.nl
cs.vu.nl.      86400  IN  SOA  star boss (9527,7200,7200,241920,86400)
cs.vu.nl.      86400  IN  MX   1 zephyr
cs.vu.nl.      86400  IN  MX   2 top
cs.vu.nl.      86400  IN  NS   star

star           86400  IN  A    130.37.56.205
zephyr         86400  IN  A    130.37.20.10
top            86400  IN  A    130.37.20.11
www            86400  IN  CNAME  star.cs.vu.nl
ftp            86400  IN  CNAME  zephyr.cs.vu.nl

flits          86400  IN  A    130.37.16.112
flits          86400  IN  A    192.31.231.165
flits          86400  IN  MX   1 flits
flits          86400  IN  MX   2 zephyr
flits          86400  IN  MX   3 top

rowboat           IN  A    130.37.56.201
                  IN  MX   1 rowboat
                  IN  MX   2 zephyr

little-sister     IN  A    130.37.62.23

laserjet          IN  A    192.31.231.216
```

В первой не закомментированной строке листинга 7.1 дается основная информация о домене, которая в дальнейшем нас интересовать не будет. Следующие две строки

определяют два хоста, с которыми следует связаться в первую очередь при попытке доставить электронную почту, посланную по адресу *person@cs.vu.nl*. Хост по имени *zephyr* (специальная машина) следует опросить первым. В случае неудачи следует попробовать доставить письмо машине по имени *top*. В следующей строке определен сервер имен для домена *star*.

После пустой строки, добавленной для удобства чтения, следуют строки, сообщающие IP-адреса для *star*, *zephyr* и *top*. Далее следует псевдоним *www.cs.vu.nl*, позволяющий не обращаться к какой-то конкретной машине. Создание этого псевдонима позволяет домену *cs.vu.nl* изменять свой WWW-сервер, не меняя адреса, по которому пользователи смогут продолжать к нему обращаться. То же справедливо и для домена *ftp.cs.vu.nl* — FTP-сервера.

В секции, предназначенной для машины *flits*, перечислены два IP-адреса и три возможных варианта адреса для обработки почты, отосланной на *flits.cs.vu.nl*. В первую очередь, естественно, следует пытаться доставить письмо самому компьютеру *flits*. Но если этот хост выключен, следует продолжать попытки, обращаясь к хостам *zephyr* и *top*.

Следующие три строки содержат типичные записи для компьютера, в данном случае для *rowboat.cs.vu.nl*. Хранящаяся в базе данных информация содержит IP-адрес, а также имена первого и второго хостов для доставки почты. Следом идет запись о машине, которая сама не способна получать почту. Последняя строка, вероятно, описывает лазерный принтер, подключенный к Интернету.

7.1.3. Серверы имен

Теоретически один сервер мог бы содержать всю базу данных DNS и отвечать на все запросы к ней. На практике этот сервер оказался бы настолько перегруженным, что был бы просто бесполезным. Более того, если бы с ним когда-нибудь что-нибудь случилось, то весь Интернет не работал бы.

Чтобы избежать проблем, связанных с хранением всей информации в одном месте, пространство имен DNS разделено на непересекающиеся **зоны (zones)**. Один возможный способ разделения пространства имен, показанного на рис. 7.1, на зоны изображен на рис. 7.2. Каждая очерченная зона содержит часть общего дерева доменов.

Расстановка границ зон целиком зависит от администратора зоны. Это решение основывается на том, сколько серверов имен требуется в той или иной зоне. Например, на рис. 7.2 у Вашингтонского университета есть зона для *washington.edu*, управляющая доменом *eng.washington.edu*, но не доменом *cs.washington.edu*, расположенным в отдельной зоне со своими серверами имен. Подобное решение может быть принято, когда факультет английского языка не хочет управлять собственным сервером имен, но этого хочет факультет вычислительной техники.

Каждая зона также ассоциируется с одним или более сервером имен. Это хосты, на которых находится база данных для зоны. Обычно у зоны есть один основной сервер имен, который получает информацию из файла на своем диске, и один или более второстепенных серверов имен, получающих информацию с основного сервера имен. Для повышения надежности некоторые серверы имен могут быть расположены вне зоны.

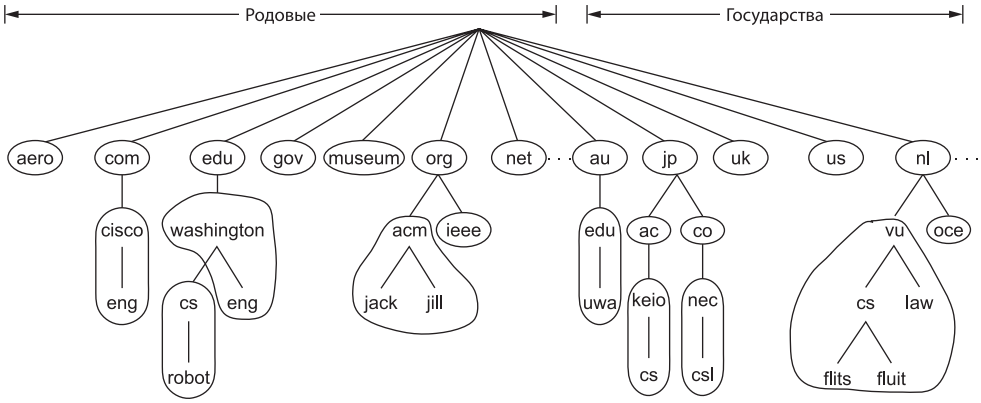


Рис. 7.2. Часть пространства имен DNS, разделенная на очерченные зоны

Процесс поиска адреса по имени называется **разрешением имен (name resolution)**. Распознаватель обращается с запросом разрешения имени домена к локальному серверу имен. Если искомый домен относится к сфере ответственности данного сервера имен, как, например, домен *top.cs.vu.nl* подпадает под юрисдикцию домена *cs.vu.nl*, тогда данный DNS-сервер сам отвечает распознавателю на его запрос, передавая ему **авторитетную запись (authoritative record)** ресурса. Авторитетной называют запись, получаемую от официального источника, хранящего данную запись и управляющего ее состоянием. Поэтому такая запись всегда считается верной, в отличие от **кэшируемых записей (cached records)**, которые могут устаревать.

Однако что происходит, если домен удаленный, как, например, в случае, когда *flits.cs.vu.nl* пытается найти IP-адрес для *robot.cs.washington.edu* в Вашингтонском университете? В этом случае, если в кэше нет информации о запрашиваемом домене, доступном локально, сервер имен посылает удаленный запрос. Поясним данный процесс на примере, показанном на рис. 7.3. На первом шаге (обозначен «1») посылается запрос локальному серверу имен. Этот запрос содержит имя искомого домена, тип (*A*) и класс (*IN*).

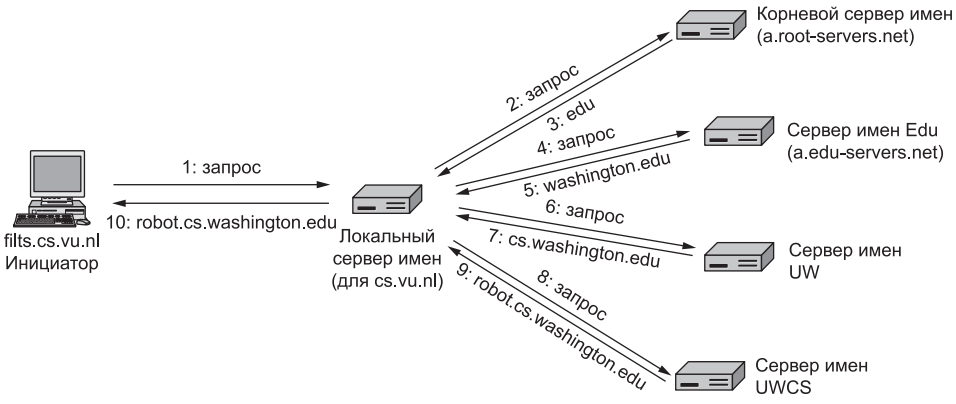


Рис. 7.3. Пример поиска распознавателем имени удаленного хоста в десяти шагах

На следующем шаге посылается запрос на один из **корневых серверов имен (root name servers)**, находящихся на вершине иерархии. На этих серверах имен хранится информация о каждом домене высшего уровня. Этот запрос показан как шаг 2 на рис. 7.3. Чтобы связаться с корневым сервером, на каждом сервере имен должна быть информация об одном или более корневых серверах имен. Обычно эта информация представлена в файле системной конфигурации, который загружается в кэш DNS, когда запускается сервер DNS. Он является просто списком записей *NS* и соответствующих записей *A*.

Существует 13 корневых серверов DNS, которые называются незамысловато — от *a-root-servers.net* до *m.root-servers.net*. Каждый корневой сервер логически мог бы быть отдельным компьютером. Однако так как весь Интернет зависит от корневых серверов, они являются мощными машинами, а информация, хранящаяся на них, неоднократно дублируется. Большинство серверов расположено в различных географических точках, и доступ к ним осуществляется посредством адресации любому устройству из группы, при этом пакет доставляется на ближайший адрес (мы описали адресацию любому устройству группы в пятой главе). Дублирование информации повышает надежность и производительность.

Маловероятно, чтобы этот корневой сервер имен знал адрес машины в Вашингтонском университете. Скорее всего, он даже не знает адреса сервера имен самого университета, однако он должен знать сервер имен домена *edu*, на котором расположен *cs.washington.edu*. Он возвращает имя и IP-адрес для части ответа на третьем шаге.

Далее локальный сервер имен продолжает этот сложный путь. Он направляет запрос серверу имен *edu* (*a.edu-servers.net*), который выдает имя сервера Вашингтонского университета. Этот процесс проиллюстрирован шагами 4 и 5. Теперь мы уже подошли ближе. Локальный сервер имен отсылает запрос на сервер имен Вашингтонского университета (шаг 6). Если искомое имя домена находится на факультете английского языка, будет получен ответ, так как зона университета этот факультет охватывает. Но факультет вычислительной техники решил запустить собственный сервер имен. Запрос возвращает имя и IP-адрес сервера имен факультета вычислительной техники Вашингтонского университета (шаг 7).

Наконец, локальный сервер имен запрашивает сервер имен факультета вычислительной техники Вашингтонского университета (шаг 8). Этот сервер отвечает за домен *cs.washington.edu*, так что он должен выдать ответ. В итоге окончательный ответ возвращается (шаг 9), и локальный сервер имен передает его на *flits.cs.vu.nl* (шаг 10). Имя получено.

Вы можете изучить этот процесс, используя стандартные программы типа `dig`, которые установлены на большинстве UNIX-систем. Например, напечатав

```
dig @a.edu-servers.net robot.cs.washington.edu
```

вы отправите запрос *robot.cs.washington.edu* на сервер имен *a.edu-servers.net* и получите распечатку результата. Так вы увидите информацию, которую мы получили на четвертом шаге в нашем примере, и узнаете имя и IP-адреса серверов имен Вашингтонского университета.

В этом длинном сценарии есть три технических момента, требующих пояснения. Во-первых, на рис. 7.3 используется два разных механизма запроса. Когда хост *flits.cs.vu.nl* отсылает запрос на локальный сервер имен, этот сервер выполняет запрос от

имени *flits*, пока не получит ответ, который можно будет вернуть. Он не возвращает частичных ответов. Они могут быть полезными, но в запросе о них нет ни слова. Этот механизм называется рекурсивным **запросом (recursive query)**.

С другой стороны, корневой сервер имен (и каждый последующий) не продолжает рекурсивно запрос локального сервера имен. Он возвращает лишь частичный ответ и переходит к следующему запросу. Локальный сервер имен отвечает за продолжение поиска ответа, направляя следующие запросы. Этот механизм называется **итеративным запросом (iterative query)**.

В одном процессе поиска имени могут быть задействованы оба механизма, как показано в этом примере. Рекурсивные запросы практически всегда кажутся предпочтительными, но многие серверы имен (особенно корневые) их не обрабатывают. Они слишком загружены. Итеративные запросы накладывают груз обработки запроса на ту машину, которая их порождает. Для локального сервера имен разумно поддерживать рекурсивные запросы, чтобы предоставлять сервис хостам на своем домене. Эти хосты не обязательно должны быть сконфигурированы таким образом, чтобы обегать все серверы имен, им нужна лишь возможность обратиться к локальному.

Второе, на чем стоит заострить внимание, — это кэширование. Все ответы, в том числе все возвращенные частичные ответы, сохраняются в кэше. Таким образом, если другой хост *cs.vu.nl* запрашивает *robot.cs.washington.edu*, ответ будет уже известен. Более того, если хост запрашивает другой хост на том же домене, например *galah.cs.washington.edu*, ответ может быть отослан напрямую на сервер имен, который отвечает за это имя. Сходным образом запросы на другие домены на *washington.edu* могут начинаться напрямую с сервера имен *washington.edu*. Использование ответов, сохраненных в кэше, серьезно сокращает количество шагов в запросе и повышает производительность. Сценарий, который мы набросали, на самом деле, является худшим из возможных вариантов, так как в кэше нет полезной информации.

Однако ответы, сохраненные в кэше, не являются авторитетными, так как изменения в домене *cs.washington.edu* не будут распространяться автоматически на все кэши, в которых может храниться копия этой информации. По этой причине записи кэша обычно долго не живут. В каждой записи ресурса присутствует поле *Time_to_live*. Оно сообщает удаленным серверам, насколько долго следует хранить эту запись в кэше. Если какая-либо машина сохраняет постоянный адрес годами, возможно, будет достаточно надежно хранить эту информацию в кэше в течение одного дня. Для более непостоянной информации, вероятно, более осмотрительно удалять все записи через несколько секунд или одну минуту[1].

Третий момент, на котором нам хотелось бы заострить внимание, — это протокол транспортного уровня, который используется для запросов и ответов, UDP. DNS-

[1] Практика показывает, что удобнее, наоборот, хранить эту информацию на своем хосте, для чего существуют специальные программы, поддерживающие мини-базы DNS для часто используемых адресов на компьютерах пользователя. — *Примеч. перев.*

Тем не менее использовать любые вариации локального кэша доменных имен или, тем более, локального сервера имен надо весьма осмотрительно, чтобы в самый не подходящий момент не увидеть вместо искомой странички нечто совершенно невостребованное (просто поменялся провайдер, и на старом IP-адресе уже совершенно другой сайт). — *Примеч. ред.*

сообщения посылаются в UDP-пакетах, имеющих простой формат для запросов, ответов и имен серверов, которые могут быть использованы для продолжения процесса получения ответа. Мы не будем углубляться в детали этого формата. Если в течение некоторого (непродолжительного) времени ответ не приходит, DNS-клиент повторяет запрос и проверяет другой сервер домена после нескольких таких попыток. Этот процесс сконструирован таким образом, чтобы работа не прекращалась, если вышел из строя один из серверов или потерялся один из пакетов. В каждый запрос включен 16-битный идентификатор; он копируется в ответ, и, таким образом, сервер имен может выдавать необходимые ответы на соответствующие им запросы даже в том случае, если в одно и то же время поступает большое количество запросов.

Хотя цель создания DNS проста и понятна, должно быть ясно, что это большая и сложная распределенная система, состоящая из миллионов совместно работающих серверов имен. DNS формирует ключевую связь между удобными для чтения пользователями именами доменов и IP-адресами машин. Она включает дублирование информации и кэширование, направленные на высокую производительность и надежность, и сконструирована таким образом, чтобы быть крайне функциональной.

Мы ничего не писали о безопасности, но, как вы понимаете, способность менять имя на адрес и обратно может иметь крайне неприятные последствия, если пользоваться ей во вред. По этой причине для DNS были разработаны расширения, обеспечивающие безопасность, под названием DNSSEC. Мы опишем их в восьмой главе.

Кроме того, иногда приложениям требуется использовать имена более гибким образом, например назвать контент и обратиться к IP-адресу ближайшего хоста, на котором этот контент есть. Так происходит, в том числе, поиск и скачивание фильмов. В этом случае нас интересует именно фильм, а не компьютер, на котором есть его копия, так что нам нужен только IP-адрес любого ближайшего компьютера, на котором есть искомый ролик. Чтобы достичь такого результата, можно использовать сеть доставки контента (CDN). Как сделать это при помощи DNS, мы опишем позднее, в разделе 7.5.

7.2. Электронная почта

Электронная почта, или как ее часто называют — **e-mail**, существует уже более трех десятилетий. Она стала популярной с первых дней развития Интернета благодаря скорости и дешевизне. До 1990 года она использовалась преимущественно в научных организациях. В девяностые годы она получила широкую известность, и с тех пор количество отправляемых с помощью электронной почты писем стало расти экспоненциально. Среднее число сообщений, посылаемых ежедневно, скоро во много раз превзошло число писем, отправляемых с помощью **обычной почты (snail mail)**, то есть бумажной. В последнее десятилетие широко распространились и другие формы сетевой коммуникации, такие как обмен мгновенными сообщениями и IP-телефония, но рабочей лошадкой интернет-коммуникации остается электронная почта. Сегодня электронная почта получила широкое применение в промышленности для обмена информацией внутри компаний, что делает возможным совместную работу над сложными проектами далеко удаленных друг от друга сотрудников. К сожалению, так же

как и в отношении обычной почты, 9 из 10 сообщений электронной представляют из себя **спам (spam)** (McAfee, 2010).

Электронной почте, как и любой форме коммуникаций, присущ определенный стиль и набор соглашений. В частности, общение по электронной почте носит очень неформальный и демократичный характер. Скажем, человек, который никогда бы не осмелился позвонить или даже написать бумажное письмо какой-нибудь Особо Важной Персоне (VIP), запросто может сесть и написать ей небрежное электронное сообщение. Часто переписка по e-mail фокусируется на содержании, а не на статусе и устраняет большинство проблем, связанных с различиями в должностном положении, возрасте и поле. Благодаря электронной почте блестящая идея, посланная студентом-практикантом по электронной почте, имеет шанс выиграть у идеи не столь блестящей, но высказанной вице-президентом компании.

В электронной почте люди обожают использовать особый жаргон и сокращения, такие как BTW (By The Way — между прочим), ROTFL (Rolling On The Floor Laughing — Катаюсь по полу от смеха), IMHO (In My Humble Opinion — По моему скромному мнению) и т. д. Кроме того, чрезвычайно популярны так называемые **смайлики (smileys)**, начиная со всем известного «:-)». Если вы его не узнали, поверните книгу на 90° по часовой стрелке. Этот и другие **эмотиконы** помогают передать тон сообщения. Они заполонили не только переписку по электронной почте, но и обмен мгновенными сообщениями.

За время использования смайликов изменились и протоколы электронной почты. Первые системы электронной почты состояли просто из протоколов передачи файлов и договоренности указывать адрес получателя в первой строке каждого сообщения (то есть файла). Со временем электронная почта отошла от передачи файлов и были добавлены многие опции, такие как возможность отослать одно сообщение нескольким адресатам. В девяностых годах добавилась возможность передавать мультимедиа, то есть посылать сообщения с изображениями и другой не текстовой информацией. Программы для чтения почты стали гораздо более сложными, переход от основанного на тексте к графическому пользовательскому интерфейсу обеспечил пользователям возможность доступа к почте при помощи ноутбуков, вне зависимости от того, где они находились территориально. Наконец, обилие спама заставило современные программы для чтения почты и протоколы передачи оной почты обратить внимание на поиск и удаление нежелательных сообщений.

В нашем описании электронной почты мы сфокусируем внимание на способе, при помощи которого почтовые сообщения курсируют между пользователями, а не на внешнем виде и особенностях программ для чтения электронных писем. Тем не менее, описав архитектуру в целом, мы перейдем к той части почтовой системы, которую видит пользователь, знакомой большинству читателей.

7.2.1. Архитектура и службы

В данном разделе мы рассмотрим возможности и организацию систем электронной почты. Архитектура почтовой системы показана на рис. 7.4. Система электронной почты (система e-mail) состоит из двух подсистем: **пользовательских агентов (user agents)**, позволяющих пользователям читать и отправлять электронную почту, и **агентов пере-**

дачи сообщений (message transfer agents), пересылающих сообщения от отправителя к получателю. Мы будем неформально называть агенты передачи сообщений **почтовыми серверами (mail servers)**.

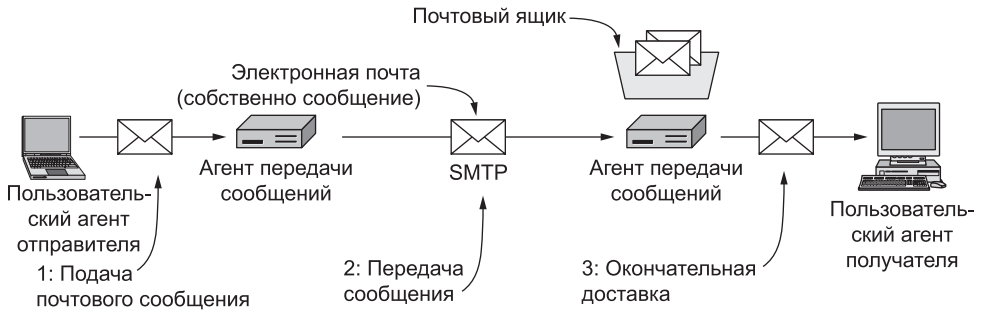


Рис. 7.4. Архитектура системы e-mail

Пользовательские агенты представляют собой программы, предоставляющие графический интерфейс или иногда интерфейс, базирующийся на тексте или командах, позволяющий пользователям взаимодействовать с системой электронной почты. В него входят средства написания сообщений и ответов на сообщения, отображения входящих сообщений и организации писем при помощи распределения их по папкам, поиска и удаления. Отсылка новых сообщений в почтовую систему для их дальнейшей доставки называется **подачей почтового сообщения (mail submission)**.

Обработка сообщений может быть частично автоматизирована с учетом желаний пользователя. Например, поступающая почта может фильтроваться, чтобы извлечь или приписать низкий приоритет сообщениям, похожим на спам. Некоторые программы включают дополнительные возможности, такие как автоматическая отправка ответных сообщений («Я в отпуске, скоро вернусь и отвечу на твое письмо»). Пользовательский агент работает на том же компьютере, на котором пользователь читает свою электронную почту. Это обычная программа, и она не обязательно должна работать все время.

Агенты передачи сообщений, как правило, являются системными процессами. Они работают в фоновом режиме на машинах почтовых серверов и всегда должны быть доступными. Они должны автоматически перемещать почтовые сообщения по системе от отправителя получателю при помощи **SMTP (Simple Mail Transfer Protocol — простого протокола передачи почтовых сообщений)**. Это шаг, на котором передается сообщение.

SMTP был впервые определен как RFC 821. Далее в него вносились изменения вплоть до текущей редакции RFC 5321. Он отсылает сообщения по соединениям и высылает обратно отчеты о статусе доставки и любых возникших ошибках. Существует множество приложений, в которых подтверждение доставки имеет большую важность и даже может иметь юридическую значимость («Ваша честь, моя электронная система не очень надежна, поэтому я полагаю, что повестка с вызовом в суд просто где-то потерялась»).

Агенты передачи сообщений также используют **списки рассылки (mailing lists)**, которые позволяют доставлять идентичные копии сообщения всем, чьи адреса были включены в список адресов электронной почты. Среди других полезных дополнитель-

ных функций можно перечислить следующие: рассылка копий писем «под копирку» (Carbon copy), рассылка копий без уведомления о других получателях (Blind carbon copy), письма с высоким приоритетом, секретная (то есть зашифрованная) почта, возможность доставки письма альтернативному получателю, если основной временно недоступен, а также возможность перепоручать обработку почты секретарям.

За связь пользовательских агентов и агентов передачи сообщений отвечают почтовые ящики и стандартный формат почтовых сообщений. **Почтовые ящики (mailboxes)** хранят почту, которая доставлена пользователю. Они поддерживаются почтовыми серверами. Пользовательские агенты просто предоставляют пользователям возможность увидеть содержимое их почтовых ящиков. Чтобы это сделать, пользовательский агент отсылает почтовым серверам команды и получает возможность манипулировать почтовыми ящиками, проверяя их содержимое, удаляя сообщения и т. д. Последний шаг в извлечении почты — это ее доставка конечному пользователю (шаг 3 на рис. 7.4). При такой архитектуре один пользователь может использовать различные пользовательские агенты на различных машинах, чтобы получить доступ к одному и тому же почтовому ящику.

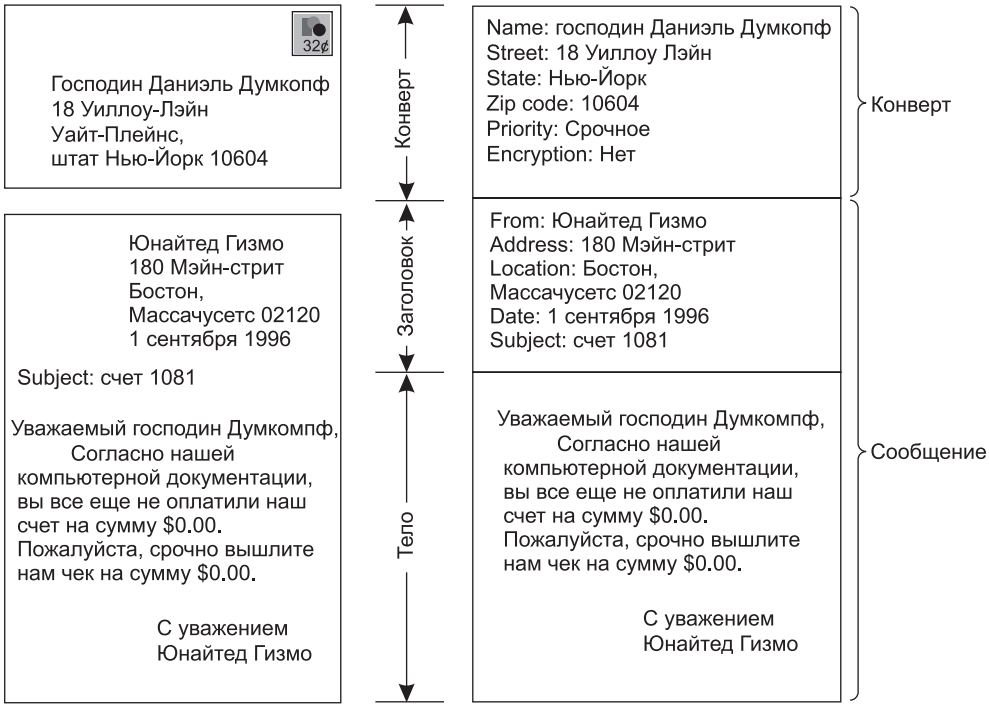


Рис. 7.5. Конверты и сообщения: *а* — обычное письмо; *б* — электронное письмо

Почта пересылается между агентами передачи сообщений в стандартном формате. В первоначально разработанный формат, RFC 822, вносились изменения. Текущая версия носит название RFC 5322; в нее включена поддержка мультимедиа-контента

и международный текст. Данная схема называется MIME, мы поговорим о ней позднее. Хотя многие все еще называют электронную почту RFC 822.

В основе всех современных систем электронной почты лежит ключевая идея о разграничении **конверта (envelope)** и содержимого письма. Конверт заключает в себе сообщение. Он содержит всю информацию, необходимую для доставки сообщения, — адрес получателя, приоритет, уровень секретности и т. п. Все эти сведения отделены от самого сообщения. Агенты передачи сообщений используют конверт для маршрутизации, аналогично тому, как это делает обычная почтовая служба.

Сообщение внутри конверта состоит из двух отдельных частей: **заголовка (header)** и **тела письма (body)**. Заголовок содержит управляющую информацию для пользовательских агентов. Тело письма целиком предназначается для человека-получателя. Примеры конвертов и сообщений показаны на рис. 7.5.

Мы поговорим об этой архитектуре более детально, рассмотрев шаги, которые необходимо пройти, чтобы отослать сообщение от одного пользователя другому. Это путешествие начинается с пользовательского агента.

7.2.2. Пользовательский агент

Пользовательский агент — это программа (иногда называемая **почтовым редактором** — **email editor** или **«читалкой»** — **email reader**), управляемая множеством команд для составления и получения сообщений, а также для ответа на сообщения и управления почтовыми ящиками. Существует много популярных пользовательских агентов, например gmail от Google, Microsoft Outlook, Mozilla Thunderbird и Apple Mail. Внешне они сильно отличаются. Графический интерфейс большинства пользовательских агентов основан на меню или значках и требует наличия мышки или, на маленьких мобильных устройствах, возможности управления при помощи касания. Более старые пользовательские агенты, такие как Elm, mh и Pine, имеют интерфейсы, основанные на тексте, и работают при помощи ввода с клавиатуры однобуквенных команд. Функциональных различий нет, по крайней мере, в отношении текстовых сообщений.

Типичные элементы интерфейса пользовательского агента показаны на рис. 7.6. Агент, которым пользуетесь вы, вероятно, выглядит гораздо более современно, но функции, скорее всего, совпадают.

Когда пользовательский агент запущен, он обычно отображает сводную информацию о сообщениях в почтовом ящике пользователя. Часто каждому сообщению соответствует одна строка, а идут они в каком-либо порядке, выбранном при сортировке. В сводной информации выделены ключевые поля сообщения, извлеченные из конверта или заголовка

На рис. 7.6 показаны семь строк сводной информации. В строках используются поля «От», «Тема» и «Дата получения» в указанном порядке, в которых отображается, от кого получено сообщение, о чем оно и когда было получено. Вся информация отформатирована удобным для пользователя образом; она базируется на полях сообщения, но не отображается буквально. Таким образом, те, кто отправляют сообщения без темы, часто сталкиваются с тем, что их письмам был приписан низкий приоритет.

Возможно наличие многих других полей и маркеров. Значки рядом с темой сообщения (см. рис. 7.6) могут обозначать, например, непрочитанную почту (конверт),

прикрепленные файлы (скрепка), важные сообщения, по крайней мере, с точки зрения отправителя (восклицательный знак).

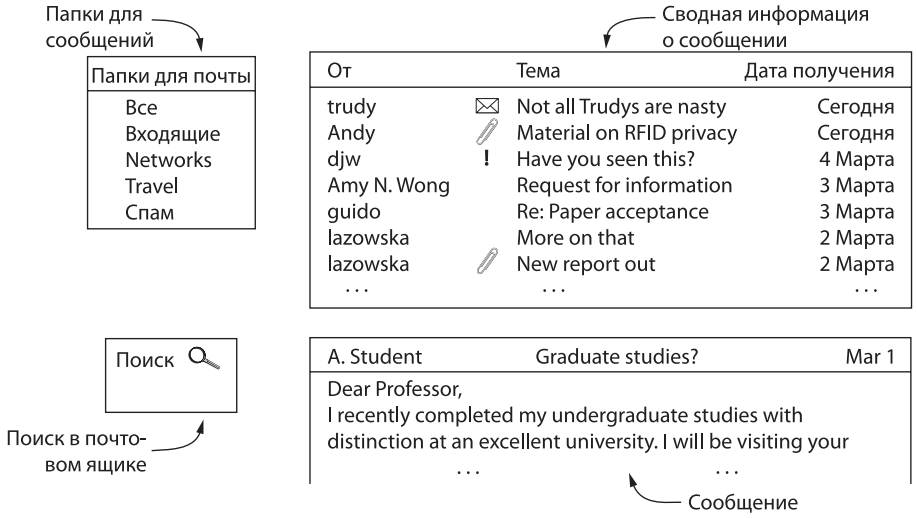


Рис. 7.6. Стандартные элементы интерфейса пользовательского агента.

Также возможны несколько вариантов сортировки. Самый распространенный основывается на времени получения, сначала более свежие, с маркером того, прочитано сообщение получателем или нет. Поля сводной информации и порядок сортировки могут быть настроены в соответствии с желаниями пользователя.

Пользовательские агенты также должны отображать входящие сообщения определенным образом, чтобы почту можно было читать. Часто предоставляется предварительный просмотр письма (см. рис. 7.6), чтобы пользователь мог решить, когда читать полученное сообщение. При предварительном просмотре могут использоваться маленькие значки или изображения, чтобы описать содержание письма. Другими вариантами обработки представления могут быть форматирование сообщения таким образом, чтобы оно помещалось на экране, перевод, преобразование содержания в более удобные формы (например, цифровую речь или распознанный текст).

После того как сообщение было прочитано, вы можете решить, что с ним делать. Это называется **размещение сообщения (message disposition)**. Среди опций есть удаление, написание ответа, пересылка сообщения другому пользователю и оставление сообщения в ящике для дальнейшей работы. Большинство пользовательских агентов снабжено одним почтовым ящиком для входящих сообщений с набором папок для сохраненной почты. Папки позволяют пользователю сохранять сообщения в разных местах в зависимости от отправителя, темы или какой-то другой категории.

Распределение по папкам также может автоматически проводить пользовательский агент до того, как пользователь прочтет сообщение. Примером этого служит проверка полей и содержания сообщения, а также отзывов пользователя о предыдущих сообщениях, предназначенные для определения того, является ли письмо спамом. Многие интернет-провайдеры и компании используют программное обеспечение,

помечающее сообщения как важные или спам, так что пользовательский агент может распределить их по соответствующим папкам. У интернет-провайдеров и компаний есть преимущество работы с множеством сообщений, так что у них могут быть списки известных спамеров. Если сотни пользователей в одно и то же время получают одинаковые сообщения, они, вероятно, являются спамом. Предварительно сортируя сообщения и помечая часть из них как «предположительно спам», пользовательский агент может избавить пользователей от массы работы по разделению нужного и ненужного.

Но какой же спам является наиболее популярным? Он генерируется набором взломанных компьютеров, которые называются **ботнет (botnet**, **«ботсеть»**), а его содержание зависит от страны, в которой вы живете. В Азии самыми популярными являются предложения поддельных дипломов, в США — дешевых лекарств и сомнительной продукции. Невостребованные счета в нигерийском банке также не теряют популярности. Ну, а таблетки для увеличения определенных частей тела входят в топлист, где бы вы не жили.

Кроме того, сами пользователи могут придумывать правила распределения писем по папкам. Каждое правило определяет состояние и действие. Например, правило может гласить, что письмо, полученное от начальника, должно перемещаться в папку для немедленного прочтения, а письма от определенного списка отправителей должны помещаться в другую папку для того, чтобы их можно было прочитать позднее. На рис. 7.6 изображено несколько папок. Самые важные из них — это Inbox (Входящие), для входящей почты, которая не была никуда перемещена при получении, и Spam (Спам), для сообщений похожих на спам.

Помимо распределения почтовых сообщений по папкам пользовательские агенты предоставляют широкие возможности поиска писем в почтовом ящике. Эта опция также показана на рис. 7.6. Возможность поиска позволяет пользователям быстро находить сообщения. Например, можно найти сообщение, содержавшее строку «where to buy Vegemite», которое было получено в прошлом месяце.

Электронная почта прошла долгий путь с тех пор, как она была просто способом передачи данных. Сегодня почти все провайдеры поддерживают почтовые ящики объемом до одного гигабайта, сохраняющие переписку пользователя за долгий период времени. А обеспечить поддержку такого большого объема сообщений позволяет набор сложных способов обработки почтовых сообщений пользовательскими агентами, включающий поиск и автоматическую обработку информации. Для тех, кто отправляет и получает тысячи сообщений в год, эти инструменты бесценны.

Еще одна полезная опция — это способность определенным образом автоматически отвечать на сообщения. Один из вариантов ответа — пересылка входящего сообщения на другой адрес, например на компьютер, принадлежащий коммерческой пейджинговой компании, чтобы вызвать пользователя, используя радио или спутниковую антенну, и отобразить строку «Тема» на его пейджере. Эти **автоответчики (autoresponders)** должны работать на почтовых серверах, так как пользовательский агент может работать не все время, а также может только иногда извлекать почту. Из-за этих факторов пользовательский агент не обеспечивает настоящего автоматического ответа. Однако интерфейс для автоматических ответов обычно предоставляется пользовательским агентом.

Еще один пример автоматического ответа — **агент на отпуск (vacation agent)**. Это программа, которая просматривает каждое входящее сообщение и посылает отправителю нейтральный ответ, например: «Привет, я в отпуске. Вернусь 24-го августа. Тогда с тобой и свяжусь». В подобных ответах также может быть сказано, каким образом действовать в экстренной ситуации, если она возникнет во время отсутствия адресата, к каким людям стоит обращаться по вопросам определенных проблем и т. д. Большинство агентов на отпуск ведут запись того, кому были отосланы автоматические ответы, и избавляют от необходимости повторно отвечать данному человеку. Однако и с этими агентами есть определенные трудности. Например, нежелательно отсылать заготовленный ответ большому списку адресатов.

Теперь давайте обратимся к сценарию отсылки сообщения одним пользователем другому. Одна из основных не обсужденных нами функций, которую поддерживают пользовательские агенты, — это написание сообщения. Данный процесс включает создание сообщений, ответы на них и отсылку этой почты остальной части почтовой системы с целью их доставки. Хотя для создания тела сообщения может быть использован любой текстовый редактор, обычно редакторы интегрируются в пользовательские агенты. Это позволяет проще добавлять адресатов и прописывать темы и другую информацию в нужные поля. Например, при ответе на сообщение почтовая система может извлечь адрес отправителя из входящего сообщения и автоматически подставить его на нужное место в ответе. Другие часто встречающиеся возможности — это добавление блока с подписью (**signature block**) в конец сообщения, исправление ошибок, вычисление цифровых подписей, подтверждающих истинность сообщения.

У сообщений, отсылаемых в почтовую систему — стандартный формат, который должен быть образован при помощи информации, предоставленной пользовательским агентом. Самая важная часть в передаче сообщения — это конверт, а самая важная часть конверта — адрес назначения. Этот адрес должен быть в формате, с которым могут работать агенты передачи сообщений.

Ожидаемый формат адреса — это *user@dns-address*. Так как система доменных имен DNS уже рассматривалась выше в этой главе, сейчас мы не станем подробно останавливаться на данном вопросе. Однако следует отметить, что существуют и другие формы адресации. В частности, адреса стандарта **X.400** абсолютно не похожи на DNS-адреса.

X.400 — это стандарт ISO для систем обработки сообщений, который одно время соревновался с SMTP. SMTP победил в этом противостоянии, хотя системы X.400 все еще используются (в основном, за пределами США). Адреса стандарта X.400 абсолютно не похожи на DNS-адреса и состоят из пар *атрибут = значение*, разделенных слэшами, например:

```
/C=US/ST=MASSACHUSETTS/L=CAMBRIDGE/PA=360 MEMORIAL DR./CN=KEN SMITH/
```

В этом адресе указано государство, штат, местоположение, личный адрес и имя получателя (Ken Smith). Возможно также использование различных других атрибутов, что делает возможным отправку электронного письма человеку, чьего имени вы не знаете, при условии что вам известны другие атрибуты (например, название компании и должность получателя). Хотя форма адресации X.400 значительно менее удобна, чем DNS, для пользовательских агентов этот вопрос является спорным, так как они поддерживают удобные с точки зрения пользователя **псевдонимы (aliases**,

иногда говорят — **nicknames**), с помощью которых он может вводить или выбирать имя адресата и получать корректный электронный адрес. Поэтому обычно пользователю не приходится вводить такие, мягко говоря, странные строки.

Последний вопрос, который мы обсудим в рамках разговора об отсылке почты, — это списки рассылки, позволяющие пользователю рассылать одно и то же сообщение группе получателей при помощи одной команды. Есть два варианта того, как может храниться список рассылки. Первый — локальное хранение пользовательским агентом. В этом случае пользовательский агент может просто послать каждому получателю отдельное сообщение. Помимо этого список может храниться удаленно на агенте передачи сообщений. Тогда сообщения будут тиражироваться в системе передачи сообщений, что позволяет множеству пользователей отсылать письма данному списку. Допустим, у группы исследователей птиц есть список рассылки, называющийся *birders* (птицеловы), установленный на агенте передачи сообщений *meadowlark.arizona.edu*. Тогда любое сообщение, посланное по адресу *birders@meadowlark.arizona.edu*, будет сначала отправляться в университет штата Аризона, а затем рассылаться оттуда в виде отдельных сообщений всем членам списка рассылки, где бы они ни находились. Для отправителя письма список рассылки внешне не отличается от обычного индивидуального адреса. Если отправитель не знает, что *birders* — это список рассылки, он вполне может подумать, что посылает письмо лично некоему профессору по имени Gabriel O. Birders.

7.2.3. Форматы сообщений

Перейдем теперь от рассмотрения пользовательского интерфейса к формату самих сообщений электронной почты. Чтобы сообщения, отсылаемые пользовательским агентом, обрабатывались агентами передачи сообщений, они должны быть оформлены в соответствии с определенными стандартами. Сначала мы рассмотрим основной ASCII-формат электронного письма стандарта RFC 5322, который является последним вариантом оригинального формата интернет-сообщений, описанного в RFC 822. Затем мы познакомимся с мультимедийным расширением этого первоначального стандарта.

RFC 5322 — формат интернет-сообщений

Сообщения состоят из примитивного конверта (описанного как часть SMTP в RFC 5321), нескольких полей заголовка, пустой строки и, наконец, тела сообщения. Каждое поле заголовка (логически) состоит из одной строки ASCII-текста, содержащей имя поля, двоеточие и (в большинстве случаев) значение поля. Первоначальный RFC 822 был создан несколько десятилетий назад и в нем нет четкого разграничения конверта и заголовка. Хотя частично стандарт был пересмотрен в RFC 5322, целиком обновить его было невозможно, поскольку RFC 822 уже был очень широко распространен. Обычно пользовательский агент формирует сообщение и передает его агенту передачи сообщений, который с помощью одного из полей заголовка создает конверт нового вида, представляющий собой некую старомодную смесь сообщения и конверта.

Основные поля заголовка, связанные с транспортировкой сообщения, перечислены в табл. 7.3. Поле *To:* содержит DNS-адрес основного получателя. Возможно наличие и нескольких получателей. В поле *Cc:* указываются адреса дополнительных получателей. С точки зрения доставки, никакой разницы между основным и дополнительными получателями нет. Разница между ними чисто психологическая и, может быть, важна для людей, но совершенно не существует для почтовой системы. Термин *Cc:* (carbon copy — экземпляр, сделанный «под копирку») несколько устарел, так как при работе с компьютерами копировальная бумага вообще-то не используется, тем не менее он прочно обосновался в электронной почте. Поле *Bcc:* (Blind carbon copy — слепая копия) аналогично предыдущему, с той разницей, что в последнем случае строка этого поля удаляется из всех экземпляров сообщения, отправленных как основному, так и дополнительным получателям. Это свойство позволяет рассылать одно письмо одновременно нескольким получателям так, что получатели не будут знать, что это письмо послано еще кому-либо кроме них.

Таблица 7.3. Поля заголовка стандарта RFC 5322, связанные с транспортировкой сообщения

Поле	Значение
To:	Электронный адрес (адреса) основного получателя (получателей)
Cc:	Электронный адрес (адреса) дополнительного получателя (получателей)
Bcc:	Электронный адрес (адреса) слепой копии
From:	Автор (авторы) сообщения
Sender:	Электронный адрес отправителя
Received:	Строка, добавляемая каждым агентом передачи сообщений на протяжении маршрута
Return-Path:	Может быть использовано для идентификации обратного пути к отправителю

Следующие два поля, *From:* и *Sender:*, сообщают соответственно, кто составил и отправил сообщение. Это могут быть разные люди. Например, написать письмо может руководитель предприятия, а отослать — его секретарь. В этом случае руководитель будет числиться в поле *From:*, а секретарь — в поле *Sender:*. Поле *From:* является обязательным, тогда как поле *Sender:* может быть опущено, если его содержимое не отличается от содержимого поля *From:*. Эти поля нужны на случай, если сообщение доставить невозможно и об этом следует проинформировать отправителя. Кроме того, по адресам, указанным в этих полях, может быть оправлен ответ.

Строка, содержащая поле *Received:*, добавляется каждым агентом передачи сообщений на пути следования сообщения. В это поле помещается идентификатор агента, дата и время получения сообщения, а также другая информация, которая может быть использована для исправления неисправностей в системе маршрутизации.

Поле *Return-Path:* добавляется последним агентом передачи сообщений. Предполагалось, что это поле будет сообщать, как добраться до отправителя. Теоретически, эта информация может быть собрана из всех полей *Received:* заголовка (кроме имени почтового ящика отправителя), однако на практике оно редко заполняется подобным образом и обычно просто содержит адрес отправителя.

Помимо полей, показанных в табл. 7.3, сообщения стандарта RFC 5322 могут также содержать широкий спектр полей заголовка, используемых пользовательским агентом или самим пользователем. Наиболее часто используемые поля заголовка приведены в табл. 7.4. Информации в таблице достаточно, чтобы понять назначение полей, поэтому мы не станем рассматривать их все подробно.

Таблица 7.4. Некоторые поля, используемые в заголовке сообщения стандарта RFC 5322

Поле	Значение
Date:	Дата и время отправки сообщения
Reply-to:	Электронный адрес, на который следует присылать ответ
Message-Id:	Уникальный номер для последующей ссылки на это сообщение
In-Reply-To:	Идентификатор Message-Id сообщения, в ответ на которое посылается это сообщение
References:	Другие важные ссылки (идентификаторы Message-Id)
Keywords:	Ключевые слова, выбираемые пользователем
Subject:	Краткое изложение темы сообщения для отображения в одной строке

Поле *Reply-to:* иногда используется в случае, если ни составитель письма, ни его отправитель не хотят получать на него ответ. Например, управляющий отделом сбыта может написать письмо, информирующее клиентов о новом продукте. Это письмо отправляется его секретарем, но в поле *Reply-to:* указан адрес менеджера отдела продаж, который может ответить на вопросы и принять заказы. Это поле также может быть полезным, если у отправителя есть два адреса электронной почты и он хочет, чтобы ответы приходили не на тот, с которого он отправляет сообщение.

Message-Id: — это автоматически генерируемое число, которое используется, чтобы связывать сообщения (например, при ответе на письмо) и избежать повторной доставки.

В документе RFC 5322 открыто сказано, что пользователям разрешается изобретать дополнительные заголовки для своих нужд. Начиная с формата RFC 822, заголовки начинаются со строки *X-*. Гарантируется, что в будущем никакие стандартные заголовки не будут начинаться с этих символов, чтобы избежать конфликтов между официальными и частными заголовками. Иногда умники-студенты включают поля вроде *X-Fruit-of-the-Day:* (сегодняшний плод) или *X-Disease-of-the-Week:* (недуг недели), использование которых вполне законно, хотя их смысл и не всегда понятен.

После заголовков идет тело самого сообщения. Пользователь может разместить в нем все, что ему угодно. Некоторые люди завершают свои послания сложными подписями, включающими популярные и малоизвестные цитаты, политическими заявлениями и разнообразными объявлениями (например, «Корпорация АБВ не несет ответственности за высказанное выше мнение. Собственно, она даже не в силах постичь его»).

MIME — многоцелевые расширения электронной почты в сети Интернет

На заре существования сети ARPANET электронная почта состояла исключительно из текстовых сообщений, написанных на английском языке и представленных символами ASCII. Для подобного использования первоначального стандарта RFC 822 было вполне достаточно: он определял формат заголовков, но оставлял содержимое сообщения полностью на усмотрение пользователей. В 1990-е годы всемирное использование Интернета и необходимость посылать более разнообразный контент через почтовую систему показали, что такой подход уже не удовлетворяет пользователей, привыкших к Интернету. Требовалось обеспечить возможность отправления сообщений на языках с надстрочными знаками (например, на французском и немецком), на языках, использующих алфавиты, отличные от латинского (например, на иврите или русском), или на языках без алфавитов (например, китайском и японском). Также требовалась возможность отсылки сообщений, не являющихся текстом (например, аудио, изображения или бинарные документы и программы).

Решением стала разработка **MIME (Multipurpose Internet Mail Extensions — многоцелевые расширения электронной почты в Интернете)**. Они широко применяются как для почтовых сообщений, посылаемых через Интернет, так и для описания контента в других приложениях, таких как веб-сайты. MIME описаны в RFC 2045–2047, 4288, 4289 и 2049.

Основная идея стандартов MIME — продолжить использование формата RFC 822 (предшественника формата RFC 5322, действовавшего в то время, когда были предложены MIME), но с добавлением структуры к телу сообщения и с определением правил кодировки для передачи не-ASCII-сообщений. Не отклоняясь от стандарта RFC 822, MIME-сообщения могут передаваться с помощью обычных агентов передачи сообщений и протоколов (ранее основанных на RFC 821, а сейчас на RFC 5321). Все, что нужно было изменить, — это отправляющие и принимающие программы, которые пользователи могли создать для себя сами.

Стандартами MIME определяются пять новых заголовков сообщения, приведенных в табл. 7.5. Первый заголовок (*MIME-Version:*) просто информирует пользовательского агента, получающего сообщение, что тот имеет дело с сообщением MIME, а также сообщает ему номер версии MIME, используемой в этом сообщении. Если сообщение не содержит такого заголовка, то оно считается написанным на английском языке (или, по крайней мере, на языке, использующем только знаки ASCII) и обрабатывается соответствующим образом.

Таблица 7.5. Заголовки сообщений, добавленные MIME

Заголовок	Описание
MIME-Version:	Указывает версию стандартов MIME
Content-Description:	Описание содержимого. Строка обычного текста, информирующая о содержимом сообщения
Content-Id:	Уникальный идентификатор
Content-Transfer-Encoding:	Указывает способ кодировки тела сообщения для его передачи
Content-Type:	Тип и формат содержимого сообщения

Заголовок *Content-Description:* представляет собой ASCII-строку, информирующую о том, что находится в сообщении. Этот заголовок позволяет пользователю принять решение о том, нужно ли ему декодировать и читать сообщение. Если в строке сказано: «Фотография тушканчика Барбары», а получивший это сообщение не является любителем тушканчиков, то, вероятнее всего, он сразу удалит это сообщение, а не станет перекодировать его в цветную фотографию высокого разрешения.

Заголовок *Content-Id:* содержит идентификатор содержимого сообщения. В нем используется тот же формат, что и в стандартном заголовке *Message-Id:*.

Заголовок *Content-Transfer-Encoding:* сообщает о способе упаковки тела сообщения для его передачи по сети. Во времена разработки MIME основной проблемой были протоколы передачи сообщений (SMTP), предполагавшие, что сообщение будет в формате ASCII, при этом в строке должно было быть не более 1000 символов. Символы ASCII используют 7 бит из каждого восьмибитного байта. Бинарные данные, такие как исполняемые программы и изображения, используют все 8 бит байта, так же как и расширенные наборы символов. Не было никакой гарантии того, что эти данные будут передаваться безопасно. В результате этого потребовался метод передачи бинарных данных, который заставлял бы их выглядеть как обычное почтовое сообщение ASCII. Расширения SMTP, введенные с момента разработки MIME, на самом деле позволяют передавать восьмибитные бинарные данные, хотя даже сегодня незакодированные бинарные данные не всегда корректно передаются почтовой системой.

MIME обеспечивает пять схем кодирования, предназначенных для передачи данных (кроме того, имеется возможность добавления новых схем; на всякий случай). Простейшая из них заключается в передаче просто текстовых сообщений ASCII. Символы ASCII используют 7 разрядов и могут передаваться напрямую протоколом электронной почты, при условии, что строка не превышает 1000 символов.

Следующая по простоте схема аналогична предыдущей, но использует 8-разрядные символы, то есть все значения байта от 0 до 255 включительно. Сообщения, использующие 8-разрядную кодировку, также должны соблюдать правило о максимальной длине строки.

Еще хуже обстоит дело с сообщениями в настоящей двоичной кодировке. К ним относятся произвольные двоичные файлы, которые не только используют все 8 разрядов в байте, но еще и не соблюдают ограничение на 1000 символов в строке. К этой категории относятся исполняемые программные файлы. Сегодня почтовые серверы могут проверять, есть ли возможность переслать данные в бинарной (или восьмибитной) кодировке, и если на обоих концах не поддерживается данное расширение, данные будут передаваться в ASCII.

Кодировка бинарных данных в формате ASCII называется **base64** (64-символьная кодировка). При использовании данного метода группы по 24 бита разбиваются на четыре единицы по 6 бит, каждая из которых посылается в виде обычного разрешенного ASCII-символа. В этой кодировке 6-разрядный символ 0 кодируется ASCII-символом «A», 1 — ASCII-символом «B» и т. д. Затем следуют 26 строчных букв, затем 10 цифр и, наконец, + и / для кодирования 62 и 63 соответственно. Последовательности == и = говорят о том, что последняя группа содержит только 8 или 16 бит соответственно. Символы перевода строки и возврата каретки игнорируются, поэтому их можно вставлять в любом месте закодированного потока символов для того, чтобы строки

выглядели не слишком длинными. Таким способом можно передать любой двоичный код, хотя и достаточно не эффективно. Эта кодировка была крайне популярна до того, как приобрели широкое распространение почтовые серверы, способные передавать бинарную информацию.

Для сообщений, почти полностью состоящих из символов ASCII, но с небольшими включениями не-ASCII-символов, подобный метод несколько неэффективен. Вместо него лучше применять кодировку **quoted-printable** (цитируемое печатное кодирование). Это просто 7-битный ASCII, в котором символы, соответствующие значениям ASCII-кода свыше 127, кодируются знаком равенства, за которым следуют две шестнадцатеричных цифры — ASCII-код символа. Кроме конечных пробелов кодируются также символы управления, некоторые пунктуационные знаки и математические символы.

Наконец, когда имеются веские причины не использовать эти методы, можно указать в заголовке *Content-Transfer-Encoding:* свою кодировку.

Последний заголовок в табл. 7.5 представляет наибольший интерес. Он указывает тип тела сообщения и его применение выходит далеко за пределы электронной почты. Например, контент, загружаемый из Интернета, помечается символами MIME, что позволяет браузеру адекватно отображать информацию. Так обстоит дело с контентом, пересылаемым через потоковое мультимедиа, и передачей в реальном времени, например голоса через IP (VoIP).

Изначально в документе RFC 1521 были определены семь типов содержимого сообщений, каждый из которых распадается на несколько доступных подтипов. Подтип отделяется от типа косой чертой, например «Content-Type: video/mpeg». С тех пор к ним было добавлено много новых типов и подтипов. Этот список пополняется всякий раз при возникновении соответствующей необходимости. В сети его поддерживает IANA, он доступен по *адресу www.iana.org/assignments/media-types*.

Типы и примеры часто используемых подтипов приведены в табл. 7.6. Давайте кратко остановимся на каждом из них, начиная с типа *text*. Комбинация *text/plain* служит для обозначения обычного текстового сообщения, которое может быть отображено на экране компьютера сразу после получения. Для этого не требуется дополнительной обработки или перекодировки. Это значение поля заголовка позволяет передавать обычные сообщения в MIME с добавлением небольшого количества дополнительных заголовков. Когда веб-технологии стали популярны, был добавлен новый тип *text/html* (в RFC 2854), который позволил пересылать веб-страницы в теле письма RFC 822. В RFC 3023 определен подтип для расширяемого языка разметки (eXtensible Markup Language — XML), *text/xml*. С развитием Интернета количество документов XML стремительно увеличилось. Ниже, в разделе 7.3, мы рассмотрим HTML и XML.

Следующим типом MIME является *image*. Он используется для передачи неподвижных изображений. На сегодняшний день существует множество различных форматов хранения и передачи изображений, как с использованием сжатия, так и без него. Некоторые форматы, в том числе GIF, JPEG и TIFF, встроены практически во все браузеры, однако существует еще множество других типов и соответствующих им подтипов.

Типы *audio* и *video* предназначены, соответственно, для передачи звука и двигающегося изображения. Обратите внимание на то, что подтип *video* включает только

визуальную информацию, а не звуковую дорожку. Если необходимо передать по электронной почте видеофильм со звуком, то, возможно, придется посылать видеоряд и звуковую дорожку отдельно друг от друга. Это зависит от системы кодирования. Первым видеоформатом, который был определен стандартом MIME, стал формат со скромным названием **MPEG (Motion Pictures Experts Group — экспертная группа по вопросам движущегося изображения)**. С тех пор было добавлено еще несколько форматов. Так, чтобы позволить пользователям пересылать по электронной почте аудиофайлы в формате MP3, в RFC 3003 помимо уже существующего *audio/basic* был добавлен тип *audio/mpeg*. Типы *video/mp4* и *audio/mp4* указывают на видео- и аудиоданные в новом формате MPEG 4.

Таблица 7.6. Типы стандарта MIME и примеры подтипов

Тип	Подтип	Описание
text	plain, html, xml, css	Текст в различных форматах
image	gif, jpeg, tiff	Изображения
audio	basic, mpeg, mp4	Звуки
video	mpeg, mp4, quicktime	Видеофильмы
model	vrml	3D-модель
application	onoctet-stream, pdf, javascript, zip	Данные, производимые приложениями
message	http, rfc822	Инкапсулированное сообщение
multipart	mixed, alternative, parallel, digest	Комбинация нескольких типов

Тип *model* был добавлен позднее, чем остальные типы контента. Он предназначен для описания 3D-моделей. Однако пока он не особо широко используется.

Тип *application* (приложение) предназначен для всех форматов, которые не покрываются остальными типами и требуют от приложения интерпретации данных. Мы перечислили подтипы *pdf*, *javascript* и *zip* как примеры PDF-документов, программ на Java-Script и архивов в формате Zip соответственно. Пользовательские агенты, получающие этот контент, используют для его отображения стороннюю библиотеку или внешнюю программу. Отображение может быть интегрировано в пользовательский агент, а может и не быть.

Используя типы MIME, пользовательские агенты расширяют возможности обработки новых типов контента по мере их появления. Это значительное преимущество. С другой стороны, многие новые формы контента обрабатываются и интерпретируются приложениями, что представляет некоторую опасность. Совершенно очевидно, что запуск произвольной исполняемой программы, которая пришла по электронной почте от «друга», влечет за собой определенный риск. Такая программа может серьезно повредить те части компьютера, к которым получает доступ, особенно если она может считывать и создавать файлы и использовать сеть. Форматы документов также могут нести определенную угрозу, хотя это менее очевидно. Подобную возможность нельзя исключать, так как форматы типа PDF являются замаскированными развитыми язы-

ками программирования. Несмотря на то что они интерпретируются и ограничены в доступной им области, ошибки в интерпретаторе часто позволяют документам избежать ограничений обходными путями.

Помимо этих примеров, существует еще множество подтипов приложений, так как и самих приложений немало. Так, например, тип *octet-stream* (байтовый поток) представляет собой просто последовательность никак не обрабатываемых байтов. Он приписывается при невозможности описать контент (если нет ни одного типа, который подходил бы лучше). Получив такой поток, пользовательский агент должен, вероятно, предложить пользователю сохранить его в виде файла. Последующая обработка файла целиком зависит от пользователя, который, предположительно, знает, с каким контентом он столкнулся.

Последние два типа, о которых пойдет речь, полезны при создании сообщений и манипулировании ими. Тип *message* позволяет помещать одно сообщение в другое. Это может быть полезно для переадресации письма. Если внутри одного сообщения заключено целое сообщение стандарта RFC 822, следует использовать подтип *rfc822*. Сходным образом обычно инкапсулируются документы в формате HTML. А подтип *partial* позволяет разбивать сообщение на части и передавать их отдельно (например, в случае, когда инкапсулированное сообщение является слишком длинным). Параметры обеспечивают восстановление сообщения получателем из отдельных фрагментов в правильном порядке.

Наконец, тип *multipart* позволяет составлять сообщение из нескольких частей, при этом начало и конец каждой части отчетливо указывается. Подтип *mixed* позволяет создавать сообщение из частей различных форматов. Многие почтовые программы позволяют пользователю прикрепить к текстовому сообщению одно или более приложений. Эти приложения пересылаются с использованием типа *multipart*. В случае применения подтипа *alternative*, напротив, каждая часть должна содержать одно и то же сообщение, но в другом виде или другой кодировке. Например, сообщение может быть послано в виде простого ASCII-текста, а также в форматах HTML и PDF. Грамотно созданный пользовательский агент, получив такое сообщение, сначала попытается отобразить его в формате, зависящем от предпочтений пользователя. Скорее всего, это будет PDF, если данный формат допустим. Если это по какой-либо причине невозможно, тогда производится попытка отобразить часть в формате HTML. Если и это невозможно, отображается ASCII-текст. Части следует располагать в порядке увеличения сложности, чтобы даже старые (до MIME) пользовательские агенты смогли отобразить сообщение, хотя бы в виде простого ASCII-текста.

Подтип *alternative* также может использоваться для сообщений, посылаемых одновременно на разных языках. В этом контексте знаменитый розеттский камень, найденный в Египте, может считаться одним из ранних вариантов сообщений типа *multipart/alternative*.

Что касается двух других примеров подтипов, подтип *parallel* используется, когда все части должны «просматриваться» одновременно. Например, в фильмах часто есть видео- и аудиодорожки. Гораздо приятнее смотреть фильм, если эти две дорожки проигрываются одновременно, а не последовательно. Подтип *digest* используется, когда несколько сообщений объединяются в одно составное. Например, какая-нибудь дис-

куссионная группа в Интернете может собирать сообщения от подписчиков, а затем высылать их в виде единого сообщения типа *multipart/digest*.

Пример того, как типы MIME могут использоваться для сообщений электронной почты, приведен в листинге 7.2. В данном случае поздравление с днем рождения посылается одновременно в двух альтернативных формах: в виде HTML и в виде аудиозаписи. Если у получателя есть возможность воспроизвести звуковой файл, пользовательский агент воспроизведет его. В этом примере звук передается как подтип *message/external-body* по ссылке, так что сначала пользовательский агент должен обратиться за звуком к файлу *birthday.snd* через FTP. В противном случае на экране получателя в полной тишине отобразится текст сообщения. Эти две части письма разделены двойными дефисами, за которыми следует (определяемая пользователем) строка, указанная как значение параметра *boundary* (граница).

Листинг 7.2. Сообщение типа multipart, содержащее HTML и аудио

```
From: alice@cs.washington.edu
To: bob@ee.uwa.edu.au
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@cs.washington.edu>
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Земля обошла вокруг Солнца целое число раз

Это преамбула. Пользовательский агент игнорирует ее. Ку-ку.

--qwertyuiopasdfghjklzxcvbnm
Content-Type: text/html

<p>Happy birthday to you<br>
Happy birthday to you<br>
Happy birthday to you</p>

--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body;
     access-type="anon-ftp";
     site="bicycle.cs.washington.edu";
     directory="pub";
     name="birthday.snd"

content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm–
```

Обратите внимание: заголовок *Content-Type* трижды встречается в данном сообщении. На верхнем уровне он указывает, что сообщение состоит из нескольких частей. Для каждой части он сообщает ее тип и подтип. Наконец, в теле второй части сообщения он указывает пользовательскому агенту тип внешнего файла. Чтобы подчеркнуть это различие, мы использовали в последнем случае строчные символы, хотя для всех заголовков регистр символа не имеет значения. Для внешнего тела в формате, отличном от 7-разрядного ASCII, также требуется заголовок *content-transfer-encoding*.

7.2.4. Пересылка сообщений

Теперь, когда мы описали пользовательские агенты и сообщения электронной почты, пора разобраться в том, как агенты передачи сообщений доставляют их от отправителя получателю. Передача почты осуществляется посредством протокола SMTP.

Для этого проще всего установить транспортное соединение от машины-источника до машины-приемника, а затем просто передать по нему сообщение. Так изначально работал SMTP. Однако за последние годы возникло два разных способа использования SMTP. Первый — для **подачи почтового сообщения (mail submission**, первый шаг в архитектуре e-mail на рис. 7.4). Этим способом пользовательские агенты переправляют сообщения в почтовую систему для их дальнейшей отправки. Второй вариант использования — передача сообщений между агентами передачи сообщений (шаг 2 на рис. 7.4).

Такая последовательность обеспечивает доставку почты на всем пути от посылающего до получающего агента передачи сообщений за один шаг. Окончательная доставка происходит при участии других протоколов. Ее мы опишем в следующем разделе.

В этом разделе мы расскажем об основах протокола SMTP и механизме его расширения. Затем мы обсудим то, как он различным образом используется для принятия подачи и передачи сообщений.

SMTP — простой протокол электронной почты и его расширения

В Интернете для доставки электронной почты машина-источник устанавливает TCP-соединение с портом 25 машины-приемника. Этот порт прослушивается почтовым демоном и их общение происходит с помощью протокола **SMTP (Simple Mail Transfer Protocol — простой протокол передачи электронной почты)**. Этот сервер принимает входящие соединения, проверяет их безопасность и принимает сообщения для доставки. Если письмо невозможно доставить, отправителю возвращается сообщение об ошибке, содержащее первую часть этого письма.

Протокол SMTP представляет собой простой ASCII-протокол. Это не недостаток, а отличительная черта. Использование текста в формате ASCII позволяет легко модифицировать, тестировать и исправлять ошибки в протоколах. Они могут тестироваться отсылкой команд вручную, при этом записи сообщений легко читать. Сейчас так работает большинство интернет-протоколов прикладного уровня (например, HTTP).

Мы поговорим о простой передаче сообщений между почтовыми серверами, их доставляющими. Установив TCP-соединение с портом 25, передающая машина, выступающая в роли клиента, ждет запроса принимающей машины, работающей в режиме сервера. Сервер начинает диалог с того, что посылает текстовую строку, содержащую его идентификатор и сообщающую о его готовности (или неготовности) к приему почты. Если сервер не готов, клиент разрывает соединение и повторяет попытку позднее.

Если сервер готов принимать почту, клиент объявляет, от кого поступила почта и кому она предназначается. Если получатель почты существует, сервер дает клиенту добро на пересылку сообщения. Затем клиент посылает сообщение, а сервер подтверждает его получение. Контрольные суммы не проверяются, так как транспортный

протокол TCP обеспечивает надежный байтовый поток. Если у отправителя есть еще почта, она также отправляется. После передачи всей почты в обоих направлениях соединение разрывается. Пример диалога клиента и сервера при передаче сообщения из листинга 7.2 показан в листинге 7.3. Строки, посланные клиентом (то есть отправителем), помечены буквой *C:*, а посланные сервером (то есть получателем) — *S:*.

Сначала клиент, естественно, посылает приветствие серверу. Таким образом, первая команда клиента выглядит как `HELO`, что представляет собой наиболее удачный из двух вариантов сокращения слова `HELLO` до четырех символов. Зачем все эти команды было нужно сокращать до четырех букв, сейчас уже никто не помнит.

В нашем примере сообщение должно быть послано только одному получателю, поэтому используется только одна команда `RCPT` (сокращение от слова recipient — получатель). Использование этой команды несколько раз позволяет посылать одно сообщение нескольким получателям. Каждое из них подтверждается или отвергается индивидуально. Несмотря на то что попытки пересылки сообщения некоторым получателям оказываются неудачными (например, из-за отсутствия адресатов), это сообщение все равно может быть доставлено остальным адресатам, числящимся в списке рассылки.

Наконец, хотя синтаксис четырехсимвольных команд строго определен, синтаксис ответов не столь строг. Правила определяют только числовой код в начале строки. Все, что следует за этим кодом, может считаться комментарием и зависит от конкретной реализации протокола.

Базовый протокол SMTP работает хорошо, но он ограничен в некоторых отношениях. Он не включает аутентификацию. Это значит, что команда `FROM` в примере может выдать любой адрес отправителя, который ей понравится. Это крайне удобно для отсылки спама. Другое ограничение заключается в том, что SMTP передает сообщения ASCII, а не бинарные данные. Именно для этого была нужна кодировка base64 для передачи контента MIME. Однако с этой кодировкой при передаче почты недостаточно используется пропускная способность, что неприятно, если отправляется объемное сообщение. Третье ограничение заключается в том, что SMTP отсылает непосредственно сообщение. Оно не шифруется и никак не защищено.

Чтобы справиться с этими и многими другими проблемами, связанными с передачей сообщений, к SMTP было добавлено расширение. Оно является обязательной частью стандарта RFC 5321. Использование SMTP с расширениями называется **ESMTP (Extended SMTP — расширенный SMTP)**.

Листинг 7.3. Передача сообщения от alice@cs.washington.edu для bob@ee.uwa.edu.au

```
S: 220 ee.uwa.edu.au служба SMTP готова
C: HELO abcd.com
S: 250 cs.washington.edu приветствует ee.uwa.edu.au
C: MAIL FROM: <alice@cs.washington.edu>
S: 250 подтверждаю отправителя
C: RCPT TO: <bob@ee.uwa.edu.au>
S: 250 подтверждаю получателя
C: DATA
S: 354 Отправляйте письмо; конец письма обозначается строкой, состоящей из символа "."
C: From: alice@cs.washington.edu
C: To: bob@ee.uwa.edu.au
```

```
C: MIME-Version: 1.0

C: Message-Id: <0704760941.AA00747@ee.uwa.edu.au>
C: Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
C: Subject: Земля обошла вокруг Солнца целое число раз
C:
C: Это преамбула. Пользовательский агент игнорирует ее. Ку-ку.
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: text/richtext
C:
C: Happy birthday to you
C: Happy birthday to you
C: Happy birthday dear <bold> Carolyn </bold>
C: Happy birthday to you
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body;
C: access-type="anon-ftp";
C: site="bicycle.cs.washington.edu";
C: directory="pub";
C: name="birthday.snd"
C:
C: content-type: audio/basic
C: content-transfer-encoding: base64
C: --qwertyuiopasdfghjklzxcvbnm
C: .
S: 250 сообщение принято
C: QUIT
S: 221 ee.uwa.edu.au разрываю соединение
```

Клиенты, желающие использовать расширенную версию, в начале высылают EHLO вместо HELO. Если этот вариант отвергается, сервер работает с обычным SMTP, а пользователь должен идти по стандартному пути. Если *EHLO* принимается, сервер отвечает, какие расширения он поддерживает. После этого клиент может использовать любое из перечисленных расширений. Несколько стандартных расширений показаны в табл. 7.7. В ней даны ключевые слова, в том виде, в котором они используются в механизме расширения, и описание новой функциональности. Более подробно рассматривать расширения мы не будем.

Таблица 7.7. Некоторые расширения SMTP

Ключевое слово	Описание
AUTH	Аутентификация клиента
BINARYMIME	Сервер принимает бинарные сообщения
CHUNKING	Сервер принимает большие сообщения по частям
SIZE	Проверка размера сообщения перед попыткой отсылки
STARTTLS	Переключение на безопасный канал (TLS; см. главу 8)
UTF8SMTP	Интернационализированный адрес

Чтобы лучше понять, как работает SMTP и другие рассмотренные в этой главе протоколы, попробуйте сами поработать с ними. В любом случае, для начала найдите машину, подключенную к Интернету. В системе UNIX (или Linux) наберите в командной строке:

```
telnet mail.isp.com 25
```

подставив вместо *mail.isp.com* DNS-имя почтового сервера провайдера. В системе Windows XP щелкните на кнопке Пуск, затем на кнопке Выполнить и наберите команду в диалоговом окне. На компьютерах с Vista или Windows 7 вам, возможно, сначала придется инсталлировать программу telnet (или ее эквивалент) и запустить ее. В результате выполнения этой команды будет установлено telnet-соединение (то есть соединение TCP) с портом 25 данной машины. Как было показано в табл. 6.4, порт 25 является SMTP-портом (см. табл. 6.4 с портами для других стандартных протоколов). В ответ на введенную команду вы получите что-то вроде этого:

```
Trying 192.30.200.66…
Connected to mail.isp.com
Escape character is '^]'.
220 mail.isp.com Smail #74 ready at Thu, 25 Sept 2002 13:26 +0200
```

Первые три строки посылаются telnet и поясняют для вас происходящее. Последняя строка посылается сервером SMTP удаленной машины и сообщает о готовности к общению с вашей машиной и приему почты. Чтобы узнать о доступных командах, наберите

```
HELP
```

Начиная с этого момента, возможен обмен последовательностями команд, показанными в листинге 7.3, если сервер готов принимать от вас почтовые сообщения.

Подача почтовых сообщений

Первоначально пользовательские агенты запускались на том же компьютере, что и агенты передачи сообщений, пересылающие почту. При таком варианте все, что необходимо для отсылки сообщения, — возможность пользовательского агента связаться с локальным почтовым сервером, используя только что описанный диалог. Однако этот вариант уже не так широко распространен.

Пользовательские агенты часто работают на ноутбуках, домашних компьютерах и мобильных телефонах, а они не всегда подключены к Интернету. Агенты передачи сообщений работают на серверах провайдеров и крупных компаний, которые постоянно подключены к Интернету. Это различие означает, что пользовательскому агенту может понадобиться обратиться из Бостона к его обычному почтовому серверу в Сиэтле, чтобы послать почтовое сообщение, если пользователь отправился в путешествие.

Эта удаленная коммуникация не вызывает проблемы сама по себе. Именно для подобных случаев был разработан протокол TCP/IP. Однако провайдер или компания обычно без энтузиазма относятся к тому, что у удаленного пользователя будет возможность подавать сообщения на их почтовый сервер для доставки в какое-то иное место. Такой сервер не является общественным. Кроме того, этот вид **открытой**

почтовой станции (open mail relay) привлекает спамеров. Это происходит из-за наличия возможности скрыть настоящего отправителя и таким образом затруднить идентификацию сообщения как спама.

Учитывая эти особенности, SMTP обычно используется для подачи писем с расширением *AUTH*. Это расширение позволяет серверу проверять данные отправителя (имя пользователя и пароль) для подтверждения того, что сервер должен обеспечить работу с почтой.

Есть еще несколько отличий в том, как SMTP используется при подаче почты. Например, задействуется порт 587, а не порт 25, и SMTP-сервер может проверять и исправлять формат сообщений, отосланных пользовательским агентом. Чтобы узнать больше об использовании SMTP при подаче писем, посмотрите RFC 4409.

Передача сообщений

Когда отсылающий агент передачи почты (сообщений) получает сообщение от пользовательского агента, он доставляет его получающему агенту передачи почты, используя SMTP. Чтобы это сделать, отсылающая сторона использует адрес назначения. Посмотрите на сообщение в листинге. 7.2, адресованное к *bob@ee.uwa.edu.au*. На какой почтовый сервер оно должно быть доставлено?

Чтобы определить верный почтовый сервер, запрашивается DNS. В предыдущем разделе мы описали то, как в состав DNS входят различные типы записей, включающие запись *MX* (mail exchanger — запись для обмена почтовыми сообщениями). В этом случае запрос DNS делается для записей *MX* домена *ee.uwa.edu.au*. Этот запрос возвращает упорядоченный список имен и IP-адресов одного или более почтовых серверов.

Затем отсылающий агент передачи почты создает TCP-соединение с IP-адресом почтового сервера на порте 25, чтобы связаться с принимающим агентом передачи почты. Для передачи сообщения используется SMTP. Принимающий агент передачи почты затем помещает сообщения для пользователя *bob* в нужный почтовый ящик, чтобы позднее Боб мог их прочитать. Этот шаг локальной доставки может включать перемещение сообщения между компьютерами, если существует разветвленная почтовая инфраструктура.

При помощи этого процесса доставки почта путешествует от начального до конечного агента передачи почтовых сообщений за один шаг. На этапе передачи сообщения нет промежуточных серверов. Однако этот процесс доставки может повторяться несколько раз. Один из примеров мы уже привели (когда агент передачи сообщений применяет список рассылки). В этом случае сообщение получает список адресатов. Затем оно распадается на сообщения для каждого члена списка и посылается на его индивидуальный адрес.

Еще один пример перенаправления почты выглядит следующим образом: Боб мог закончить Массачусетский технологический институт и, соответственно, быть доступным по адресу *bob@alum.mit.edu*. Вместо того чтобы собирать почту с разных аккаунтов, Боб может сделать так, чтобы почта, посылаемая на данный адрес, переправлялась на *bob@ee.uwa.edu*. В этом случае сообщения, отправленные на *bob@alum.*

mit.edu, будут доставляться дважды. Сначала они будут посылаться на почтовый сервер *alum.mit.edu*. Затем на сервер *ee.uwa.edu.au*. Каждый из этих шагов является полноценной отдельной доставкой, если мы рассматриваем их с точки зрения агентов передачи почтовых сообщений.

Также не стоит забывать о спаме. Сегодня девять из десяти отсылаемых сообщений принадлежат именно к этой категории (McAfee, 2010). Мало кто хочет получать подобную почту, но ее сложно избежать, так как нежелательные сообщения маскируются под обычные. До приема сообщения могут быть проведены дополнительные проверки, чтобы сократить возможности спама. Сообщение для Боба было отослано с адреса *alice@cs.washington.edu*. Принимающий агент передачи почты может обратиться к агенту передачи почты, отправившему сообщение, через DNS. Это позволяет проверить, совпадает ли IP-адрес на другом конце TCP-соединения и DNS-имя. В общем случае, получающий агент может проверить отсылающий домен в DNS и выяснить, придерживается ли он политики отправки почты. Эта информация часто выдается в формате TXT- и SPF-записей. В ней может говориться о том, что возможно проведение и других проверок. Например, почта, посылаемая с *cs.washington.edu*, может всегда отправляться с хоста *june.cs.washington.edu*. Если высылающий агент передачи почты не *june*, возникает проблема.

Если проваливается любая из этих проверок, вероятно, почта была отослана с поддельного адреса. В этом случае сообщение не принимается. Однако если даже оно проходит все эти проверки, нет гарантии, что оно не является спамом. Проверки просто подтверждают, что почта, вероятно, приходит из той части сети, из которой и должна приходить. Смысл этого заключается в том, что при отсылке почты спамерам необходимо отправлять сообщения с верного адреса. Это позволяет проще распознать и удалить спам, если он нежелателен.

7.2.5. Окончательная доставка сообщений

Наше почтовое сообщение почти доставлено. Оно прибыло в почтовый ящик Боба. Осталось только передать копию сообщения пользовательскому агенту Боба, чтобы оно могло отобразиться. Это шаг 3 в архитектуре на рис. 7.4. Когда пользовательский агент и агент передачи почты работали на одном и том же компьютере и представляли собой всего лишь разные процессы, эта задача была несложной. Агент передачи почты просто писал новые сообщения в конце файла почтового ящика, а пользовательский агент просто проверял файл почтового ящика на наличие новых сообщений.

Сегодня пользовательский агент, работающий на персональном компьютере, ноутбуке или мобильном телефоне, вероятно, будет размещаться не на почтовом сервере компании или провайдера. Пользователи хотят иметь доступ к своей почте вне зависимости от того, где они находятся. Им нужен доступ к почте на работе, с домашнего компьютера, с ноутбука в командировках или из интернет-кафе, когда они отдыхают. Им нужна возможность работать без постоянного соединения с сетью, периодически подключаясь к Интернету для получения входящих сообщений и отправки исходящих. Более того, каждый пользователь может запускать несколько агентов в зависимости от того, каким компьютером ему удобнее пользоваться в данный момент. Несколько пользовательских агентов даже могут работать одновременно.

В данном случае пользовательский агент должен отображать содержимое почтового ящика и позволять работать с ним удаленно. Для этой цели может задействоваться несколько разных протоколов, но только не SMTP. SMTP — это протокол, основанный на проталкивании данных. Для передачи сообщения ему необходимо соединение с удаленным сервером. Таким образом, окончательная доставка не может быть осуществлена по двум причинам: из-за того, что почтовый ящик должен храниться на агенте передачи почты, и из-за того, что пользовательский агент может быть не подключен к Интернету, когда SMTP пытается передать сообщение.

IMAP — протокол доступа к электронной почте в Интернете

Один из главных протоколов, использующихся для конечной доставки, — это **IMAP (Internet Mail Access Protocol — протокол доступа к электронной почте в Интернете)**. Четвертая версия этого протокола определена в RFC 3501. Чтобы использовать IMAP, почтовый сервер запускает IMAP-сервер, который проверяет порт 143. Пользовательский агент запускает IMAP-клиент. Этот клиент соединяется с сервером и начинает передавать команды из перечисленных в табл. 7.8.

Сначала клиент запускает защищенный транспорт, если он используется (чтобы сохранить конфиденциальность сообщений и команд), а затем вводит логин и пароль или иначе авторизуется на сервере. После входа в систему можно воспользоваться множеством команд, чтобы просматривать папки и сообщения или даже части сообщений, помечать письма галочками для дальнейшего удаления и разбирать их по папкам. Обратите внимание на то, что во избежание неразберихи мы в этой главе используем слово «*папка*» (*folder*), основываясь на том, что почтовый ящик пользователя состоит из нескольких таких папок. Однако в спецификации IMAP вместо папки используется термин *mailbox* («*почтовый ящик*»). Таким образом, получается, что у пользователя есть много почтовых ящиков IMAP, каждый из которых пользователь видит как папку.

Протокол IMAP обладает разнообразными возможностями, как, например, способность упорядочивать почту не по порядку ее поступления, а по атрибутам писем (например, «сначала дайте мне письмо от Элис»). На сервере также могут быть поисковые инструменты, чтобы можно было найти сообщения, удовлетворяющие определенным критериям, так что клиент увидит только их.

IMAP — это улучшенная версия ранее разработанного протокола окончательной доставки **POP3 (Post Office Protocol, version 3 — протокол почтового отделения, версия 3)**, который определен в RFC 1939. Протокол POP3 проще, но он поддерживает меньше возможностей и является менее безопасным при обычном использовании. Обычно почта загружается на компьютер с пользовательским агентом, а не остается на почтовом сервере. Это облегчает жизнь серверу, но усложняет ее пользователю. Читать почту с нескольких компьютеров становится гораздо сложнее, к тому же, если компьютер с пользовательским агентом сломается, вся почта будет безвозвратно утеряна. Тем не менее POP3 все еще используется.

Кроме того, могут использоваться и частные протоколы, так как протокол работает между почтовым сервером и пользовательским агентом, который может обеспечиваться той же компанией. Примером почтовой системы с частным протоколом является Microsoft Exchange.

Таблица 7.8. Команды IMAP (версия 4)

Команда	Описание
CAPABILITY	Перечислить возможности сервера
STARTTLS	Запустить безопасный транспорт (TLS, см. главу 8)
LOGIN	Войти на сервер, используя имя пользователя и пароль
AUTHENTICATE	Авторизоваться иным способом
SELECT	Выбрать папку
EXAMINE	Выбрать папку, предназначенную только для чтения
CREATE	Создать папку
DELETE	Удалить папку
RENAME	Переименовать папку
SUBSCRIBE	Добавить папку к активному набору
UNSUBSCRIBE	Удалить папку из активного набора
LIST	Перечислить доступные папки
LSUB	Перечислить активные папки
STATUS	Узнать статус папки
APPEND	Добавить сообщение в папку
CHECK	Просмотреть состояние выбранной папки (что именно входит в понятие «состояние», зависит от конкретной реализации сервера). Создать контрольную точку для папки
FETCH	Просмотреть сообщения, находящиеся в папке
SEARCH	Найти сообщения, находящиеся в папке
STORE	Изменить метки сообщения
COPY	Сделать копию сообщения в папке
EXPUNGE	Удалить отмеченные сообщения
UID	Вызвать команды, используя уникальные идентификаторы
NOOP	Ничего не делать
CLOSE	Удалить помеченные сообщения и закрыть папку
LOGOUT	Выйти из системы и закрыть соединение

Веб-почта

Альтернативой IMAP и SMTP для предоставления почтовых услуг, набирающей все большую популярность, стало использование веб-интерфейса для отсылки и получения сообщений. Широко используемые системы **веб-почты (Webmail)** включают Google Mail, Microsoft Hotmail и Yahoo! Mail[1]. Веб-почта — это один из примеров про-

[1] Необходимо отметить, что перечень этот далеко не исчерпывающий, а популярность различных сервисов веб-почты может отличаться в разных регионах. Например, в России Яндекс. Почта или Mail.Ru известны, возможно, больше, чем последние два пункта приведенного перечня. — *Примеч. ред.*

граммного обеспечения (в данном случае пользовательский почтовый агент), которое предоставляется как услуга, используя веб-технологии.

При этой архитектуре провайдер управляет почтовыми услугами как обычно, чтобы принимать сообщения для пользователей при помощи SMTP через порт 25. Однако пользовательский агент отличается. Он является не отдельной программой, а пользовательским интерфейсом, который предоставляется через веб-страницы. Это значит, что пользователи могут использовать любой понравившийся им браузер, чтобы получить доступ к своей почте и отсылать новые сообщения.

Мы пока не говорили о веб-технологиях в Интернете и о Всемирной паутине, но краткое описание выглядит примерно следующим образом: когда пользователь заходит на почтовую веб-страницу провайдера, он видит форму, в которой ему нужно заполнить поля *Имя пользователя* и *Пароль*. Пароль и имя пользователя отправляются на сервер, где проверяется их правильность. Если вход в систему осуществлен корректно, сервер находит почтовый ящик пользователя и строит веб-страницу, на которой отображается содержание почтового ящика. Эта веб-страница отсылается на браузер клиента.

Пользователь может щелкать на многих элементах страницы, отображающей его почтовый ящик, так что сообщения можно читать, удалять и т. д. Чтобы интерфейс лучше откликался на действия пользователя, веб-страницы часто включают программы на JavaScript. Эти программы запускаются локально на машине пользователя в ответ на определенные события (например, клики мышкой) и могут загружать на сервер и подгружать с сервера сообщения в фоновом режиме, чтобы подготовить к показу следующее письмо или подать новое. В этой модели подача почты происходит при использовании обычных веб-протоколов при помощи отправки данных по URL. Веб-сервер помещает сообщения в обычную, уже описанную нами систему доставки почты. Для безопасности также могут использоваться стандартные веб-протоколы. Эти протоколы связаны с шифрованием веб-страниц, а не с тем, является ли содержание страницы почтовым сообщением.

7.3. Всемирная паутина (WWW)

Всемирная паутина (WWW, World Wide Web, часто для краткости просто «веб») — это архитектура, являющаяся основой для доступа к связанному контенту, находящемуся на миллионах машин по всему Интернету. За 10 лет своего существования из средства координации структуры экспериментов на тему физики высоких энергий в Швейцарии она превратилась в приложение, о котором миллионы людей с разными интересами думают, что это и есть «Интернет». Огромная популярность этого приложения стала следствием того, что даже новички не встречают затруднений при его использовании. Кроме того, при помощи цветного графического интерфейса Всемирная паутина предоставляет огромное количество информации практически по любому вопросу, от африканских муравьедов до яшмового фарфора.

Всемирная паутина была создана в 1989 году в Европейском центре ядерных исследований **CERN (Conseil Europeen pour la Recherche Nucleaire)** в Швейцарии. Сначала главной целью было наладить общение внутри больших групп, участники которых зачастую жили в разных странах и разных часовых поясах. Им нужно было пользоваться

постоянно меняющимися отчетами о работе, чертежами, рисунками, фотографиями и другими документами, появляющимися по ходу ведения экспериментов в области физики элементарных частиц. Предложение создать паутину из связанных друг с другом документов пришло от физика центра CERN Тима Бернерс-Ли (Tim Berners-Lee). В декабре 1991 года на конференции Hypertext'91 в Сан-Антонио в штате Техас была произведена публичная демонстрация. Эта демонстрация привлекла внимание других ученых. Марк Андрессен (Marc Andreessen) в университете Иллинойса начал разработку первого графического браузера, Mosaic. Программа увидела свет в феврале 1993 года.

Остальное, как говорится, уже история. Mosaic стала настолько популярной, что год спустя ее автор Марк Андрессен решил сформировать собственную компанию Netscape Communications Corp., чьей целью была разработка программного обеспечения для Всемирной паутины (или короче — ПО для веб). В течение последующих трех лет между Netscape Navigator и Internet Explorer от Microsoft развернулась настоящая «война браузеров». Разработчики с той и с другой стороны в безумном порыве пытались заполучить как можно большую часть нового рынка, пичкая свои программы как можно большим числом функций (а следовательно, и ошибок), и в этом превзойти соперника.

На протяжении 1990-х и 2000-х годов количество веб-сайтов и веб-страниц, так же как и веб-контента, росло по экспоненте, пока количество сайтов не стало исчисляться миллионами, а количество страниц миллиардами. Небольшое число этих сайтов стало невероятно популярным. Эти сайты и стоящие за ними компании сегодня в серьезной мере определяют веб в том виде, в котором его видят люди. В качестве примеров можно перечислить: книжный магазин (Amazon, основанный в 1994 году, рыночная цена $50 млрд), блошиный рынок (eBay, 1995, $30 млрд), поисковик (Google, 1998, $150 млрд) и социальную сеть (Facebook, 2004, частная компания, оцениваемая более чем в $15 млрд). У того периода в 2000-х, когда стоимость многих веб-компаний подскочила до сотен миллионов долларов за одну ночь, только для того, чтобы обрушиться на следующий день (когда оказывалось, что их создание было просто рекламным ходом), даже есть имя. Он называется «**эрой доткомов**» (**dot com era**). Новые идеи до сих пор развиваются в веб подобным образом. Многие из них исходят от студентов. Например, Марк Цукерберг был студентом Гарварда, когда придумал и запустил проект Facebook, а Сергей Брин и Лари Пейдж были студентами Стенфорда, когда запустили Google. Возможно, вы будете следующим.

В 1994 году CERN и Массачусетский технологический институт (M.I.T., Massachusetts Institute of Technologies) подписали соглашение об основании **WWW-консорциума** (**World Wide Web Consortium**, иногда применяется сокращение **W3C**) — организации, цель которой заключалась в дальнейшем развитии Всемирной паутины, стандартизации протоколов и поощрении взаимодействия между отдельными сайтами. Бернерс-Ли стал директором консорциума. Хотя о Всемирной паутине уже написано очень много книг, лучшее место, где вы можете получить самую свежую информацию о ней, это сама Всемирная паутина. Домашнюю страницу консорциума можно найти по адресу *http://www.w3.org*. На этой странице заинтересованный читатель найдет ссылки на другие страницы, содержащие информацию обо всех документах консорциума и о его деятельности.

7.3.1. Представление об архитектуре

С точки зрения пользователя Всемирная паутина состоит из огромного количества контента в форме **веб-страниц (Web pages)**, которые часто называются просто **страницами (pages)** для краткости. Каждая страница может содержать ссылки (указатели) на другие связанные с ней страницы в любой точке мира. Пользователи могут переходить по ссылке (например, просто щелкнув на ней мышью), при этом страница, на которую указывает ссылка, загружается и появляется в окне браузера. Этот процесс можно повторять бесконечно. Идея страниц, связанных между собой, ныне называемых **гипертекстом (hypertext)**, была впервые пророчески предложена в 1945 году, задолго до появления Интернета, Ванневаром Бушем (Vannevar Bush), профессором из Массачусетского университета, занимавшимся электротехникой. На самом деле, это случилось даже до того, как появились коммерческие компьютеры, хотя в университетах уже существовали грубые прототипы, которые занимали огромные помещения, а по мощности уступали современным карманным калькуляторам.

Страницы просматриваются специальной программой, называемой **браузером (browser)**. Самыми популярными браузерами являются программы Firefox, Internet Explorer и Chrome. Браузер предоставляет пользователю запрашиваемую страницу, интерпретирует ее контент и выводит должным образом отформатированные страницы на экран. Контент может быть представлен в виде сочетания текста, изображений и команд форматирования и выглядеть как обычный документ или как видео или программы с определенным графическим интерфейсом, в рамках которого может работать пользователь.

Пример веб-страницы дан в верхнем левом углу рис. 7.7. Это страница факультета информатики и вычислительной техники в Вашингтонском университете. На ней присутствуют текст и графические элементы (правда, не в том масштабе, чтобы все можно было хорошо разобрать). Некоторые части страницы связаны с другими страницами ссылками. Строки текста, значки, изображения и т. д., представляющие собой ссылки на другие страницы, называются **гиперссылками (hyperlink)**. Чтобы перейти по ссылке, пользователь перемещает указатель мыши в выделенную область, при этом меняется форма указателя, и щелкает на ней. Переход по ссылке является просто сообщением браузеру того, что нужно загрузить другую страницу. Поначалу ссылки выделялись либо подчеркиванием, либо другим цветом, чтобы их можно было различить на странице. Сегодня создатели веб-страниц постоянно контролируют внешний вид ссылок, так что они могут быть выглядеть как значки, или менять внешний вид, когда в их области появляется курсор мышки. Сделать ссылки визуально отличимыми и обеспечить им подходящий интерфейс — задача создателей сайтов.

Студенты факультета могут получить дополнительную информацию, пройдя по ссылке, предназначенной специально для них. Ссылка активируется по щелчку на обведенном поле. При этом браузер получит из сети новую страницу и отобразит ее (как показано в левом нижнем углу рис. 7.7) на экране. Помимо этого на первой странице есть еще десятки ссылок. Каждая новая страница может состоять из контента, расположенного как на той же самой машине, что и первая страница, так и на компьютере, расположенном на противоположном конце Земли. Для пользователя это не заметно. Браузер добывает запрашиваемые страницы без участия пользователя.

Таким образом, пользователь может постоянно перемещаться между компьютерами, просматривая контент.

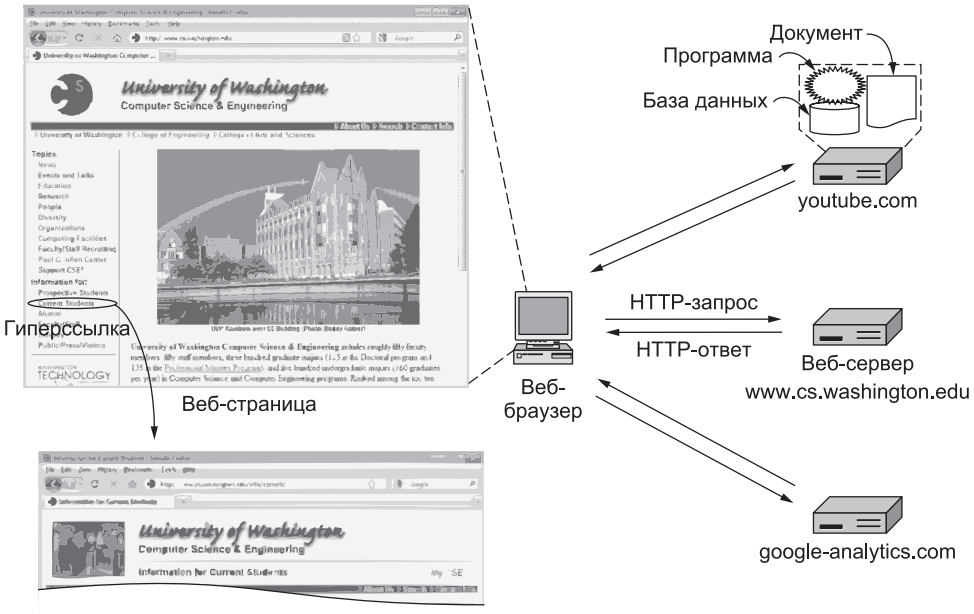


Рис. 7.7. Архитектура Всемирной паутины

Основной принцип, стоящий за отображением страниц, также показан на рис. 7.7. Браузер отображает веб-страницу на клиентской машине. Каждая страница отображается посредством отсылки запроса на один или более серверов, который отвечает, передавая контент страницы. Протокол запроса-ответа для отображения страниц — это простой текстовый протокол, который работает через TCP, так же как и в случае с SMTP. Он называется **HTTP (HyperText Transfer Protocol — протокол передачи гипертекста)**. В качестве контента может выступать простой документ, который считывается с диска, или результат работы программы или запроса, отосланного в базу данных. Страница называется **статичной (static page)**, если она представляет из себя статичный документ, который всегда отображается одинаково. Если страница, напротив, создается по требованию программы или сама содержит какую-либо программу, она называется **динамической (dynamic page)**.

Контент динамической страницы может быть разным при каждом вызове. Например, главная страница электронного магазина может выглядеть различно для каждого посетителя. Если клиент книжного магазина в прошлом покупал мистические романы, при посещении главной страницы магазина он, вероятно, увидит изображения новых триллеров, в то время как человек, больше интересующийся новыми рецептами, обнаружит на этой странице новые кулинарные книги. Вкратце мы расскажем, как веб-

сайты следят за тем, кто что покупает. Если не вдаваться в детали, дело заключается в файлах cookie (даже в отношении тех, кто не особо интересуется кулинарией)[1].

На рисунке браузер обращается к трем серверам, чтобы загрузить две страницы, *cs.washington.edu*, *youtube.com* и *google-analytics.com*. Контент с этих серверов интегрируется и отображается браузером. При отображении запускается обработка, тип которой зависит от типа контента. Кроме отображения текста и графических элементов, может возникнуть необходимость проиграть видеофайл или запустить скрипт, в котором прописан отдельный пользовательский интерфейс, являющийся частью страницы. В нашем случае с сервера *cs.washington.edu* загружается главная страница, с сервера *youtube.com* — размещенное на ней видео, а с сервера *google-analytics.com* — ничего из того, что видно пользователю; этот сервер ведет запись посетителей сайта. Позднее мы еще поговорим о трекерах.

Сторона клиента

Давайте теперь более детально рассмотрим сторону веб-браузера, основываясь на рис. 7.7. По сути дела, браузер — это программа, которая может отображать веб-страницы и распознавать щелчки мыши на элементах активной страницы. При выборе элемента браузер следует по гиперссылке и получает с сервера запрашиваемую страницу.

Когда была создана Всемирная паутина, сразу стало очевидным, что наличие ссылок с одних страниц на другие требует создания механизма именования и расположения страниц. Самыми важными вопросами, на которые необходимо было дать ответ перед отображением выбранной страницы, были следующие:

1. Как называется страница?
2. Где она расположена?
3. Как можно получить к ней доступ?

Если каждой странице приписать уникальное имя, в их идентификации не было бы никакой неоднозначности. Тем не менее это не решило бы проблему. Рассмотрим параллель между людьми и страницами. В США почти у каждого есть номер социальной страховки, который является уникальным идентификатором, так как у двух разных людей не может быть одного номера социальной страховки. Тем не менее, если у вас есть только номер этой страховки какого-то человека, вы не сможете выяснить его адрес, и уж конечно вы никак не сможете определить, на каком языке следует писать этому человеку, на английском, испанском или китайском. Во Всемирной паутине возникают примерно те же проблемы.

Избранное решение идентифицирует страницы таким образом, что сразу были решены все проблемы. Каждой странице был приписан **URL (Uniform Resource Locator — унифицированный указатель информационного ресурса)**, который служит именем страницы во Всемирной паутине. URL делится на три части: протокол (который также называют **схемой — scheme**), DNS-имя машины, на которой расположена страница, и путь, уникально определяющий отдельную страницу (файл для чтения

[1] Один из вариантов перевода слова «cookie» — «печенье». — *Примеч. ред.*

или программу, предназначенную для запуска на машине). В общем случае у пути есть иерархическое имя, которое моделирует структуру каталогов файлов. Однако интерпретация пути — это работа сервера. Действительная структура каталогов может и не отображаться.

В качестве примера приведем URL страницы, указанный на рис. 7.7:

http://www.cs.washington.edu/index.html

Этот URL состоит из трех частей: протокола (*http*), DNS-имени хоста (*www.cs.washington.edu*) и имени пути (*index.html*).

Когда пользователь щелкает мышью на гиперссылке, браузером выполняется ряд действий, приводящих к загрузке страницы, на которую указывает ссылка. Рассмотрим каждое действие, происходящее после выбора этой ссылки.

1. Браузер определяет URL (по выбранному элементу страницы).
2. Браузер запрашивает у службы DNS IP-адрес сервера *www.cs.washington.edu*.
3. DNS дает ответ 128.208.3.88.
4. Браузер устанавливает TCP-соединение с 80-м портом (общеизвестным портом для HTTP-протокола) машины 128.208.3.88.
5. Браузер отправляет HTTP-запрос на получение файла */index.html*.
6. Сервер *www.cs.washington.edu* высылает страницу, как HTTP-ответ, например, отправляя файл */index.html*.
7. Если страница содержит URL, которые необходимы для отображения, браузер получает другие URL, используя тот же процесс. В этом случае URL включают множество размещенных изображений, также полученных с *www.cs.washington.edu*, размещенное видео с *youtube.com* и скрипт с *google-analitics.com*.
8. Браузер отображает страницу */index.html* в том виде, в котором она представлена на рис. 7.7.
9. Если в течение некоторого времени на те же серверы не поступает других запросов, TCP-соединения обрываются.

Многие браузеры отображают текущее выполняемое ими действие в строке состояния внизу экрана. Это позволяет пользователю понять причину низкой производительности: например, не отвечает служба DNS или сервер или просто сильно перегружена сеть при передаче страницы.

URL-дизайн не ограничен в том смысле, что он позволяет браузерам использовать различные протоколы для того, чтобы обрабатывать разные виды ресурсов. На самом деле, были определены URL и для многих других протоколов. Несколько упрощенные формы стандартных схем URL приведены в табл. 7.9.

Давайте коротко остановимся на этом списке. Протокол *http* — это родной язык Всемирной паутины, тот, на котором разговаривают веб-серверы. **HTTP** — это сокращение выражения **HyperText Transfer Protocol (протокол передачи гипертекста)**. Мы поговорим о нем более детально немного позднее.

Протокол *ftp* используется для доступа к файлам через FTP, протокол передачи файлов в Интернете. FTP предшествует возникновению Всемирной паутины. Он используется уже более трех десятилетий. Веб позволяет легко получить доступ к фай-

лам, расположенным на различных FTP-серверах по всему миру, предоставляя простой интерфейс, основанный на работе с мышью, вместо интерфейса, построенного на вводе команд в командную строку. Упрощение способа доступа к информации является одной из причин повышения популярности Всемирной паутины и ее быстрого роста.

Таблица 7.9. Некоторые стандартные схемы URL

Имя	Используется	Пример
http	Гипертекст (HTML)	http://www.ee.uwa.edu/~rob/
https	Гипертекст с обеспечением безопасности	https://www.blank.com./accounts/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Локальный файл	file:///usr/suzanne/prog.c
mailto	Отсылка почты	mailto:JohnUser@acm.org
rtsp	Потоковая передача мультимедиа	rtsp://youtube.com/montypython.mpg
sip	Мультимедийный звонок	sip:eve@adversary.com
about	Информация браузера	about:plugins

Можно получить доступ к локальному файлу как к веб-странице, используя протокол *file* или просто написав его имя. Для применения этого способа не нужен сервер. Конечно, это работает только для локальных файлов, а не для удаленных.

Протокол *mailto*, на самом деле, не позволяет запрашивать веб-страницы, но он все равно полезен. С его помощью можно отсылать почту через веб-браузер. Большинство браузеров отвечают на переход пользователя по ссылке *mailto* запуском пользовательского агента, в котором можно начинать писать сообщение с уже добавленным адресом.

Протоколы *rtsp* и *sip* предназначены для установления сессий передачи мультимедийных потоков, а также аудио- и видеозвонков.

Наконец, протокол *about* предоставляет информацию о браузере. Например, если вы пройдете по ссылке *about:plugins*, большинство браузеров отобразит страницу, перечисляющую типы MIME, которые доступны благодаря расширениям браузера (так называемым плагинам).

Вкратце, URL были разработаны не только для навигации во Всемирной паутине, но и для того, чтобы запускать более старые протоколы, такие как FTP и электронная почта, и новые для аудио и видео и, конечно, для того, чтобы предоставить удобный доступ к локальным файлам и информации браузера. Благодаря этому подходу теряется необходимость во всех программах, предоставляющих пользовательский интерфейс для вышеперечисленных нужд, а доступ в Интернет практически полностью интегрируется в одну программу: веб-браузер. Если бы не тот факт, что эта идея пришла в голову британского физика, работающего в исследовательской лаборатории в Швейцарии, можно было бы легко предположить, что это был прекрасный план, разработанный рекламным отделом компании, выпускающей программное обеспечение.

Несмотря на все эти выдающиеся качества, постоянно увеличивающееся использование Всемирной паутины выявило некоторые недостатки, заложенные в схему URL.

URL указывает на один отдельный хост, но иногда имеет смысл ссылаться на страницу, не указывая того, где она находится. Например, было бы неплохо, если бы страницы, на которые идет множество ссылок, многократно копировались в разных частях сети, чтобы уменьшить трафик. Но мы не можем сказать: «Мне нужна страница *xyz* и мне все равно, откуда она возьмется».

Чтобы решить эту проблему, URL были обобщены до **URI (Uniform Resource Identifier — универсальный идентификатор ресурса)**. Некоторые URI указывают, как определить место нахождения ресурса. Это URL. Другие URI указывают имя ресурса, но не место его нахождения. Эти URI называются **URN (Uniform Resource Name — унифицированное имя ресурса)**. Правила написания URI отражены в RFC 3986, другие схемы использования URI определяются IANA. Существует много различных типов URI, помимо перечисленных в табл. 7.9; мы перечислили только те, что сегодня наиболее часто используются во Всемирной паутине.

Типы MIME

Для отображения новой (и каждой) страницы браузер должен понять ее формат. Чтобы все браузеры могли отображать любые страницы, они пишутся на стандартизованном языке HTML. На сегодняшний день он является общепринятым в Интернете. Более детально мы рассмотрим его ниже.

Несмотря на то что браузер по сути дела представляет собой интерпретатор HTML, большинство браузеров оснащается многочисленными кнопками и функциями, облегчающими навигацию по Всемирной паутине. У многих браузеров есть кнопки для возврата на предыдущую страницу и перехода на следующую страницу (последняя доступна только в том случае, если пользователь уже возвращался назад), а также кнопка для прямого перехода на выбранную пользователем начальную страницу. Большинство браузеров поддерживают в меню команды для установки закладки на текущей странице и отображения списка закладок, что позволяет попадать на любую страницу при помощи всего одного щелчка мышью.

Как показано в нашем примере, HTML-страницы могут содержать разнообразные элементы контента, а не только текст и гипертекст. Строго говоря, не всем страницам необходимо содержать HTML. Страница может содержать видео в формате MPEG, документ в формате PDF, фотографию в формате JPEG, песню в формате MP3 и еще сотни различных типов файлов. Поскольку стандартные HTML-страницы могут иметь ссылки на любые файлы, у браузера возникает проблема обработки страницы, которую он не может интерпретировать.

Вместо того чтобы наращивать возможности и размеры браузеров, встраивая в них интерпретаторы для различных типов файлов (количество которых быстро растет), обычно применяется более общее решение. Когда сервер возвращает в ответ на запрос какую-либо страницу, вместе с ней высылается некоторая дополнительная информация о ней. Эта информация включает MIME-тип страницы (см. табл. 7.6). Страницы типа *text/html* выводятся браузером напрямую, как и страницы некоторых других встроенных типов. Если же для данного MIME-типа внутренняя интерпретация невозможна, браузер определяет, как выводить страницу, по своей таблице MIME-типов. В данной таблице в соответствие каждому типу ставится программа просмотра.

Существует два способа отображения: с помощью подключаемого модуля, **плагина (plug-in)**, или вспомогательных приложений. Подключаемый модуль представляет собой особый сторонний код, который браузер извлекает из специального каталога на жестком диске и устанавливает в качестве своего расширения, как показано на рис. 7.8, *а*. Стандартными примерами являются плагины для PDF, Flash и Quick-time, позволяющие работать с документами и проигрывать аудио и видео. Поскольку подключаемые модули работают внутри браузера, у них есть доступ к текущей странице, вид которой они могут изменять. После завершения своей работы (обычно это связано с переходом пользователя на другую страницу), подключаемый модуль удаляется из памяти браузера.

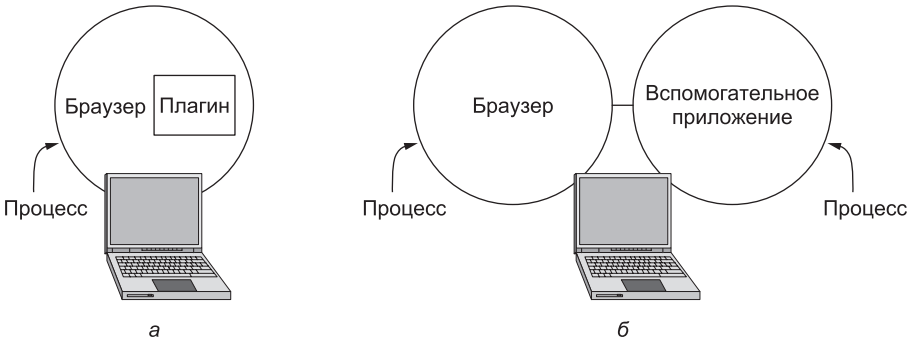


Рис. 7.8. Браузер с подключаемым модулем (*а*); вспомогательное приложение (*б*)

Каждый браузер имеет набор процедур, которые должны реализовывать все подключаемые модули. Это нужно для того, чтобы браузер мог обращаться к последним. Например, существует стандартная процедура, с помощью которой базовый код браузера передает подключаемому модулю данные для отображения. Набор этих процедур образует интерфейс подключаемого модуля и является специфичным для каждого конкретного браузера.

Кроме того, браузер предоставляет подключаемому модулю определенный набор своих процедур. Среди них в интерфейс браузера обычно включаются процедуры распределения и освобождения памяти, вывода сообщений в строке статуса браузера и опроса параметров браузера.

Перед использованием подключаемого модуля его нужно установить. Этот процесс подразумевает, что пользователь копирует с веб-сайта производителя модуля установочный файл. Запуск установочного файла распаковывает подключаемый модуль, регистрирует в браузере MIME-тип и ассоциирует этот тип с модулем. Обычно в браузерах заранее загружены популярные плагины.

Вторым способом расширения возможностей браузера является использование **вспомогательных приложений (helper application)**. Вспомогательное приложение — это полноценная программа, работающая как отдельный процесс. Это показано на рис. 7.8, *б*. Поскольку она никак не связана с браузером, между ними отсутствует какой бы то ни было интерфейс. Вспомогательное приложение обычно принимает от браузера имя временного файла, содержащего данные для отображения, открывает

файл и отображает контент. Обычно в роли вспомогательных приложений выступают большие программы, которые существуют отдельно от браузера, например Microsoft Word или PowerPoint.

Многие вспомогательные приложения используют MIME-тип *application* (приложение). В результате было определено существенное число подтипов, например *application/vnd.ms-powerpoint* для файлов PowerPoint; *vnd* обозначает формат, зависящий от поставщика. Таким образом, URL может напрямую указывать на файл PowerPoint, и когда пользователь щелкает на нем, автоматически загружается PowerPoint, обрабатывает нужный файл и отображает его. Вспомогательные приложения могут пользоваться не только MIME-типом *application*. Так, например, Adobe Photoshop использует *image/x-photoshop*.

Получается, что браузер можно настроить на обработку практически любого количества типов файлов, не внося в него никаких изменений. Современные веб-серверы часто содержат сотни комбинаций типов и подтипов файлов. Новые типы файлов появляются всякий раз при установке новых программ.

Источником конфликтов является то, что большое количество плагинов и вспомогательных приложений могут обрабатывать один и тот же подтип, например *video/mpg*. В результате программа, которая регистрируется последней, затирает своей записью существующую ассоциацию с типом MIME и оставляет его за собой. Следствием этого является то, что каждая устанавливаемая программа может изменить метод отображения браузером некоторых типов.

Браузеры могут работать и с локальными файлами без соединения с сетью, не запрашивая информацию с удаленных серверов. Однако браузеру нужно каким-то хитрым образом определить MIME-тип файла. Стандартный метод заключается в ассоциировании на уровне операционной системы расширения файла с типом MIME. В стандартной конфигурации попытка открыть файл foo.pdf приведет к его открытию в браузере с использованием плагина *application/pdf*, а файл bar.doc будет открыт во вспомогательном приложении Word.

Здесь также возможны конфликты, поскольку многие программы страстно желают поддерживать, например, .mpg. Профессиональные программы при установке обычно выводят флажки, позволяющие выбрать поддерживаемые типы MIME и расширения. Таким образом, пользователь может выбрать то, что ему требуется, и таким образом избежать случайного затирания существующих ассоциаций. Программы, нацеленные на массового потребителя, полагают, что большинство пользователей понятия не имеют о типах MIME и просто захватывают все, что могут, совершенно не обращая внимания на ранее установленные программы.

Расширение возможностей браузера по поддержке новых типов файлов — это удобно, но может также привести к возникновению некоторых проблем. Когда браузер на компьютере с Windows получает файл с расширением exe, он думает, что это исполняемая программа и никаких вспомогательных средств для нее не требуется. Очевидно, следует просто запустить программу. Однако такой подход может оказаться серьезной дырой в системе защиты информации. Злоумышленнику требуется лишь создать нехитрый сайт с фотографиями, скажем, знаменитых киноактеров или спортсменов, и поставить ссылки на вирусы. Один-единственный щелчок мышкой на фотографии может в этом случае привести к запуску непредсказуемой и, возможно, опасной программы, которая будет действовать на машине пользователя. Для предотвращения

подобных нежелательных ситуаций Firefox и другие программы настроены на избирательный запуск неизвестных программ, однако не все пользователи понимают, что безопасный выбор важнее удобного.

Сторона сервера

О стороне клиента сказано уже достаточно много. Поговорим теперь о стороне сервера. Как мы уже знаем, когда пользователь вводит URL или щелкает на гиперссылке, браузер производит структурный анализ URL и интерпретирует часть, заключенную между *http://* и следующей косой чертой, как имя DNS, которое следует искать. Вооружившись IP-адресом сервера, браузер устанавливает TCP-соединение с 80 портом этого сервера. После этого отсылается команда, содержащая оставшуюся часть URL, в которой указывается путь к странице на сервере. Сервер возвращает браузеру запрашиваемый файл для отображения.

В первом приближении простой веб-сервер напоминает сервер, представленный в листинге 6.1. Этому серверу, как и настоящему веб-серверу, передается имя файла, который следует найти и отправить по сети. В обоих случаях в основном цикле сервер выполняет следующие действия:

1. Принимает входящее TCP-соединение от клиента (браузера).
2. Получает путь к странице, являющийся именем запрашиваемого файла.
3. Получает файл (с диска).
4. Высылает содержимое файла клиенту.
5. Разрывает TCP-соединение.

Современные веб-серверы обладают более широкими возможностями, однако существенными в их работе являются именно перечисленные шаги, предпринимаемые в случае запроса контента, содержащегося в файле. В том случае, если контент является динамическим, третий шаг может быть заменен запуском программы (определенной по пути), возвращающей контент.

Однако веб-серверы реализуются иным способом, для того чтобы они отвечали на множество запросов за одну секунду. Одна из проблем, связанных с простым способом реализации, заключается в том, что доступ к файлам часто затруднен. Считывание с диска идет слишком медленно по сравнению с работой программы, и одни и те же файлы могут считываться с диска несколько раз из-за вызовов операционной системы. Другая проблема заключается в том, что одновременно не могут обрабатываться несколько запросов. Файл может быть большим, и пока он передается, другие запросы блокируются.

Очевидным способом решения проблемы является кэширование в памяти *n* последних запрошенных файлов или определенного количества гигабайт контента. Прежде чем обратиться за файлом к диску, сервер проверяет содержимое кэша. Если файл обнаруживается в кэше, его можно сразу выдать клиенту, не обращаясь к диску. Несмотря на то что для эффективного кэширования требуются большие объемы памяти и некоторое дополнительное время на проверку кэша и управление его содержимым, суммарный выигрыш во времени почти всегда оправдывает эти накладные расходы и стоимость.

Одним из способов решения проблемы последовательной обработки запросов является создание **многопоточных (multithreaded)** серверов. Одна из реализаций подразумевает, что сервер состоит из входного модуля, принимающего все входящие

запросы, и k обрабатывающих модулей, как показано на рис. 7.9. Все $k + 1$ потоков принадлежат одному и тому же процессу, поэтому у обрабатывающих модулей есть доступ к кэшу в адресном пространстве процесса. Когда приходит запрос, входящий модуль принимает его и создает краткую запись с его описанием. Затем запись передается одному из обрабатывающих модулей.

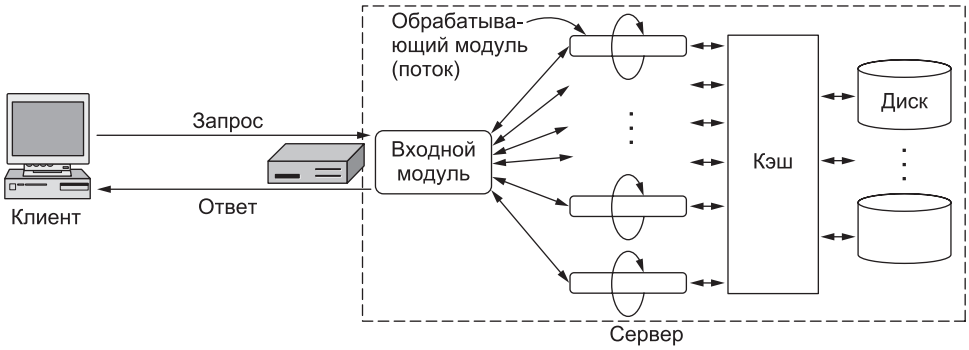


Рис. 7.9. Многопоточный веб-сервер с входным и обрабатывающими модулями

Обрабатывающий модуль вначале проверяет кэш на предмет нахождения там нужных файлов. Если они там действительно есть, он обновляет запись, включая в нее указатель на файл. Если искомого файла в кэше нет, обрабатывающий модуль обращается к диску и считывает файл в кэш (при этом, возможно, затирая некоторые хранящиеся там файлы, чтобы освободить место). Считанный с диска файл попадает в кэш и отсылается клиенту.

Преимущество такой схемы заключается в том, что пока один или несколько обрабатывающих модулей заблокированы в ожидании окончания дисковой или сетевой операции (при этом такие модули не потребляют мощности центрального процессора), другие модули могут активно обрабатывать другие запросы. Имея k обрабатывающих модулей, производительность можно повысить в k раз по сравнению с однопоточным сервером. Конечно, когда диск или сеть являются ограничивающим фактором, необходимо наличие нескольких дисков или более быстрой сети, чтобы действительно улучшить однопоточную модель.

Современные веб-серверы выполняют гораздо больше функций, чем просто прием имен путей и отправка файлов. На самом деле, реальная обработка каждого запроса может оказаться достаточно сложной. По этой причине на многих серверах каждый обрабатывающий модуль выполняет последовательности действий. Входной модуль передает каждый входящий запрос первому доступному модулю, который обрабатывает его путем выполнения некоторого подмножества указанных ниже шагов, в зависимости от того, что именно требуется для данного запроса. Эти шаги появляются после того, как было создано TCP-соединение и какой-либо защищенный транспортный механизм (как SSL/TLS, который будет описан в главе 8).

1. Вычисление имени запрашиваемой веб-страницы.
2. Осуществление контроля доступа для веб-страницы.
3. Проверка кэша.

4. Получение запрошенной страницы с диска или запуск программы, создающей ее.
5. Определение оставшейся части ответа (например, типа MIME).
6. Возвращение ответа клиенту.
7. Добавление записи в журнал активности сервера.

Шаг 1 необходим, потому что входящий запрос может и не содержать реального имени файла или программы в виде строкового литерала. Он может содержать встроенные ярлыки, которые необходимо передать. Например, URL может быть вот таким: *http://www.cs.vu.nl/*. Здесь имя файла отсутствует. Этот URL необходимо дополнить неким именем файла по умолчанию. Обычно это *index.html*. Другое общее правило — отображать *~user/* в веб-каталог пользователя. Эти правила могут быть использованы вместе. Таким образом, домашняя страница одного из авторов (AST) будет доступна по адресу:

http://www.sc.vu.nl/~ast/

несмотря на то что реальным именем файла является *index.html* в определенном каталоге. К тому же современные браузеры могут указывать информацию о конфигурации, такую как язык пользователя и ПО браузера по умолчанию (например, итальянский или английский), что позволяет серверу выбирать веб-страницу с маленькими картинками для мобильного устройства на соответствующем языке, если таковая существует. Вообще говоря, расширение имени — задача не такая уж тривиальная, как может показаться, поскольку существует множество соглашений о том, как отображать пути в файловые каталоги и программы.

Шаг 2 состоит в проверке наличия ограничений доступа, ассоциированных со страницей. Не все страницы доступны широкой публике. Определение того, имеет ли право клиент просматривать страницу, может зависеть от личности клиента (например, заданной при помощи имени пользователя и пароля) или расположения клиента в пространстве DNS или IP. Например, право просмотра страницы могут иметь только сотрудники какой-либо компании. То, как это достигается, зависит от устройства сервера. Так, на популярном сервере Apache в каталоге, в котором расположены файлы с ограничением доступа, существует файл под названием *.htaccess*, перечисляющий эти ограничения.

Шаги 3 и 4 подразумевают получение страницы. Правила обработки определяют, можно ли ее получить из кэша. Например, страницы, созданные работающими программами, не всегда могут сохраняться в кэше, так как при каждом запуске они могут выдавать различные результаты. Время от времени должны проверяться даже файлы, чтобы выяснить, не изменилось ли их содержимое (тогда старые версии можно будет удалить из кэша). Если страница требует запуска программы, также возникает проблема с определением параметров программы или входных данных. Эти данные извлекаются из пути или других частей запроса.

На шаге 5 определяются другие части ответа, связанные с содержимым страницы. Например, MIME-тип. Он может следовать из расширения файла, файла конфигурации и, возможно, других источников.

Шаг 6 возвращает страницу по сети. Чтобы повысить производительность, одно и то же TCP-соединение может быть использовано клиентом и сервером для многократного получения различных страниц. Такой тип использования предполагает

создание некоторой логики для отображения запроса на совместно используемое соединение и возвращения каждого ответа таким образом, чтобы он ассоциировался с нужным запросом.

Шаг 7 создает для административных целей запись в системном журнале и сохраняет другую важную статистику. Такие журналы могут быть исследованы на наличие полезной информации о поведении пользователей, например о том, в каком порядке люди посещают страницы на сайте.

Файлы cookie

Использование сети таким образом, который мы только что описали, включает набор независимых запросов страниц. Отсутствует понятия сеанса связи. Браузер клиента посылает запрос на сервер и получает в ответ файл. После этого сервер забывает о том, что он когда-либо видел этого клиента.

Эта модель прекрасно подходит для получения документов, предназначенных для широкой публики, и она хорошо работала в то время, когда Всемирная паутина была только создана. Однако такой вариант не подходит для возвращения различных страниц разным пользователям в зависимости от того, что они уже сделали с сервером. Такое поведение необходимо для многих непрерывных взаимодействий с веб-сайтами. К примеру, на некоторых сайтах (например, сайтах газет) требуется регистрация пользователей; а иногда получение информации с сайта является платным. Возникает вопрос различения запросов от зарегистрированных пользователей и от всех остальных. Вторым примером является электронная коммерция. Как серверу собрать информацию о состоянии корзины пользователя, если тот время от времени пополняет ее содержимое? Третий пример — веб-порталы типа Yahoo!. Пользователям предоставляется возможность индивидуально настраивать вид начальной страницы (например, так, чтобы им сразу показывались последние новости о любимой спортивной команде или курсы ценных бумаг). Однако как сервер сможет корректно отобразить страницу, если он не знает, с каким пользователем имеет дело?

На первый взгляд может показаться, что решение очевидно: сервер может различать пользователей по их IP-адресам. Однако эта идея не работает. Во-первых, многие пользователи работают на компьютерах с разделяемыми ресурсами (например, дома), и с помощью IP-адреса можно идентифицировать лишь компьютер, а не пользователя. Что еще хуже, многие компании используют NAT, поэтому исходящие пакеты от всех клиентов имеют один и тот же IP-адрес. То есть все компьютеры, которые работают через NAT, кажутся серверу одинаковыми. К тому же многие ISP приписывают IP-адреса посетителям при помощи DHCP. IP-адрес со временем может измениться, так что для сервера вы внезапно станете выглядеть так же, как ваш сосед. По всем этим причинам сервер не может использовать IP-адрес, чтобы отслеживать пользователей.

Для решения этой проблемы был предложен сильно раскритикованный метод **cookie-файлов**. Такое название произошло от старинного программистского сленгового словечка. Программа вызывала процедуру и получала взамен нечто, что могло понадобиться впоследствии для выполнения какой-либо задачи. Это «нечто» и называлось cookie. В этом смысле файловый дескриптор в UNIX и дескриптор объектов в Windows

можно рассматривать как cookie. Эти специальные маркеры были впервые применены в браузере Netscape в 1994 году. Сейчас они формализованы в RFC 2109.

Когда пользователь запрашивает страницу, сервер может снабдить свой ответ дополнительной информацией в форме cookie-файла, который представляет собой маленькую именованную строку (до 4 Кб), которую сервер может ассоциировать с браузером. Эта ассоциация не позволяет наверняка определить пользователя, но она является гораздо более точной, чем IP-адрес. Браузеры обычно на некоторое время сохраняют полученные маркеры в каталоге cookie на жестком диске клиента, так что cookie-файлы сохраняются при повторных вызовах браузера, если только пользователь не отключил данную функцию. Итак, cookie — это файлы или строки, а не исполняемые программы. В принципе, в них может содержаться вирус, но поскольку они рассматриваются всего лишь как информационные данные, нет какой-либо официальной возможности для активации вирусов и нанесения ущерба системе. Однако у хакера всегда есть возможность, используя ошибки в браузере, заставить активироваться вирусы, содержащиеся в cookie.

В маркерах cookie может содержаться до пяти полей, как показано в табл. 7.10. Поле *Домен* (*Domain*) содержит имя домена, с которого пришел маркер. Предполагается, что браузеры проверяют тот факт, что серверы не лгут относительно имен доменов. Каждый домен должен хранить не более 20 маркеров, связанных с одним клиентом. Поле *Путь* (*Path*) содержит путь в структуре каталогов на сервере, указывающий те части дерева каталогов, которые могут использовать маркер. Часто в этом поле содержится знак «/», означающий, что доступно дерево целиком.

Поле *Содержимое* (*Content*) имеет вид *имя = значение*. Как *имя*, так и *значение* могут быть совершенно произвольными, на усмотрение сервера. В этом поле хранится основная информация, которую несет в себе маркер.

Поле *Годен до* (*Expires*) указывает срок годности маркера. Если это поле отсутствует, браузер отбрасывает cookie сразу после выхода из программы. Такой маркер называется **непостоянным (nonpersistent cookie)**. Если же указано время и дата, то о таком маркере говорят, что он **постоянный (persistent cookie)**. Он хранится до тех пор, пока не выйдет срок годности. Время, указываемое в поле *Годен до*, — гринвичское. Чтобы удалить cookie с жесткого диска клиента, сервер просто посылает его заново, указывая вышедший срок годности.

Таблица 7.10. Несколько примеров cookie

Домен	Путь	Содержимое	Годен до	Защищенный
toms-casino.com	/	CustomerID=297793521	15-10-10 17:00	Да
jills-store.com	/	Cart=1-00501;1-07031;2-13721	11-1-11 14:22	Нет
aportal.com	/	Prefs=Stk;CSCO+ORCL;Spt:Jets	31-12-20 23:59	Нет
sneaky.com	/	UserID=4627239101	31-12-19 23:59	Нет

Наконец, поле *Защищенный* (*Secure*) может быть установлено для индикации того, что браузер может вернуть его только на сервер, использующий защищенный канал передачи информации, а именно SSL/TLS (который мы опишем в главе 8). Это свой-

ство используется в электронной коммерции, банковском деле и других приложениях, в которых важна защита информации.

Итак, мы узнали о получении маркеров. Как же они используются? Непосредственно перед отправкой браузером запроса на получение страницы на какой-нибудь веб-сайт проверяется каталог с cookie. Ищутся маркеры, пришедшие с того (и только того) домена, на который отправляется запрос. Все найденные маркеры отправляются вместе с запросом. Получив их, сервер может интерпретировать содержащуюся в них информацию так, как ему нужно.

Рассмотрим возможные применения cookie. В табл. 7.10 первый cookie был прислан доменом *toms-casino.com* и используется для идентификации клиента. Если через неделю тот же клиент возвращается на сайт, чтобы просадить очередную сумму денег, браузер отправляет cookie, так что сервер узнает старого знакомого. Вооружившись идентификатором пользователя, сервер может найти запись о нем в базе данных и использовать эту информацию для генерации соответствующей веб-страницы. В зависимости от азартности игрока ему может быть предложен покер, таблица результатов сегодняшних скачек или просто игровой автомат.

Второй cookie-маркер пришел с *jills-store.com*. Сценарий в этом случае заключается в том, что клиент бродит по магазину, выбирая себе покупки. Найдя что-нибудь привлекательное, он щелкает на значке товара. При этом сервер добавляет товар к списку покупок (который поддерживается на сервере), а также создает cookie-маркер, содержащий код заказанного товара, и отсылает cookie клиенту. Клиент продолжает бродить по электронному магазину, щелкая мышью на новых страницах, и в ответ на каждый новый запрос страницы на сервер отсылается cookie. По мере накопления выбранных товаров дополняется информация в cookie. Наконец, когда пользователь щелкает ПЕРЕЙТИ К РАСЧЕТАМ, cookie, содержащий теперь уже полную информацию о покупках, отсылается вместе с запросом на сервер. Таким образом, серверу точно известно, какие товары хочет приобрести клиент.

Третий cookie-маркер прибыл с веб-портала. Когда пользователь щелкает по ссылке на портал, браузер отсылает ему cookie, в котором говорится о том, что надо показать страницу, содержащую котировки акций Cisco и Oracle, а также результаты футбольного матча New York Jets. Так как максимальный размер cookie-файла равен 4 Кб, то остается еще много места для более детальной настройки страницы. Например, в нее можно включить сводку погоды, специальные предложения, заголовки статей в крупных газетах и т. п.

Более спорным использованием cookie-файлов является отслеживание действий пользователя. Операторы веб-сайтов узнают, как пользователи перемещаются по их страницам, и рекламщики строят профили сайтов и реклам, которые просматривал отдельный пользователь. Загвоздка заключается в том, что пользователи обычно не знают, что их поведение отслеживается, даже в случае детализированных профилей и веб-сайтов, которые кажутся несвязанными. Тем не менее **веб-отслеживание (Web-tracking)** является серьезным бизнесом. DoubleClick, который предоставляет и отслеживает рекламные баннеры, признан компанией Alexa, занимающейся вебмониторингом, одним из 100 крупнейших веб-сайтов в мире. Google Analitics, отслеживающий использование сайтов для операторов, используется более чем половиной из крупнейших 100 000 сайтов.

Сервер может легко отследить активность пользователя при помощи файлов cookie. С их помощью можно отслеживать число различных посетителей сайта, узнавать, сколько страниц просмотрел каждый из них, и составлять по этим данным статистику. Когда на сервер приходит первый запрос от пользователя, вместе с ним, разумеется, не высылается никакой маркер. Поэтому сервер отсылает обратно cookie со значением счетчика, равным 1. Последующие просмотры страниц сайта уже будут сопровождаться отсылкой cookie. Всякий раз счетчик будет инкрементироваться и отсылаться пользователю. Таким образом, по счетчикам можно узнать, сколько пользователей покинуло сайт, просмотрев только первую страницу, сколько посетителей просматривают по две страницы и т. д.

Отслеживание навигации пользователей по сайтам является чуть более сложной задачей. Это делается так. Рекламное агенство, скажем, Черный Рекламщик, связывается с крупнейшими веб-сайтами и размещает на них рекламные баннеры продуктов своих клиентов, за что сайту выплачиваются денежные взносы. Вместо того чтобы предоставлять сайту баннер в виде GIF с рекламой, который можно разместить на каждой из страниц, ему дается URL, который следует поместить на всех страницах. Каждый из этих URL содержит уникальный идентификатор в виде пути, например:

```
http://www.sneaky.com/382674902342.gif
```

Когда пользователь впервые посещает страницу *P*, содержащую такую рекламу, браузер, как водится, принимает HTML-файл. Просматривая его, браузер находит ссылку на изображение на *www.sneaky.com*. Разумеется, он отправляет запрос на получение изображения. Вместе с GIF приходит cookie с уникальным идентификатором пользователя, 4627239101 (см. табл. 7.10). Черный Рекламщик отмечает, таким образом, тот факт, что пользователь с таким идентификатором посетил страницу *P*. Это делается очень просто, так как ссылка на запрошенный файл (*382674902342.gif*) существует, на самом деле, только на странице *P*. Конечно, одна и та же реклама может располагаться на тысячах разных страниц, но каждая из них имеет свое имя. При этом за доставку каждого экземпляра рекламная компания может взимать с заказчика небольшую сумму.

Затем, когда пользователь оказывается на другой странице, содержащей баннер от Черного Рекламщика, браузер, скачав HTML-файл с сервера, видит ссылку на изображение с именем, скажем, *http://www.sneaky.com/193654919923.gif*, и запрашивает данный файл. Поскольку с домена *sneaky.com* уже был получен cookie, браузер отсылает его обратно с идентификатором пользователя. Так Черный Рекламщик (ЧР) узнает о том, что пользователь посетил вторую страницу с его рекламой.

Со временем ЧР может составить подробное описание пристрастий пользователя, при этом вовсе не обязательно, чтобы тот щелкал на баннерах. Конечно, остается неизвестным имя пользователя (хотя имеется IP-адрес, и этого может оказаться достаточно для вычисления имени с помощью баз данных). Однако стоит пользователю указать свое имя на одном из сайтов, сотрудничающих с ЧР, как появляется возможность составить и продать целое веб-досье на пользователя. Продажа таких досье оказывается делом настолько прибыльным, что ЧР выгодно сотрудничать с максимально возможным количеством сайтов и собирать как можно больше информации

И если ЧР хочет стать Суперчерным Мегарекламщиком, его объявления не должны выглядеть как обычные классические баннерные ссылки. «Объявление» размером

в один пиксел, сливающееся по цвету с задним фоном страницы (то есть невидимое), будет иметь ровно такой же эффект при слежении за пользователями: браузер будет запрашивать gif-изображение размером 1 × 1 пиксел и отправлять обратно cookie.

Cookie стали центральной точкой дебатов по поводу конфиденциальности в сети из-за описанного выше отслеживания поведения пользователей. Наиболее коварной частью всего бизнеса является то, что многие пользователи не имеют не малейшего представления о том, что идет сбор какой-то информации, и даже могут считать, что они защищены от подобного, так как не щелкают мышью ни на какие баннеры. По этой причине cookie, которые отслеживают поведение пользователей на сайтах, многими воспринимаются как **программы-шпионы (spyware)**. Посмотрите на cookie, которые уже хранятся в вашем браузере. Большинство браузеров покажет эту информацию вместе с текущими настройками конфиденциальности. Кроме непонятных идентификаторов вы можете с удивлением обнаружить имена, адреса и пароли. Будем надеяться, что номеров кредитных карт там не будет, но потенциальные возможности злоупотребления информацией очевидны.

Для самоуспокоения некоторые пользователи настраивают свои браузеры так, чтобы они отвергали любые cookie. Однако это может породить проблемы, так как многие веб-сайты не смогут корректно работать без обмена с пользователями cookie-маркерами. Помимо этого большинство браузеров позволяет пользователям блокировать **cookie «третьей стороны» (third-party cookies)**. То есть те, которые исходят не с сайта, на главной странице которого вы находитесь, например cookie *sneaky.com*, который используется при взаимодействии со страницей *P*, а с совершенно другого веб-сайта. Блокирование этих cookie позволяет избежать отслеживания переходов между сайтами. Также для обеспечения надлежащего контроля за тем, как используются (или не используются) cookie, можно инсталлировать расширения браузеров. Пока продолжаются споры, многие компании разрабатывают варианты политики конфиденциальности, которые определяют то, каким образом они будут делиться информацией, чтобы избежать злоупотреблений. Конечно, такая политика определяет лишь то, как компании формулируют взаимодействие с получаемой информацией. Например, фраза «Мы можем использовать собранную о вас информацию в собственных целях» может означать, что компания может ее продать.

7.3.2. Статичные веб-страницы

Основная идея Всемирной паутины состоит в перемещении веб-страниц от сервера клиенту. Простейшие веб-страницы являются статическими, то есть это просто размещенные на каком-либо сервере файлы, которые при каждом просмотре отображаются одинаковым образом. Однако то, что страницы являются статичными, еще не значит, что при отображении в браузере с ними ничего не происходит. Так, страница, содержащая видео, может быть статичной.

Как уже упоминалось ранее, HTML является родным языком Всемирной паутины, на котором написано большинство страниц. Домашние страницы учителей обычно являются статичными HTML-страницами. Домашние страницы компаний обычно являются динамическими, сконструированными по заказу компаниями, занимающимися дизайном. В этом разделе мы вкратце расскажем о статичных HTML-страницах,

так как они являются основой того, о чем пойдет речь далее. Читатели, уже знакомые с HTML, могут сразу перейти к следующему разделу, где мы описали динамический контент и веб-сервисы.

HTML — язык разметки веб-страниц

HTML (HyperText Markup Language) появился параллельно со Всемирной паутиной. С помощью HTML можно размещать на веб-страницах текст, графику, видео, а также указатели на другие страницы и т. п. Он является языком разметки, то есть языком, описывающим способ форматирования документа. Термин «разметка» (markup) восходит к тем дням, когда литературный редактор с помощью специальной разметки указывал типографу (это такой человек когда-то был), какой шрифт использовать для печати документа. Таким образом, языки разметки содержат подробные команды форматирования. Например, в языке HTML команда `<b>` **означает начало участка текста, печатаемого полужирным шрифтом**, а `</b>` означает конец такого участка. Другими примерами языков разметки, хорошо известными академическим авторам, являются LaTeX и TeX.

Главное преимущество языка разметки перед языком, не имеющим явных команд форматирования, заключается в том, что он отделяет контент от того, каким образом он должен быть представлен. Таким образом, написание браузера является простой задачей: браузер должен понимать и применять содержащиеся в тексте команды разметки к контенту. С помощью встроенных стандартизированных команд разметки в HTML-файлах становится возможным читать и переформатировать любую веб-страницу веб-браузером. Способность изменять форматирование крайне важна, так как должна быть возможность просматривать веб-страницу, созданную на экране с установленным разрешением 1600 × 1200 точек при 24 битах на точку, на экране с разрешением, например, 640 × 320 точек при 8 битах на точку на мобильном телефоне.

Хотя в принципе можно создавать подобные документы с помощью стандартных текстовых редакторов, и многие так и делают, также есть возможность использовать специальные программы редактирования текстов (текстовые процессоры) и HTML-редакторы, берущие на себя большую часть работы (за счет снижения возможностей пользователя детально контролировать получаемый результат).

Простая веб-страница, написанная на HTML, и то, как она выглядит в браузере, показано на рис. 7.10. Страница состоит из заголовка и тела. Вся страница размещается между командами форматирования, называемыми в языке HTML **тегами (tags)**, `<html>` и `</html>`. Впрочем, большинство браузеров правильно отобразят страницу и в отсутствие этих тегов. Как видно из рис. 7.10, *а*, заголовок веб-страницы заключен в скобки тегов `<head>` и `</head>`, а тело располагается между тегами `<body>` и `</body>`. Команды внутри тегов называют **директивами (directives)**. Пусть не все, но большинство HTML-тегов имеют такой формат, то есть `<something>` помечает начало чего-либо, а `</something>` — его конец.

Регистр символов в тегах не имеет значения. Например, `<head>` и `<HEAD>` означают одно и то же, однако у нижнего регистра лучше характеристики совместимости. Формат самого HTML-текста, то есть расположение строк и т. д., не имеет значения. Программы обработки HTML-текстов игнорируют лишние пробелы и переносы строк, так как они все равно форматируют текст так, чтобы он помещался в заданной области

отображения. Соответственно, для того чтобы исходные HTML-документы легче читались, в них можно добавлять произвольное количество знаков табуляции, пробелов и символов переноса строк. И наоборот, для разделения абзацев в тексте в исходный HTML-текст недостаточно вставить пустую строку, так как она просто игнорируется браузером. В этом случае необходимо явное использование специального тега.

```
<html> <head> <title> AMALGAMATED WIDGET, INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page </h1>
<img src="http://www.widget.com/images/logo.gif" ALT="AWI Logo"> <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's</b>
home page. We hope <i> you </i> will find all the information you need here.
<p>Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by email. </p>
<hr> <h2> Product information </h2>
<ul> <li> <a href="http://widget.com/products/big"> Big widgets </a> </li>
<li> <a href="http://widget.com/products/little"> Little widgets </a> </li>
</ul> <h2> Contact information </h2> <ul>
<li> By telephone: 1-800-WIDGETS </li>
<li> By email: info@amalgamated-widget.com </li>
</ul> </body> </html>
```

а

б

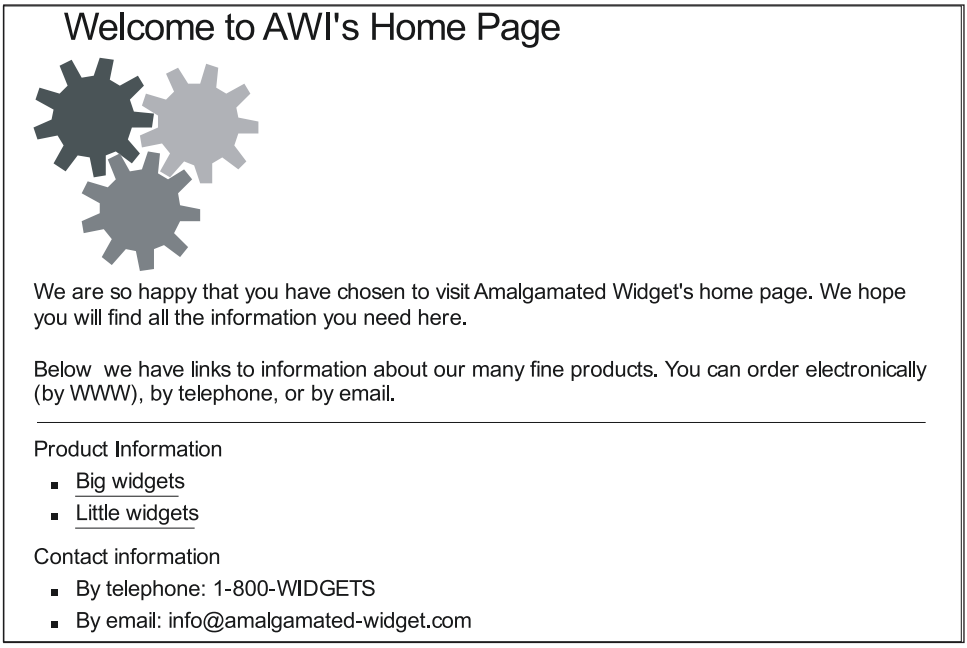


Рис. 7.10. HTML для примера веб-страницы (*а*). Отформатированная страница (*б*)

Некоторые теги могут иметь именованные параметры, называемые **атрибутами (attributes)**. Например, тег `<img>` на рис. 7.10 используется для того, чтобы наравне с текстом расположить на странице изображение. У этого тега есть два атрибута `src` и `alt`. Первый атрибут предоставляет URL для изображения. HTML-стандарт не определяет, какие форматы изображений разрешены. На самом деле, практически все

браузеры поддерживают файлы GIF и JPEG. Браузеры могут поддерживать и другие форматы, но это палка о двух концах. Если пользователь привык к браузеру, который поддерживает, скажем, файлы TIFF, он может разместить их на своих веб-страницах, а потом с удивлением обнаружить, что другие браузеры просто игнорируют все эти замечательные картинки.

При помощи второго атрибута можно задать текст, который будет выводиться на страницу, если не удастся отобразить изображение. Для каждого тега стандарт HTML устанавливает список допустимых атрибутов и их значение. Поскольку все атрибуты являются именованными, их порядок не имеет значения.

Формально при написании HTML-документов должен использоваться набор символов Latin-1 международного стандарта ISO 8859-1, но для пользователей, чьи клавиатуры поддерживают только ASCII-символы, для ввода специальных символов, таких как, например, é, могут использоваться специальные управляющие последовательности символов. Эти последовательности должны начинаться со знака амперсанда и заканчиваться точкой с запятой. Например, ` ` означает пробел, `&egrave;` означает символ è, а `&eacute` — символ é. Так как сами символы <, > и & оказываются зарезервированными, для их отображения в тексте также применяются управляющие последовательности: `<` (less than — знак «меньше»), `>` (greater than — знак «больше») и `&` (ampersand — амперсанд).

Главным элементом заголовка является название страницы, располагающееся между тегами `<title>` и `</title>`. В него можно поместить также некоторую метаинформацию, хотя в нашем примере ее нет. Она не отображается на странице, а используется некоторыми браузерами для того, чтобы помечать окно страницы.

Несколько заголовков используются на рис. 7.10. Каждый из них создается при помощи тега вида `<h`*n*`>`, где *n* — цифра от 1 до 6. `<h1>` является самым важным заголовком, `<h6>` — наименее важным. Как это отобразить на экране, зависит от браузера. Обычно заголовки с меньшими номерами отображаются более крупными шрифтами. Браузер может также выделять различные заголовки различными цветами. Обычно заголовки `<h1>` выводятся на экран крупным полужирным шрифтом и выделяются, по меньшей мере, одной пустой строкой над и под заголовком. В отличие от них заголовки `<h2>` отображаются шрифтом меньшего размера, отступы перед заголовком и после него также уменьшаются.

Теги `<b>` и `<i>` обозначают соответственно полужирный шрифт (boldface) и курсив (italics). При использовании тега `<hr>` отображается разрыв, а также горизонтальная черта на странице.

Тегом `<p>` отмечается начало абзаца. Браузер может это отобразить, например, добавив пустую строку и отступ. Заметим, что многие ленивые HTML-программисты не пользуются тегом `</p>`, обозначающим конец абзаца.

Язык HTML предоставляет несколько механизмов создания списков, включая вложенные списки. Неупорядоченные списки (unordered list), такие как на рис. 7.10, начинаются тегом `<ul>`. Отдельные пункты раньше помечались тегом `<li>`. Тег `<ol>` (ordered list) соответствует упорядоченному списку. При его использовании абзацы, отдельные пункты неупорядоченного списка помечаются знаком «•». Элементы упорядоченных списков автоматически нумеруются браузером.

И наконец, мы подошли к гиперссылкам. Примеры их использования вы можете увидеть на рис. 7.10. Гиперссылки вводятся при помощи пары тегов: <a> (anchor — закладка) и . У этого тега также могут быть различные параметры, самым значимым из которых является href (гиперссылка, URL). Текст, располагающийся между тегами <a> и , отображается на экране. Если этот текст выбирается, браузер открывает страницу, на которую указывает гиперссылка. Между тегами <a> и можно также размещать и другие элементы, например изображение (тег <img>). В этом случае, если пользователь щелкнет на изображении, будет произведен переход по ссылке.

Существует множество других тегов и атрибутов HTML, которые мы не привели в этом простом примере. Так, тег <a> может содержать параметр name, что позволяет создать гиперссылку посреди текста, на которую можно ссылаться из другого места этой же страницы. Это может быть полезным в том случае, если веб-страницы начинаются с оглавления, состоящего из «локальных» гиперссылок. Щелчок мышью на пункте оглавления позволяет быстро переместиться в нужное место страницы. В качестве другого примера можно привести тег <br>. Он заставляет браузер вставить перевод строки.

Вероятно, лучший способ разобраться в тегах — посмотреть на них в действии. Чтобы это сделать, вы можете выбрать любую веб-страницу, открыть ее в вашем браузере в виде HTML и выяснить, как она устроена. В большинстве браузеров есть пункт меню Посмотреть исходный код страницы или что-то в этом духе. При выборе этого пункта вместо отформатированной страницы отображается ее исходный HTML-текст.

Мы вкратце рассказали о тегах, которые существуют с ранних этапов развития Всемирной паутины. HTML продолжает развиваться. В табл. 7.11 отображены некоторые возможности, которые были добавлены в следующих версиях HTML. HTML 1.0 относится к версии HTML, которая использовалась при возникновении Всемирной паутины. Версии 2.0, 3.0 и 4.0 сменяли друг друга достаточно быстро в течение нескольких лет после того, как Всемирная паутина обрела популярность. После HTML 4.0 прошло почти десять лет, прежде чем путь для новой версии, HTML 5.0, был открыт. Так как данный стандарт является основным обновлением, объединяющим способы, которыми браузеры обрабатывают разнообразный контент, HTML 5.0 до сих пор находится в стадии разработки, которая вряд ли закончится раньше 2012 года. Тем не менее основные браузеры уже поддерживают возможности HTML 5.

Изменения, вносимые в версии HTML, связаны с добавлением новых возможностей, которые люди хотели бы видеть, но пока они не вошли в стандарт, использовать данные возможности приходится нестандартным образом (например, при помощи плагинов). Так, например, в первых двух версиях не существовало таблиц, они были добавлены только в HTML 3.0. HTML-таблица состоит из нескольких **строк**, каждая из которых состоит из нескольких **ячеек**, которые могут содержать широкий спектр данных (например, текст, изображения и даже другие таблицы). До введения HTML 3.0 авторам, которым нужна была таблица, приходилось прибегать к особым методам, таким как включение картинки, на которой изображалась таблица.

HTML 4.0 отличается от предыдущих версий некоторыми новыми свойствами. Они включают в себя специальные методы доступа для людей с ограниченными возможностями, внедрение объектов (обобщение тега <img>, позволившее включать в со-

став страниц не только изображения, но и другие объекты), поддержка языков написания сценариев (скриптов), что дало толчок к развитию динамических страниц и т. д.

Таблица 7.11. Некоторые отличия между версиями HTML

Объект	HTML 1.0	HTML 2.0	HTML 3.0	HTML 4.0	HTML 5.0
Гиперссылки	×	×	×	×	×
Изображения	×	×	×	×	×
Списки	×	×	×	×	×
Активные карты и изображения		×	×	×	×
Формы		×	×	×	×
Математические формулы			×	×	×
Панели инструментов			×	×	×
Таблицы			×	×	×
Возможности по обеспечению доступности				×	×
Встраивание объектов				×	×
Таблицы стилей				×	×
Скрипты				×	×
Видео и аудио					×
Векторная графика					×
Отображение XML					×
Фоновые потоки					×
Хранение информации в браузере					×
Область для рисования					×

HTML 5.0 включает множество опций, предназначенных для обработки мультимедиа, которое на сегодняшний день широко используется во Всемирной паутине. Видео и аудио может размещаться на страницах и проигрываться браузером, и это не потребует от пользователя установки плагинов. Вместо использования растровых форматов графических объектов (таких как JPEG и GIF), можно строить рисунки в браузере в виде векторной графики. Также расширились возможности поддержки выполнения скриптов в браузерах, таких как фоновые потоки вычислений и доступ к хранилищу. Все эти возможности помогают поддерживать веб-страницы, которые больше похожи на традиционные приложения с пользовательским интерфейсом, чем на документы. Именно в этом направлении развивается Всемирная паутина.

Ввод информации и формы

Есть одна важная возможность, которую мы еще не обсудили, — ввод информации. Первая версия языка HTML фактически обеспечивала лишь одностороннюю связь. Пользователи могли получать страницы от поставщиков информации, но отправлять информацию обратно было довольно трудно. Достаточно быстро стало очевидным то, что для того, чтобы обеспечить возможности размещения заказов на продукты при помощи веб-страниц, заполнения учетных карточек, поиска по ключевым словам и многого другого, необходим двусторонний трафик.

Отсылка вводимой информации от пользователя серверу (через браузер) требует двух типов поддержки. Во-первых, необходимо, чтобы HTML передавал данные в этом направлении. То, как это происходит, мы опишем в следующем разделе; в этом процессе задействован метод *POST*. Во-вторых, нужно предоставить элементы пользовательского интерфейса, которые собирают и пакуют вводимую информацию. **Формы (forms)** с этой функциональностью были введены в HTML 2.0.

Формы могут содержать кнопки и поля для ввода текста, позволяющие пользователям делать выбор или вводить необходимую информацию, которую затем можно отсылать владельцу страницы. Формы написаны так же, как и другие части HTML, как видно из примера в листинге 7.4. Внешний вид формы, соответствующей данному HTML-тексту, приведен на рис. 7.11. Обратите внимание на то, что формы являются статическим контентом. Они не меняют поведения в зависимости от того, кто их использует. Динамический контент, о котором мы поговорим позднее, предоставляет более сложные способы сбора вводимой информации при помощи пересылки программы, поведение которой может зависеть от среды браузера.

Как и все формы, она заключена между тегами `<form>` и `</form>`. В атрибутах этого тега прописано, что делать с вводимыми данными, в данном случае используется метод *POST* для пересылки данных на заданный URL. Текст, не заключенный в теги, просто отображается. Внутри формы разрешено использование всех обычных тегов (например, `<b>`), чтобы позволить автору страницы контролировать вид формы на экране.

В данной форме для ввода данных используются три типа окон, каждый из которых использует тег `<input>`. У данного тега есть множество параметров для определения размера, особенностей и использования отображаемой области. Самые распространенные формы — это пустые поля, в которые пользователь может ввести текст, флажки и кнопки «*отправить*», которые инициируют передачу данных на сервер.

Окно первого типа — это текстовая область, которая следует за текстом «Имя». Ширина этого окна 46 символов. Предполагается, что пользователь введет здесь свое имя, которое будет храниться в виде текстовой строки в переменной `customer` для последующей обработки. В следующих окнах формы спрашивается адрес заказчика, то есть улица, город, штат и страна. Так как между этими полями не вставляются теги `<p>`, браузер по возможности пытается отобразить их все в одной строке (а не в качестве отдельных параграфов). С точки зрения браузера, этот абзац представляет собой просто шесть отдельных элементов — три строки, перемежающиеся тремя окнами. В следующей строке у пользователя запрашивается номер кредитной карты и срок ее действия. Передавать номера кредитных карт по Интернету следует только в том случае, если приняты все соответствующие меры предосторожности. Более подробно этот аспект будет обсуждаться в главе 8.

Следом за датой истечения срока действия кредитной карты мы обнаруживаем новые для нас элементы управления — переключатели. Они используются, когда требуется выбрать только один вариант из нескольких. Они напоминают кнопки на автомагнитолах, служащие для быстрого доступа к заданным радиостанциям. Переключатели этого типа всегда объединяются в группы. При этом включение одного переключателя автоматически выключает все остальные переключатели этой группы. Внешний вид переключателя зависит от используемого графического интерфейса. В пункте бланка «Размер штуковины» также используется два переключателя. Группы переключателей определяются по значению параметра `name` (имя) тега `<input>`. Специальных скобок из тегов, вроде `<radiobutton>` ... `</radiobutton>`, для определения групп переключателей не предусмотрено.

Параметры `value` указывают, какая кнопка была нажата пользователем. Например, в зависимости от того, какую кредитную карту выберет пользователь для своих расчетов, переменной `cc` будет присвоено значение текстовой строки «mastercard» или «visacard».

Листинг 7.4. HTML-текст для бланка заказа

```
<html>
<head> <title> Бланк заказа клиента AWI </title> </head>
<body>
<h1> Бланк заказа штуковины </h1>
<form action="http://www.widget.com/cgi-bin/order.cgi" method=post>
<p> Имя <input name="customer" size=60> </p>
<p> Адрес <input name="address" size=58> </p>
<p> Город <input name="city" size=21>
Штат <input name="state" size=4>
Страна <input name="country" size=10> </p>
<p> Номер кредитной карты <input name="cardno" size=10>
Срок действия <input name="expires" size=4>
M/C <input name="cc" type=radio value="mastercard">
VISA <input name="cc" type=radio value="visacard"> </p>
<p> Размер штуковины:  большой <input name="product" type=radio value="дорогая">
маленький <input name="product" type=radio value="дешевая">
Доставка  <input name="express" type=checkbox> </p>
<p> <input type=submit value="Отправить заказ"> </p>
Благодарим вас за то, что вы заказали у нас штуковины фирмы AWI. Это правильный выбор!
</form>
</body>
</html>
```

Следом за двумя наборами переключателей в бланке используется элемент управления типа `checkbox` (флажок). Он похож на переключатель предыдущего типа, так как тоже может находиться в одном из двух состояний (установлен/сброшен), но не объединяется в группы и включается и выключается щелчком мыши на нем, независимо от других элементов управления того же типа.

Наконец, мы добираемся до кнопки, имеющей тип `submit` (подтверждение). Строка параметра `value` в данном случае содержит надпись на кнопке. Когда пользователь нажимает кнопку `submit`, браузер упаковывает всю собранную информацию в одну большую строку и отправляет ее на сервер, на URL, прописанный как часть тега `<form>`,

для обработки. Используется простой способ кодировки. Поля с данными разделяются амперсандами (&), а вместо пробелов ставятся знаки +. В нашем примере такая строка, отсылаемая на сервер, может выглядеть так, как показано ниже в листинге 7.5.

Рис. 7.11. Форматированная страница бланка заказа

Листинг 7.5. Возможный ответ браузера серверу, содержащий введенную пользователем информацию.

```
customer=John+Doe&address=100+Main+St.&city=White+Plains&state=NY&country=USA&cardno=12345
67890&expires=6/14&cc=mastercard&product=cheap&express=on
```

Это сообщение отправляется на сервер в виде одной текстовой строки (вы видите несколько строк, а не одну только из-за недостаточной ширины бумажного листа). Сервер сам решает, что ему делать с полученной строкой, вероятнее всего он передаст ее программе, обрабатывающей такие данные. Мы обсудим это позднее.

Есть и другие типы вводимой информации, которые не приведены в этом простом примере. Так, например, существуют типы `password` (пароль) и `textarea` (текстовая область). Окно `password` похоже на окно `text` (текстовый тип, который устанавливается по умолчанию и не нуждается в именовании), однако при наборе текста символы не отображаются на экране. Область `textarea` тоже близка к типу по умолчанию, однако она может содержать несколько строк.

Для отображения длинных списков вариантов, из которых должен быть сделан выбор, используются теги `<select>` и `</select>`. Такие списки часто представляются в виде выпадающего меню. Если задан параметр `multiple` (множественный), то их можно сравнить с флажками, если же он не задан — с переключателями.

Наконец, есть способы обозначить начальные значения (или значения по умолчанию), которые пользователь может менять. Например, если в области `text` задано поле `value`, содержимое отображается таким образом, чтобы пользователь мог его редактировать или стирать.

CSS — каскадные таблицы стилей

Изначальной целью HTML было определение именно *структуры* документа, а вовсе не его *внешнего вида*. Например, строка

```
<h1>Фотографии Натальи</h1>
```

сообщает браузеру о том, что следует выделить заголовок, однако ничего не говорит о его гарнитуре, размере или цвете. Все эти детали реализует браузер по своему усмотрению: у него есть преимущество, состоящее в том, что он знает свойства конкретного монитора (например, сколько на нем точек). Однако дизайнерам веб-страниц в какой-то момент захотелось получить тотальный контроль над видом создаваемых страниц. Тогда были добавлены новые теги, уточняющие, как должен выглядеть документ. Например,

```
<font face="Helvetica" size="24" color="red"> Фотографии Натальи </font>
```

Были добавлены также методы точного позиционирования элементов на экране. Проблема, присущая данному подходу, заключается в том, что такие страницы создавать слишком утомительно. В результате получается огромный HTML-текст, не обладающий свойством переносимости. И хотя страница может замечательно смотреться на браузере у создателя, однако на другом браузере, другой версии того же браузера или просто на экране с другим разрешением она может выглядеть совершеннейшей кашей.

Альтернативным и гораздо более удобным способом является использование таблиц стилей. Таблицы стилей в текстовых редакторах позволяют авторам ассоциировать текст с логическим, а не с физическим стилем, например «первый абзац» вместо «курсива». Внешний вид каждого стиля определен отдельно. Таким образом, если автор решит изменить шрифт первого абзаца с синего курсива кеглем 14 пт на розовый жирный кеглем 18 пт, потребуется изменить лишь одно определение, а не править весь документ.

CSS (Cascading Sryle Sheets — каскадные таблицы стилей) ввели таблицы стилей во Всемирную паутину с появлением HTML 4, однако они стали широко использоваться и поддерживаться в браузерах только в 2000 году. CSS определяет простой язык для описания правил, которым подчиняется внешний вид размеченного тегами контента. Давайте разберем пример. Предположим, что AWI хочет шикарные веб-страницы, текст на которых будет набран темно-синим цветом, шрифтом Arial на кремовом фоне, кегль заголовков первого уровня на 100 % больше, чем основной текст, а второго уровня — на 50 %. CSS определяет эти правила в листинге 7.6.

Листинг 7.6. Пример CSS

```
body{background-color:linen;color:navy;font-family:Arial;}
h1{font-size:200%;}
h2{font-size:150%;}
```

Как вы видите, определения стилей могут быть компактными. Каждая строка указывает элемент, к которому она относится, и задает значения свойств. Свойства элемента применяются по умолчанию ко всем элементам, которые содержатся в HTML. Таким образом, `body` задает стиль абзацев текста в теле документа. Существуют также удобные условные обозначения названий цветов (например, `red` — красный). Любые параметры стилей, которые не определены, заполняются браузером по умолчанию.

Такое поведение делает определения таблицы стилей факультативными; страница будет адекватно отображаться и без них.

Таблицы стилей можно поместить в файл HTML (например, используя тег <style>), но обычно они хранятся в отдельном файле, на который дается ссылка. Например, тег <head> страницы AWI можно изменить так, чтобы она считывала информацию о каскадном стиле из файла *awistyle.css*, как показано в листинге 7.7. В этом примере также определяется тип MIME как *text/css*.

Листинг 7.7. Добавление таблицы стилей CSS

```
<head>
<title>AMALGAMATED WIDGET, INC.</title>
<link rel="stylesheet" type="text/css" href="awistyle.css"/>
</head>
```

У такой стратегии имеются два преимущества. Во-первых, она позволяет применить один набор стилей к нескольким страницам веб-сайта. Таким образом мы получаем единообразный внешний вид страниц, даже если они разрабатывались разными авторами в разное время, кроме того, мы можем изменить внешний вид всего сайта, отредактировав только файл CSS, а не все файлы HTML. Этот метод можно сравнить с подключением файла заголовка директивой #include в программе на языке C. Изменив в нем одно определение макрокоманды, вы измените его во всех программных файлах, в которых подключается этот заголовок. Второе преимущество заключается в том, что загружаемые файлы HTML оказываются гораздо менее громоздкими, так как браузер может загрузить одну копию файла CSS для всех страниц, которые на нее ссылаются. Ему не нужно загружать новые копии всех определений для каждой отдельной веб-страницы.

7.3.3. Динамические веб-страницы и веб-приложения

Статичная модель страницы, которую мы использовали до сих пор, рассматривает страницы как мультимедийные документы, удобно связанные между собой. Эта модель соответствовала поставленным целям в ранние дни развития Всемирной паутины, когда большое количество информации размещалось онлайн. Сегодня же веб в большой мере используется для приложений и сервисов. В качестве примеров можно привести покупку товаров в интернет-магазинах, поиск по каталогам библиотек, изучение карт, чтение и отсылку почты, а также совместную работу с документами нескольких пользователей.

Эти новые типы использования похожи на традиционные приложения (например, программы для работы с почтой и текстовые редакторы). Отличие состоит в том, что эти приложения запускаются в браузере, а пользовательские данные хранятся на серверах в центрах обработки данных Интернета. Они используют веб-протоколы, получают информацию через Интернет, и браузер отображает пользовательский интерфейс. Преимущество такого подхода состоит в том, что пользователю не нужно устанавливать отдельные приложения, и он может получить доступ к своим данным с разных компьютеров, причем данные сохраняются у оператора сервиса. Конечно, немаловажен и тот факт, что эти приложения предоставляются крупными провайдерами

бесплатно. Эта модель является распространенной формой **облачных вычислений (cloud computing)**, при которых вычисления перемещаются с пользовательских компьютеров на совместно используемые кластеры серверов в Интернете.

Веб-страницы больше не могут быть статичными, если они должны работать как приложения. Требуется динамический контент. Например, страница библиотечного каталога должна показывать, какие книги доступны на данный момент, а какие находятся на руках и потому недоступны. Сходным образом у пользователя должна быть возможность взаимодействовать со страницей фондовой биржи, чтобы посмотреть курсы акций на разные периоды времени и вычислить прибыли и потери. Как можно понять по этим примерам, динамический контент может генерироваться программами, запущенными на сервере или в браузере (или и там, и там).

В этом разделе мы рассмотрим каждый из этих случаев по очереди. В общем, ситуация выглядит так, как показано на рис. 7.12. Например, представим себе сервис, работающий с картами, который позволяет пользователю ввести название улицы, после чего предоставляет ему карту местности. Получив запрос, веб-сервер должен использовать программу для создания страницы, которая показывает карту запрашиваемой местности из базы данных улиц и другой географической информации. Это действие показано как шаги 1–3. Запрос (шаг 1) вызывает запуск программы на сервере. Программа опрашивает базу данных и генерирует нужную страницу (шаг 2) и возвращает ее в браузер (шаг 3).

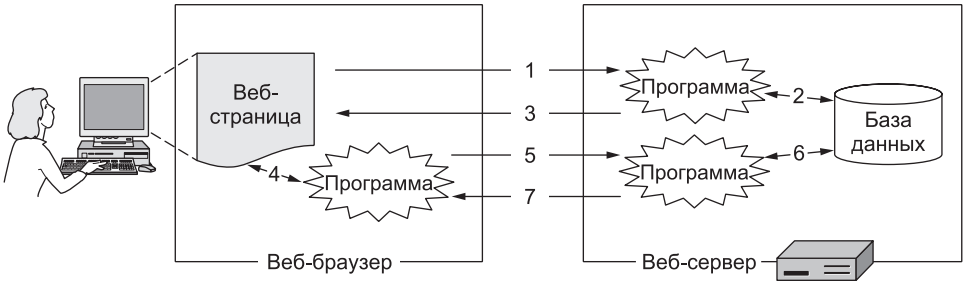


Рис. 7.12. Динамические страницы

Но это не весь динамический контент. Возвращаемая страница сама может содержать программы, которые запускаются в браузере. В нашем примере программа позволяет пользователю находить маршруты и исследовать прилежащие области с разными уровнями детализации. Она обновляет страницу, увеличивая или уменьшая масштаб в соответствии с запросами пользователя (шаг 4). Чтобы провести некоторые операции, программе может понадобиться больше данных с сервера. В этом случае программа отошлет запрос на сервер (шаг 5), который отыщет нужную информацию в базе данных (шаг 6) и вернет ответ (шаг 7). Затем программа продолжит вносить изменения на страницу (шаг 4). Запросы и ответы обрабатываются в фоновом режиме; пользователь может даже не знать о них, так как URL и название страницы обычно не изменяются. Страница с программами, выполняющимися на стороне клиента, может предоставить более удобный интерфейс, чем страница, включающая только программы, выполняющиеся на сервере.

Динамическая генерация содержимого веб-страниц на стороне сервера

Давайте рассмотрим динамическую генерацию веб-страниц на стороне сервера подробнее. Простая ситуация, при которой необходима генерация страниц на стороне сервера, — это использование форм. Рассмотрим ситуацию, при которой пользователь заполняет форму заказа (см. рис. 7.11) и нажимает на кнопку Отправить заказ. Когда пользователь нажимает на кнопку, на сервер, на URL, определенный в форме, отсылается запрос (в этом случае методом *POST* на *http://widget.com/cgi-bin/order.cgi*), содержащий в себе данные, введенные пользователем. Эти данные должны быть переданы программе или скрипту для обработки. Таким образом, URL вызывает запуск определенной программы, в которую данные предоставляются в качестве входной информации. В этом случае обработка включает в себя введение заказа во внутреннюю систему AWI, обновление записей клиента и списание денег с кредитной карты. Страница, которая возвращается в ответ на этот запрос, зависит от того, что произойдет в процессе обработки. Результат не фиксирован, как в случае со статичными страницами. Если заказ успешно обрабатывается, возвращаемая страница может содержать дату доставки товара. Если запрос не был успешно обработан, возвращаемая страница может гласить, что запрашиваемых товаров нет в наличии или по какой-то причине не была принята кредитная карта.

То, как именно сервер запускает программу вместо поиска файла, зависит от устройства веб-сервера. Это не определяется самими веб-протоколами. Именно поэтому интерфейс может быть разработан в соответствии с требованиями компании-собственника сайта. Браузеру не нужно знать детали. И коли уж мы говорим о браузере, стоит отметить, что он просто создает запрос и получает страницу.

Тем не менее для веб-серверов были разработаны стандартные API, чтобы запускать программы. Существование этих интерфейсов позволяет разработчикам тратить меньше усилий на расширение различных серверов за счет веб-приложений. Мы кратко рассмотрим два API, чтобы вы получили о них некоторое представление.

API является методом обработки запросов динамических страниц. Он был доступен с момента возникновения Всемирной паутины. Он называется **CGI (Common Gateway Interface — общий шлюзовой интерфейс)** и определен в RFC 3875. CGI предоставляет интерфейс, позволяющий веб-серверам общаться с прикладными программами и скриптами, которые могут получать данные (например, из формы) и в ответ генерировать HTML-страницы. Эти программы могут быть написаны на любом выбранном разработчиком языке, обычно с использованием скриптов для простоты разработки. Выберите Python, Ruby, Perl или другой язык, который вам по нраву.

Существует договоренность, в соответствии с которой программы, запускаемые через CGI, должны размещаться в каталоге CGI-BIN, который виден в URL. Сервер отображает запрос в этот каталог на имя программы и запускает программу как отдельный процесс. Он предоставляет программе любые данные, отосланные с запросом, как входные. На выходе программы получается веб-страница, передаваемая в браузер.

В нашем примере программа *order.cgi* вызывается с данными, введенными в форму, как показано в листинге 7.5. Она проанализирует параметры и обработает заказ.

Полезной представляется договоренность о том, что программа вернет HTML для формы заказа, если эта форма не была заполнена. Таким образом, программа неизбежно будет знать, в каком виде представлена форма.

Второй API, о котором мы поговорим, серьезно отличается от уже описанного. Этот способ заключается во внедрении небольших скриптов в HTML-страницы. Они выполняются на сервере, в их задачу входит генерирование страницы. Популярным инструментом для написания таких скриптов является **PHP (PHP:Hypertext Preprocessor — PHP:Гипертекстовый препроцессор)**. При его использовании требуется, чтобы сервер понимал PHP (точно так же, как браузер должен понимать CSS, чтобы интерпретировать страницы, написанные с применением таблиц стилей). Обычно серверы определяют веб-страницы, написанные на PHP, по расширению *php*, а не *htm* или *html*.

PHP проще использовать, чем CGI. Пример обработки формы с помощью PHP показан в листинге 7.8, *а*. В верхней части листинга мы видим обычную HTML-страницу с простой формой. На этот раз тег `<form>` указывает на то, что *action.php* должен быть запущен для обработки параметров после нажатия кнопки подтверждения. Форма в этом примере состоит из двух текстовых полей ввода, в одном из которых запрашивается имя пользователя, а в другом — его возраст. По окончании работы пользователя с формой на сервер отсылается стандартная строка, пример которой мы уже видели ранее. Эта строка обрабатывается, из нее извлекаются значения переменных `name` и `age`. Затем начинает свою работу скрипт *action.php*, показанный в листинге 7.8, *б*. Он генерирует ответ. Работа скрипта заключается в исполнении php-команд. Если пользователь предоставил данные «Барбара» и «24», ему будет прислан HTML-файл, код которого показан в листинге 7.8, *в*. Как видите, обработка форм с помощью PHP производится элементарно.

Листинг 7.8. Веб-страница с формой (*а*); PHP-скрипт для обработки формы (*б*); результат работы PHP-скрипта при исходных данных «Барбара» и «24» соответственно (*в*)

(а)

```
<html>
<body>
<form action="action.php" method="post">
<p> Введите свое имя: <input type="text" name="name"> </p>
<p> Введите свой возраст: <input type="text" name="age"> </p>
<input type="submit" value="Подтверждение">
</form>
</body>
</html>
```

(б)

```
<html>
<body>
<h1> Ответ: </h1>
Привет, <?php echo $name; ?>!
Предсказываю: в следующем году тебе будет <?php echo $age+1; ?>
</body>
</html>
```

продолжение ⇩

Листинг 7.8 (*продолжение*)

```
(в)
<html>
<body>
<h1> Ответ: </h1>
Привет, Барбара!
Предсказываю: в следующем году тебе будет 25
</html>
</body>
```

Несмотря на простоту использования, PHP — это мощный язык программирования для взаимодействия со Всемирной паутиной и серверными базами данных. В PHP есть переменные, строки, массивы и большинство управляющих структур, присущих языку C, однако ввод/вывод гораздо мощнее, чем обычный *printf*. PHP имеет открытый исходный код, распространяется бесплатно и широко используется. PHP был разработан специально для сервера Apache, который также обладает открытым исходным кодом и является самым распространенным веб-сервером в мире. Более подробную информацию по PHP можно найти в (Valade, 2009).

Итак, мы знаем уже два различных способа генерации динамических HTML-страниц: с помощью CGI-скриптов и внедрения PHP. Есть и еще несколько методов на выбор. Так, **JSP (JavaServer Pages — Страницы сервера Java)** в целом схож с PHP и отличается только тем, что динамическая часть программируется на языке Java. Файлы страниц, написанных с помощью JSP, имеют одноименное расширение: *.jsp*. **ASP.NET (Active Server Pages .NET — активные серверные страницы .NET)** — это ответ Microsoft на PHP и JSP. Здесь для генерации динамического контента используются программы, написанные в собственной среде разработки сетевых приложений .NET, созданной Microsoft. Соответственно, файлы страниц, написанных с использованием этого метода, имеют расширение *.aspx*. Вопрос выбора между этими тремя техниками в основном политический (открытый исходный код против *Microsoft*). С точки зрения технологий все эти методы вполне сравнимы по возможностям.

Создание динамических веб-страниц на стороне клиента

Скрипты CGI и PHP решают вопросы обработки вводимых данных и взаимодействия с базами данных, расположенными на сервере. Они могут принимать входящую информацию из форм, осуществлять поиск по одной или нескольким базам данных и в качестве результата генерировать HTML-страницы. Но ни один из этих методов не позволяет напрямую взаимодействовать с пользователем, например реагировать на движения мышкой. Для этих целей необходимы скрипты, внедренные в HTML-страницы и выполняющиеся не на серверной, а на клиентской машине. Начиная с HTML 4.0, появилась возможность включать скрипты такого типа с помощью тега `<script>`. Технологии, которые использовались для создания этих интерактивных веб-страниц, часто ошибочно называют **динамическим HTML (dynamic HTML)**.

Наиболее популярный язык написания сценариев для клиентской стороны — это **JavaScript**. Его мы вкратце и рассмотрим ниже. Несмотря на схожесть названий, JavaScript практически не похож на язык программирования Java. Как и другие языки написания скриптов, он очень высокоуровневый. Так, одной строкой JavaScript можно создать диалоговое окно, войти в цикл ожидания пользовательского ввода и сохранить

полученную строку в переменной. Столь высокий уровень языка идеально подходит для разработки интерактивных веб-страниц. С другой стороны, тот факт, что JavaScript не стандартизован и мутирует быстрее, чем мушка-дрозофила в рентгеновском луче, сильно усложняет написание программ, независимых от конкретной платформы. Надо, впрочем, надеяться, что рано или поздно этот язык дойдет до более или менее устойчивого состояния.

Пример программы на JavaScript показан в листинге 7.9. Как и в листинге 7.8, *а*, программа создает форму с запросом имени и возраста пользователя и гениальным образом предсказывает, исходя из этих данных, каков будет возраст человека в следующем году. Тело скрипта почти такое же, как и в примере PHP. Основная разница состоит в объявлении кнопки Подтверждение и определении присваивания в этом объявлении. Оператор присваивания сообщает браузеру о том, что в случае нажатия кнопки необходимо запустить скрипт `response` и передать ему форму в качестве параметра.

Совершенно по-новому здесь объявляется функция `response`. Объявление находится в заголовке HTML-файла, который обычно хранит информацию о заголовках, цвете фона и т. п. Функция извлекает из формы значение поля `name` и сохраняет его в виде строки в переменной `person`. Также извлекается значение поля `age`. Оно приводится к целочисленному типу с помощью функции `eval`, затем к значению добавляется 1, и результат сохраняется в `years`. После этого документ открывается для записи, в него записываются четыре строки (для этого используется метод `writeln`), и документ закрывается. Документ представляет собой HTML-страницу, как видно по многочисленным тегам HTML. Браузер выводит готовый документ на экран.

Листинг 7.9. Применение JavaScript для обработки формы

```
<html>
<head>
<script language="javascript" type="text/javascript">
function response(test_form) {
    var person = test_form.name.value;
    var years = eval(test_form.age.value) + 1;
    document.open()
    document.writeln("<html> <body>");
    document.writeln("Привет, " + person + "!<br>");
document.writeln("Предсказываю: в следующем году тебе будет " + years +".");
    document.writeln("</body> </html>");
    document.close();
}
</script>
</head>

<body>
<form>
Введите свое имя: <input type="text" name="name">
<p>
Введите свой возраст: <input type="text" name="age">
<p>
<input type="button" value="Подтверждение" onclick="response(this.form)">
</form>
</body>
</html>
```

Важно понимать, что, несмотря на то что PHP и JavaScript схожи в том, что они оба размещают код в файлах HTML, обработка идет совершенно разными путями. В примере PHP в листинге 7.8, после того как пользователь нажимает кнопку Подтверждение, браузер собирает всю введенную информацию в одну длинную строку и отправляет ее на сервер, в качестве запроса PHP-страницы. Сервер загружает PHP-файл и запускает PHP-скрипт, который размещен там для того, чтобы создать новую HTML-страницу, отсылаемую браузеру для отображения. Браузер не знает о том, что она была сгенерирована программой. Обработка показана как шаги 1–4 на рис. 7.13, *а*.

В примере JavaScript (см. листинг 7.9) после нажатия кнопки Подтверждение браузер сам выполняет действия функции JavaScript, содержащейся на странице. Вся работа производится локально, внутри браузера. С сервером никакого взаимодействия не осуществляется. Это показано как шаги 1 и 2 на рис. 7.13, *б*. Как следствие, результат появляется практически мгновенно, тогда как при использовании PHP задержка прибытия страницы с результатом может составлять несколько секунд.

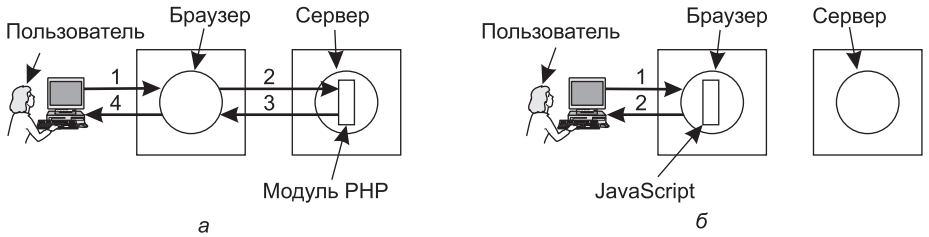


Рис. 7.13. PHP-скрипт на стороне сервера (*а*); сценарий на JavaScript на стороне клиента (*б*)

Эти отличия вовсе не означают, что JavaScript «лучше», чем PHP. Просто у них различные сферы применения. PHP (а с ним неявно и JSP, и ASP) применяется тогда, когда необходимо взаимодействовать с удаленной базой данных. JavaScript (и другие языки со стороны клиента, о которых мы поговорим, такие как VBScript) используют тогда, когда требуется взаимодействие с пользователем в пределах его компьютера. Разумеется, возможно одновременное использование PHP и JavaScript. Мы вкратце расскажем об этом.

JavaScript — это не единственный инструмент для создания веб-страниц с высокой степенью интерактивности. На платформе Windows альтернативой ему служит **VBScript**, базирующийся на Visual Basic. Еще один метод, который можно использовать на разных платформах, связан с использованием **апплетов (applets)**. Это небольшие программы на Java, скомпилированные в машинные инструкции виртуального компьютера, называемого **JVM (Java Virtual Machine — виртуальная машина Java)**. Апплеты могут внедряться в HTML-страницы (между тегами `<applet>` и `</applet>`) и интерпретироваться JVM-совместимыми браузерами. Поскольку перед выполнением Java-апплеты проходят стадию интерпретации, интерпретатор может помочь избежать выполнения «нехороших действий». По крайней мере, теоретически такая возможность существует. На практике создатели апплетов обнаружили почти бесконечный поток ошибок в библиотеках ввода/вывода Java.

Ответ корпорации Microsoft на Java-апплеты фирмы Sun состоял в подключении к веб-страницам **управляющих элементов ActiveX (ActiveX controls)** — программ,

скомпилированных для машинного языка процессора x86 и выполняемых на аппаратном уровне. Это свойство делает их значительно более быстрыми и гибкими, нежели интерпретируемые Java-апплеты, поскольку они могут делать все то же самое, что и обычная программа. Когда Internet Explorer видит на странице управляющий элемент ActiveX, он загружает его, идентифицирует и исполняет. Однако загрузка инородных программ может породить проблемы защиты информации, к которым мы обратимся в главе 8.

Поскольку почти все браузеры могут интерпретировать программы как на Java, так и на JavaScript, разработчик, желающий создать страницу с высокой степенью интерактивности, может выбирать по крайней мере из двух этих технологий. А если вопрос платформонезависимости значения для него не имеет, то к ним добавляется еще и ActiveX. Общее правило таково:

- JavaScript обеспечивает простоту программирования.
- Java-апплеты обеспечивают быстроту выполнения.
- Управляющие элементы ActiveX обгоняют по скорости выполнения все остальные технологии.

Во всех браузерах реализована строго одна и та же версия JVM, а вот найти два браузера с одинаковой реализацией JavaScript почти невозможно. Поэтому Java-апплеты легче переносятся с платформы на платформу, чем JavaScript. Существует множество увесистых (часто более 1000 страниц) книг, посвященных JavaScript. Например, можете посмотреть (Flanagan, 2010).

AJAX — асинхронный JavaScript и XML

Сложным веб-приложениям требуются удобные пользовательские интерфейсы и простой доступ к данным, хранящимся на удаленных веб-серверах. Написание скриптов со стороны клиента (например, при помощи JavaScript) и со стороны сервера (например, при помощи PHP) — это основные технологии, которые могут предоставить решение этих задач. Эти технологии обычно используются вместе с несколькими другими в комбинации под названием **AJAX (Asynchronous JavaScript and XML — асинхронный JavaScript и XML)**. Многие полнофункциональные веб-приложения от Google, такие как Gmail, Maps и Docs, написаны на AJAX.

AJAX несколько странен, так как это не язык. Это набор технологий, которые работают совместно, чтобы создать веб-приложения, которые были бы такими же удобными и функциональными, как традиционные настольные прикладные системы. Это следующие технологии:

1. HTML и CSS, чтобы представлять информацию в виде страниц.
2. DOM (Document Object Model — объектная модель документов), чтобы менять части страниц при просмотре.
3. XML (eXtensible Markup Language — расширяемый язык разметки), чтобы программам обмениваться данными приложений с сервером.
4. Асинхронный способ, при помощи которого программы отсылают и получают XML-данные.
5. JavaScript — язык, который связывает все эти технологии.

Так как все это представляет из себя некоторый набор, мы рассмотрим каждую из частей, чтобы оценить ее вклад в создание веб-приложений. Мы уже рассмотрели HTML и CSS. Они являются стандартами описания контента и того, как он должен отображаться. Любая программа, которая может создавать HTML и CSS, может использовать веб-браузер, как средство отображения.

DOM — это представление HTML-страницы, доступное программам. Это представление имеет структуру дерева, отражающего структуру HTML-элементов. Например, DOM-дерево HTML-текста, приведенного в листинге 7.8, *a*, дано на рис. 7.14. В корне находится элемент *html*, который представляет весь блок HTML. Этот элемент является родительским по отношению к элементу *body*, который, в свою очередь, является родительским для элемента *form*. У элемента *form* есть два атрибута, которые вы можете увидеть справа, один из них относится к методу формы (*POST*) и один предназначен для действия, которое нужно провести с формой (URL-запрос). У этого элемента есть три дочерних, включающих два тега абзацев и один тег для ввода информации, которая содержится в форме. В нижней части расположены листья, являющиеся либо элементами, либо константами, такими как текстовые строки. Значение модели DOM состоит в том, что она предоставляет программам простой способ менять части страницы. Таким образом, не возникает необходимости полностью переписывать страницу. Заменяется только тот узел, в который вносятся изменения. Когда они внесены, браузер обновит соответствующую часть страницы. Например, если в DOM была изменена часть страницы, содержащая изображение, браузер обновит это изображение, оставив остальные части страницы неизменными. Мы уже видели, как работает DOM, когда пример JavaScript из листинга 7.9 добавлял строки к элементу *document*, чтобы вызвать появление новых строк в нижней части окна браузера. DOM — это отличный способ создания страниц, которые могут меняться.

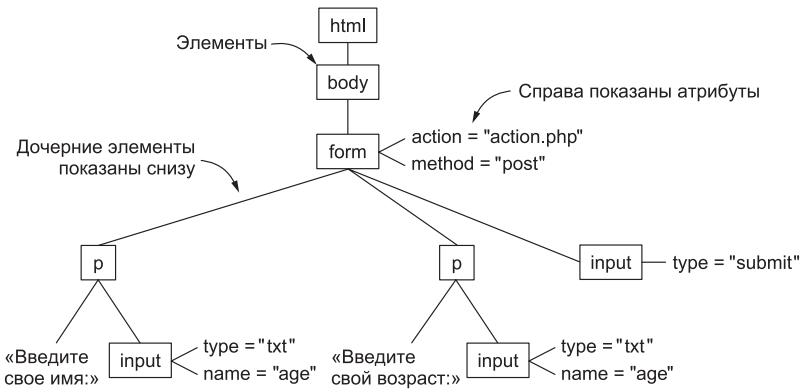


Рис. 7.14. Дерево DOM для HTML из листинга 7.8, *a*

Третья технология, **XML**, — это язык, предназначенный для описания структурированного контента. HTML мешает контент с форматированием, так как он связан со способом представления информации. Однако по мере того, как веб-приложения становятся более распространенными, нарастает потребность разделить структурированный контент и его представление. Например, рассмотрим программу, которая ищет

в сети лучшую цену на какую-либо книгу. Когда веб-страницы написаны на HTML, программе очень трудно выяснить, где указаны название и цена.

По этой причине консорциум WWW (W3C) разработал XML (Bray и др. 2006), чтобы веб-контент можно было структурировать для автоматической обработки. В отличие от HTML для XML не существует четко определенных тегов. Каждый пользователь может создавать свои. Простой пример XML-документа приведен в листинге 7.10. Он определяет структуру под названием book_list, являющуюся списком книг. У каждой книги есть 3 поля: название, автор и год издания. Эти структуры очень просты. В структурах могут повторяться поля (например, в случае коллектива авторов), могут быть дополнительные поля (например, URL аудиокниги) и альтернативные поля (например, URL книжного магазина, если книга есть в наличии, или URL сайта аукциона, если она распродана).

Листинг 7.10. Простой документ XML

```
<?xml version="1.0" ?>
<book_list>
<book>
  <title> Поведение человека и принцип экономии усилий </title>
  <author> Джордж Зипф </author>
  <year> 1949 </year>
</book>
<book>
  <title> Математическая теория коммуникаций </title>
  <author> Клод Э. Шенон </author>
  <author> Уоррен Вивер </author>
<year> 1949 </year>
</book>
<book>
  <title> Тысяча девятьсот восемьдесят четыре </title>
  <author> Джорж Оруэл </author>
  <year> 1949 </year>
</book>
</book_list>
```

В этом примере каждое из трех полей является неделимой сущностью, но теоретически поля можно делить и дальше. Например, чтобы хорошо контролировать поиск и форматирование, поле «автор» может быть оформлено следующим образом:

```
<author>
<first_name> Джордж  </first_name>
<last_name> Зипф </last_name>
</author>
```

Каждое поле может быть подразделено на подполя (а подполя на подподполя и т. д.) произвольное количество раз.

Файл, проиллюстрированный листингом 7.10, просто определяет список из трех книг. Он хорошо подходит для передачи информации между программами, работающими в браузерах и на серверах, но ничего не говорит о том, каким образом документ должен быть представлен в виде веб-страницы. Чтобы сделать это, программа, которая собирает информацию и решает, что 1949 год был весьма урожаен на книги,

может выдать HTML, в котором названия выделены курсивом. Кроме того, язык под названием **XSLT (eXtensible Stylesheet Language Transformations — стилевые трансформации расширяемого языка разметки)** может использоваться для того, чтобы определить, каким образом трансформировать XML в HTML. XSLT похож на CSS, но у него гораздо больше возможностей. Мы, пожалуй, избавим вас от деталей.

Еще одно преимущество представления данных в виде XML, а не HTML состоит в том, что программам проще их анализировать. На HTML изначально писали вручную (да и сейчас не многое изменилось), так что часто он выглядит несколько сырым. Иногда закрывающие теги, такие как </p>, опускаются. У других, таких как <br>, нет соответствующих закрывающих тегов. Кроме того, другие теги могут размещаться не там, где следует, а положение имен тегов и атрибутов может варьироваться. Большинству браузеров приходится нелегко, когда они пытаются разобраться в таком коде. У XML гораздо более строгая структура. Имена тегов и атрибутов всегда пишутся в нижнем регистре, теги всегда должны закрываться в порядке, обратном порядку их появления (или же должно быть строго определено, что определенному тегу не нужен соответствующий закрывающий тег), а значения атрибутов должны быть заключены в кавычки. Эта точность определений делает разбор проще, а результат получается однозначный.

HTML даже был определен в терминах XML. Этот подход называется **XHTML (eXtended HyperText Markup Language — расширенный язык гипертекстовой разметки)**. В общем-то это просто версия HTML с очень жесткими требованиями. Страницы XHTML должны строго соответствовать правилам XML, иначе они не принимаются браузером. Так что с низкокачественными веб-страницами и несовместимости страниц с браузерами покончено. Так же как в случае с XML, целью было создание страниц, которые бы легче обрабатывались программами (в данном случае веб-приложениями). Хотя XHTML существовал с 1998 года, он долго не мог завоевать популярность. Те, кто пользовался HTML, не понимали, зачем им нужен XHTML, и он долго не поддерживался браузерами. Сегодня HTML 5.0 определен таким образом, что страница может быть написана либо на HTML, либо на XHTML, чтобы переход на новый стандарт не был таким резким. В итоге XHTML должен заменить HTML, но пройдет еще много времени, прежде чем этот переход будет завершен.

XML также обрел популярность, как язык коммуникации между программами. Когда это взаимодействие проводится при помощи протокола HTTP (описанного в следующем разделе), оно называется веб-сервисом (Web service). Стоит обратить внимание на **SOAP (Simple Object Access Protocol — простой протокол доступа к объектам)**, который является способом реализации веб-сервисов, выполняющих вызов удаленных процедур между программами, независимо от их языка и платформы. Клиент просто создает запрос в форме XML-сообщения и отсылает его на сервер, используя протокол http. Сервер отсылает ответ в такой же форме. Таким образом приложения на различных платформах могут работать вместе.

Возвращаясь к AJAX, мы лишь подчеркиваем, что XML является полезным форматом для обмена данными между программами, работающими в браузере и на сервере. Однако, чтобы создать быстро откликающийся интерфейс в браузере при отсылке или получении данных, у скриптов должна быть возможность осуществлять асинхронные операции ввода/вывода, которые не блокируют дисплей при ожидании ответа на

запрос. Например, рассмотрим вариант с картой, которую можно прокручивать в браузере. Когда скрипт получает сообщение о том, что пользователь начал прокручивать карту, он может запросить больше данных о карте с сервера, если просматриваемый кусок близок к границе уже полученной части карты. Интерфейс не должен застывать, пока получаются эти данные, иначе ему не светит получить место под солнцем. Вместо этого прокрутка должна продолжаться без помех. Когда данные будут получены, скрипт получает об этом сообщение и может их использовать. Если все идет по плану, новые данные о карте оказываются получены до того, как в них возникает необходимость. В современных браузерах имеется поддержка такой модели коммуникации.

Последняя часть головоломки — это скриптовый язык, который собирает AJAX в единое целое, предоставляя доступ к описанным выше технологиям. В большинстве случаев этот язык — JavaScript, но существуют и такие альтернативы, как VBScript. Мы приводили простой пример JavaScript ранее. Но пусть вас не обманывает эта кажущаяся простота. У JavaScript есть множество индивидуальных особенностей, но это полноценный язык программирования со всеми возможностями C и Java. В нем есть переменные, строки, массивы, объекты, функции и все обычные управляющие структуры. У него также есть интерфейсы, связанные с особенностями браузера и веб-страницы. JavaScript может отслеживать передвижение мышки по объектам на экране, что позволяет легко создать внезапно выпадающее меню и делает страницы более живыми. Он может использовать DOM, чтобы получить доступ к страницам, манипулировать HTML и XML и проводить асинхронные HTTP-коммуникации.

Перед тем как оставить вопрос динамических страниц, позвольте нам вкратце обобщить технологии, о которых мы успели поговорить, проиллюстрировав их отдельным примером. Целые веб-страницы могут быть созданы при помощи различных скриптов, размещенных на сервере. Скрипты могут быть написаны на языках серверных расширений, таких как PHP, JSP или ASP.NET, или запускать отдельные процессы CGI и таким образом быть написаны на любом языке. Эти возможности показаны на рис. 7.15.

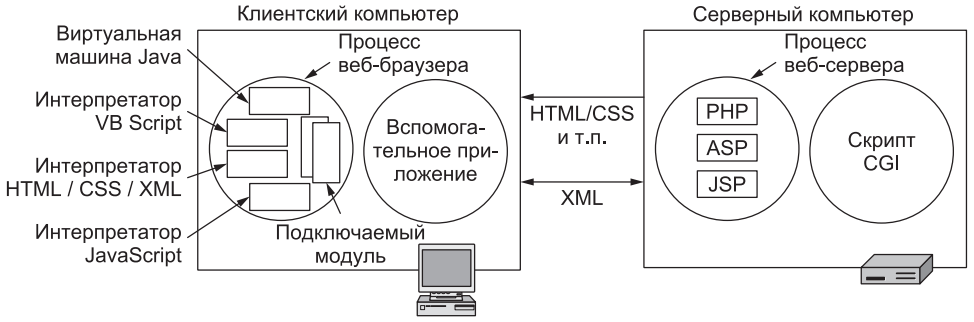


Рис. 7.15. Различные технологии, используемые для создания динамических страниц.

Когда браузер получает веб-страницы, они обрабатываются как нормальные страницы на HTML, CSS и других типах MIME и просто отображаются. Плагины, которые запускаются в браузере, и вспомогательные приложения, запускаемые вне браузера, могут быть инсталлированы, чтобы расширить типы MIME, которые поддерживаются браузером.

Создание динамического контента также возможно на стороне клиента. Программы, которые размещены на веб-страницх, могут быть написаны на JavaScript, VBScript, Java и других языках. Эти программы могут проводить произвольные вычисления и обновлять дисплей. При помощи AJAX программы на веб-страницах могут асинхронно обмениваться XML-файлами и другими видами данных с сервером. Эта модель поддерживает различные веб-приложения, которые выглядят так же, как и традиционные приложения. Отличие лишь в том, что они работают в браузере и получают информацию, хранящуюся на серверах в Интернете.

7.3.4. HTTP — протокол передачи гипертекста

Теперь, когда мы примерно понимаем, что такое веб-контент и веб-приложения, пора посмотреть на протокол, который используется для передачи всей этой информации от веб-сервера к клиенту и обратно. Это **HTTP (HyperText Transfer Protocol — протокол передачи гипертекста)**, определенный в RFC 2616.

HTTP — простой протокол, работающий по принципу запрос-ответ, который обычно запускается через TCP. Он определяет, какие сообщения клиент может отсылать на серверы и какие получать ответы. Заголовки запросов и ответов, так же как в SMTP, даны в ASCII. Их содержимое задано в формате, похожем на MIME, так же как и в SMTP. Эта простая модель частично была причиной успеха Всемирной паутины на раннем этапе, так как повлекла ее быстрое развитие и распространение.

В этом разделе мы поговорим о наиболее важных свойствах HTTP в том виде, в котором он используется сегодня. Однако, прежде чем переходить к подробному рассказу, мы отметим, что способ его использования в Интернете эволюционирует. HTTP — это протокол прикладного уровня, так как он работает поверх TCP и близко ассоциируется с веб. Именно поэтому мы решили поговорить о нем в этой главе. Однако в другом смысле HTTP становится больше похож на транспортный протокол, который предоставляет способ передачи контента из одной сети в другую. Эти процессы не всегда идут в рамках общения веб-браузера с веб-сервером. Медиаплеер может использовать HTTP, чтобы обратиться к серверу и получить информацию об альбоме. Антивирус может использовать HTTP для того, чтобы загрузить последние обновления. Разработчики — для получения файлов по какому-либо проекту. Бытовая электроника, такая как рамки для цифровых фотографий, часто используют HTTP-сервер как интерфейс, связывающий их с внешним миром. Коммуникации между компьютерами все чаще идут при помощи HTTP. Например, сервер авиакомпании может использовать SOAP (XML RPC через HTTP) для связи с сервером аренды машин и таким образом резервировать автомобили в качестве услуги, предоставляемой в пакете при покупке тура. Вероятно, развитие в этом направлении продолжится наряду с расширением использования HTTP.

Соединения

Обычный способ взаимодействия браузера с сервером заключается в установке TCP-соединения с портом 80 сервера, хотя формально эта процедура не является обязательной. Ценность использования TCP в том, что ни браузерам, ни серверам не приходится

беспокоиться о том, что делать со слишком длинными сообщениями, надежностью и контролем перегрузки. Все это обеспечивается протоколом TCP.

Ранее, в HTTP 1.0 после установки соединения посылался один запрос, на который приходил один ответ. После этого TCP-соединение разрывалось. В то время типичная веб-страница целиком состояла из HTML-текста, и такой способ взаимодействия был адекватным. Однако вскоре средняя веб-страница стала изобиловать большим количеством ссылок на различные значки и другие украшения. Очевидно, что установка TCP-соединения для передачи одного значка оказалась нерациональной и слишком дорогой.

Это соображение привело к созданию протокола HTTP 1.1, который поддерживал **постоянные соединения (persistent connection)**. Это означало, что появилась возможность установки TCP-соединения, отправки запроса, получения ответа, а затем передачи и приема дополнительных запросов и ответов. Эта стратегия называется **повторным использованием соединения (connection reuse)**. Таким образом, снизились накладные расходы, возникавшие при постоянных установках и разрывах соединения. Стало возможным также конвейеризировать запросы, то есть отправлять запрос 2 еще до прибытия ответа на запрос 1.

Разница в производительности между этими тремя случаями показана на рис. 7.16. В части *а* мы видим три запроса, отсылаемых один за другим, при этом для каждого устанавливается отдельное соединение. Предположим, что это представление страницы с двумя размещенными на ней картинками, находящимися на одном сервере. URL изображений определяется при получении главной страницы, так что они получаются позднее ее. Сегодня на обычной странице присутствует около 40 других объектов, которые должны быть получены для ее отображения. Но это сделает наш рисунок огромным. Так что в нашем примере мы используем всего два размещенных объекта.

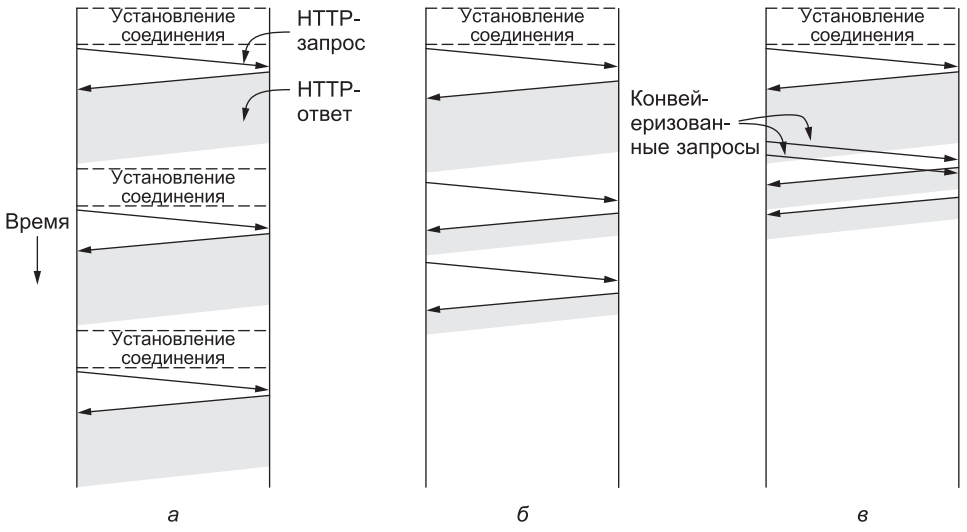


Рис. 7.16. HTTP: *а* — множественные соединения и последовательные запросы; *б* — постоянное соединение и последовательные запросы; *в* — постоянное соединение и конвейеризованные запросы

На рис. 7.16, *б* страница получена при помощи постоянного соединения. То есть TCP-соединение открывается с самого начала, затем посылаются те же самые 3 запроса, как и раньше один за другим, и только после этого соединение закрывается. Заметьте, что загрузка завершается быстрее. Это происходит по двум причинам: во-первых, не тратится время на установку дополнительных соединений. Для каждого TCP-соединения требуется дополнительное время, как минимум, для его подтверждения. Во-вторых, передача тех же картинок проходит быстрее. Спросите, почему? Все это из-за контроля перегрузок сети в TCP. Сразу после установки соединения TCP использует процедуру медленного старта, чтобы увеличить пропускную способность, пока он не изучит поведение сетевого пути. Вследствие этого периода «разогрева» на передачу информации посредством множественных коротких TCP-соединений уходит гораздо больше времени, чем в случае с использованием одного длительного TCP-соединения. Наконец, на рис. 7.16, *в* установлено одно постоянное соединение, а запросы передаются конвейеризовано. Второй и третий запросы отсылаются за особенно короткий промежуток времени, так как была получена достаточно большая часть главной страницы, чтобы можно было сделать вывод о том, что изображения должны быть загружены и отображены. В итоге приходят ответы на эти запросы. Этот метод сокращает время, в течение которого сервер не занят делом, так что он и дальше повышает производительность. Однако за постоянные соединения приходится платить. Новый вопрос, который возникает вокруг них, — это «когда закрывать соединение?» Соединение с сервером не должно разрываться, пока загружается страница. А что потом? Высок шанс того, что пользователь нажмет на ссылку, которая запросит еще одну страницу с сервера. Если соединение остается открытым, следующий запрос может быть отослан немедленно. Однако нет гарантии того, что клиент создаст запрос к серверу в ближайшее время. На практике клиенты и серверы обычно сохраняют постоянное соединение, пока не пройдет какого-то небольшого промежутка времени (например, 60 с), в течение которого не будет отослано ни одного запроса и не будет принято ни одного ответа, или же в том случае, если открыто слишком много соединений и некоторые следует закрыть.

Внимательный читатель мог заметить, что существует одна комбинация, о которой мы пока не упомянули. Можно также посылать один запрос по одному TCP-соединению, но устанавливать эти соединения параллельно. Этот метод **параллельных соединений (parallel connection)** широко использовался браузерами до появления постоянных соединений. И у него тот же недостаток, что и у последовательных соединений — дополнительные служебные операции, — но производительность гораздо выше. Так происходит из-за того, что параллельная установка и увеличение количества соединений скрадывает некоторое количество времени. В нашем примере соединения для обоих размещенных изображений могут быть установлены в одно и то же время. Однако запуск большого числа TCP-соединений с одним и тем же сервером — не лучшая идея. Причина кроется в том, что TCP реализует отслеживание перегрузок отдельно для каждого соединения. В результате соединения соревнуются друг с другом, вызывая дополнительные потери пакетов, и в сочетании являются более агрессивными пользователями сети, чем индивидуальные соединения. Постоянные соединения стоят на уровень выше и их использование более предпочтительно, чем использование параллельных, так как они избегают ненужных издержек и не страдают от проблем с перегрузками.

Методы

Несмотря на то что HTTP был разработан специально для использования в веб-технологиях, он был намеренно сделан более универсальным, чем это было необходимо, так как рассчитывался на будущее применение в объектно-ориентированных приложениях. По этой причине в дополнение к обычным запросам веб-страниц были разработаны специальные операции, называемые **методами**. Они обязаны своим существованием технологии SOAP.

Каждый запрос состоит из одной или нескольких строк ASCII-текста, причем первое слово в первой строке является именем вызываемого метода. Встроенные методы перечислены в табл. 7.12. Имена методов чувствительны к регистру символов, то есть метод *GET* существует, а *get* — нет.

Таблица 7.12. Встроенные методы HTTP-запросов

Метод	Описание
GET	Запрос чтения веб-страницы
HEAD	Запрос чтения заголовка веб-страницы
PUT	Запрос сохранения веб-страницы
POST	Добавить к именованному ресурсу (например, к веб-странице)
DELETE	Удалить веб-страницу
TRACE	Отобразить входящий запрос
CONNECT	Зарезервирован для будущего использования
OPTIONS	Опрос определенных параметров

Метод *GET* запрашивает у сервера страницу (под которой в общем случае подразумевается объект, но на практике обычно это просто файл), закодированную согласно стандарту MIME. Большую часть запросов к серверу составляют именно запросы *GET*. Вот самая типичная форма *GET*:

```
GET filename HTTP/1.1
```

где `filename` указывает на запрашиваемую страницу, а `1.1` — на используемую версию протокола.

Метод *HEAD* просто запрашивает заголовок сообщения, без самой страницы. С помощью этого метода можно собрать индексную информацию или просто проверить работоспособность данного URL.

Метод *POST* используется, когда подтверждаются формы. Он, так же как и метод *GET,* используется для веб-сервисов SOAP. В нем также хранится URL, но вместо того, чтобы просто найти страницу, он передает данные на сервер (то есть содержимое формы или параметры RPC). Затем сервер в зависимости от URL что-то делает с этими данными, обычно прикрепляет их к объекту. В результате может быть, к примеру, что-то продано или вызвана процедура. Наконец, метод возвращает страницу с полученным результатом.

Оставшиеся методы редко используются для просмотра сетевых ресурсов. Метод *PUT* является противоположностью метода *GET*: он не читает, а записывает страницу.

Этот метод позволяет создать набор веб-страниц на удаленном сервере. Тело запроса содержит страницу. Она может быть закодирована с помощью MIME. В этом случае строки, следующие за командой *PUT*, могут включать различные заголовки, например заголовки аутентификации, подтверждающие права абонента на запрашиваемую операцию.

Метод *DELETE*, что неудивительно, удаляет страницу или, по крайней мере, указывает на то, что веб-сервер удалит страницу. Как и в методе *PUT*, здесь особую роль могут играть аутентификация и разрешение на выполнение этой операции.

Метод *TRACE* предназначен для отладки. Он приказывает серверу отослать назад запрос. Этот метод особенно полезен, когда запросы обрабатываются некорректно и клиенту хочется узнать, что за запрос реально получает сервер.

Метод *CONNECT* позволяет пользователю подключиться к серверу через устройство-посредник, такое как веб-кэш.

Метод *OPTIONS* позволяет клиенту запросить у сервера страницу и получить методы и заголовки, которые можно на ней использовать.

В ответ на каждый запрос от сервера поступает ответ, содержащий строку состояния, а также, возможно, дополнительную информацию (например, веб-страницу или ее часть). Строка состояния может содержать трехразрядный код состояния, сообщающий об успешном выполнении запроса или о причинах неудачи. Первый разряд предназначен для разделения всех ответов на пять основных групп, как показано в табл. 7.13. Коды, начинающиеся с 1 (1xx), на практике используются редко. Коды, начинающиеся с 2, означают, что запрос был обработан успешно и данные (если их запрашивали) отосланы. Коды 3xx сообщают клиенту о том, что нужно попытать счастья в другом месте — используя либо другой URL, либо свой собственный кэш (будет обсуждаться далее).

Таблица 7.13. Группы кодов состояния, содержащиеся в ответах сервера

Код	Значение	Примеры
1xx	Информация	100 = сервер согласен обрабатывать запросы клиента
2xx	Успех	200 = запрос успешно обработан; 204 = содержимое отсутствует
3xx	Перенаправление	301 = страница перемещена; 304 = кэшированная страница все еще доступна
4xx	Ошибка клиента	403 = ошибка доступа; 404 = страница не найдена
5xx	Ошибка сервера	500 = внутренняя ошибка сервера; 503 = попробуйте еще раз позднее

Коды, начинающиеся с 4, означают, что запрос по какой-либо причине, связанной с клиентом, потерпел неудачу: например, была запрошена несуществующая страница или сам запрос был некорректен. Наконец, коды 5xx сообщают о внутренних ошибках сервера, возникших либо вследствие ошибки программы, либо из-за временной перегрузки.

Заголовки сообщений

За строкой запроса (например, содержащей название метода *GET*) могут следовать другие строки с дополнительной информацией. Они называются **заголовками запро-**

сов (request headers). Эту информацию можно сравнить с параметрами, предоставляемыми при вызове процедуры. В свою очередь, ответы могут содержать **заголовки ответов (response headers)**. Некоторые заголовки могут встречаться и там, и там. Наиболее важные из них перечислены в табл. 7.14. Этот список достаточно длинный, так что, как вы понимаете, каждому запросу и ответу может соответствовать набор заголовков.

Таблица 7.14. Некоторые заголовки сообщений протокола HTTP

Заголовок	Тип	Содержимое
User-Agent	Запрос	Информация о браузере и его платформе
Accept	Запрос	Тип страниц, поддерживаемых клиентом
Accept-Charset	Запрос	Поддерживаемые клиентом наборы символов
Accept-Encoding	Запрос	Поддерживаемые клиентом типы кодирования
Accept-Language	Запрос	Естественные языки, понимаемые клиентом
If-Modified-Since	Запрос	Время и дата последнего обновления
If-None-Match	Запрос	Теги, отосланные с последнего обновления
Host	Запрос	DNS-имя сервера
Authorization	Запрос	Список персональных идентификаторов клиента
Referer	Запрос	URL, с которого был отправлен предыдущий запрос
Cookie	Запрос	Отправка ранее принятого cookie-файла на сервер
Set-Cookie	Ответ	Сервер хочет, чтобы клиент сохранил cookie
Server	Ответ	Информация о сервере
Content-Encoding	Ответ	Тип кодирования содержимого (например, gzip)
Content-Language	Ответ	Естественный язык, используемый на странице
Content-Length	Ответ	Размер страницы в байтах
Content-Type	Ответ	Тип MIME страницы
Content-Range	Ответ	Идентифицирует часть контента страницы
Last-Modified	Ответ	Время и дата внесения последних изменений в страницу
Expires	Ответ	Время и дата, когда страница перестанет считаться действительной
Location	Ответ	Команда клиенту на пересылку его запроса по другому адресу
Accept-Ranges	Ответ	Сервер готов принимать запросы на страницы указанного размера
Date	Запрос/ Ответ	Дата и время отправки сообщения
Range	Запрос/ Ответ	Идентифицирует часть страницы
Cache-Control	Запрос/ Ответ	Указание на то, как обрабатывать кэш
ETag	Запрос/ Ответ	Тег для контента страницы
Upgrade	Запрос/ Ответ	Протокол, на который хочет переключиться отправитель

Заголовок *User-Agent* позволяет клиенту информировать сервер о версии своего браузера (например, *Mozilla/5.0* и *Chrome/5.0.375.125*). Эта информация позволяет серверу приспосабливать свои ответы к конкретному браузеру, так как поведение и способности разных браузеров серьезно отличаются.

Четыре заголовка, начинающиеся с *Accept*, сообщают серверу о типах информации, которые клиент готов принять (если их набор ограничен). Первый приведенный в таблице заголовок определяет типы MIME, которые будут корректно приняты клиентом (например, *text/html*). Заголовок *Accept-Charset* сообщает о том, какой набор символов клиент хотел бы видеть (например, ISO-8859 или Unicode-1-1). В заголовке *Accept-Encoding* речь идет о приемлемых методах сжатия (например, gzip). Наконец, *Accept-Language* сообщает, на каком языке клиент готов читать документы (например, на испанском). Если сервер имеет возможность выбирать из нескольких страниц, он подберет наиболее подходящий для клиента вариант в соответствии с полученной информацией. Если запрос удовлетворить невозможно, возвращается код ошибки, и запрос считается неудавшимся.

Заголовки *If-Modified-Since* и *If-None-Match* используются кэшем. Они позволяют клиенту запрашивать отсылку страницы только в том случае, если в кэше нет доступной копии. Позднее мы еще поговорим о кэшировании.

Заголовок *Host* описывает сервер. Его значение берется из URL. Этот заголовок обязателен. Почему? Потому что некоторые IP-адреса могут обслуживать несколько имен DNS одновременно, и серверу необходимо каким-то образом различать, кому передавать запрос.

Заголовок *Authorization* требуется в тех случаях, когда запрашивается защищенная страница. С его помощью клиент может подтвердить свои права на просмотр запрашиваемой страницы.

Клиент использует неверно написанный заголовок *Referer*, чтобы выдать URL, связанный с запрашиваемым. Чаще всего это URL предыдущей страницы.

Этот заголовок крайне полезен при отслеживании переходов от страницы к странице, так как он позволяет серверу узнать, каким образом клиент прибыл на определенную страницу.

Несмотря на то что cookie описываются в RFC 2109, а не в RFC 2616, для их описания также существуют заголовки. Заголовок *Set-cookie* определяет то, как серверы отсылают файлы cookie клиентам. Если этот заголовок установлен сервером, предполагается, что, увидев его, клиент сохранит у себя cookie и вернет его вместе со следующим запросом на сервер при помощи заголовка *Cookie*. (Обратите внимание на то, что существует и более поздняя спецификация для файлов cookie с обновленными заголовками, RFC 2965, но она не особо широко распространена.)

В ответах используются и многие другие заголовки. Заголовок *Server* позволяет серверу описать версию своего программного обеспечения. Следующие пять заголовков, начинающиеся со слова *Content-*, позволяют серверу описать свойства страницы, которую он отсылает.

Заголовок *Last-modified* содержит дату и время внесения последних изменений в отправляемую страницу, а заголовок *Expires* говорит о том, сколько времени страница будет доступна. Они оба играют важную роль при кэшировании страницы.

Заголовок *Location* вставляется сервером для информирования клиента о том, что стоит попробовать осуществить свой запрос повторно по другому URL. Такая ситуация может возникать при «переезде» страницы или тогда, когда несколько URL ссылаются на одну и ту же страницу (возможно, на зеркало страницы, расположенное на другом сервере). Этот трюк также нередко применяется теми компаниями, главная веб-страница которых прописана в домене *com*, однако клиенты перенаправляются с нее на национальные или региональные страницы, основываясь на IP-адресе клиента или выбранном клиентом языке.

Если страница очень велика по размеру, клиент может не захотеть принимать ее сразу целиком. Некоторые серверы могут принимать запросы, ограничивающие размеры страниц, отсылаемых за один раз. Если страница оказывается слишком большой, она будет разбита на более мелкие единицы и выслана в несколько приемов. Заголовок *Accept-Ranges* сообщает о том, что сервер готов поддерживать такие запросы частей страниц.

Теперь перейдем к заголовкам, которые могут быть использованы в обоих направлениях. Заголовок *Date* может применяться как в запросах, так и в ответах. Он содержит время и дату отправки сообщения, а заголовок *Range* сообщает размер страницы в байтах и высылает его в качестве ответа.

Заголовок *ETag* выдает короткий тег, который служит в качестве имени контента страницы. Заголовок *Cache-Control* выдает другие четкие инструкции о том, как кэшировать (или, чаще, как не кэшировать) страницы.

Заголовок *Upgrade* используется для перехода на новый протокол коммуникации, такой как следующий протокол HTTP, или на защищенный способ передачи данных. Он позволяет клиенту сообщить, какие варианты он может поддерживать, а серверу заявить, каким вариантом пользуется он.

Кэширование

Мы часто возвращаемся на страницы, которые уже просматривали ранее, а на связанных веб-страницах часто размещаются одни и те же ресурсы. В качестве примера можно привести изображения, которые используются для навигации по сайту, а также стандартные таблицы стилей и скрипты. Было бы крайне неэкономично получать все эти ресурсы страниц каждый раз, когда они отображаются, так как у браузера уже есть их копии.

Сохранение полученных страниц для дальнейшего использования называется **кэшированием (caching)**. Преимущество этого метода состоит в том, что страница, сохраненная в кэше, может быть повторно использована, при этом не обязательно повторять передачу. У HTTP есть встроенная поддержка кэширования, чтобы помочь клиентам узнать, могут ли они безопасно использовать страницы повторно. Эта поддержка повышает производительность, сокращая и трафик и время ожидания. Все это достается ценой того, что теперь браузер должен хранить страницы, что практически всегда оправдано, так как локальное хранение информации не требует серьезных затрат ресурсов. Обычно страницы хранятся на диске, так что они могут быть использованы, когда браузер будет запущен в следующий раз.

Спорным вопросом в кэшировании HTTP является то, как определить, что сохраненная копия страницы соответствует тому, как страница выглядит при следующем

вызове. Такой вывод не может быть сделан только на основании совпадения URL. Например, URL может выдавать страницу, на которой отображаются последние новости. Контент этой страницы будет часто обновляться, хотя URL останется тем же. С другой стороны, на странице может содержаться список богов греческой и римской мифологии. Эта страница будет обновляться не так часто.

HTTP использует две стратегии решения этой проблемы. Они отбражены на рис. 7.17 как формы обработки между запросом (шаг 1) и ответом (шаг 5). Первая стратегия — это проверка действительности страницы (шаг 2). Запрашивается кэш, и если в нем есть свежая (то есть действительная) копия страницы заданного URL, получать новую копию с сервера нет необходимости. Вместо этого сохраненная в кэше страница может быть возвращена напрямую. Когда сохраненная в кэше страница была первоначально получена с сервера, вместе с ней возвращался заголовок *Expires*. Его содержание вместе с текущей датой и временем могут использоваться, чтобы сделать заключение о действительности страницы.

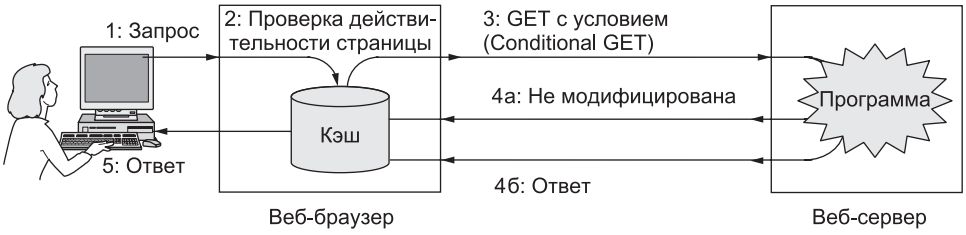


Рис. 7.17. Кэширование HTTP

Однако не все страницы приходят с удобным заголовком *Expires*, который указывает на то, когда страница должна быть получена снова. В конце концов, строить догадки тяжело, особенно догадки на будущее. В этом случае браузер может использовать эвристические правила. Например, если страница не была изменена за последний год (как сказано в заголовке *Last-Modified*), довольно логично предположить, что она не изменится за ближайший час. Тем не менее нет гарантии того, что эта ставка не оправдается. Например, биржа может закрыться на ночь, и ее страница не будет обновляться несколько часов, но, как только начнется следующая торговая сессия, изменения будут вноситься постоянно. Таким образом, оправданность кэширования может серьезно варьироваться в зависимости от периода времени. По этой причине эвристические правила должны использоваться осторожно, хотя они зачастую работают очень хорошо.

Поиск страниц, которые не были изменены за определенный промежуток времени, — наиболее полезный способ использования кэширования, так как в этом случае не нужно связываться с сервером. К сожалению, он не всегда работает. Сервер должен использовать заголовок *Expires* осторожно, так как то, когда на страницу будут внесены изменения, может быть неизвестно. Таким образом, копии, сохраненные в кэше, могут быть свежими, но клиент этого не знает.

В этом случае используется вторая стратегия. Она заключается в запрашивании с сервера информации о действительности сохраненной копии страницы. Этот запрос обозначается как **conditional GET** (GET с условием). Он показан на рис. 7.17 как шаг 3. Если сервер знает, что сохраненная в кэше копия все еще действительна, он может от-

править короткий ответ, подтверждая это (шаг 4а). В ином случае он должен отослать полный ответ (шаг 4б).

Есть еще несколько полей заголовков, которые используются для того, чтобы сервер мог проверить, действительна ли копия на момент запроса. Клиент знает, когда на страницу были внесены изменения в последний раз благодаря заголовку *Last-Modified*. Он может выслать это время серверу, используя заголовок *If-Modified-Since*, чтобы запросить страницу только в том случае, если она была изменена после последнего кэширования. В ином случае сервер может вернуть вместе со страницей заголовок *ETag*. Он высылает тег, являющийся кратким именем контента страницы, похожим на контрольную сумму, только лучше. (Это может быть криптографический хэш, который мы опишем в главе 8.) Клиент может выяснить, действительна ли сохраненная копии, выслав на сервер заголовок *If-None-Match*, перечисляющий теги сохраненной копии. Если любой тег совпадает с контентом, который сервер должен переслать в ответ, может быть использована соответствующая копия, сохраненная в кэше. Этот метод может быть использован, когда определять, свежая ли страница, неудобно или бессмысленно. Например, сервер может вернуть различный контент для одного и того же URL, в зависимости от того, какие языки и типы MIME являются предпочтительными. В этом случае только дата модификации не поможет серверу определить, свежая ли копия страницы хранится в кэше. Наконец, обратите внимание на то, что обе эти стратегии кэширования сводятся на нет директивами заголовка *Cache-Control*. Они могут использоваться, чтобы ограничить кэширование (например, не кэшировать) при отсутствии необходимости. В качестве примера можно привести динамические страницы, которые изменяются при каждом получении. Страницы, на которых требуется авторизация, также не кэшируются.

Можно еще много чего рассказать о кэшировании, но мы несколько ограничены в объеме, так что отметим лишь два важных момента. Во-первых, кэширование может использоваться не только в браузере. Обычно HTTP-запросы могут передаваться через несколько кэшей. Использование кэширования, не относящееся к браузеру, называется **прокси-кэшированием (proxy caching)**. Каждый новый уровень кэширования может помочь сократить количество запросов, передающихся вверх по цепочке. Запускать прокси-кэширование, чтобы извлечь пользу из кэширования страниц, запрашиваемых разными пользователями, — обычная практика провайдеров и организаций. Мы обсудим прокси-кэширование, рассмотрев его более подробно, в разделе 7.5 в конце главы. Во-вторых, кэширование серьезно повышает производительность, но не настолько, как многие надеются. Причина в том, что существуют как популярные веб-ресурсы, так и непопулярные, и многие из них очень длинные (например, видео). «Длинный хвост» редко посещаемых веб-ресурсов занимает место в кэше, а количество запросов, обрабатываемых кэш-памятью, не сильно растет с увеличением ее размера. Веб-кэш с высокой вероятностью не сможет обработать больше половины запросов (Breslau и др., 1999).

Пример использования HTTP

Поскольку HTTP является текстовым протоколом, взаимодействие с сервером посредством терминала (который в данном случае выступает как противоположность браузеру) можно организовать достаточно просто. Необходимо лишь установить TCP-соединение с портом 80 сервера. Читателю предоставляется возможность по-

экспериментировать со следующим сценарием. Он будет работать с большинством оболочек UNIX и в командном окне Windows (когда запущена программа telnet). Итак, последовательность команд такова:

```
telnet www.ietf.org 80 >log
GET /rfc.html HTTP/1.1
Host: www.ietf.org
```

Эта последовательность команд устанавливает telnet-соединение (то есть TCP-соединение) с портом 80 веб-сервера IETF, расположенного по адресу *www.ietf.org*. Далее следует команда *GET*. Указывается путь URL и протокол передачи. Попробуйте подставить свои варианты серверов и URL. Следом идет обязательная строка с заголовком *Host*. Пустая строка, которая находится за ней, также обязательна. Она сигнализирует серверу о том, что заголовки запросов закончились. Меняя названия серверов и URL, можно посмотреть много разных видов заголовков и страниц.

7.3.5. Мобильный веб

Всемирная паутина используется на большинстве компьютеров, в том числе и на мобильных телефонах. Просмотр сетевых ресурсов при помощи устройства мобильной связи может быть крайне полезным. Однако с этим связан и ряд технических проблем, так как большая часть контента была разработана в расчете на персональные компьютеры и широкие каналы связи. В этом разделе мы опишем, как разрабатывается доступ во Всемирную паутину через мобильные устройства, или **мобильный веб (mobile Web)**.

По сравнению с настольными компьютерами на работе и дома, мобильные телефоны несколько сложнее использовать для просмотра сетевых ресурсов. Причинами тому служат:

1) относительно маленькие экраны, на которых должны отображаться большие страницы и большие изображения;
2) ограниченные возможности ввода информации, которые не позволяют удобно вводить URL и другие длинные строки;
3) полоса пропускания канала, которая ограничена возможностями радиоканалов, например 3G, кроме того, это достаточно дорого;
4) связь, которая может работать нестабильно;
5) ограниченная вычислительная мощность из-за размеров, цены и емкости аккумулятора.

Эти сложности ведут к тому, что простое использование контента, предназначенного для настольных компьютеров, мобильными устройствами, вероятно, будет не самым приятным опытом.

Первые способы решения этих проблем основывались на создании новых протоколов, разработанных для беспроводных устройств с ограниченными возможностями. **WAP (Wireless Application Protocol — протокол беспроводного доступа)** наиболее широко известен как пример этой стратегии. Попытки создания WAP начались в 1997 году крупнейшими производителями мобильных устройств, такими как Nokia, Ericsson и Motorola. Однако их ожидал сюрприз. За следующие 10 лет пропускная

способность сети и возможности мобильных устройств выросли в десятки раз. Появились сервисы с данными в формате 3G и мобильные телефоны с большими цветными дисплеями, быстрыми процессорами и возможностью работать в беспроводной сети 802.11. Внезапно у мобильных устройств появилась возможность использовать обычные веб-браузеры. Разрыв между телефонами и настольными компьютерами, который никогда не будет преодолен, все еще существует, но многие технические проблемы, которые лежали в основе создания различных протоколов, исчезли.

Обретает популярность подход, предполагающий использование одних и тех же протоколов для персональных компьютеров и мобильных устройств, таким образом, чтобы сайты предоставляли удобный для отображения на экране мобильного телефона контент. Веб-серверы могут определять, какую версию веб-страниц предоставить для телефона или для компьютера, просмотрев заголовки запроса. Заголовок *User-Agent* особенно полезен в этом отношении, так как указывает на версию браузера. Таким образом, когда веб-сервер получает запрос, он может просмотреть заголовки и вернуть страницу с маленькими изображениями, меньшим количеством текста и более простой навигацией для iPhone и страницы с полным набором свойств для ноутбука.

W3C некоторым образом поддерживает этот подход. Например, он стандартизует лучшие практики для мобильного веб-контента. Список из 60 таких практик дан в первой спецификации (Рабин и МакКетиНевил, 2008). Большинство этих практик сделали серьезный вклад в максимизацию эффективности кэширования и сокращение размера страниц, в том числе при помощи сжатия, так как коммуникации обходятся дороже, чем вычисления. Благодаря этому подходу владельцы сайтов, особенно больших, создают мобильные версии контента, так как это требуется тем, кто выходит в сеть при помощи телефона. Чтобы помочь таким пользователям, существует обозначение страниц, которые можно (успешно) просматривать с мобильного телефона.

Еще один полезный инструмент — это сокращенная версия HTML, которая называется **XHTML Basic**. Этот язык является частью XHTML, которая предназначена для использования при помощи мобильных телефонов, телевизоров, карманных компьютеров, торговых автоматов, пейджеров, автомобилей, игровых автоматов и даже часов. По этой причине такая версия не поддерживает таблиц стилей, скриптов или фреймов, но большинство стандартных тегов в ней присутствует. Они сгруппированы в 11 модулей. Некоторые необходимы, некоторые опциональны. Все они определены в XML. Эти модули и некоторые примеры тегов приведены в табл. 7.15.

Однако не все страницы будут спроектированы так, чтобы хорошо работать на мобильных устройствах. Таким образом, дополнительный подход состоит в **трансформации контента (content transformation)** или **транскодировании (transcoding)**. Он подразумевает, что компьютер, обеспечивающий связь компьютера и сервера, забирает страницу с сервера и трансформирует ее таким образом, чтобы контент подходил для мобильного устройства. Простой трансформацией является переформатирование больших изображений в более низкое разрешение для сокращения их размера. Возможны и многие другие простые, но полезные трансформации. Транскодирование с определенным успехом использовалось с появлением мобильного веба (см., например, Fox и др. 1996). Однако при использовании обоих подходов возникает некоторая несогласованность между мобильным контентом и решениями, принимаемыми сервером и транскодером. Например, веб-сайт может выбрать определенную комбинацию

изображений и текста для пользователя мобильного Интернета только для того, чтобы транскодер просто изменил формат изображения.

Таблица 7.15. Модули и теги XHTML Basic.

Модуль	Необходимый?	Функция	Примеры тегов
Structure (Структура)	Да	Структура документа	`body`, `head`, `html`, `title`
Text (Текст)	Да	Информация	`br`, `code`, `dfn`, `em`, `h`*n*, `kbd`, `p`, `strong`
Hypertext (Гипертекст)	Да	Гиперссылки	`a`
List (Списки)	Да	Списки элементов	`dl`, `dt`, `dd`, `ol`, `ul`, `li`
Forms (Формы)	Нет	Заполнение форм	`form`, `input`, `label`, `option`, `textarea`
Tables (Таблицы)	Нет	Прямоугольные таблицы	`caption`, `table`, `td`, `th`, `tr`
Image (Изображения)	Нет	Размещение изображений	`img`
Object (Объекты)	Нет	Апплеты, карты и т. д.	`object`, `param`
Meta-information (Метаинформация)	Нет	Дополнительная информация	`meta`
Link (Ссылка)	Нет	Аналогично <a>	`link`
Base (База)	Нет	Точка отсчета URL	`base`

Пока мы говорили о контенте, а не о протоколах, так как именно контент представляет наибольшую проблему при реализации мобильного веба. Однако мы вкратце упомянем и протоколы. Протоколы HTTP, TCP и IP, использующиеся сетью, могут заполнять значительную часть канала, передавая информацию, которая не будет отображать, например, заголовки. Чтобы решить эту проблему, в WAP и других методах определены протоколы для специальных целей. В этом, по большей части, нет необходимости. Технологии сжатия заголовков, такие как ROHC (RObust Header Compression), описанные в главе 6, могут сократить объем побочной информации в таких протоколах. Таким образом, можно пользоваться одним набором протоколов (HTTP, TCP, IP) по связям как с высокой, так и с низкой пропускной способностью. Использование связей с низкой пропускной способностью требует всего лишь включения механизма сжатия заголовков.

7.3.6. Веб-поиск

Чтобы закончить наше обсуждение Всемирной паутины, мы поговорим о том, что по мнению многих людей является наиболее успешным веб-приложением — веб-поиск. В 1998 году Сергей Брин и Лари Пейдж, студенты Стенфорда, создали компанию под названием Google, чтобы написать лучший механизм поиска. Они были вооружены идеей, впоследствии показавшей себя крайне успешно. А заключалась она в том, что алгоритм поиска, оценивающий, сколько раз ссылки с других страниц указывали на

каждую страницу, является лучшим показателем ее важности, чем то, сколько раз на ней встречаются ключевые слова. Например, многие страницы ссылаются на главную страницу Cisco, что делает эту страницу более значимой для пользователя, который ввел «Cisco» как ключевое слово, чем страница, не относящаяся к компании, но включающая это слово много раз.

Они были правы. Такой подход действительно позволил написать лучший механизм поиска, и пользователи это оценили. Поддержанная венчурным капиталом компания Google невероятно разрослась. В 2004 году она стала акционерной и стоила $23 млрд. К 2010 году она охватывала более миллиона серверов в центрах обработки и хранения данных. По всему миру.

В некотором роде, поиск — это просто еще одно приложение, хотя и одно из наиболее старых, так как он развивался с появления Всемирной паутины. Однако веб-поиск оказался крайне необходимым. Ежедневно обрабатывается более миллиарда запросов. Люди, ищущие совершенно разную информацию, используют поиск как отправную точку. Например, очевидно, что мы не знаем, с какой страницы начать просмотр, чтобы выяснить, где купить вегемит в Сиэтле, но есть шанс, что поисковая система знает о странице с желаемой информацией и сможет быстро переправит вас на нее.

Чтобы осуществить веб-поиск в традиционном виде, пользователь открывает в своем браузере сайт веб-поиска. Основные поисковые сайты — это Google, Yahoo! и Bing[1]. Затем пользователь отправляет нужный запрос используя форму. Тогда поисковая система передает этот запрос базе данных, чтобы найти релевантные страницы, изображения и т. д., и возвращает ответ в виде динамической страницы. После чего пользователь может перейти по найденным ссылкам.

Веб-поиск — интересный предмет для обсуждения, так как он влияет на организацию и использование сетей. Во-первых, интересен вопрос, как веб-поиск находит нужные страницы. У системы веб-поиска должна быть база данных, содержащая страницы, чтобы обработать запрос. Каждая HTML-страница может содержать ссылки на другие страницы, так что на все интересное (или, по крайней мере, часто появляющееся в поисковых запросах) где-то уже стоит ссылка. То есть теоретически возможно, начав с небольшого набора страниц, просмотреть все страницы, размещенные в вебе, переходя по всем ссылкам. Этот процесс называется **веб-кроулингом** (**Web crawling**, часто произносится и пишется как **веб-краулинг**) или **обходом страниц**. Все поисковые системы используют его.

Одна из сложностей с веб-кроулингом заключается в том, что найти можно не все виды страниц. Получение статичных документов и переходы по ссылкам — это просто. Однако многие веб-страницы содержат программы, которые выдают разные страницы в зависимости от того, какая информация была отправлена пользователем. Примером может послужить каталог онлайн-магазина. Каталог может содержать динамические веб-страницы, созданные при помощи базы данных, содержащих продукты, и запро-

[1] В России (точнее, для поиска ресурсов, использующих русский язык) между Google и Yahoo! вполне можно поставить Яндекс (yandex.ru). Тем более что до совсем недавнего времени он существенно превосходил Google по качеству учета русской орфографии и грамматики при индексировании ресурсов и обработке поисковых запросов пользователей. Впрочем, хуже он и не стал — Google догнал. — *Примеч. ред.*

сов разных продуктов. Этот тип контента отличается от статичных страниц, которые легко обнаружить. Как системы веб-поиска находят такие динамические страницы? Ответ прост: по большей части не находят. Этот тип скрытого контента называется **глубокой Всемирной паутиной** (**deep Web**). Как искать такой контент — открытая проблема, над которой сейчас бьются ученые (см., например, Мадхаван и др. 2008). Также существует договоренность, в соответствии с которой сайты создают страницу (известную как *robots.txt*), чтобы сообщить поисковым системам, какие части сайтов должны или не должны посещаться.

Второй серьезный вопрос — как обрабатывать данные с перекрестными гиперссылками. Чтобы алгоритмы индексации работали на наборе данных, где-то должны храниться страницы. Оценки расходятся, но основные поисковые системы, вероятно, содержат каталоги десятков миллиардов страниц, скопированных с видимой части Всемирной паутины. Средний размер страницы оценивается в 320 Кбайт. Это значит, что посещенная поисковыми системами копия Всемирной паутины занимает около 20 петабайт или 2×10^{16} байт. Хоть это число и крайне велико, именно такой объем данных может храниться и обрабатываться в центрах обработки данных Интернета (Чанг и др. 2006). Например, если хранение информации на диске стоит 20 долларов за терабайт, хранение 2×10^4 Тбайт стоит $400 000, что не много для таких огромных компаний, как Google, Microsoft и Yahoo!. Кроме того, в то время как сеть расширяется, цены на диски стремительно падают, так что в обозримом будущем хранение всей Всемирной паутины представляется решаемой задачей в рамках больших компаний.

Извлечь из этих данных полезную информацию — другая важная задача. Вы можете оценить то, как XML помогает программам легко извлекать структуру данных, в то время как специальные форматы добавляют неразрешенных проблем. Также важен вопрос конверсии форматов и даже перевода с одного языка на другой. Но даже выяснение структуры данных является лишь частью проблемы, а сама она заключается в том, чтобы понять, что подразумевает данная структура. Решение этой задачи принесет огромную пользу, начиная с того, что поможет подбирать релевантные страницы в ответ на поисковые запросы. Главная цель — получить возможность ответа на вопросы, например, где купить дешевый, но качественный тостер в вашем городе.

Третий аспект веб-поиска заключается в том, что он создает более высокий уровень именования. Вам не нужно запоминать длинный URL, если с тем же успехом (или даже бóльшим) можно найти веб-страницу по имени человека. Мы предполагаем, что имя запомнить проще, чем URL. Эта стратегия оказывается крайне удачной. Тем же способом, которым DNS-имена вытеснили IP-адреса компьютеров, веб-поиск вытесняет URL. К плюсам поиска можно отнести и то, что он исправляет ошибки и опечатки, в то время как неверно напечатав URL, вы получите неверную страницу.

Наконец, веб-поиск предоставляет нам возможности, которые мало влияют на непосредственное устройство сети, но оказывают большое воздействие на интернет-сервисы — финансирование рекламы. Реклама — это двигатель экономики, благодаря которому сеть разрастается. Главное отличие от печатной рекламы заключается в способности выдавать рекламу, основываясь на том, что ищет пользователь, и таким образом повысить ее релевантность. Такое изменение механизма направлено на добавление к результатам поиска наиболее релевантной рекламы (Edelman и др., 2007).

Конечно, эта новая модель послужила причиной возникновения новых проблем, таких как **накручивание числа кликов по ссылке (click fraud)**, при котором программы имитируют пользователей, щелкающих мышкой по рекламе, что приводит к перечислению средств, заработанных не самым честным путем.

7.4. Потоковая передача аудио и видео

Веб-приложения и мобильный веб — это, конечно, замечательное изобретение последних лет в области использования сети, однако далеко не единственное. Для многих аудио и видео служат чашей святого Грааля сетевых технологий. Слово «мультимедиа» возбуждает и разработчиков, и коммерсантов. Одни видят в мультимедиа бесконечный источник интересных технических проблем, связанных, например, с передачей голоса через IP или доставкой видео по заказу, возможно интерактивного, другие — не меньший источник прибыли.

Хотя идея отправки аудио и видео через Интернет появилась еще в семидесятых, только в начале XXI века передача **мультимедиа в реальном времени (real-time audio** и **real-time video)** стала возможной. Трафик в реальном времени отличается от вебтрафика тем, что передача должна идти с определенной скоростью, чтобы иметь смысл. Кроме того, просмотр постоянно прерывающегося видео в замедленном темпе вряд ли кому-то доставит удовольствие. В противоположность, при обращении к Всемирной паутине в сети могут быть короткие перебои, а на загрузку страницы может уходить больше или меньше времени (в определенных пределах), и это не станет серьезной проблемой.

Чтобы обеспечить такие изменения, произошло два события. Во-первых, компьютеры стали гораздо мощнее, они стали оснащаться микрофонами и камерами, так что появилась возможность легко вводить, обрабатывать и выводить аудио- и видеоданные. Во-вторых, были решены все проблемы с пропускной способностью Интернета. Длинные ссылки, ведущие к центру Интернета, работают со скоростью нескольких гигабайт в секунду, а широкополосный доступ и беспроводная связь 802.11 доходят до пользователей на периферии Интернета. Это развитие позволило интернет-провайдерам переносить огромные объемы трафика через обычные телефонные модемы на 56 Кбит/с, благодаря чему обычные пользователи могут подключаться к Интернету в 100–1000 раз быстрее, .

С расширением пропускной способности канала вырос аудио- и видеотрафик, но это произошло по разным причинам. Телефонные разговоры занимают сравнительно небольшую часть канала (64 Кбит/с, а при сжатии еще меньше), но телефонные услуги всегда были дорогими. Компании обратили внимание на возможность переложить телефонный трафик на Интернет, чтобы сократить телефонные счета. Такие проекты, как Skype, дали пользователям возможность совершать телефонные звонки бесплатно, используя интернет-соединения. Быстро развивались телефонные компании, которые увидели дешевый способ передавать обычные голосовые звонки, используя оборудование, предназначенное для выхода в сеть. В результате многократно возросло количество передаваемых по интернет-сетям голосовых данных, которые называются «**Голос поверх IP**» (**Voice over IP**, **VoIP**) или **интернет-телефонией (Internet telephony)**.

В отличие от аудио видео занимает значительную часть пропускной способности канала. интернет-видео в хорошем качестве кодируется со сжатием до примерно 1 Мбит/с, а ведь в обычном DVD-фильме около 2 Гбайт. До появления широкополосного доступа в Интернет пересылка фильмов была невозможной. Но теперь все изменилось. С распространением скоростного доступа у пользователей впервые появилась возможность смотреть размещенное в сети видео из дома. И многие именно так и делают. Около четверти пользователей каждый день заходят на YouTube, популярный видеосайт. Прокат видеофильмов сегодня заменили загрузки из Интернета. А общий размер размещенных видеороликов совершенно изменил структуру интернет-трафика. Уже сегодня большая часть информации, размещенной в Интернете, — это видео, а через несколько лет на долю видео будет приходиться около 90 % интернет-трафика (Cisco, 2010).

Учитывая то, ширины полосы пропускания достаточно для передачи видео и аудио, ключевым вопросом разработки приложений для проведения конференций и передачи мультимедиа является сетевая задержка. Для аудио и видео требуется представление в реальном времени, то есть они должны проигрываться с определенной скоростью, чтобы быть полезными. Долгие задержки предполагают, что звонки, которые должны быть интерактивными, таковыми не являются. Эта проблема будет вам понятна, если вы хоть раз разговаривали по спутниковому телефону. Ведь в этом случае задержка даже на полсекунды ужасно раздражает. При проигрывании музыки и фильмов через сеть абсолютная задержка не играет роли, так как она влияет только на то, когда начнется проигрывание файла. Но варьирование задержки, которое называется **неустойчивой синхронизацией (jitter)**, все еще имеет значение. Оно должно маскироваться плеером, иначе аудио будет звучать неразборчиво, а видео будет постоянно дергаться.

В этом разделе мы обсудим некоторые стратегии, позволяющие разобраться с этой проблемой, а также протоколы для установки аудио- и видеосессий. После того как мы расскажем об аудио и видео, мы остановимся на трех случаях, для которых используются разные методы. Первый и самый простой случай, который мы рассмотрим, — это передача сохраненного в сети видео, как, например, при просмотре файлов на YouTube. Следующий по сложности случай — это передача потокового живого мультимедийного контента. Примерами этого служат онлайн-радио и телевидение (IPTV), когда радио- и телестанции вещают через Интернет. Последний и самый сложный случай — это интерактивные аудио- и видеоконференции, такие как звонки через Skype,

Часто термин **мультимедиа (multimedia)** используется в контексте Интернета и обозначает аудио и видео. Буквально мультимедиа означает использование двух или более средств аудиовизуальной информации. По этому определению выходит, что данная книга является объектом мультимедиа, так как в ней есть текст и графика (рисунки). Тем не менее вы, вероятно, представляете себе мультимедиа несколько иначе, так что мы будем использовать этот термин для обозначения *медиа*, *длящегося во времени* (*continuous media*), то есть проигрываемого в течение определенного интервала времени. На практике чаще всего это аудио и видео, то есть звук, плюс движущееся изображение.

Несмотря на такое довольно понятное определение, многие, говоря о «мультимедиа», имеют в виду только чисто звуковую информацию, например интернет-теле-

фонию или интернет-радио. Строго говоря, они не правы. Более удачным термином в данном случае является словосочетание **потоковая информация** (**streaming media**), однако мы, поддавшись стадному инстинкту, все же будем именовать аудиоданные, передающиеся в реальном масштабе времени, «мультимедиа».

7.4.1. Цифровой звук

Звуковая волна представляет собой одномерную акустическую волну (волну давления). Когда такая волна достигает уха, барабанная перепонка начинает вибрировать, вызывая вибрацию тонких костей внутреннего уха, в результате чего в мозг по нерву посылается пульсирующий сигнал. Эта пульсация воспринимается слушателем как звук. Подобным образом, когда акустическая волна воздействует на микрофон, им формируется электрический сигнал, представляющий собой амплитуду звука как функцию времени.

Человеческое ухо способно слышать сигналы в диапазоне частот от 20 до 20 000 Гц, хотя некоторые животные, например собаки, могут слышать и более высокие частоты. Громкость, воспринимаемая ухом, изменяется логарифмически по отношению к амплитуде, поэтому отношение силы двух звуков с амплитудами A и B обычно измеряется в **децибелах (дБ)**, как $10 \log_{10}(A/B)$. Если принять нижний порог слышимости (давление около 2×10^5 Па) для синусоидальной волны частотой 1 кГц за 0 дБ, то громкости обычного разговора будет соответствовать 50 дБ, а болевой порог наступит при силе звука около 120 дБ, что соответствует отношению амплитуд равному 1 млн.

Человеческое ухо удивительно чувствительно к изменениям звука, длящимся всего несколько миллисекунд. Глаз, напротив, не в состоянии заметить такие кратковременные изменения. Таким образом, флуктуация (джиттер) в несколько миллисекунд при передаче мультимедиа влияет в большей степени на качество звука, чем на качество изображения.

Цифровое аудио — это цифровое представление аудиоволны, которое можно использовать для ее воссоздания. Звуковые волны можно преобразовывать в цифровую форму при помощи **аналогово-цифрового преобразователя** (**АЦП**). На вход АЦП подается электрическое напряжение, а на выходе формируется двоичное число. На рис. 7.18, *а* показан пример синусоидальной волны. Чтобы представить этот сигнал в цифровом виде, мы можем измерять значения сигнала (отсчеты) через равные интервалы времени ΔT, как показано на рис. 7.18, *б*. Если звуковая волна не является чисто синусоидальной, а представляет собой сумму нескольких синусоидальных волн и самая высокая частота ее составляющих равна f, тогда, согласно теореме Найквиста (см. главу 2), для последующего восстановления сигнала достаточно измерять значения сигнала с частотой дискретизации $2f$. Производить замеры сигнала с большей частотой нет смысла, так как более высокие частоты отсутствуют в сигнале.

При обратном процессе цифровые значения переводятся в аналоговое электрическое напряжение. Это делается при помощи **цифро-аналогового преобразователя** (**ЦАП**). Потом репродуктор может перевести аналоговое напряжение в акустические волны, и люди услышат звуки. Оцифрованные отсчеты (сэмплы) никогда не бывают точными. Например, отсчеты на рис. 7.18, *в* могут принимать только 9 значений — от

–1,00 до +1,00 с шагом 0,25. При 8-битовом квантовании каждый отсчет может принимать одно из 256 различных значений. При 16 битах на отсчет можно кодировать сигнал с еще более высокой точностью, так как каждому значению сигнала можно сопоставить одно из 65 536 различных значений. Ошибка, возникающая в результате неточного соответствия квантованного сигнала, способного принимать конечное число значений, исходному сигналу, называют **шумом квантования (quantization noise)**. При недостаточном количестве битов, которыми представляется каждый отсчет сигнала, этот шум может быть настолько велик, что будет различим на слух как искажение исходного сигнала или как посторонние шумы.

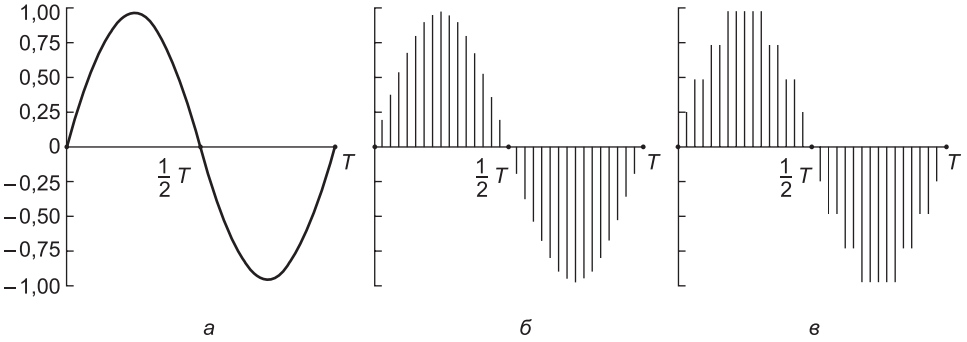


Рис. 7.18. Волна: *а* — синусоидальная; *б* — дискретизация; *в* — квантование отсчетов 4 битами

Двумя хорошо известными примерами использования цифрового звука являются телефон (если применяются новые цифровые АТС) и аудио-компакт-диски. В импульсно-кодовой модуляции, применяемой в телефонной системе, используются восьмибитовые отсчеты, замеряемые 8000 раз в секунду. Шкала является нелинейной, чтобы минимизировать воспринимаемое искажение, и при 8000 замеров в секунду частоты выше 4кГц теряются. В Северной Америке и Японии при кодировании используется **закон μ (μ-law)**. В Европе, а также во многих странах по всему миру при кодировании используется **закон A (A-law)**. Каждая кодировка обеспечивает поток данных в 64 000 бит/с.

Аудио-компакт-диски содержат звуковой сигнал, оцифрованный с частотой дискретизации 44 100 Гц, в результате чего они могут хранить звуки с частотами до 22 кГц, что воспринимается как достаточно качественный звук людьми, но считается весьма низким качеством среди собак, ценящих хорошую музыку. Каждому отсчету выделяется 16 бит, его значение пропорционально амплитуде сигнала. Обратите внимание, что 16-битовый отсчет может принимать всего 65 536 различных значений, хотя измерения показывают, что динамический диапазон человеческого уха составляет более одного миллиона значений. Таким образом, несмотря на то что аудио в CD-качестве гораздо лучше, чем аудио, передаваемое через телефон, использование 16 бит на отсчет дает существенный шум квантования (хотя полный динамический диапазон и не охвачен, качество звучания компакт-дисков обычно не вызывает нареканий). Некоторые фанатичные аудиофилы по прежнему выбирают записи в формате долгоиграющих пластинок с 33 оборотами в минуту, а не CD-записи, так как у пластинок

нет ограничения предельной частоты в 22 кГц и нет шума квантования (однако, если с ними не обращаться очень бережно, на них появляются царапины). При 44 100 отсчетах в секунду по 16 бит каждый для несжатого аудио в CD-качестве требуется пропускная способность в 705,6 Кбит/с для монофонического сигнала и 1,411 Мбит/с для стереофонического.

Сжатие звука

Аудио часто сжимается для того, чтобы сократить требуемую полосу пропускания канала и время передачи, несмотря на то что аудиоданные требуют не такой большой пропускной способности, как видео. Во всех системах сжатия должны присутствовать 2 алгоритма: один для сжатия данных в месте их размещения и второй для их распаковки. В литературе эти алгоритмы называют алгоритмами **кодирования (encoding)** и **декодирования (decoding)** соответственно. Мы также будем использовать эту терминологию.

В алгоритмах сжатия присутствует определенная асимметричность, о которой необходимо знать. Хотя сейчас мы рассматриваем аудио, этот аспект также относится и к видео. Для многих приложений мультимедиа-документ кодируется единожды (когда сохраняется на мультимедиа-сервере). Эта асимметричность означает, что алгоритм кодирования может быть медленным и требовать дорогого оборудования, при этом алгоритм декодирования должен быть быстрым и работающим на дешевом оборудовании. Оператор популярного аудио- или видеосервера может склониться к мысли о покупке нескольких компьютеров, чтобы закодировать всю свою библиотеку, но требовать того же от посетителей сайта, которые зашли, чтобы послушать музыку или посмотреть фильмы, — не лучшая мысль. Многие используемые сегодня системы сжатия являются крайне объемными, все это делается для того, чтобы сделать декодирование быстрым и простым, даже ценой медленного и сложного кодирования. С другой стороны, для живого видео и аудио, такого как звонки через Skype, медленно кодирование неприемлемо. Оно должно работать в реальном времени. Следовательно, мультимедиа в реальном времени использует алгоритмы или параметры, отличные от аудио или видео, хранящегося на диске. Часто используется гораздо меньшее сжатие. Второе нарушение симметрии состоит в том, что процесс кодирования/декодирования не обязательно должен быть обратимым. То есть при сжатии файла, его передаче и последующей декомпрессии пользователь должен получать всю информацию до последнего бита. В отношении мультимедиа это требование не актуально. Обычно аудио- или видеосигнал может отличаться от оригинала после кодирования и последующего декодирования, при условии, что звучать (или выглядеть) он будет так же. Когда раскодированный файл не полностью соответствует закодированному оригиналу, передача информации идет с **потерями (lossy)**. Если входной и выходной файлы совпадают, передача идет **без потерь (lossless)**. Системы с потерями важны, потому что, потеряв небольшую часть информации, мы получаем гораздо лучшее сжатие.

Исторически междугородняя связь по телефонным сетям была очень дорогой, так что есть еще масса работы, которую необходимо проделать над **голосовыми кодерами (voice coders — vocoders)**, сжимающими такое аудио, как человеческая речь. Речь занимает диапазон от от 600 до 6000 Гц и порождается благодаря механическому

процессу, особенности которого зависят от голосового аппарата говорящего, языка и челюстей. Некоторые голосовые кодеры используют модели голосовой системы для сведения речи к небольшому набору параметров (например, размеры и форма резонаторов) и объема данных, равного всего лишь 2,4 Кбит/с. Однако рассмотрение устройства этих голосовых кодеров выходит за рамки данной книги.

Мы сконцентрируемся на передаче через Интернет аудио, которое обычно близко к CD-качеству. При передаче такого аудио тоже желательно сократить объем данных. Без сжатия для передачи аудио в стереокачестве потребуется ширина канала в 1,411 Мбит/с, что будет тормозить многие широкополосные каналы, оставляя меньше места для видео и другого веб-трафика. Скорость передачи данных можно уменьшить на порядок благодаря сжатию, при этом потери качества будут незначительными или же их не будет вовсе.

Сжатие и распаковка требуют обработки сигнала. К счастью, цифровой звук и видеоролики могут легко обрабатываться компьютерами при помощи специального ПО. На самом деле, существуют десятки программ, позволяющих пользователям записывать, отображать, редактировать и хранить медиаданные из различных источников. Это привело к тому, что большое количество музыки и фильмов доступно через Интернет — правда, это не всегда законно, — в результате чего артисты и правообладатели подают множество исков.

Для сжатия аудиофайлов было разработано множество алгоритмов. Вероятно, самые популярные форматы — это **MP3 (MPEG audio layer 3 — MPEG аудио, уровень 3)** и **ACC (Advanced Audio Coding — усовершенствованное кодирование аудио)**, использующееся в файлах **MP4 (MPEG-4)**. Чтобы не путаться, учтите, что MPEG обеспечивает сжатие и аудио и видео. MP3 относится к блоку сжатия аудио (часть 3) стандарта MPEG-1, а не к третьей версии. На самом деле, третья версия формата MPEG не выходила, вышли только MPEG-1, MPEG-2 и MPEG-4. AAC — это формат, который последовал за NP3. Он по умолчанию используется для кодирования аудио в MPEG-4. В MPEG-2 возможны оба варианта кодирования аудио, MP3 и AAC. Теперь понятно? Что хорошо в стандартах, так это то, что всегда есть выбор. И если какой-то вам не нравится, просто подождите год-другой.

Существует две концепции сжатия звука. При **кодировании формы сигналов (waveform coding)** сигнал раскладывается на компоненты при помощи преобразования Фурье. В главе 2 мы привели пример в виде временной функции и амплитуд, получающихся в результате ее разложения в ряд Фурье (см. рис. 2.1, *а*). Амплитуда каждой компоненты кодируется с минимальными искажениями. Задачей является максимально аккуратная передача формы сигнала с минимально возможной затратой битов.

Другая концепция называется **перцепционным кодированием (perceptual coding)**. Она основана на некоторых недостатках слухового аппарата человека, позволяющих кодировать сигнал таким образом, что слушатель не ощутит никакой разницы по сравнению с настоящим сигналом, хотя на осциллографе эта разница будет весьма заметна. Наука, на которой базируется перцепционное кодирование, называется **психоакустикой (psychoacoustics)**. Она изучает восприятие звука человеком. Оба формата, MP3 и AAC, используют перцепционное кодирование.

Ключевым свойством перцепционного кодирования является то, что одни звуки могут **маскировать** другие. Представьте себе, что теплым летним вечером вы медити-

руете на лужайке, слушая живой концерт для флейты с оркестром. Затем, откуда ни возьмись, появляется бригада рабочих с отбойными молотками в руках, которая начинает вскрывать асфальт на близлежащей улице. Расслышать флейту, к сожалению, уже никто не в состоянии. Нежные звуки, издаваемые ею, подверглись *маскированию* звуками отбойных молотков. Если рассматривать ситуацию с точки зрения передачи данных, то в этот момент достаточно кодировать лишь диапазон частот, в котором работают отбойные молотки, — все равно флейту за этим грохотом не слышно. Способность громких звуков определенного диапазона частот «прятать» более тихие звуки других диапазонов (которые были бы слышны при отсутствии громких звуков) называется **частотным маскированием (frequency masking)**. На самом деле, даже после того как рабочие выключат отбойные молотки, слушатели не будут слышать флейту в течение некоторого небольшого периода времени. Это связано с тем, что при появлении очень громкого звука коэффициент усиления человеческого уха резко снизился, и после прекращения работы отбойных молотков требуется время на его возвращение в нормальное состояние. Этот эффект называется **временным маскированием (temporal masking)**.

Чтобы перейти от качественного описания этих эффектов к количественным, представим себе проведение некого эксперимента 1. Человек, находящийся в тихом помещении, надевает наушники, соединенные со звуковой картой компьютера. Компьютер генерирует звук (чистую синусоидальную звуковую волну) с частотой 100 Гц, сила которого постепенно возрастает. Испытуемый должен нажать клавишу на клавиатуре, как только он услышит звук. Компьютер запоминает силу звука, при которой была нажата клавиша, и повторяет эксперимент на частотах 200 Гц, 300 Гц и т. д., доходя до верхнего предела слышимых частот. Эксперимент необходимо провести над большим количеством испытуемых. На рис. 7.19, *а* показан график с логарифмическим масштабом на обеих осях, показывающий усредненную зависимость порога слышимости от частоты звука. Наиболее очевидный вывод, который можно сделать при взгляде на эту кривую, состоит в том, что нет никакой необходимости когда бы то ни было кодировать частоты, амплитуда которых ниже порога слышимости. Например, если сила звука на частоте 100 Гц равна 20 дБ, этот звук можно не кодировать, и качество звучания при этом не ухудшится, так как уровень 20 дБ при 100 Гц находится ниже порога слышимости (см. рис. 7.19, *а*).

Теперь рассмотрим эксперимент 2. Пусть компьютер повторяет действия эксперимента 1, но на этот раз на каждую тестовую частоту будет накладываться синусоидальная звуковая волна постоянной амплитуды с частотой, скажем, 150 Гц. Мы обнаружим, что порог слышимости для частот, расположенных вблизи 150 Гц, резко возрастает. Это отражено на графике на рис. 7.19, *б*.

Из последнего наблюдения можно сделать следующий вывод: зная, какие сигналы маскируются более мощными сигналами на близлежащих частотах, мы можем пренебречь соответствующими частотами и не кодировать их, экономя тем самым биты. Из рис. 7.19, *б* очевидно, что сигналом с частотой 125 Гц можно полностью пренебречь, и никто не заметит разницы. Даже после прекращения звучания громкого сигнала в каком бы то ни было частотном диапазоне знание свойств временного маскирования позволяет в течение некоторого времени (пока ухо настраивается на меньшую мощность звука) продолжать пренебрегать кодированием этой частоты. Суть алгоритмов

MP3 и AAC состоит в разложении сигнала в ряд Фурье для получения силы звука на каждой из частот с последующей передачей исключительно немаскированных частот, кодируемых минимально возможным числом бит.

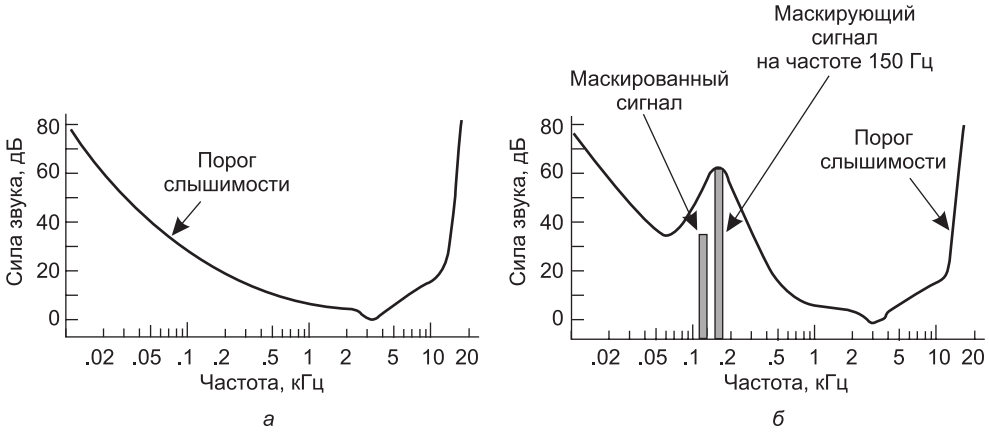


Рис. 7.19. Порог слышимости как функция частоты (а); эффект маскирования (б)

Теперь, зная основной принцип, мы можем рассмотреть, как производится само кодирование. Сжатие звука выполняется путем замеров амплитуды сигнала, производимых с частотой от 8 до 96 кГц для AAC, а часто просто с частотой 44,100 кГц, чтобы звук получался в качестве, близком к CD. Замеры могут сниматься по одному (моно) или двум (стерео) каналам. Затем выбирается желаемая выходная битовая скорость. С помощью алгоритма MP3 можно сжать записанную на компакт-диск стереофоническую запись рок-н-ролла до 96 Кбит/с с потерей качества, едва заметной даже для фанатов рок-н-ролла, не лишенных слуха. Если мы хотим «перегнать в MP3» фортепианный концерт, нам понадобится формат AAC с битовой скоростью по крайней мере 128 Кбит/с. Разница обусловлена тем, что соотношение сигнал/шум в рок-н-ролле гораздо выше, чем в фортепианном концерте (только в техническом смысле, разумеется). Можно, впрочем, выбрать меньшую битовую скорость и получить более низкое качество воспроизведения.

После этого отсчеты обрабатываются небольшими группами. Каждая группа предварительно проходит через набор цифровых фильтров, выделяющих частотные диапазоны. Информация о частоте заводится в психоакустическую модель для определения маскирующих частот. Следующим шагом является распределение имеющегося запаса бит между частотными диапазонами. При этом большее число бит отводится под диапазон с наибольшей немаскированной спектральной мощностью, меньшее — под немаскируемые диапазоны с меньшей спектральной мощностью, и совсем не отводятся биты под маскируемые диапазоны. Наконец, битовые последовательности шифруются с помощью кода Хаффмана (Huffman), который присваивает короткие коды числам, появляющимся наиболее часто, и длинные — появляющимся редко. Если вас заинтересовала эта тема и вы хотите узнать о ней побольше — обратитесь к книге Бранденбурга (Brandenburg, 1999).

7.4.2. Цифровое видео

Теперь, когда мы узнали об ухе все, пора перейти к глазу. Предвосхищая ваши вопросы скажу, что в следующем разделе мы не будем рассказывать вам о носе. У человеческого глаза есть одна особенность: когда изображение появляется на сетчатке, оно на сохраняется на ней на несколько миллисекунд, прежде чем уступить место другому. Если картинки сменяются со скоростью 50 изображений в секунду, глаз не заметит того, что изображение дискретно. Этот принцип используется всеми видеосистемами для отображения двигающихся картинок.

Самое простое представление видео — это последовательность кадров, каждый из которых состоит из набора пикселей, прямоугольных элементов, составляющих изображение. Каждый пиксель может быть отдельным битом, чтобы отображаться либо белым, либо черным. Однако качество такой системы ужасно. Попробуйте конвертировать цветную картинку в черно-белую (используйте именно черно-белый вариант, а не оттенки серого) при помощи вашего любимого графического редактора. Следующий шаг — это использование 8 бит на пиксель для представления 256 уровней серого. По этой схеме получается высококачественное черно-белое видео. Для цветного видео многие системы используют по 8 бит на каждый из основных компонентов цвета, красный, зеленый и синий (RGB). Такое представление возможно, так как любой цвет может быть представлен при помощи линейного наложения оттенков красного, зеленого и синего определенной интенсивности. При использовании 24 бит на пиксель мы получаем около 16 млн цветов, что гораздо больше, чем может различить человеческий глаз.

На цветном жидкокристаллическом дисплее компьютера или телевизора каждый отдельный пиксель состоит из разделенных промежутками подпикселей красного, зеленого и синего цветов. Картинки отображаются при помощи определения интенсивности подпикселей, а глаз смешивает цветовые компоненты.

Обычно картинки сменяются со скоростью 24 кадра в секунду (так же как на 35-мм кинопленке), 30 кадров в секунду (как на американском телевидении — система NTSC) и 25 раз в секунду (как на телевизорах с системой PAL, распространенных во всем мире). Если уж быть совсем точными, в США обычное телевидение вещает со сменой картинки 29,97 раза в секунду. Когда телевидение было черно-белым, кадры менялись 30 раз в секунду, но, когда был введен цвет, инженерам понадобилось добавить эту информацию к передаваемому потоку, так что они сократили частоту смены кадров до 29,97 раза в секунду. Компьютеры же должны менять кадры 30 раз в секунду. PAL был изобретен после того, как появился NTSC; там фактически используется 25,000 кадров в секунду. Чтобы в нашем рассказе не осталось пробелов, стоит упомянуть о третьей системй, SECAM, которая используется во Франции, в франкоговорящей части Африке и восточной Европе. Она появилась в восточной Европе из Восточной Германии, чтобы у тех, кто в ней жил, не было возможности смотреть западногерманское телевидение (PAL) и воспринимать чуждые идеи[1]. Но сейчас многие страны переключаются на PAL.

[1] Следует заметить, что существуют и другие версии истории распространения различных стандартов телевещания. В частности, версия, согласно которой SECAM изначально был разработан как раз во Франции. — *Примеч. ред.*

Вообще-то, для телевещания 25 кадров в секунду — это не достаточно хорошее качество для того, чтобы передавать плавные движения, так что изображения разбиваются на два поля (**fileds**) по четным и нечетным строкам развертки. Два поля (каждое с половиной разрешения) передаются последовательно, выдавая почти 60 (NTSC) или ровно 50 (PAL) полей в секунду. Такая система известна, как **чередование (interlacing)**. Видео, предназначенное для просмотра на компьютере, **последовательное (progressive)**, то есть чередование не используется, так как на видеокартах мониторов есть буферы, что позволяет помещать в буфер новое изображение 30 раз в секунду, при этом изображение на экране может обновляться 50 или даже 100 раз в секунду, чтобы избежать мерцания. У аналоговых телевизоров нет буфера, такого как у компьютеров. Когда на компьютере отображается видео с быстрыми движениями, использующее чередование, рядом с четко выделенными краями можно увидеть горизонтальные линии — этот эффект называется **комбингом (combing)**.

Размеры кадров, передаваемых через Интернет, сильно варьируются по той простой причине, что для больших фреймов требуется большая ширина полосы пропускания, а это не всегда доступно. Видео в низком разрешении может быть 320 на 240 пикселей, а полноэкранное — 640 на 480. Эти размеры относятся к первым компьютерным мониторам и телевидению в формате NTSC соответственно. **Соотношение размеров (aspect ratio)** или отношение ширины изображения в пикселях к высоте 4 : 3 такое же как у стандартного телевизора. Видеофайлы **HDTV (High-Definition TeleVision — телевидение в высоком разрешении)** могут загружаться в разрешении 1280 на 720 пикселей. У этих широкоэкранных файлов соотношение размеров равно 16 : 9, чтобы более точно соответствовать соотношению размеров для фильмов (3 : 2). Для сравнения можно привести стандартные DVD-видео. У них соотношение размеров обычно 720 на 480 пикселей, а у дисков высокого разрешения Blu-ray (работающих на основе сине-фиолетового лазера) обычно 1080 на 720 пикселей.

В Интернете количество пикселей уже отошло в историю, так как медиаплееры могут выдавать одну и ту же картинку в разных размерах. Видео является просто еще одним окном на экране компьютера, которое можно увеличить или уменьшить. Чем больше пикселей, тем выше качество изображения, так что оно не выглядит размытым при увеличении. Однако многие мониторы могут отображать картинки (а следовательно, и видео) с использованием даже большего количества пикселей, чем в HDTV.

Сжатие видео

Из нашей дискуссии о цифровом видео должно быть очевидно, что сжатие критично для пересылки видео через Интернет. Даже на передачу видео в стандартном качестве с размером картинки 640 на 480 пикселей, 24 битами информации о цвете на пиксель и 30 кадрами в секунду уходит больше 200 Мбит/с. Это серьезно превосходит скорость, с которой большинство компаний подключены к Интернету, не говоря уже о простых пользователях. И ведь пока мы говорим только об одном видеопотоке. Так как передача несжатого видео практически невозможна, по крайней мере через глобальную сеть, остается надеяться лишь на то, что возможно хорошее сжатие. К счастью, за последние несколько лет в этом направлении было проведено множество

исследований, так что появилось множество техник и алгоритмов сжатия, которые делают передачу видео возможной.

Для передачи видео через Интернет используется множество форматов, и патентованных, и стандартных. Самая популярная кодировка — это MPEG и ее формы. Это открытый формат использующийся в файлах с расширениями mpg, mp4, так же как и в файлах других контейнерных форматов. В этом разделе мы рассмотрим MPEG, чтобы выяснить, как сжимается видео. Для начала мы рассмотрим сжатие изображений в формате JPEG. Видео — это просто последовательность изображений (вкупе со звуком). Один из способов сжать видео — это сжать каждое из передаваемых изображений. В первом приближении MPEG — это просто сжатие с помощью JPEG каждого из передаваемых кадров и некоторые дополнительные опции, позволяющие исключить избыточность.

Стандарт JPEG

Стандарт **JPEG (Joint Photographic Experts Group — объединенная группа экспертов по машинной обработке фотографических изображений)** для сжатия полутоновых изображений (например, фотографий) был разработан группой экспертов, давшей ему название, которая работала от ITU, ISO, IEC и других организаций, отвечающих за стандарты. Этот формат широко используется (посмотрите на файлы с расширением jpg) и часто сжимает файлы в 10 и более раз.

JPEG определен в международном стандарте 10918. На самом деле он больше напоминает список покупок, чем отдельный алгоритм, но из четырех входящих в него методов нас в рамках нашего повествования интересует только один, последовательный метод с частичной потерей информации. Кроме того, мы рассмотрим то, каким образом JPEG обычно используется для кодирования 24-битных RGB-видеоизображений, выпустив некоторые детали, чтобы излишне не усложнять наш рассказ.

Алгоритм проиллюстрирован на рис. 7.20. Шаг 1 — подготовка блоков. Чтобы добавить конкретики, предположим, что на входе алгоритма JPEG — это изображение RGB с размерами 640 × 480, 24 бита на пиксель, как показано на рис. 7.21, *a*. RGB — это не лучшая модель цвета для сжатия. Глаз гораздо более чувствителен к **яркости (luminance)**, а не к **хроматическим данным (chrominance)**, то есть к цвету видеосигнала. Таким образом, мы сначала вычисляем яркость, *Y*, и две хроматические характеристики, *Cb* и *Cr*, компонентов *R*, *G* и *B*. Следующие формулы используются для 8-битных значений (от 0 до 255):

$$Y = 16 + 0{,}26R + 0{,}50G + 0{,}09B;$$
$$Cb = 128 + 0{,}15R - 0{,}29G - 0{,}44B;$$
$$Cr = 128 + 0{,}44R - 0{,}37G + 0{,}07B.$$

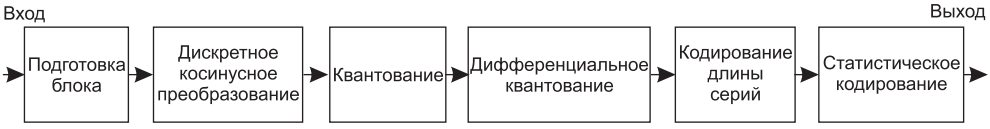


Рис. 7.20. Шаги последовательного кодирования JPEG с потерями

Для *Y*, *Cb* и *Cr* конструируются отдельные матрицы. Затем в матрицах *Cb* и *Cr* квадратные блоки по 4 пикселя усредняются и сводятся в один, и изображение сокращается до размеров 320 × 240. Сжатие идет с потерями, но глаз это едва ли заметит, так как главное для него — яркость. Тем не менее общий объем данных сокращается в 2 раза. Теперь из каждого элемента всех трех матриц вычитается 128, чтобы поместить ноль в центр диапазона. Наконец, каждая матрица разделяется на блоки 8 × 8. В матрице *Y* — 4800 блоков, в оставшихся двух — по 1200, как показано на рис. 7.21, *б*.

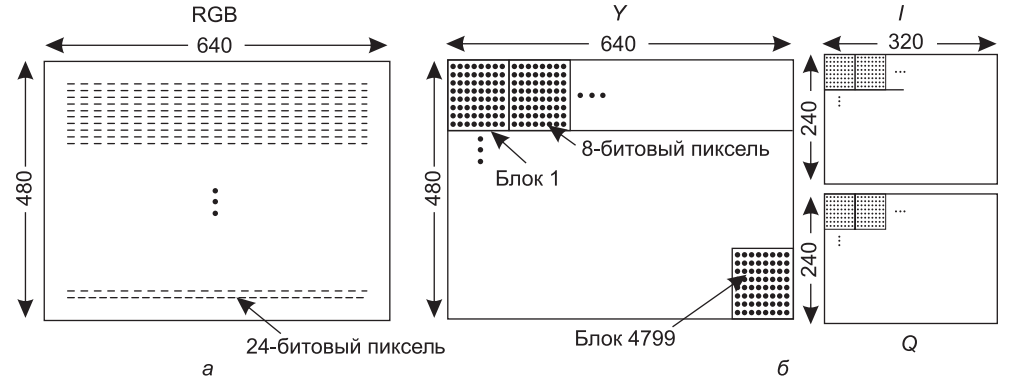


Рис. 7.21. Входные данные RGB (*а*). После подготовки блоков (*б*)

Второй шаг кодирования JPEG — **применение ДКП (DCT, Discrete Cosine Transformation — дискретное косинусное преобразование)** отдельно к каждому из 7200 блоков. Выход каждого ДКП — это матрица ДКП-коэффициентов 8 × 8. Элемент ДКП (0,0) — это среднее значение блока. Остальные элементы указывают на то, какая спектральная мощность у каждой пространственной частоты. Обычно эти элементы быстро уменьшаются до нуля при отдалении от начала координат (0,0), как показано на рис. 7.22.

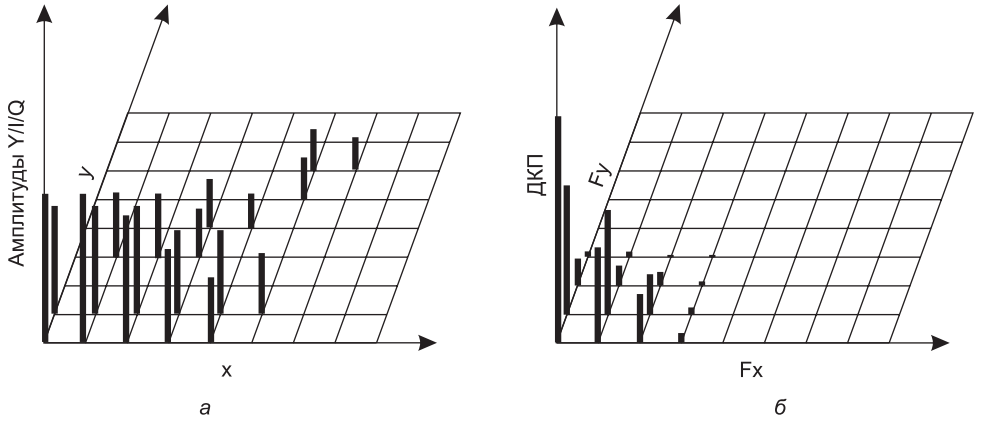


Рис 7.22. Один блок матрицы Y (*а*). Коэффициенты ДКП (*б*)

Как только ДКП завершено, кодирование JPEG переходит к шагу 3, который называется **квантованием (quantization)**. На этом этапе стираются наименее важные

коэффициенты ДКП. Эта трансформация с потерями проводится при помощи деления каждого из коэффициентов в матрице ДКП 8 × 8 на вес, взятый из таблицы. Если все веса равны 1, при трансформации ничего не происходит. Однако если веса резко возрастают, высокая разрешающая способность падает.

Пример этого шага приведен на рис. 7.23. На нем мы видим начальную матрицу ДКП, таблицу дискретизации и результат, полученный при помощи деления каждого элемента ДКП на соответствующий элемент таблицы дискретизации. Значения в этой таблице не являются частью стандарта JPEG. Каждое приложение должно предоставлять их самостоятельно, чтобы была возможность обеспечения сжатия с потерями.

ДКП-коэффициенты

150	80	40	14	4	2	1	0
92	75	36	10	6	1	0	0
52	38	26	8	7	4	0	0
12	8	6	4	2	1	0	0
4	3	2	0	0	0	0	0
2	2	1	1	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Квантованные коэффициенты

150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Таблица квантования

1	1	2	4	8	16	32	64
1	1	2	4	8	16	32	64
2	2	2	4	8	16	32	64
4	4	4	4	8	16	32	64
8	8	8	8	8	16	32	64
16	16	16	16	16	16	32	64
32	32	32	32	32	32	32	64
64	64	64	64	64	64	64	64

Рис 7.23. Вычисление квантованных коэффициентов ДКП

На четвертом шаге уменьшается значение (0,0) каждого блока (в левом верхнем углу), которое заменяется на величину разности с соответствующим элементом в предыдущем блоке. Так как эти элементы являются средними значениями соответствующих блоков, они должны меняться медленно, так что при переходе к разностным значениям идет уменьшение их значений. Для остальных значений разности не вычисляются.

На пятом шаге 64 элемента линеаризуются и ко всему массиву применяется групповое кодирование. Сканирование блока слева направо, а затем сверху вниз не соберет вместе все нули, так что используется зигзагообразная модель сканирования, как показано на рис. 7.24. В этом примере она выдает 38 последовательных нулей в конце матрицы. Эта строка может быть сокращена до одного указателя, говорящего, что этих нулей 38. Данная техника называется **групповым кодированием** или кодированием длины серий (**run-length encoding**).

Теперь у нас есть список чисел, который представляет собой картинку (в трансформированном виде). На шестом шаге числа для хранения или передачи кодируются по методу Хаффмана, отдавая часто встречающимся числам более короткие коды.

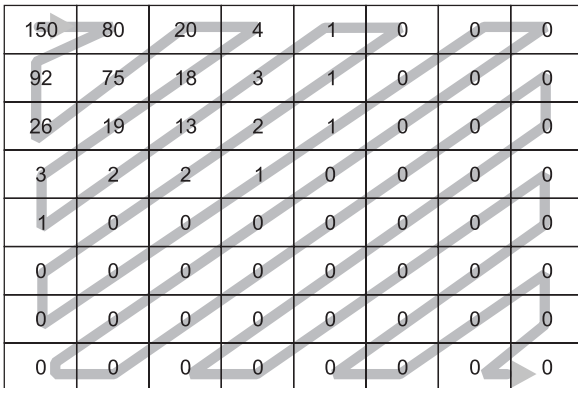


Рис. 7.24. Порядок, в котором передаются квантованные значения.

JPEG может казаться сложным, но именно таким он и является на самом деле. Тем не менее возможность сжатия вплоть до двадцатикратного стоит того. Декодирование изображения JPEG требует запуска алгоритма в обратном порядке. JPEG практически симметричен: на декодирование и на кодирование уходит одинаковое время. Но не все алгоритмы сжатия симметричны, как мы сейчас увидим.

Стандарт MPEG

Наконец, мы подошли к самой сути: стандартам **MPEG (Motion Picture Experts Group — экспертная группа по вопросам движущегося изображения)**. Хотя существует множество частных алгоритмов сжатия видео, эти стандарты определяют основные алгоритмы, используемые для сжатия видео. Они стали международными стандартами с 1993 года. Так как фильмы содержат и изображения и звук, MPEG может сжимать и аудио, и видео. Мы уже говорили о сжатии аудио и изображений, так что давайте теперь перейдем к сжатию видео.

Стандарт MPEG-1 (включающий MP3-аудио) был впервые опубликован в 1993 году, и он до сих пор широко используется. Его целью было производство выходной видеоинформации в таком же качестве, как у видеомагнитофона со сжатием в 40 раз, которая сможет передаваться со скоростью 1 Мбит/с. Такое видео подходит для широкого использования в Интернете на веб-сайтах. Не беспокойтесь, если вы не помните, что такое видеомагнитофон — MPEG-1 также использовался для хранения видео на CD. Если вы не застали и существования CD, нам придется перейти к MPEG-2. Этот стандарт был выпущен в 1996 году. Его разработали для сжатия видео в качестве телетрансляции. Сейчас этот стандарт очень распространен, так как является базовым для кодирования видео на DVD (а такое видео почти всегда попадает в Интернет) и для цифрового телевещания (DVB). Обычно видео в качестве DVD кодируется исходя из скорости передачи 4–8 Мбит/с.

В стандарте MPEG-4 есть два видеоформата. Первый из их, выпущенный в 1999 году, кодирует видео при помощи объектно-ориентированного представления. Это позволяет накладывать съемку на искусственно сконструированное изображение,

например, ведущего, рассказывающего о погоде, на карту. Такая структура позволяет программам легко взаимодействовать с видеоданными. Второй формат, выпущенный в 2003 году, называется **H.264** или **AVC (Advanced Video Coding — продвинутое кодирование видео)**. Его целью является кодирование видео с в 2 раза меньшей требуемой пропускной способностью, чем предыдущие кодировщики, сохраняя то же качество, чтобы передавать в сети видео в лучшем качестве. Этот кодировщик используется для видео в высоком качестве.

Эти стандарты отличаются множеством различных деталей. В более поздних стандартах также есть гораздо больше возможностей и опций кодирования, чем в ранних. Однако мы не будем погружаться в детали. По большей части, видеосжатие со временем улучшилось благодаря множеству маленьких усовершенствований, а не фундаментальному изменению алгоритма сжатия. Так что мы обрисуем все это в общем.

MPEG сжимает и аудио, и видео. Так как аудио- и видеокодировщики работают независимо, на приемнике необходима их синхронизация. Решение этой проблемы состоит во введении единого отсчета времени и передаче меток обоим кодировщикам. Эти метки включаются в закодированный выход и передаются приемнику, который таким образом может синхронизировать аудио- и видеопотоки.

Сжатие MPEG-видео использует два вида избыточности, присутствующие в фильмах: пространственную и временную. Пространственная избыточность может быть устранена при помощи кодирования каждого кадра отдельно с помощью JPEG. Этот подход используется не очень часто, как правило, когда нужен случайный доступ к каждому кадру (например, при редактировании видео). При этом варианте достигаются уровни сжатия JPEG.

Дополнительное сжатие может быть достигнуто при помощи использования того факта, что следующие друг за другом кадры часто почти одинаковые. Этот эффект не так полезен, как может показаться на первый взгляд, так как многие режиссеры делают переходы от одной сцены к другой каждые 3–4 с (подсчитайте сцены, сменяющиеся за пару минут в любом фильме). Тем не менее массив из 75 или более похожих кадров предполагает, что видео может быть сжато гораздо сильнее, чем при кодировании каждого JPEG отдельно.

Для сцен, в которых камера и задний план статичны, а один-два актера медленно передвигаются, почти все пиксели соседних кадров будут совпадать. В таком случае просто извлечение каждого кадра из предыдущего и запуск JPEG с различиями сильно упростят ситуацию. Однако для сцен, где камера идет за актером или приближается, эта техника мало чем нам поможет. Необходим способ компенсировать это движение. Именно это и делает MPEG, и в этом заключается главное отличие MPEEG от JPEG.

На выходе MPEG получается 3 вида кадров:

1. I- (Intracoded — закодированные в себе) кадры: содержащие сами себя сжатые неподвижные картинки.
2. P- (Predictive — предсказательные) кадры: поблочная разница с предыдущим кадром.
3. B- (Bidirectional — двунаправленные) кадры: поблочная разница с предыдущим и последующим кадрами.

Кадры первого типа, I-кадры, являются неподвижными картинками. Они могут кодироваться при помощи JPEG или чего-то похожего. Периодически такие кадры должны встречаться (например, пару раз в секунду) по трем причинам. Во-первых, MPEG может использоваться для многоадресной передачи, когда количество пользователей может увеличиться или уменьшиться. Если все кадры, начиная со второго, зависят от предшествующих, тот, кто пропустил первый кадр, не сможет декодировать ни один из следующих. Во-вторых, если какой-то кадр был получен с ошибкой, последующие будут нечитаемыми. В третьих, без кадров первого типа декодеру придется рассчитывать каждый кадр, чтобы картинка не сбивалась.

P-кадры, напротив, кодируют различия между кадрами. Они основаны на идее макроблоков (**macroblocks**), которые покрывают, например, 16 × 16 пикселей для яркости и 8 × 8 для цветности. Макроблок кодируется при помощи поиска на предыдущем кадре такого же или немного отличающегося. Пример того, где могут быть полезны P-кадры, приведен на рис. 7.25. На нем мы видим три последовательных кадра с одинаковым задним планом, но разной позицией героя. Макроблоки, содержащие человека, будут смещены на какое-то расстояние и их придется отыскать.

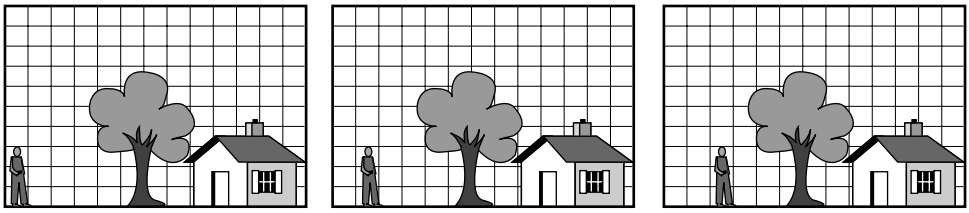


Рис. 7.25. Три последовательных кадра

Стандарты MPEG не определяют то, на каком расстоянии должен вестись поиск или насколько должны быть похожи области, чтобы считать их условно совпадающими. Все зависит от каждого отдельного случая его применения. Например, в некоторой реализации поиск макроблока может вестись на той же позиции и на позициях, отстоящих от нее на $\pm\Delta x$ по оси x и $\pm\Delta y$ в направлении y. Для каждой позиции может вычисляться количество совпадений в матрице яркости. Позиция с наибольшим числом совпадений (при условии, что это число перекроет минимально необходимое) будет объявлена победителем. В ином случае будет считаться, что макроблок отсутствует. Конечно, возможно внедрение и гораздо более сложных алгоритмов.

Если макроблок найден, он кодируется при помощи его разницы с тем же макроблоком в предыдущем кадре (по яркости и хроматическим характеристикам). Обычно эти матрицы различий затем подвергаются дискретному косинусному преобразованию, квантованию, кодированию длины серий и кодированию по методу Хаффмана.

Значение макроблока в выходном потоке сопровождается вектором перемещения (на сколько макроблок переместился в каждом направлении), а затем идет кодирование этой разницы. Если макроблок не найден в предыдущем кадре, его текущее значение кодируется как в I-кадре. Конечно, этот алгоритм крайне ассиметричен. На практике он может проверять каждую спорную позицию в предыдущем кадре в отчаянной попытке локализовать каждый макроблок, вне зависимости от того, как

далеко он переместился. Этот подход минимизирует закодированный поток MPEG за счет очень медленного кодирования. Он подходит для однократного кодирования библиотеки фильмов, но ни в коем случае не для кодирования видеоконференций.

В каждом отдельном случае по-своему определяется, что такое «найденный» макроблок. Благодаря этому качество и скорость алгоритмов кодирования варьируются, но подходящий выходной MPEG получается всегда.

Пока декодирование MPEG прямое. Декодирование I-кадров похоже на декодирование изображений JPEG. Декодирование P-кадров требует от декодера буферизовывать предыдущие кадры, чтобы построить следующие в отдельном буфере из полностью декодированных макроблоков и макроблоков, содержащих различия с предыдущим кадром. Новый кадр собирается макроблок за макроблоком.

B-кадры похожи на P-кадры, но в случае их применения эталонный макроблок может находиться как в предыдущем, так и в последующем кадре. Эта дополнительная свобода позволяет лучше компенсировать движение. Это полезно, например, когда объекты на некоторое время перекрываются. Чтобы закодировать B-кадр, кодировщик должен держать в памяти последовательность кадров: прошедшие, текущий (кодируемый) и следующий. Декодирование также несколько усложняется и занимает больше времени. Это происходит из-за того, что отдельный B-кадр не может быть декодирован, пока не декодированы последующие кадры, от которых он зависит. Таким образом, хотя B-кадры позволяют максимально сжать видео, они не всегда используются из-за сложности и необходимости специфической буферизации.

Стандарты MPEG содержат много улучшений этих техник, что позволяет достичь очень хорошего сжатия. AVC может использоваться, чтобы сжать видео в соотношении 50 : 1, что сокращает требования по пропускной способности канала в те же 50 раз. Чтобы узнать о AVC больше, почитайте книгу Салливана и Виганда (Sullivan and Wiegand, 2005).

7.4.3. Потоковая передача сохраненных медиафайлов

Теперь давайте перейдем к сетевым приложениям. В первую очередь мы поговорим о потоковой передаче медиаинформации, которая уже хранится в файлах. Наиболее известным примером можно считать просмотр видео через Интернет. Это одна из форм **VoD (Video on Demand — видео по запросу)**. Другие формы видео по запросу используют для передачи фильмов провайдерскую сеть, которая не является частью Интернета (например, кабельная сеть).

В следующем разделе мы рассмотрим передачу медиа в реальном времени, например передача телесигнала через IP (IPTV) и интернет-радио. Затем мы рассмотрим третий случай — конференции в реальном времени. Примером являются звонки с передачей голоса через IP или видеоконференции через Skype.

В этих случаях требования к тому, как мы можем передавать аудио и видео по сети, крайне высоки, так как мы должны уделять особое внимание задержкам и ее варьированию (неустойчивости синхронизации).

В Интернете крайне много сайтов с музыкой и видео, которые транслируют сохраненные медиафайлы. На самом деле, проще всего работать с сохраненными медиафай-

лами *не* транслируя их. Представьте себе, что вы хотите создать сайт проката видео типа iTunes от Apple. Обычный веб-сайт предоставит пользователям возможность загрузить, а потом посмотреть видео (после оплаты, конечно). Эта последовательность шагов показана на рис. 7.26. Мы остановимся на них подробно, чтобы была лучше видна разница со следующим примером.

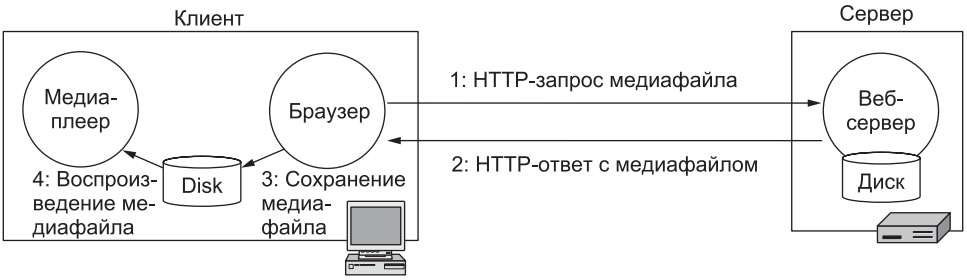


Рис. 7.26. Проигрывание медиафайлов при помощи Всемирной паутины через загрузку

Браузер начинает действовать, когда пользователь щелкает на названии фильма. На первом шаге отсылается HTTP-запрос фильма на веб-сервер, на который указывает ссылка фильма. На втором шаге сервер получает фильм (который представляет собой обычный файл в формате MP4 или каком-нибудь другом) и отсылает ее браузеру.

По MIME-типу файла (например, *video/mp4*) браузер определяет способ его воспроизведения. В нашем случае используется медиаплеер, который показан как вспомогательное приложение, хотя это также может быть и плагин. Браузер сохраняет весь фильм, чтобы запустить файл с диска (шаг 3). Затем запускается медиаплеер, которому передается имя полученного файла. Наконец, на четвертом шаге медиаплеер начинает читать файл и проигрывать фильм.

В принципе, такой подход совершенно корректен. Фильм пользователь увидит. Загрузка остается простой загрузкой файла, никаких сетевых проблем реального времени. Единственная серьезная проблема заключается в том, что вся видеозапись должна быть предварительно передана по сети. Большинство клиентов не хотят ждать час, прежде чем они смогут посмотреть «видео по заказу». С этой моделью возникают проблемы, даже когда она применяется к аудио. Если музыкальный файл занимает 4 Мбайт (типичный размер аудиофайла с песней в формате MP3), а пользователь может принимать информацию со скоростью 1 Мбит в секунду, он будет наслаждаться тишиной в течение почти 30 с, прежде чем услышит музыку. Далеко не все меломаны приходят от этого в восторг.

Как обойти эту проблему, не внося изменений в работу браузера? Видеосайты могут использовать схему, показанную на рис. 7.27. Файл, связанный гиперссылкой с названием фильма, на самом деле является не видеофайлом, а **метафайлом (metafile)**. Метафайл обычно очень короткий. Он включает название (и, возможно, несколько ключевых характеристик). В типичной ситуации он состоит всего лишь из одной текстовой строки, которая выглядит примерно так:

```
rtsp://joes-movie-server/movie-0025.mp4
```

Браузер, как обычно, получает страницу. Затем следуют шаги 1 и 2. Затем он запускает проигрыватель и передает ему, как обычно, имя временного файла (шаг 3).

Проигрыватель видит, что временный файл содержит URL того места, где можно получить фильм. Он соединяется с сервером *joes-video-server* и запрашивает видеоролик (шаг 4). Затем фильм передается на медиаплеер (шаг 5). Преимущество этого способа заключается в том, что медиаплеер запускается быстро, сразу после загрузки короткого метафайла. После этого браузер не используется. Медиафайл передается плееру напрямую, и просмотр можно начинать до того, как будет загружен весь файл.

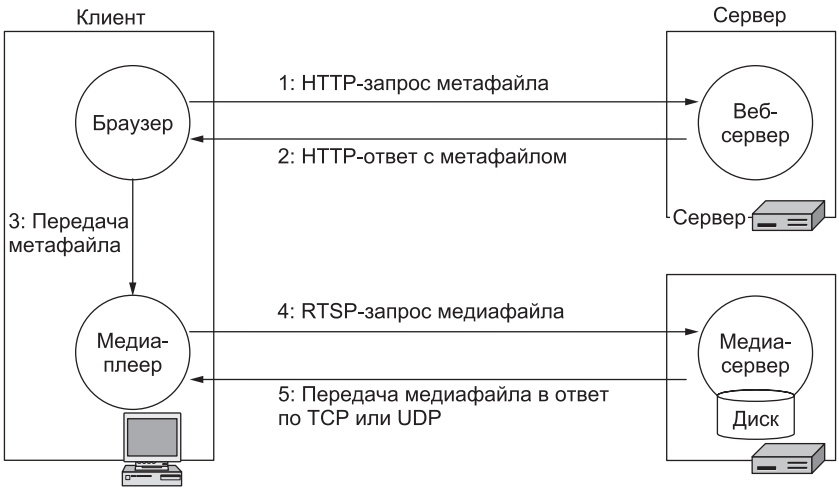


Рис. 7.27. Проигрывание медиафайлов при помощи Всемирной паутины и медиасервера

На рис. 7.27 мы показали два сервера, так как сервер, указанный в метафайле, часто не совпадает с веб-сервером, содержащим ссылку на метафайл. Более того, обычно это даже не HTTP-сервер, а специализированный мультимедийный сервер. В приведенном примере этот сервер использует протокол **RTSP (Real Time Streaming Protocol — потоковый протокол реального времени)**, это становится понятно при взгляде на ссылку — она начинается с названия схемы, *rtsp*.

Проигрыватель мультимедиа (медиаплеер) решает следующие 4 основные задачи.

1. Управление интерфейсом пользователя.
2. Обработка ошибок передачи.
3. Распаковка сжатых данных.
4. Устранение неустойчивости синхронизации (джиттера).

Большинство современных программ для воспроизведения мультимедиа имеют привлекательный интерфейс. Часто они имеют панели управления с изменяемым внешним видом, называемые «скинами» **(skins)**, которые пользователь может менять. Как мы уже сказали, в задачи проигрывателя входит общение с пользователем.

Другие задачи связаны с сетевыми протоколами. Мы поговорим о всех них, начиная с преодоления ошибок при передаче. Способ их преодоления зависит от того, используется ли транспорт на основе TCP, такой как HTTP, или основанный на UDP, такой как RTP. На практике используются оба. При использовании передачи на основе TCP медиаплееру не придется исправлять ошибки, так как TCP обеспечивает высокую надежность при помощи повторной передачи. Это довольно простой способ

преодоления ошибок, по крайней мере, для медиаплеера, но он усложняет удаление джиттера на более позднем шаге. Передача, основанная на UDP, такая как RTP, предлагает альтернативный вариант. Мы рассмотрели его в главе 6. Эти протоколы не используют повторные передачи. Таким образом, потеря пакета из-за затора или ошибки при передаче приведет к тому, что часть медиа не будет доставлена. С этой проблемой должен справляться медиаплеер. Давайте разберем проблему, с которой мы столкнулись. Потери при передаче являются проблемой, так как пользователи явно не обрадуются большим пропускам в песнях или фильмах. Однако эта проблема не так серьезна, как потери при передаче обычного файла, так как потеря небольшой части медиа не поставит крест на всей передаче медиафайла. В случае с видео пользователь вряд ли заметит, если за секунду будет проиграно 24 вместо 25 кадров. В случае с аудио небольшие прерывания могут маскироваться звуками, близкими по времени. И пользователь не заметит подмены, если не будет *очень* внимательно прислушиваться.

Однако все это справедливо только в том случае, если прерывания в передаче очень короткие. Ошибки при передаче или получении часто приводят к тому, что теряется весь пакет или даже несколько пакетов. Чтобы сократить влияние потери пакетов при передаче медиа, может быть использовано две стратегии: FEC и интерливинг. Мы опишем каждую из них.

FEC (Forward Error Correction — заблаговременное исправление ошибок) является просто кодированием исправления ошибок, которое мы рассмотрели в главе 3, применительно к прикладному уровню. Примером является межпакетный контроль четности (Shacham и McKenny, 1990). Для каждых четырех отсылаемых пакетов данных может быть сконструирован и отослан пятый **пакет контроля по четности (parity packet)**, как показано на рис. 7.28 (пакеты *A*, *B*, *C* и *D*). Пакет контроля по четности, *P*, содержит избыточную информацию по принципу суммарного контроля по четности или исключающего ИЛИ, при этом учитывается сумма битов в каждом из четырех пакетов данных. Остается лишь надеяться, что будут доставлены все пакеты в большинстве групп из пяти пакетов. В этом случае пакет контроля по четности просто игнорируется на принимающей стороне. А если теряется только контрольный пакет, проблемой это не становится.

Рис. 7.28. Использование пакета с контролем по четности для предотвращения потерь

Однако иногда при передаче может быть утерян какой-либо пакет, как, например *B* на рис. 7.28. Медиаплеер получает только 3 пакета данных, *A*, *C* и *D*, а также контрольный пакет, *P*. При использовании данной схемы биты отсутствующего пакета

могут быть восстановлены при помощи данных пакета *P*. А точнее, если символом «+» обозначить «исключающее ИЛИ» или «сложение по модулю2», *B* может быть реконструирован по формуле $B = P + A + C + D$ при помощи свойств исключающего ИЛИ (то есть $X + Y + Y = X$).

FEC может сократить уровень потерь, заметных для медиаплеера, восстанавливая некоторые из утерянных пакетов, но он работает только до определенного уровня. Если теряются два пакета из группы, содержащей пять, утерянную информацию восстановить не удастся. Еще одной важной особенностью FEC является та цена, которую мы платим за защиту от потерь. Каждая четверка пакетов превращается в пятерку, так что требования по ширине канала возрастают на 25 %. Время ожидания декодирования также возрастает, так как нам придется ожидать доставки контрольного пакета перед тем, как сможет быть реконструирован пакет, пришедший до этого.

В технике, упомянутой выше, также есть интересный нюанс. В главе 3 мы описали использование контроля по четности для обнаружения ошибок. Здесь мы говорим об исправлении ошибок. Как возможно и то и другое? Ответ заключается в том, что на сей раз нам известно, который из пакетов потерян. Потерянные данные называются **разрушенными (erasure)**. В главе 3, когда мы рассматривали кадр, в котором несколько битов содержали ошибки, мы не знали, какие именно. Этот случай сложнее, чем разрушенные пакеты. Таким образом, при потере пакетов пакет контроля по четности может обеспечить исправление ошибок, а в случае, описанном в главе 3, возможно лишь обнаружение ошибок. Скоро, когда речь пойдет о многоадресной передаче, мы познакомимся еще с одной неожиданной выгодой, которую можно извлечь из контрольного пакета.

Вторая стратегия называется **интерливингом (interleaving — чередование)**. Этот подход базируется на смешивании или интерливинге порядка медиа перед передачей и сортировке или деинтерливинге при его получении. Таким образом, благодаря перемешиванию не будет потеряно следующих друг за другом пакетов, и один большой разрыв при проигрывании медиа не образуется. Например, пакет может содержать 220 сэмплов стерео, в каждом по паре 16-битных чисел, чего обычно достаточно для 5 мс проигрывания музыки. Если эти образцы были посланы по порядку, потеря пакета повлечет перерыв в проигрывании на 5 мс. Вместо этого передача осуществляется так, как показано на рис. 7.29. Все четные сэмплы интервала в 10 мс посылаются в одном пакете, а нечетные во втором. Теперь потеря третьего пакета означает не пропуск в музыке в 5 мс, а чередование пустых и заполненных музыкой коротких промежутков в течение 10 мс. С потерей можно легко справиться, если плеер использует интерполяцию, учитывая предыдущий и следующий образцы. В результате мы получим временную потерю в разрешении на 10 мс, но значительного прерывания не будет.

Эта схема интерливинга работает только при отсутствии сжатия. Однако схема интерливинга (для промежутков времени, а не отдельных сэмплов) может также применяться после сжатия, если есть возможность обнаружить границы сэмплов в сжатом потоке. RFC 3119 предлагает схему, которая работает со сжатым аудио.

Интерливинг является привлекательной техникой, так как при его использовании не требуется дополнительной пропускной способности канала пропускания в отличие от FEC. Однако он, так же как и FEC, провоцирует увеличение времени ожидания, так

как необходимо ожидать доставки нескольких пакетов (для того чтобы сэмплы можно было выстроить по порядку).

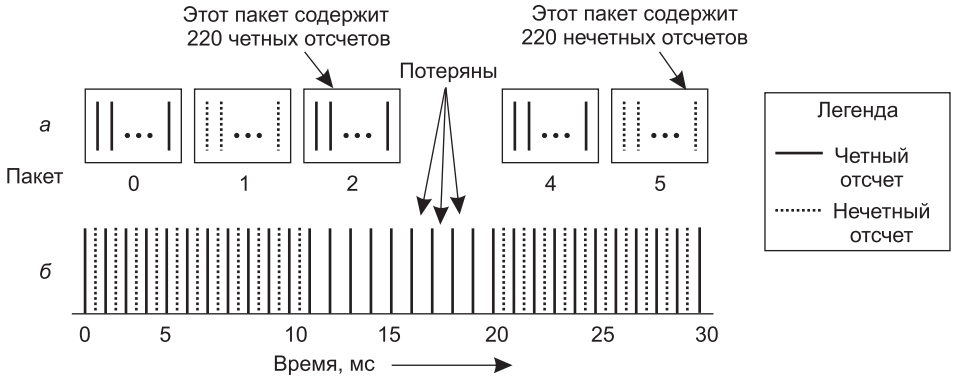


Рис. 7.29. Когда в пакетах передаются только четные или нечетные примеры, потеря пакета временно ухудшает разрешение, но не создает перерыва

Третья задача медиаплеера — восстановление сжатых данных. Хотя эта задача интенсивна в плане вычислений, она достаточно проста. Сложность заключается лишь в декодировании медиа, если сетевой протокол не исправляет ошибки при передаче. Во многих схемах сжатия более поздние данные не могут быть декодированы, если не декодированы более ранние, использованные при их сжатии. При передаче данных на основе UDP пакеты могут теряться. Таким образом, процесс кодирования должен быть разработан так, чтобы позволить декодирование, несмотря на потерю пакетов. Именно из-за этого требования используются I-, P- и B-кадры. Каждый I-кадр может быть декодирован независимо от других, чтобы преодолеть потерю любого из предыдущих кадров.

Четвертая задача — это устранение джиттера, бича всех систем, работающих в реальном времени. Общее решение, которое мы описали в разделе 6.4.3, заключается в использовании буфера. Все потоковые системы начинают с буферизации 5–10 с медиа перед началом проигрывания, как показано на рис. 7.30. При проигрывании медиа регулярно извлекается из буфера, таким образом, чтобы аудио было отчетливым, а видео не подрагивало. Задержка на старте дает буферу шанс не опуститься ниже **нижнего предела (low-water mark)**. Смысл состоит в том, что теперь данные должны приходить достаточно регулярно для того, чтобы буфер никогда не оказывался пустым. Если же это случится и буфер опустеет, проигрывание будет остановлено. Плюс буферизации состоит в том, что если новые данные запаздывают из-за заторов, буферизованные медиаданные позволят продолжить нормальное воспроизведение, пока не подоспели новые данные и буфер не пополнился.

То, сколько информации должно быть буферизовано и как быстро медиасервер наполняет буфер, зависит от сетевых протоколов. Есть много вариантов. Важным фактором реализации является то, какой транспорт используется, на основе TCP или UDP.

Предположим, что используется транспорт на основе UDP, такой как RTP. Далее предположим, что ширина полосы пропускания достаточна для передачи пакетов

с медиасервера на медиаплеер с небольшими потерями и также небольшим количеством дополнительной информации. В этом случае пакеты могут пересылаться с той же скоростью, с которой проигрывается медиа. Каждый пакет будет передаваться по сети и после задержки при начале воспроизведения прибывать в нужное время, чтобы медиаплеер мог проигрывать медиа. Необходимо буферизовать лишь небольшую часть информации, так как задержка постоянна. Если используется интерливинг или FEC, требуется буферизовать несколько бóльшую часть, по крайней мере, набор пакетов, которые невозможно декодировать по отдельности. Однако количество буферизованной информации увеличится лишь незначительно.

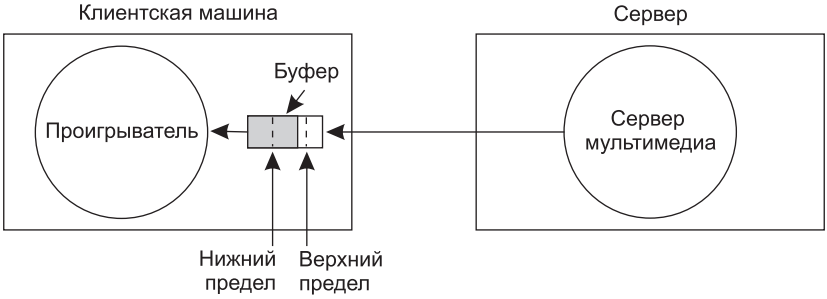


Рис. 7.30. Медиаплеер буферизует входную информацию с медиасервера и проигрывает медиа из буфера, а не напрямую из сети

К сожалению, этот сценарий нереален по двум причинам. Во-первых, из-за того, что ширина канала варьируется в зависимости от сетевых маршрутов, так что для медиасервера не очевидно, достаточна ли ширина полосы пропускания, пока не начнется передача. Простое решение заключается в кодировании медиа в нескольких разрешениях и предоставление пользователю возможности выбора разрешения, которое больше подходит для скорости его Интернета. Часто существуют только два уровня: высокое качество, скажем, 1,5 Мбит/с или больше, и низкое качество, скажем, в 512 Кбит/с или меньше.

Во-вторых, будет небольшой джиттер, или разница в том, сколько времени уходит у части медиа на то, чтобы пройти путь от передающего сервера до медиаплеера. Этот джиттер происходит по двум причинам. В сети часто передается несколько потоков, соревнующихся за место под солнцем. Это происходит из-за того, что пользователь продолжает просматривать веб-страницы, смотря фильм на одной из них. Такой трафик вызовет колебания в скорости получения медиа. Более того, нам важна доставка видеокадров и отрезков аудио, а не пакетов. Благодаря компрессии видеокадры могут быть больше или меньше в зависимости от их содержания. На передачу движения обычно требуется больше бит, чем на передачу спокойного пейзажа. Если пропускная способность сети постоянна, количество доставленного медиа за определенный временной период может варьироваться. Чем больше джиттер или варьирование задержки, тем больше информации должно храниться в буфере.

Теперь предположим, что транспорт, основанный на TCP, такой как HTTP, используется для отсылки медиа. При помощи повторных передач и ожидания доставки пакетов, чтобы выстроить их по порядку, TCP увеличит джиттер в медиаплеере,

возможно даже значительно. В результате нужен буфер большего размера и большее количество хранящейся в нем информации. Однако в этом есть и преимущество. TCP будет отсылать данные с той скоростью, с которой они могут передаваться по сети. Иногда воспроизведение может быть приостановлено, если при передаче возникли потери. Но в большей части случаев сеть сможет доставить медиа быстрее, чем плеер проигрывает файл. В эти периоды будет заполняться буфер, что поможет справиться с задержками в дальнейшем. Если сеть значительно быстрее, чем требуется для проигрывания медиа, а часто так и бывает, буфер быстро заполнится при запуске медиаплеера и вскоре ему уже не будет грозить опустошение.

При использовании TCP или UPD и скорости передачи, которая выше скорости, требуемой для воспроизведения, вопрос состоит в том, насколько большую часть медиафайла плеер подгрузит в буфер. Часто загружается весь файл целиком.

Однако загрузка такой большой части не является необходимостью, требует много места для хранения и не обязательно предотвращает опустошение буфера. Когда это нежелательно, решением является установка проигрывателем **верхнего предела (high-water mark)** заполнения буфера. Суть в том, что сервер выдает данные лишь до тех пор, пока буфер не заполнится до верхнего предела. После этого проигрыватель просит сервер приостановить передачу. Поскольку данные будут продолжать прибывать в течение времени передачи запроса приостановки, расстояние между верхним пределом и концом буфера должно быть больше некоторого X — количества данных, которые успеют передаться за этот промежуток времени. X соответствует задержкам в канале, возникающим вследствие ограниченной пропускной способности (то есть произведению пропускной способности на задержку). После приостановки сервера буфер начнет опустошаться. Когда количество данных в нем достигнет нижнего предела, проигрыватель попросит сервер мультимедиа возобновить передачу. Чтобы предотвратить опустошение буфера, при расчете нижнего предела также должно приниматься во внимание произведение пропускной способности на задержку, и в соответствии с ним должен рассчитываться момент, когда на сервер нужно отослать запрос на возобновление передачи.

Чтобы начинать и останавливать поток медиа, проигрыватель должен осуществлять удаленное управление сервером. Оно обеспечивается протоколом RTSP, предоставляющим соответствующий механизм управления. Он определен в RFC 2326. Кроме начала и остановки воспроизведения, файл может прокручиваться назад и вперед до определенной позиции, могут проигрываться определенные части, а также использоваться ускоренное или замедленное проигрывание. Однако этот стандарт не определяет поток данных, хотя обычно это RTP через UDP или RTP через HTTP через TCP. Основные команды RTSP приведены в табл. 7.16. Они представлены в простом текстовом формате, как HTTP-сообщения, и обычно передаются при помощи TCP. RTCP может также работать через UDP, так как каждая команда подтверждается (и может быть повторно отправлена, если она не подтверждена).

Несмотря на то что кажется, что TCP плохо подходит для трафика в реальном времени, он часто используется на практике. Основная причина заключается в том, что он проще проходит через межсетевые экраны, чем UDP, особенно если он запускается через HTTP-порт.

Таблица 7.16. Команды RTSP, посылаемые проигрывателем на сервер

Команда	Действие сервера
DESCRIBE	Перечисляет параметры мультимедийных данных
SETUP	Устанавливает логическое соединение между проигрывателем и сервером
PLAY	Начинает отправлять данные клиенту
RECORD	Начинает прием данных от клиента
PAUSE	Приостанавливает передачу данных
TEARDOWN	Удаляет логическое соединение

Большинство администраторов настраивают межсетевые экраны таким образом, чтобы защитить сети от нежелательного проникновения в нее извне. Почти всегда разрешается установка TCP-соединений с удаленного порта 80 (HTTP для Всемирной паутины). Блокирование этого порта быстро выливается в недовольство клиентов. Однако большинство других портов блокируется, включая RSTP и RTP, которые, помимо других, используют и порты 554 и 5004. Таким образом, простейший способ передать сигнал через межсетевой экран — это заставить веб-сайт притвориться HTTP-сервером, отсылающим обычные HTTP-ответы (по крайней мере, для межсетевого экрана). Есть и другие преимущества TCP. Так как он обеспечивает надежность, он предоставляет клиенту точную копию медиа. Это позволяет пользователю легко перемотать файл на уже просмотренный кадр, не беспокоясь о потере данных. Наконец, TCP буферизует максимальную часть файла с максимально возможной скоростью. Когда место в буфере дешево (например, если для хранения используется диск), медиаплеер может загружать медиа по ходу просмотра. Когда загрузка завершена, пользователь может смотреть фильм без перерывов, даже если пропадает связь. Это качество полезно при использовании мобильных телефонов, так как возможность подключения может резко меняться при движении.

Недостаток TCP — это увеличение времени ожидания при начале просмотра (из-за запуска TCP), а также более высокий нижний предел. Однако это редко является проблемой, в том случае, если скорость передачи значительно превышает скорость, требуемую для воспроизведения.

7.4.4. Передача медиа в реальном времени

Не только записанное видео крайне популярно во Всемирной паутине. Передача медиа в реальном времени также пользуется большой популярностью. Как только стало возможным передавать аудио и видео через Интернет, коммерческие радио- и телестанции воспользовались этим. Вскоре появились студенческие станции, вещающие через Интернет. И студенты начали вести собственные программы.

Сегодня люди и компании всех размеров передают аудио и видео. Вещание в реальном времени стало колыбелью инноваций, возникли новые стандарты и технологии. Живое вещание через Интернет используется основными телестанциями. Его называют **IPTV (IP TeleVision — IP-телевидение)**. Такое же вещание используется и радиостанциями, например BBC. Оно называется **интернет-радио**. И IPTV, и ин-

тернет-радио вещают на весь мир, освещая различные события, от показов мод до мировых чемпионатов по футболу и отборочных состязаний по крикету. Живое вещание через IP используется провайдерами кабельного телевидения как технология построения собственных вещательных систем. И она часто используется малобюджетными проектами от сайтов для взрослых до зоопарков. С современной технологией практически каждый может начать живое вещание быстро и дешево.

Один из подходов к живому вещанию основан на записи программ на диск. Зрители могут подсоединиться к архивам сервера, выбрать любую программу, загрузить и прослушать ее. **Подкаст (podcast)** — это файл, полученный таким образом. Для программ, идущих по расписанию, также можно хранить контент после окончания передачи, так что архив будет отставать от живого вещания где-то на полчаса или даже меньше.

На самом деле, этот подход полностью совпадает с тем, что мы обсуждали касательно передачи мультимедиа. Он легко применим, все упомянутые нами техники работают в его рамках, и пользователи могут выбирать нужную им программу из архива.

Другой подход заключается в живом вещании через Интернет. Зрители подключаются к идущему медиапотоку, так же как к идущей по телевидению передаче. Однако медиаплееры предоставляют дополнительные возможности, такие как приостановка или перемотка назад. Живой медиаконтент будет продолжать передаваться и буферизовываться плеером, пока пользователь не будет готов к просмотру. Со стороны браузера это выглядит в точности как передача сохраненного видео. Плееру не важно, отсылается ли контент в реальном времени, или же его источником является файл. Обычно плеер и не сможет этого определить (единственное отличие при живой передаче будет заключаться в отсутствии возможности перемотать контент вперед).

Учитывая сходство механизма с тем, что мы обсуждали выше, большая часть нашей дискуссии будет применима и к этому случаю, но есть и важные различия.

Важно отметить, что все еще есть необходимость в буферизации на стороне клиента, чтобы сгладить неустойчивость синхронизации (джиттер). На самом деле, при передаче контента в реальном времени часто потребуется буферизовывать бóльшую часть данных (и это не с вязано с тем, что пользователь может приостанавливать воспроизведение). Когда передается файл, медиа может отправляться с более высокой скоростью, чем требуется для воспроизведения. В этом случае буфер быстро заполнится, что позволит компенсировать джиттер (и плеер остановит поток, если буферизация более не требуется). В отличие от этого при передаче контента в реальном времени скорости приема (и воспроизведения) совпадают со скоростями передачи (и генерации) контента. Быстрее контент отправляться не может. Следовательно, буфер должен быть больше, чтобы справиться со всем возможным джиттером. Как правило, задержки начала воспроизведения на 10–15 с обычно достаточно, так что это не является серьезной проблемой.

Другое важное различие заключается в том, что события, передающиеся в реальном времени, обычно одновременно просматриваются сотнями тысяч зрителей. В этом случае решением проблемы является использование групповой адресации. Этого не требуется при потоковой передаче сохраненных медиафайлов, так как пользователи обычно запрашивают разный контент в разное время. Передача множеству пользователей, таким образом, состоит из многих отдельных сессий передачи, протекающих в одно и то же время.

Схема многоадресной потоковой трансляции работает следующим образом. Сервер отправляет каждый пакет единожды, используя многоадресную передачу по IP группе адресатов (IP multicast). Все клиенты, желающие подключиться к передаче, присоединяются к группе при помощи IGMP, а не отсылая RTSP-сообщения на медиасервер, так как медиасервер уже отсылает живой поток (кроме случая, когда присоединяется первый зритель). Что и нужно для того, чтобы поток получался локально.

Так как многоадресная передача является сервисом доставки от одного ко многим, медиа передается в RTP-пакетах через UDP-транспорт. TCP работает только между одним отправителем и одним получателем. Так как UDP не обеспечивает надежности, некоторые пакеты могут быть потеряны. Чтобы сократить уровень потерь до приемлемого, мы можем использовать FEC и интерливинг, как и раньше.

В случае использования FEC возникает полезное взаимодействие с многоадресной передачей, как показано в примере с контрольным пакетом на рис. 7.31. Когда пакеты передаются на множество адресов, разные клиенты могут терять разные пакеты. Например, клиент 1 потерял пакет *B*, клиент 2 потерял контрольный пакет *P*, клиент 3 потерял пакет *D*, а клиент 4 не потерял ни одного пакета.

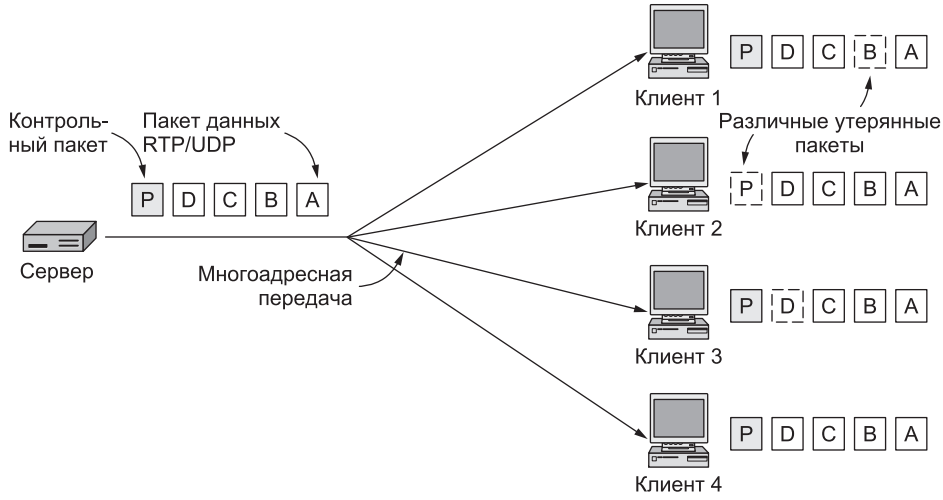


Рис. 7.31. Многоадресная передача медиа с контрольным пакетом

Однако, несмотря на то что клиенты потеряли разные пакеты, в данном примере все пользователи смогут восстановить утерянную информацию. Все что требуется — это чтобы ни один из клиентов не терял больше одного пакета, тогда отсутствующий сможет быть восстановлен при помощи информации из полученных. Нонненмахер и др. (1997) описали, как можно использовать эту идею для повышения надежности.

Для сервера с большим количеством клиентов многоадресная передача медиа в RTP- и UDP-пакетах является самым эффективным способом. Иначе сервер должен передавать *N* потоков, когда у него есть *N* клиентов, что потребует очень большой пропускной способности сети со стороны сервера.

Вы может удивитесь, но на практике Интернет так не работает. Обычно каждый пользователь устанавливает TCP-соединение с сервером, и медиа передается по этому соединению. Для клиента это выглядит так же, как передача сохраненных медиафайлов. И так же как при ней, существует несколько причин для такого, казалось бы, неудачного решения.

Первая причина состоит в том, что многоадресная передача по IP не так уж распространена в Интернете. Некоторые интернет-провайдеры и сети ее поддерживают, но обычно она не является доступной за границами отдельной сети, что ставит крест на глобальном вещании. Другой причиной являются преимущества TCP над UDP, которые мы обсуждали ранее. Передачу при помощи TCP смогут принять практически все пользователи Интернета, особенно в том случае, если передача будет идти как HTTP для прохода через межсетевые экраны, а надежная доставка медиа позволит пользователям легко перематывать контент назад.

Тем не менее существует один важный случай, в котором могут использоваться UDP и многоадресная передача: внутри сети провайдера. Например, компания, занимающаяся кабельным вещанием, может решить передавать телевизионные каналы пользователю через телевизионную приставку, используя IP-технологию вместо традиционного телевещания. Использование IP для передачи видеопрограмм в общем называется IPTV, как мы уже говорили. Так как компания полностью контролирует свою собственную сеть, она может построить эту сеть так, чтобы та поддерживала многоадресную передачу и пропускной способности ее канала было достаточно для вещания на основе UDP. Все это незаметно клиенту, так как IP-технология существует внутри **области свободной для навигации**.

Такая сеть выглядит просто как кабельное телевидение в отношении обслуживания, но вещание идет через IP, при помощи телеприставки, являющейся компьютером, на котором работает UDP, а телевизор выступает просто в роли монитора, подсоединенного к компьютеру.

Возвращаясь к случаю Интернета, недостаток передачи в реальном времени через TCP заключается в том, что сервер должен отсылать отдельную копию медиа каждому клиенту. Это реально, если клиентов не много, особенно если передается аудио, скользкий момент заключается в размещении сервера в месте с хорошим подключением к Интернету, чтобы пропускной способности канала было достаточно. Обычно это предполагает аренду сервера у провайдера, а не использование домашнего сервера. Рынок таких услуг довольно широк, так что цена вопроса не будет велика.

На самом деле, кто угодно, даже студент может установить медиасервер, а затем управлять им, например таким как радиостанция. Главные компоненты такой станции перечислены на рис. 7.32. Основой является обычный персональный компьютер с качественной звуковой картой и микрофоном. Для получения и кодирования аудио в разных форматах (например, MP4) используется популярное ПО, а для прослушивания, как обычно, используются медиаплееры.

Поток аудиоданных, создаваемый станцией, отправляется на сервер мультимедиа в Интернете с хорошим подключением к сети или как подкасты для передачи сохраненных файлов или для передачи в реальном времени. Сервер распространяет медиа по большому количеству TCP-соединений. На нем также располагается веб-сайт со страницами о станции и ссылками на доступный контент. Существуют как коммерче-

ские программы, включающие в себя все необходимые компоненты, так и открытые программные средства, такие как icecast.

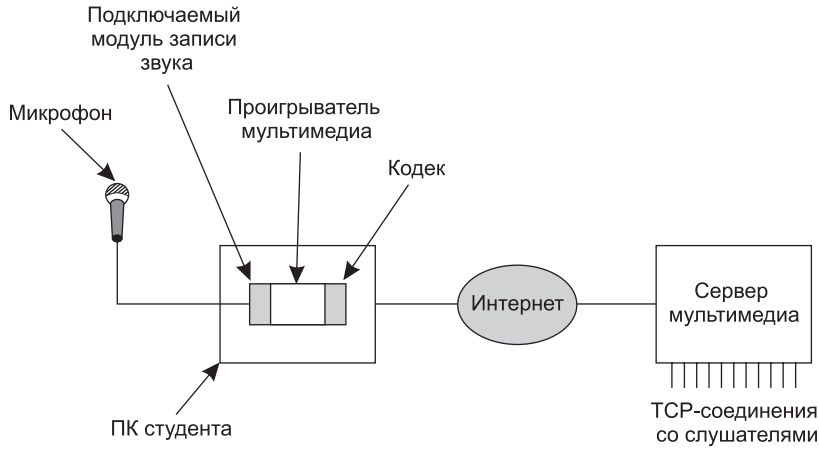


Рис. 7.32. Студенческая радиостанция

Однако при большом числе клиентов использовать TCP для передачи медиа каждому клиенту с одного сервера невозможно. Для этого будет не достаточно полосы пропускания канала одного сервера. Для больших сайтов потокового вещания используется несколько серверов, рассредоточенных географически, так чтобы клиент мог подключиться к ближайшему. Данная схема является сетью распределения контента, о которой мы поговорим в конце этой главы.

7.4.5. Конференции в реальном времени

Когда-то в стародавние времена голосовые звонки передавались коммутируемой телефонной сетью общего пользования. Затем появились Интернет и Всемирная паутина. С годами объем передаваемых данных все возрастал, и к 1999 году объемы данных и речи в телефонной сети уравнялись (и то и другое можно измерить количеством бит в секунду, поскольку сейчас в глубинах телефонной системы используется цифровое PCM-кодирование). К 2002 году объем информационного трафика стал на порядок больше объема речевого трафика, и его рост (экспоненциальный!) продолжается. Между тем, объем речевого трафика сохраняется практически неизменным.

В результате этого телефонные сети стали вытесняться Интернетом. Голосовой трафик сегодня передается при помощи использования интернет-технологий и занимает лишь малую часть сетевой полосы пропускания. Эта подрывная технология известная как **IP-телефония** (**VoIP** — **Voice over IP**, **передача голоса поверх IP**), или **интернет-телефония**.

IP-телефония используется в нескольких формах, на что влияют серьезные экономические факторы. (То есть: ее используют потому, что она экономит деньги.)

Одна из форм похожа на обычный (старомодный?) телефон, который включается в Ethernet и посылает вызовы по сети. Пер Андерсон (Pehr Anderson) был студентом

последнего курса в M.I.T., когда он и его друзья разработали эту модель для учебного проекта. Они получили за него оценку «B». Не удовлетворившись этим, в 1996 году он основал компанию, названную NBX, стал первооткрывателем этого вида IP-телефонии и через три года продал 3Com за 90 млн долларов. Компании любят этот подход, потому что он позволяет им избавиться от отдельных телефонных линий и довольствоваться уже имеющимися сетями.

Другой подход — использование технологии IP при построении телефонной сети для междугородных звонков. В таких странах, как США, такие сети доступны для альтернативной связи на большие расстояния, для этого перед номером необходимо набрать специальный код. Голосовые отсчеты (сэмплы) помещаются в пакеты, которые отправляются в сеть и извлекаются из пакетов, покидая ее. Так как IP-оборудование намного дешевле, чем обычное телекоммуникационное оборудование, услуга получается более дешевой.

В качестве отступления: разница в цене не исключительно техническая. В течение многих десятилетий услуги телефонной связи были регулируемой монополией, которая гарантировала телефонным компаниям определенный процент дохода от их затрат. Не удивительно, что они увеличивали затраты, например, имели громадное количество избыточных технических средств, что оправдывалось повышением надежности (допустимые сбои телефонной системы предполагались суммарно не более чем на два часа в течение 40 лет или в среднем на три минуты в год). Этот эффект часто называли «синдром позолоченного телефонного столба». После прекращения регулирования этот эффект, конечно, уменьшился, но унаследованное оборудование все еще существует. Индустрия информационных технологий никогда не действовала подобным образом, поэтому она всегда была более эффективной и не содержала лишнего.

Мы, однако, сосредоточимся на той форме IP-телефонии, которая, вероятно, наиболее видима потребителям: использование одного компьютера, чтобы позвонить на другой компьютер. Такая форма стал распространенной, так как компьютеры начали снабжаться микрофонами, динамиками, камерами, а также достаточно быстрыми процессорами, чтобы обработать аудио- и видеофайлы, и люди начали подключаться к Интернету из дома по каналам с большой пропускной способностью. Известный пример — программное обеспечение Skype появившееся в 2003 году. Skype и другие компании также предоставляют возможность легко звонить на обычные телефонные номера, как и на компьютеры с IP-адресами.

С увеличением сетевой пропускной способности, к голосовым звонкам присоединились видеовызовы. Изначально видеовызовы совершались в пределах компаний. Системы проведения видеоконференций были разработаны для обмена видео между двумя или несколькими местами, что позволяло сотрудникам, находящимся в разных местах, видеть друг друга во время собрания. Однако с хорошим широкополосным подключением к Интернету и программным обеспечением, выполняющим сжатие видео, видеоконференции стали возможны и дома. Такие инструменты, как например Skype, который предназначался исключительно для аудио, сейчас обычно включает возможность вызовов с видео, таким образом, друзья и родственники в разных концах света могут не только слышать, но и видеть друг друга.

С нашей точки зрения, голосовые или видеовызовы в Интернете относятся к проблеме потокового мультимедиа, однако имеют больше ограничений, чем передача

сохраненного файла или прямая трансляция события. Дополнительное ограничение — низкое время ожидания, которое необходимо для двухстороннего разговора. Телефонная сеть позволяет обеспечить одностороннее время ожидания до 150 мс для приемлемого использования, бóльшая задержка начинает быть заметна и раздражает участников. (Международные вызовы могут иметь время ожидания вплоть до 400 мс, по этому параметру они далеки от удобных для пользователя.)

Такое низкое время ожидания трудно достичь. Конечно, буферизация 5—10 секунд мультимедиа не сработает (так, как это работает при прямой трансляции спортивного события по радио). Вместо этого видео и голосовая IP-телефония должны быть разработаны с использованием разнообразных методов минимизации времени ожидания. Такая задача ведет к выбору UDP вместо TCP, потому что повторные передачи TCP добавляют к задержке как минимум один двойной проход. Однако некоторые виды времени ожидания не могут быть уменьшены даже и при использовании UDP. Например, расстояние между Сиэтлом и Амстердамом приблизительно 8000 км. Задержка распространения со скоростью света в оптическом волокне для этого расстояния составляет 40 мс. Пожелаем удачи тому, кто попробует ее сократить! На практике задержка распространения через сеть будет больше, потому что пройдет еще большее расстояние (биты перемещаются не по дуге большого круга), и также имеются задержки передачи, так как каждый IP-маршрутизатор сохраняет и пересылает пакет. Эти фиксированные задержки съедают общее допустимое время ожидания.

Другой источник задержки связан с размером пакета. Обычно использование больших пакетов представляет собой лучший способ использовать сетевую пропускную способность, потому что они более эффективны. Однако для звукового сигнала с частотой дискретизации 64 Кбит/с пакет размером 1 Кбайт будет заполняться 125 мс (и даже дольше, если отсчеты сжаты). Такая задержка превзошла бы общее время ожидания. Кроме того, если пакет в 1 Кбайт будет послан по широкополосному каналу, скорость которого 1 Мбит/с, передача займет 8 мс. Затем добавятся еще 8 мс, для того чтобы пакет прошел через широкополосное соединение на другом конце. Ясно, что большие пакеты не годятся.

Вместо этого системы IP-телефонии используют короткие пакеты, чтобы сократить время ожидания за счет уменьшения эффективности использования пропускной способности. Они загружают звуковые отсчеты в меньшие порции, обычно по 20 мс. При скорости 64 Кбит/с это 160 байт данных и еще меньше при сжатии. Однако, определенно, задержка при использовании такого пакета составит 20 мс. Задержка при передаче также будет меньше, потому что пакет короче. В нашем примере она сократится примерно до 1 мс. При использовании коротких пакетов минимальная задержка в одном направлении для пакета Сиэтл—Амстердам уменьшается от неприемлемой 181 мс (40 + 125 + 16) до приемлемой 62 мс (40 + 20 + 2).

Мы еще не сказали о дополнительных расходах времени программным обеспечением, а оно также израсходует часть допустимой задержки. Это особенно верно для видео, так как для соответствия видео имеющейся пропускной способности обычно необходимо сжатие. В отличие от потоковой передачи сохраненного файла, тут нет времени для кодера со сложными вычислениями, обеспечивающего максимальный уровень сжатия. Оба — и кодер, и декодер должны действовать быстро.

Буферизация по-прежнему необходима для своевременного проигрывания медиасэмплов (чтобы избежать неразборчивого звука или прерывистого видео), но количество данных в буфере должно быть очень небольшим, так как оставшееся допустимое время задержки измеряется в миллисекундах.

Если пакет не прибывает слишком долго, проигрыватель пропустит отсутствующие отсчеты, возможно, заменив окружающим шумом или повтором фрагмента, чтобы маскировать потерю для потребителя. Необходим компромисс между размером буфера, используемого, чтобы управлять джиттером и количеством потерянного мультимедиа. Буфер меньшего размера сокращает время ожидания, но увеличивает потери из-за джиттера. Таким образом, чем меньше размер буфера, тем потери заметнее потребителю.

Наблюдательные читатели, возможно, заметили, что до сих пор в этом разделе мы не сказали ничего о протоколах сетевого уровня. Сеть может сократить время ожидания или хотя бы джиттер, используя механизмы обеспечения качества обслуживания. Причина, по которой эта проблема не поднималась прежде, — в том, что потоковая передача может выполняться с существенным временем ожидания, даже в случае живой трансляции. Если время ожидания — не главное, то буфера на стороне хоста (узла) достаточно, чтобы решить проблему джиттера. Однако для конференций в реальном времени обычно важно, чтобы сеть сокращала и задержку, и джиттер (как варьирование задержки), чтобы остаться в пределах допустимой задержки. Это не важно только в том случае, если сетевая пропускная способность настолько велика, что каждый получает хорошее обслуживание.

В главе 5 мы описали два механизма обеспечения качества обслуживания, которые помогают достичь этой цели. Один механизм — это ДО (дифференцированное обслуживание), в котором пакеты отмечены как принадлежащие различным классам и получают различную обработку в пределах сети. Соответствующая маркировка для пакетов IP-телефонии — низкая задержка. На практике системы устанавливают коды ДО в общеизвестные значения следующим образом: класс обслуживания — «*Срочная пересылка*» (*Expedited Forwarding*); тип обслуживания — «*Низкая задержка*» (*Low Delay*). Это особенно полезно для широкополосных каналов доступа, так как эти каналы могут перегружаться, когда веб- или другой трафик конкурирует за использование связи. Определяемые устойчивым сетевым путем, задержка и джиттер увеличиваются затором. Каждому пакету в 1 Кбайт необходимо 8 мс для его передачи по каналу со скоростью 1 Мбит/с, и пакет IP-телефонии примет на себя эти задержки, если он находится в очереди после веб-трафика. Однако, если пакеты IP-телефонии имеют маркер низкой задержки, они прыгнут в начало очереди, обходя пакеты веб-трафика и уменьшая время ожидания.

Второй механизм, который может сократить задержку, — убедиться, что пропускная способность достаточна. Если доступная пропускная способность не постоянна или изменяется скорость потока передаваемых данных (как со сжатым видео) и иногда пропускная способность не достаточна, возникнут очереди и задержка увеличится. Это будет происходить даже с ДО. Чтобы гарантировать достаточную пропускную способность, часть сети может быть зарезервирована. Такая возможность обеспечивается интегрированным обслуживанием. К сожалению, оно не является широко распространенным. Вместо этого сети проектируются для ожидаемого объема передачи

информации, или клиентам предоставляются соглашения об уровне обслуживания для определенного объема передачи информации. Приложения должны действовать ниже этого уровня, чтобы избегать порождения затора и возникновения ненужных задержек. Для повседневного проведения видеоконференций дома потребитель может выбрать качество видео, определяемое пропускной способностью сети, или же программное обеспечение может проверить сетевой путь и выбрать соответствующее качество автоматически.

Любой из упомянутых выше факторов может сделать время ожидания неприемлемым, поэтому конференц-связь в реальном времени требует внимания к каждому из них. Краткий обзор IP-телефонии и анализ этих факторов см. в Goode (2002).

Теперь, когда мы обсудили проблему времени ожидания для потокового мультимедиа, мы перейдем к другой важной проблеме систем проведения конференций. Это проблема того, как устанавливать и прекращать вызовы. Мы рассмотрим два протокола, которые широко используются для этой цели — H.323 и SIP. Skype — это ещё одна важная система, но его внутреннее устройство закрыто.

H.323

Еще до того как голосовые и видеозвонки стали совершаться при помощи Интернета, всем было понятно, что если каждый производитель станет изобретать собственный стек протоколов, система никогда работать не будет. Во избежание возникновения этой проблемы заинтересованные стороны объединились под покровительством Международного союза телекоммуникаций (ITU) и начали разработку единого стандарта. В 1996 году ITU выпустил рекомендации с индексом **H.323** под заголовком «Видеотелефонные системы и оборудование локальных вычислительных сетей, не предоставляющих гарантированное качество обслуживания». Такое название могло родиться только в телефонной индустрии. Данные рекомендации были пересмотрены в 1998 году, и новый вариант H.323 стал носить название «Системы мультимедиа-коммуникаций, основанные на пакетах».

H.323 скорее дает общее представление об архитектуре систем интернет-телефонии, нежели описывает некий конкретный протокол. В документе можно найти множество ссылок на различные специализированные протоколы кодирования речи, установки соединения, передачи сигналов, данных и т. п., однако их описание не приводится. Общая модель изображена на рис. 7.33. В центре находится **шлюз (gateway)**, соединяющий Интернет с телефонной сетью. Он поддерживает протокол H.323 со стороны Интернета и протоколы коммутируемой телефонной сети общего пользования с «телефонной» стороны. Коммуникационные устройства называются **терминалами**. В локальной вычислительной сети может быть **машина-привратник (gatekeeper)**, управляющая конечными узлами, находящимися под ее юрисдикцией (в ее **зоне**).

Работу телефонной сети обеспечивает множество протоколов. Во-первых, необходим протокол кодирования и декодирования аудио и видео. Стандартное телефонное представление одного голосового канала кодируется как цифровое аудио с потоком 64 Кбит/с (8 бит на отсчет с частотой 8000 раз в секунду), что определено в **G.711**. Все системы H.323 обязаны поддерживать G.711. Тем не менее разрешена (но не является обязательной) поддержка и других протоколов кодирования речи. Они используют

иные алгоритмы сжатия и приводят к несколько отличающемуся компромиссу между качеством и использованием пропускной способности. Для видео поддерживаются формы сжатия MPEG, которые мы обсуждали ранее, включая H.264.

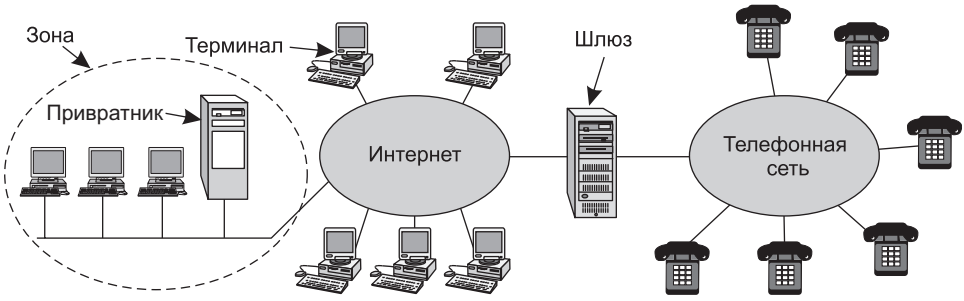


Рис. 7.33. Модель архитектуры H.323 для интернет-телефонии

Поскольку разрешено использование нескольких алгоритмов сжатия, необходим отдельный протокол, который позволил бы терминалам договориться об использовании одного из этих протоколов. Такой протокол называется **H.245**. Он позволяет согласовать также другие параметры соединения, например битовую скорость. RTCP требуется для управления каналами RTP. Кроме того, нужны протоколы для установления и разрыва соединений, обеспечения тонального вызова, генерирования звуков звонков и других стандартных функций телефонной системы. Используется стандарт ITU **Q.931**. Терминалам нужен протокол для ведения переговоров с машиной-привратником (если оный присутствует в локальной сети). Для этого в системе работает протокол **H.225**. Канал между ПК и привратником, которым этот протокол управляет, называется каналом **RAS (Registration/Admission/Status — Регистрация/Доступ/Статус)**. Он позволяет терминалам, кроме всего прочего, входить в зону и покидать ее, запрашивать и освобождать пропускную способность, обновлять данные о состоянии и т. п. Наконец, нужен протокол для непосредственной передачи данных. На этом участке работает RTP через UDP. Как обычно, управляется он RTCP. Иерархия всех этих протоколов показана на рис. 7.34.

<table>
<tr><td>Аудио (речь)</td><td>Видео</td><td colspan="4">Управление</td></tr>
<tr><td>G.7xx</td><td>H.26x</td><td rowspan="2">RTCP</td><td rowspan="2">H.225 (RAS)</td><td rowspan="2">Q.931 (Сигнализация)</td><td rowspan="2">H.245 (Управление вызовами)</td></tr>
<tr><td colspan="2">RTP</td></tr>
<tr><td colspan="4">UDP</td><td colspan="2">TCP</td></tr>
<tr><td colspan="6">IP</td></tr>
<tr><td colspan="6">Протокол канального уровня</td></tr>
<tr><td colspan="6">Протокол физического уровня</td></tr>
</table>

Рис. 7.34. Стек протоколов H.323

Чтобы понять, как эти протоколы взаимодействуют друг с другом, рассмотрим случай персонального компьютера (ПК), являющегося терминалом локальной сети (с привратником) и звонящего на удаленный телефон. Вначале компьютеру нужно найти привратника, поэтому он рассылает широковещательным образом специальный UDP-пакет через порт 1718. Из ответа привратника ПК узнает его IP-адрес. Теперь компьютер должен зарегистрироваться у привратника. Для этого он посылает ему сообщение RAS в пакете UDP. После регистрации компьютер обращается к привратнику с просьбой (сообщение доступа RAS) о резервировании пропускной способности. Только после выделения этого ресурса можно начинать установку соединения. Предварительное резервирование пропускной способности позволяет привратнику ограничить число соединений, устанавливаемых на исходящей линии, что, в свою очередь, служит для обеспечения необходимого качества обслуживания.

Строго говоря, телефонные системы выполняют ту же работу. Когда вы поднимаете трубку, на местный абонентский пункт отсылается сигнал. Если на пункте достаточно мощности для обработки еще одного звонка, он генерирует непрерывный гудок. В ином случае вы ничего не услышите. На сегодняшний день размер системы настолько велик, что вы практически всегда услышите непрерывный гудок, но раньше, когда телефония только зарождалась, на это почти всегда требовалось несколько секунд. Так что если ваши внуки когда-нибудь спросят, зачем нужны непрерывные гудки до начала набора, теперь вы будете знать, что им ответить. Хотя к тому времени стационарных телефонов, скорее всего, не останется.

Теперь ПК устанавливает TCP-соединение с привратником, чтобы осуществить телефонный звонок. При установлении телефонного соединения используются традиционные протоколы телефонной сети, ориентированные на соединение. Поэтому требуется протокол TCP. С другой стороны, в телефонной системе нет никаких RAS, которые позволяли бы телефонным аппаратам заявлять о своем присутствии, поэтому разработчики H.323 могли применять как UDP, так и TCP для передачи сообщений RAS, и они выбрали протокол с наименьшими накладными расходами — UDP.

Когда терминалу уже выделена пропускная способность, он может послать по TCP-соединению сообщение *SETUP* (стандарт Q.931). В нем указывается номер вызываемого абонента (или IP-адрес и порт, если вызывается удаленный компьютер). Привратник отвечает Q.931-сообщением *CALL PROCEDING*, подтверждая тем самым факт корректного приема запроса. Затем привратник пересылает сообщение *SETUP* на шлюз.

Шлюз, который является, с одной стороны, компьютером, а с другой — телефонным коммутатором, осуществляет обычный звонок на обычный телефон. Оконечная телефонная станция вызываемого абонента выполняет свою обычную работу (у абонента звенит звонок), а кроме этого отсылает обратно Q.931-сообщение *ALERT*, извещая ПК о том, что началась серия звонков. Когда абонент поднимает трубку, оконечная телефонная станция отправляет сообщение *CONNECT*, сообщая компьютеру о том, что соединение установлено.

После установления соединения привратник перестает принимать участие в этом процессе, хотя шлюз, конечно, продолжает работать, обеспечивая двустороннюю связь. Пакеты идут в обход привратника и направляются напрямую по IP-адресу шлюза. Такую ситуацию можно сравнить с обычным каналом между двумя сторонами. Это

действительно просто соединение физического уровня, по которому передаются биты, и все. Ни одна из сторон не в курсе того, что представляет собой противоположная сторона.

Для переговоров о предпочитаемых параметрах соединения используется протокол H.245. При этом используется специальный управляющий канал H.245, который всегда открыт. Каждая из сторон начинает с объявления своих возможностей. Например, может сообщаться о поддержке видео (H.323 может поддерживать видео), конференц-связи, используемых кодеках и т. п. После того как каждая из сторон узнает возможности противоположной стороны, организуются два однонаправленных канала, с которыми связываются определенные кодеки и которым присваиваются определенные параметры. Поскольку на каждой из сторон может быть установлено разное оборудование, вполне возможна ситуация, когда каждый из однонаправленных каналов использует свой кодек. По достижении договоренности по всем вопросам можно начинать передачу данных (по протоколу RTP). Управление производится RTCP, контролирующим перегрузку. Если передаются видеоданные, RTCP занимается синхронизацией звукового и видеоряда. На рис. 7.35 показаны различные виды логических каналов. После того как на одной из сторон вешают трубку, по каналу Q.931 передается сигнал окончания связи, для того чтобы высвободить не требующиеся более ресурсы, после того как звонок завершен.

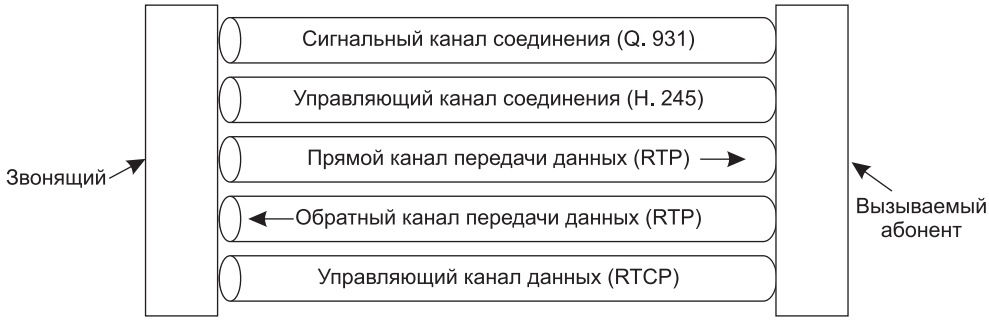


Рис. 7.35. Логические каналы между звонящим и вызываемым абонентами во время разговора

После разрыва соединения вызывающий ПК должен снова связаться с привратником и послать ему сообщение RAS с запросом освобождения зарезервированной пропускной способности. Впрочем, вместо этого он может осуществить новый звонок.

Мы до сих пор ничего не говорили о качестве обслуживания, как части H.323, а ведь на самом деле это довольно существенный аспект успешной передачи конференций в реальном времени. Дело в том, что QoS не входит в область рассмотрения H.323. Если сеть, по которой передаются данные, способна обеспечить стабильное соединение без джиттера между ПК и шлюзом, значит, нам повезло и качество обслуживания будет хорошим. В противном случае качество будет, увы, плохое. Однако любой телефонный звонок будет лишен перебоев благодаря дизайну телефонной сети.

SIP — протокол установления соединения

Стандарт H.323 был разработан ITU. В интернет-сообществе многим он показался типичным телекоммуникационным продуктом: громоздким, сложным и недостаточно гибким. Было решено организовать специальный комитет IETF для создания более простой и гибкой системы передачи речи поверх IP. Основным результатом деятельности этого комитета стал протокол **SIP (Session Initiation Protocol — протокол установления соединения)**, последняя версия которого описана в RFC 3261 в 2002 году. Протокол оговаривает способ установки телефонных соединений через Интернет, технологию организации систем для видеоконференций и способы создания других мультимедийных соединений. В отличие от H.323, представляющего собой целый набор протоколов, SIP — это единый модуль, способный взаимодействовать с разнообразными интернет-приложениями. Например, номера телефонов определяются в виде URL, то есть на веб-страницах можно размещать гиперссылки, щелкнув по которым пользователь сможет установить телефонное соединение (примерно так же схема *mailto* позволяет написать электронное письмо и отправить его по указанному в ссылке адресу).

Протокол SIP позволяет устанавливать и двухсторонние соединения (то есть обычные телефонные соединения), и многосторонние (когда каждый из участников может как слушать собеседников, так и говорить), и широковещательные (когда один из участников говорит, а остальные могут только слушать). Во время сеанса связи могут передаваться аудио-, видео- или другие данные. Эта возможность используется, например, при организации сетевых игр с большим количеством участников в реальном времени. SIP занимается только установлением, управлением и разрывом соединений. Для передачи данных также используются другие протоколы, например RTP/RTCP. SIP — это протокол прикладного уровня, работающий поверх TCP или UDP.

Протокол SIP предоставляет разнообразные услуги, включая поиск вызываемого абонента (который может в данный момент быть далеко от своего домашнего компьютера), определение его возможностей, поддержку механизмов установления и разрыва телефонного соединения. В простейшем случае SIP устанавливает сеанс связи между компьютерами звонящего и вызываемого абонентов. Именно этот случай мы сейчас и рассмотрим.

Телефонные номера в SIP представляются в виде URL со схемой *sip*. Например, *sip:ilse@cs.university.edu* свяжет вас с пользователем по имени Ilse, хост которого определяется DNS-именем *cs.university.edu*. SIP URL могут содержать также адреса формата IPv4, IPv6 или реальные номера телефонов.

Протокол SIP является текстовым, он построен по модели HTTP. Одна из сторон посылает ASCII-сообщение, в котором первая строка содержит имя метода, а ниже следуют дополнительные строки, содержащие заголовки для передачи параметров. Многие заголовки взяты из стандарта MIME, что позволяет SIP взаимодействовать с существующими интернет-приложениями. Шесть методов, определяемых базовой спецификацией, перечислены в табл. 7.17.

Для установки сеанса связи звонящий должен либо создать TCP-соединение с вызываемым абонентом и послать по нему сообщение *INVITE*, либо послать это же сообщение в UDP-пакете. В обоих случаях заголовки, содержащиеся во второй и всех

последующих строках, описывают структуру тела сообщения, содержащего информацию о возможностях звонящего, типах мультимедиа и форматах. Если вызываемый абонент принимает звонок, он посылает в качестве ответа трехразрядный код результата, подобный HTTP (группы этих кодов перечислены в табл. 7.13, код 200 означает прием вызова). Следом за строкой с кодом результата вызываемый абонент может также сообщить данные о своих возможностях, типах мультимедиа и форматах.

Таблица 7.17. Методы SIP

Метод	Описание
INVITE	Запрос установления сеанса связи
ACK	Подтверждение установления сеанса
BYE	Запрос окончания сеанса
OPTIONS	Опрос возможностей хоста
CANCEL	Отмена запроса
REGISTER	Информирование сервера переадресации о текущем местоположении пользователя

Соединение устанавливается с использованием протокола тройного рукопожатия, звонящий высылает *ACK* для окончания работы протокола и подтверждения приема кода 200.

Любая из сторон может послать запрос окончания сеанса связи, для этого используется метод *BYE*. Сеанс считается законченным после получения подтверждения от противоположной стороны.

Метод *OPTIONS* применяется для опроса возможностей машины. Обычно это делается перед запуском сеанса связи для того, чтобы определить, поддерживается ли тип сеанса, на который рассчитывает вызывающая сторона (например, передача голоса поверх IP).

Метод *REGISTER* относится к возможности протокола SIP разыскивать пользователя и соединяться с ним, даже если его нет дома. Сообщение, содержащее данный метод, отправляется на поисковый сервер SIP (location server), хранящий данные о том, кто где находится в данный момент. Впоследствии с помощью этого сервера можно попробовать найти абонента. Операция переадресации, используемая при этом, показана на рис. 7.36. На этом рисунке мы видим, что звонящий отправляет сообщение *INVITE* на прокси-сервер. Это делает возможную переадресацию незаметной. Прокси пытается разыскать абонента и посылает *INVITE* по найденному адресу. Дальнейшее общение представляет собой коммутацию последовательности сообщений при тройном рукопожатии. Сообщения *LOOKUP* и *REPLY* не входят в протокол SIP; на этой стадии может использоваться любой подходящий протокол в зависимости от типа поискового сервера.

SIP обладает еще множеством других возможностей, которые мы здесь не стали описывать подробно. Среди них есть функции ожидания вызова, отображения звонка, шифрования и аутентификации звонящего. Кроме того, есть возможность звонить с компьютера на обычный телефон, если есть доступ к соответствующему шлюзу между Интернетом и телефонной системой.

Рис. 7.36. Использование прокси-сервера и переадресации в протоколе SIP

Сравнительный анализ H.323 и SIP

H.323 и SIP поддерживают как двухстороннюю, так и многостороннюю связь. Оконечным оборудованием могут служить как компьютеры, так и обычные телефоны. И там, и там стороны предварительно договариваются о параметрах, возможно использование шифрования данных и протоколов RTP/RTCP. Сводная сравнительная табл. 7.18 показывает все сходства и различия.

Таблица 7.18. Сравнение H.323 и SIP

Аспект	H.323	SIP
Разработчик	ITU	IETF
Совместимость с телефонной системой	Полная	В большой мере
Совместимость с Интернетом	Присутствует, по прошествии длительного времени	Присутствует
Архитектура	Монолитная	Модульная
Завершенность	Полный стек протоколов	SIP обеспечивает лишь установление соединения
Переговоры относительно параметров	Ведутся обеими сторонами	Ведутся обеими сторонами
Сигналы при вызове	Q.931 поверх TCP	SIP поверх TCP или UDP
Формат сообщений	Двоичный	ASCII
Передача мультимедийных данных	RTP/RTCP	RTP/RTCP
Многосторонняя связь	Есть	Есть
Мультимедийные конференции	Возможны	Невозможны
Адресация	URL или номер телефона	URL
Разрыв связи	Явный или разрыв TCP-соединения	Явный или по тайм-ауту

продолжение ⇛

Таблица 7.18. (продолжение)

Аспект	H.323	SIP
Обмен сообщениями (instant messaging)	Нет	Есть
Шифрование данных	Есть	Есть
Объем описания стандарта	1400 страниц	250 страниц
Реализация	Громоздкая и сложная	Посредственная, но коммерчески выгодная
Статус	Широко распространен, особенно видео	Хорошая альтернатива, особенно для речи

Несмотря на схожий набор свойств и характеристик, протоколы разительно отличаются друг от друга концепцией и философией. H.323 — это типичный тяжеловесный стандарт, характерный для телефонной индустрии. Он описывает целый стек протоколов и очень точно указывает, что разрешено, а что запрещено. Такой подход приводит к хорошо определенным протоколам на каждом уровне, тем самым упрощается задача взаимодействия сетей. Однако платой за это оказывается большой, сложный и жесткий стандарт, тяжело адаптируемый к приложениям, которые появятся в будущем.

SIP, наоборот, представляет собой типичный интернет-протокол, работа которого основана на обмене короткими текстовыми строками. Это небольшой модуль, который хорошо взаимодействует с другими протоколами Интернета, однако несколько хуже согласуется с существующими сигнальными протоколами телефонной системы. Поскольку модель системы передачи данных поверх IP, предложенная IETF, использует модульный принцип, она оказывается достаточно гибкой и может легко адаптироваться к новым приложениям. Недостаток этого протокола связан с возможными проблемами межсетевого взаимодействия в тех случаях, когда люди пытаются интерпретировать, что обозначает стандарт.

7.5. Доставка контента

Интернет претендовал на то, чтобы захватить все области коммуникаций, подобно телефонной сети. Сначала академики связывались с отдаленными машинами, регистрируясь через сеть, чтобы выполнять задачи. Люди долго использовали электронную почту, чтобы связываться друг с другом, и сейчас используют еще и видео-, и голосовую IP-телефонию. Однако с ростом Всемирной паутины Интернет стал скорее хранилищем контента, чем средством коммуникации. Многие люди используют Всемирную паутину, чтобы найти информацию, и имеется огромное количество файлообменных сетей, предоставляющих совместный доступ к фильмам, музыке и программам. Смещение акцента на контент было столь явным, что большая часть пропускной способности Интернета сейчас используется для передачи сохраненного видео. Поскольку задачи распространения контента отличны от задач коммуникации, они предъявляют другие требования к сети. Например, если Салли хочет поговорить с Иту, она может позвонить на его мобильный телефон с помощью IP-телефонии. Коммуникация должна быть с определенным компьютером; нет смысла звонить на компьютер Пола.

Но если Иту хочет посмотреть последний матч своей команды по крикету, он будет рад получить это видео с любого компьютера, который может его предоставить. Ему не важно, был ли это компьютер Салли, или Пола, или, наиболее вероятно, неизвестный сервер в Интернете. Таким образом, местоположение не имеет значения для контента, за исключением того, что это воздействует на качество работы (и законность).

Еще одно отличие заключается в том, что некоторые веб-узлы, которые предоставляют контент, стали чрезвычайно популярны. Яркий пример — YouTube. Этот сайт позволяет пользователям делиться созданными ими видео на любую мыслимую тему. Многие люди хотят сделать это. Все остальные хотят увидеть. Посчитано, что YouTube со всеми этими требовательными к пропускной способности видео производит до 10 % всего ежедневного интернет-трафика. Ни один сервер не может обеспечить достаточную мощность и надежность, чтобы управлять таким потрясающим уровнем спроса. Вместо этого YouTube и другие большие поставщики контента строят свои собственный сети распределения контента. Эти сети используют центры хранения данных, расположенные по всему миру, чтобы предоставлять контент чрезвычайно большому количеству клиентов, обеспечивая хорошую производительность и доступность.

Методы, которые используются для распределения контента, со временем развивались. В начале роста Паутины ее популярность вела почти что к ее уничтожению. Растущее число запросов к контенту приводило к тому, что серверы и сети часто были перегружены. Люди начали расшифровывать WWW как World Wide Wait (Всемирное Ожидание). В ответ на потребительский спрос увеличивалась пропускная способность ядра Интернета, и более быстрая широкополосная связь вела к краям сети. Это увеличение пропускной способности было ключевым для улучшения работы, но — только частью решения. Чтобы сократить бесконечные задержки, исследователи развивали и различные архитектуры использования пропускной способности для распределения контента.

Одна из архитектур — **CDN (Content Distribution Network — сеть распределения контента)**. В ней поставщик рассредоточивает совокупность машин в Интернете и использует их, чтобы предоставлять контент клиентам. Это выбор больших игроков. Альтернативная архитектура — сеть **P2P (Peer-to-Peer — пиринговая сеть, сеть равноправных узлов)**. В ней совокупность компьютеров вносит свои ресурсы в объединенный фонд, чтобы предоставлять контент друг другу, без специально установленных серверов или какого-либо центрального пункта управления. Эта идея захватила воображение людей, потому что, действуя сообща, много маленьких игроков могут создать большой эффект.

В этом разделе мы рассмотрим проблему распределения контента в Интернете и некоторые решения, которые используются на практике. После короткого обсуждения популярности содержимого и интернет-трафика мы опишем, как построить мощные веб-серверы и использовать кэширование для улучшения производительности для веб-клиентов. Затем мы рассмотрим две главные архитектуры для распространения контента: сети CDN и P2P. Как мы увидим, их устройство и свойства весьма различны.

7.5.1. Контент и интернет-трафик

Чтобы проектировать и строить сети, которые работают хорошо, нам нужно понимать, какой трафик они должны нести. При смещении задач в сторону доступа к контенту

серверы, например, мигрировали из офисов компаний в центры хранения и обработки данных в Интернете, которые предоставляют большое число машин с превосходной возможностью подключения к сети. Чтобы поддерживать даже небольшой сервер, в настоящее время легче и дешевле арендовать виртуальный сервер в центре хранения и обработки данных в Интернете, чем работать с реальной машиной дома или в офисе с широкополосным подключением к Интернету.

К счастью, важно знать только два факта, касающиеся интернет-трафика. Первый — он быстро изменяется, не только в деталях, но и в целом. До 1994 года большая часть трафика представляла собой традиционную передачу файлов по FTP (для перемещения программ и данных между компьютерами) и электронную почту. Затем появился и начал экспоненциально расти веб. Веб-трафик оставил FTP и электронную почту далеко позади задолго до «пузыря доткомов» в 2000 году. Около 2000 года стартовали P2P-сети, предоставляющие музыку, а затем и фильмы. К 2003 году наибольшую часть интернет-трафика занимал P2P-трафик, оставивший позади веб. Где-то в конце 2000-х потоковое видео, использующее методы распределения контента, с таких сайтов, как YouTube, начало превосходить P2P. К 2014 году, предсказывает Cisco, 90 % всего интертрафика будет видео в той или иной форме (Cisco, 2010).

Не всегда имеет значение объем трафика. Например, во время бума IP-телефонии, даже до Skype, запущенного в 2003 году, она всегда оставалась незначительным всплеском на диаграмме, потому что требования пропускной способности для звука на два порядка величины ниже, чем для видео. Тем не менее IP-телефония нагружает сеть в другом аспекте, потому что она чувствительна ко времени ожидания. Другой пример, он-лайновые социальные сети, растущие неистово, начиная с Facebook, запущенного в 2004 году. В 2010 году Facebook впервые получил больше пользователей в день, чем Google. Даже не рассматривая трафик (а трафик там очень большой), социальные сети важны, потому что они меняют способ взаимодействия людей через Интернет.

Вывод, который мы делаем: «сейсмические сдвиги» — существенные изменения в интернет-трафике — происходят быстро и с некоторой регулярностью. Что будет следующим? Мы обязательно сообщим вам об этом в 6-м издании нашей книги.

Второй существенный факт об интернет-трафике — он чрезвычайно ассиметричный. Многие значения, с которыми мы имеем дело, близки к своей средней величине. Например, рост большинства взрослых людей близок к среднему. Есть некоторое количество высоких людей и некоторое количество низких и совсем мало очень высоких или очень низких. При таких свойствах можно найти диапазон, который захватит большую часть населения.

Интернет-трафик устроен не так. В течение длительного времени было известно, что есть небольшой ряд веб-узлов с большим трафиком и большая часть сайтов с намного меньшим трафиком. Эта особенность стала частью языка создания сетей. Раньше в статьях говорилось о движении в терминах **поездов пакетов (pocket trains)**, об идее существования экспресс-поездов с большим количеством пакетов, внезапно проходящем через связь (Jain и Routhier, 1986), что было формализовано как понятие **самоподобия (selfsimilarity)**, которое для наших целей можно понимать как сетевой трафик, который представляет собой много коротких и много длинных промежутков даже при рассмотрении в различных масштабах времени (Leland и др., 1994). Более

поздние работы называют длинные потоки трафика **слонами (elephants)**, а короткие — **мышами (mice)**. Идея заключается в том, что есть лишь несколько слонов и много мышей, но слоны имеют значение, потому что они такие большие.

Возвращаясь к веб-контенту, очевидна аналогичная ассиметрия. Опыт работы с пунктами видеопроката, библиотеками и другими подобными организациями показывает, что не все фильмы (книги) одинаково популярны. Экспериментально доказано, что если в пункте проката есть N фильмов, то доля заявок на конкретный фильм, стоящий на k-м месте в списке популярности, примерно равна C/k. Здесь C — это число, дополняющее сумму долей до 1, а именно:

$$C = 1/(1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/N).$$

Таким образом, например, самый популярный фильм берут примерно в семь раз чаще, чем седьмой в списке популярности. Такая зависимость называется **законом Ципфа (Zipf's law)** (Zipf, 1949). Он назван в честь Джорджа Ципфа, профессора лингвистики в Гарвардском университете, который заметил, что частота использования слова в большом тексте инверсионно пропорциональна его рангу. Например, сороковое в списке самых частотных слов используется в два раза чаще, чем восьмидесятое, и в три раза чаше, чем сто двадцатое.

Распределение Ципфа показано на рис. 7.37, *а*. Он иллюстрирует утверждение о том, что существует небольшое количество популярных элементов и огромное — непопулярных. Чтобы опознавать распределения такого вида, удобно размещать данные в логарифмическом масштабе по обеим осям, как показано на рис. 7.64, *б*. В результате должна получаться прямая.

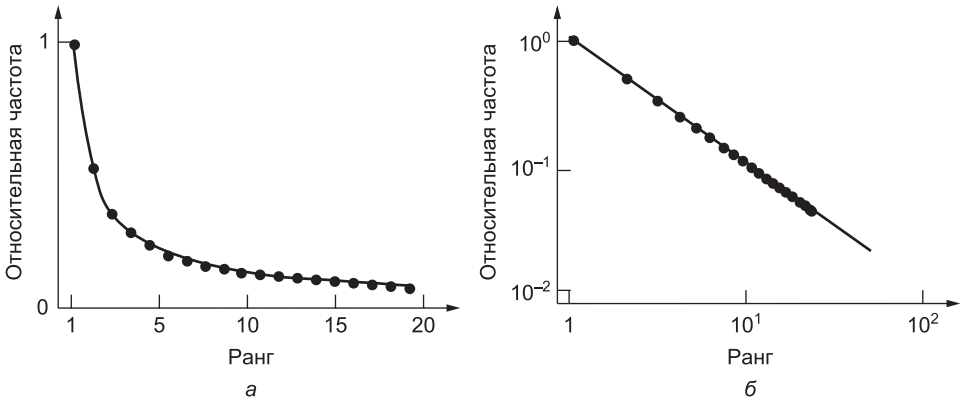


Рис. 7.37. Распределение Ципфа: *а* — линейный масштаб; *б* — логарифмический по обеим осям масштаб

Изучение популярности веб-страниц показало ее приблизительное соответствие законам Ципфа (Breslau и др., 1999). Распределение Ципфа — это одно из распределений, известных как **степенные законы (power laws)**. Степенные законы очевидны во многих сферах человеческой деятельности, например распределение городского населения и распределение богатства. Они также описывают ситуацию, когда имеются несколько больших игроков и много более мелких игроков, и они представляют

собой прямую линию на логарифмическом по обеим осям масштабе. Вскоре было обнаружено, что топология Интернета может быть приблизительно описана с помощью степенных законов (Faloutsos и др., 1999). Затем исследователи начали изображать все мыслимые свойства Интернета на логарифмическом масштабе и, увидев прямую линию, восклицали: «Степенной закон!»

Однако более важным, чем прямая линия на логарифмическом масштабе, было то, что эти распределения означают для проектирования и использования сетей. Так как большое количество контента имеет распределение Ципфа или подчиняется степенным законам, кажется фундаментальным, что популярность веб-узлов в Интернете подчиняется похожим законам. Это, в свою очередь, означает, что понятие «средний сайт» — это не удачное представление. Сайты лучше описывать как популярные и непопулярные. И те и другие значимы для рассмотрения. Популярные сайты, очевидно, значимы, так как небольшое количество популярных сайтов могут отвечать за большую часть интернет-трафика. Возможно, это покажется удивительным, но непопулярные сайты тоже значимы. Причина в том, что суммарное количество трафика, направленного к непопулярным сайтам, может составлять большую долю в общем трафике. Потому что непопулярных сайтов очень много. Это понятие — суммарное значение многих непопулярных вариантов было популяризировано, например, книгой *The Long Tail* («длинный хвост») (Anderson, 2008a).

Кривые, показывающие убывание, такое как на рис. 7.37, *а*, типичные, но все-таки не все одинаковые. В частности, ситуации, в которых уровень убывания пропорционален оставшемуся количеству материала (как например с нестойкими радиоактивными атомами) представляют собой **экспоненциальное убывание** (**exponential decay**), которое убывает намного быстрее, чем закон Ципфа. Число элементов, скажем атомов, оставшихся после момента времени t, обычно выражается, как $e^{-t/\alpha}$, где константа α показывает, насколько быстро происходит убывание. Разница между экспоненциальным убыванием и законом Ципфа заключается в том, что в первом случае можно пренебречь концом хвоста, а в случае закона Ципфа общий вес хвоста существенен и его нельзя проигнорировать.

Чтобы эффективно работать в этом ассиметричном мире, мы должны уметь строить оба вида веб-узлов. Непопулярные сайты легки в управлении. С использованием DNS, много различных сайтов могут фактически указывать на один и тот же компьютер в Интернете, который управляет ими всеми. С другой стороны, популярные сайты трудно поддерживать. Для этого нет ни одного достаточно мощного компьютера; кроме того, использование одного компьютера приведет к тому, что в случае его поломки сайт окажется недоступным для миллионов пользователей. Чтобы управлять этими сайтами, мы должен построить системы распределения контента. Мы приступим к этому далее.

7.5.2. Серверные фермы и веб-прокси

Сетевые проекты, которые мы до сих пор рассматривали, состояли из одной серверной машины, общающейся со многими клиентскими машинами. Чтобы построить большие веб-узлы с хорошими характеристиками, мы можем увеличивать скорость обработки или на серверной стороне, или на клиентской стороне. На серверной стороне более

мощный веб-сервер может быть построен путем организации серверной фермы, в которой кластер компьютеров действует как единый сервер. На клиентской стороне лучшая производительность может быть достигнута с применением лучших методов кэширования. В частности, кэширующий прокси (веб-прокси) обеспечивает большой общий кэш для группы клиентов. Мы опишем каждый из этих методов. Заметим, однако, что ни один из этих способов не достаточен для построения самых больших веб-сайтов. Для самых популярных сайтов необходимы методы распределения контента, использующие компьютеры, находящиеся в различных местах; эти методы мы опишем в следующих разделах.

Серверные фермы

Сколько бы пропускной способности не имела одна машина, она может обслуживать только определенное количество сетевых запросов, затем она будет перегружена. Решение в данном случае — использовать несколько компьютеров, чтобы сделать веб-сервер. Это приводит к модели **серверной фермы (server farm)**, показанной на рис. 7.38.

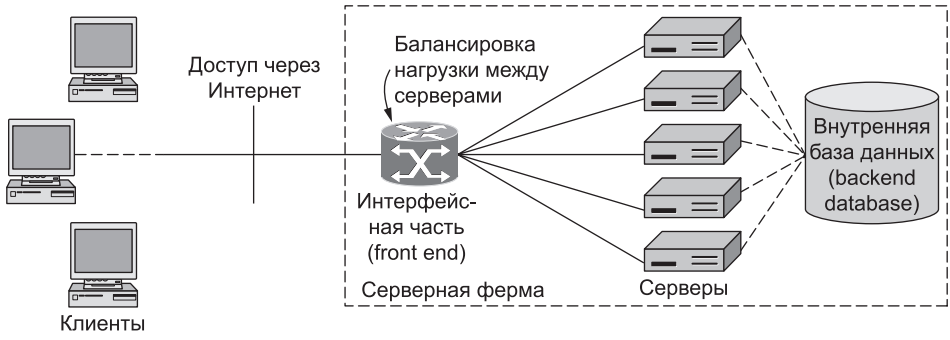


Рис. 7.38. Серверная ферма

Трудность с этой с виду простой моделью заключается в том, что набор компьютеров, образующих серверную ферму, должен выглядеть для клиентов как единый логический веб-сайт. В противном случае мы имеем дело просто с несколькими веб-сайтами, работающими параллельно. Существует несколько возможных решений, чтобы набор серверов выглядел как один веб-сайт. Все решения предполагают, что каждый из серверов может обрабатывать запрос от любого клиента. Для этого каждый сервер должен иметь копию веб-сайта. Пунктирными линиями показано соединение с этой целью серверов с общей внутренней базой данных.

Одно из решений — использовать DNS для распределения запросов по серверам серверной фермы. Когда сделан запрос DNS для URL веб-сайта, DNS-сервер возвращает меняющийся список IP-адресов серверов. Каждый клиент обращается к одному IP-адресу, обычно к первому в списке. Таким образом, разные клиенты обращаются к разным серверам с целью достижения одного и того же веб-узла, как и задумано. Метод DNS лежит в основе технологии CDN, мы снова вернемся к нему позже в этом разделе.

Другие решения основаны на **интерфейсной части (front end)**, которая распределяет поступающие запросы по пулу серверов в серверной ферме, даже если клиент

контактирует с серверной фермой, используя один IP-адрес назначения. Интерфейсная часть обычно представляет собой сетевой коммутатор на канальном уровне или IP-маршрутизатор, то есть устройство, которое управляет фреймами или пакетами. Все решения основаны на том, что эти устройства (или серверы) просматривают заголовки сетевого или транспортного уровня или уровня приложений и используют их нестандартным образом. Веб-запрос и ответ переносятся как TCP-соединение. Чтобы работать правильно, интерфейсная часть должна отправить все пакеты одного запроса к одному серверу.

Простая схема интерфейсной части — передавать все входящие запросы всем серверам. Каждый сервер отвечает только на часть запросов по предварительному соглашению. Например, 16 серверов могут смотреть на IP-адрес источника данных и отвечать только на те запросы, последние 4 бита IP-адресов которых соответствуют их определенному селектору. Другие пакеты отбрасываются. Хотя это и расточительно с точки зрения расхода пропускной способности, но так как зачастую ответы намного длиннее, чем запросы, это вовсе не так нелепо, как может показаться.

В более общем варианте структуры, интерфейсная часть, возможно, проверяет IP-, TCP-, и HTTP-заголовки пакетов и произвольно распределяет их по серверам. Распределение называют политикой **балансировки нагрузки (load balancing)**, так как его цель — балансировать рабочую нагрузку серверов. Политика может быть простой или сложной. Простая политика может состоять в том, чтобы использовать серверы один за другим по очереди или по кругу циклически. В этом случае интерфейс должен помнить отображение каждого запроса, чтобы последующие пакеты, являющиеся частью того же запроса, были отправлены к тому же серверу. Кроме того, для того чтобы сделать сайт надежнее, чем один сервер, интерфейс должен получать информацию о случаях отказов серверов и прекращать посылать запросы на отказавшие серверы.

Во многом, как и NAT, эта общая схема является опасной или, как минимум, хрупкой, так как мы только что создали устройство, которое нарушает самый основной принцип многоуровневых протоколов: каждый уровень должен использовать свой собственный заголовок для целей управления и не может проверять или использовать информацию из полезной нагрузки для каких бы то ни было целей. Но люди все равно проектируют такие системы и склонны удивляться, когда они выходят из строя вследствие изменений в более высоких уровнях. Интерфейс в данном случае — сетевой коммутатор или маршрутизатор, но он может действовать основываясь на информации транспортного или более высокого уровня. Такое устройство называется «**промежуточным узлом**» (**middlebox**), потому что оно вставлено в середине сетевого пути, где у него нет никакого дела согласно стеку протоколов. В этом случае интерфейс лучше всего рассматривать как внутреннюю часть серверной фермы, которая заканчивает все уровни вплоть до прикладного уровня (и поэтому может использовать всю информацию заголовков этих уровней).

Тем не менее, как и NAT, эта схема полезна на практике. Причина для просмотра TCP-заголовков заключается в том, что в этом случае можно достигнуть лучшего баланса загрузки, чем имея только информацию IP. Например, один IP-адрес может представлять целую компания и делать много запросов. И только просматривая TCP или информацию более высоких уровней, можно направить эти запросы на различные серверы.

Причина для просмотра заголовков HTTP несколько иная. Очень много веб-взаимодействий являются доступом к базам данных и их модификацией, например когда клиент просматривает свою самую недавнюю покупку. Серверу, который получит этот запрос, придется запросить общую внутреннюю базу данных. Полезно направить последующие запросы от этого же пользователя на тот же сервер, потому что он уже имеет информацию о пользователе в кэше. Самый простой путь для этого — использовать веб-куки (или другую информацию для различения пользователей) и просматривать заголовки HTTP для их обнаружения.

В заключение заметим, что хотя мы описали этот проект для веб-сайтов, серверная ферма может быть построена и для других видов серверов. Примером могут быть серверы потокового медиа поверх UDP. Единственное изменение, которое потребуется, — интерфейс должен быть способен поддерживать баланс нагрузки этих запросов (у которых будут другие поля заголовка протокола, чем у веб-запросов).

Веб-прокси

Сетевые запросы и ответы посылаются с использованием HTTP. В разделе 7.3 мы описали, как браузеры могут кэшировать ответы и использовать их многократно для ответов на последующие запросы. Различные поля заголовков и правила используются браузером, чтобы определить, является ли кэшированная копия веб-страницы все еще актуальной. Здесь мы не будем повторять этот материал.

Кэширование улучшает работу, сокращая продолжительность времени ответа и загруженность сети. Если браузер может сам определить, что кэшированная страница актуальна, он может немедленно выбрать ее из кэша, вообще без сетевого трафика. Однако, даже если браузер должен запросить сервер о подтверждении актуальности страницы, продолжительность времени ответа сокращена и сетевая нагрузка меньше, особенно для больших страниц, так как посылается только маленькое сообщение.

Тем не менее лучшее, что может сделать браузер, — кэшировать все веб-страницы, которые посетил пользователь. Из нашего обсуждения популярности, вы, возможно, помните, что кроме небольшого количества популярных страниц, которые неоднократно посещает много людей, существует очень много непопулярных страниц. На практике это ограничивает эффективность кэширования браузером, потому что существует большое количество страниц, которые данный пользователь посетит только один раз. Эти страницы всегда надо получать с сервера.

Единственный способ использовать кэш более эффективно — сделать его общим для нескольких пользователей. Таким образом страница, уже полученная для одного пользователя, может быть возвращена другому пользователю, когда он сделает такой же запрос. Без кэширования браузером обоим пользователям необходимо получить страницы с сервера. Конечно, этот общий доступ не может быть сделан для зашифрованного трафика, страниц, которые требуют аутентификацию, и некэшируемых страниц (например, текущие биржевые цены), которые возвращаются программами. Динамические страницы, особенно созданные программами, — это тот растущий случай, для которого кэширование не эффективно. Однако есть большое количество веб-страниц, которые видимы для многих пользователей и выглядят одинаково, независимо от того, кто к ним обращается (например, изображения).

Веб-прокси (Web proxy) используется для того, чтобы обеспечить пользователям общий доступ к кэшу. Прокси — это агент, который действует от имени кого-то другого, например пользователя. Есть много видов прокси. Например, ARP-прокси отвечает на запросы ARP от имени пользователя, который находится где-то в другом месте (и не может ответить за себя). Веб-прокси выполняет веб-запросы от имени его пользователей. Он обычно обеспечивает кэширование веб-ответов, и так как он находится в совместном доступе, веб-прокси имеет существенно больший кэш, чем браузер.

Когда используется прокси, типичный вариант для организации — задействовать один веб-прокси для всех пользователей. Организация — это компания или интернет-провайдер. В обоих случаях выгодно увеличить скорость веб-запросов для пользователей, а также сократить потребности в пропускной способности. Для конечных пользователей обычно устанавливается постоянная цена, независимо от использования, а плата с большинства компаний и интернет-провайдеров взимается в зависимости от использованной пропускной способности.

На рис. 7.39 показана конфигурация с использованием прокси. Каждый браузер настроен так, что он отправляет запросы к прокси, а не к реальному серверу страницы. Если прокси имеет эту страницу, он возвращает ее немедленно. В противном случае, прокси берет страницу с сервера, добавляет ее к кэшу для будущего использования и возвращает клиенту то, что он запросил.

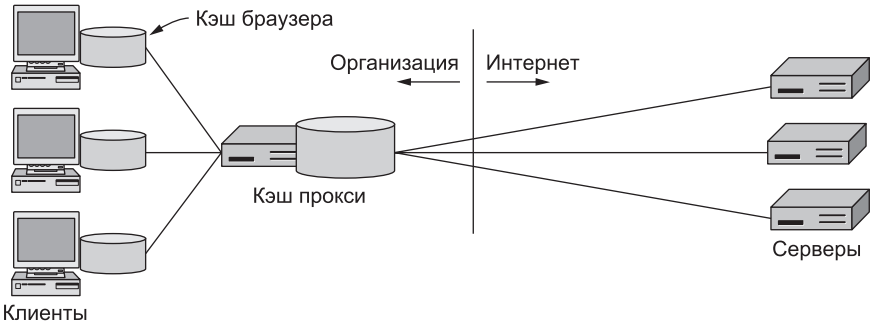


Рис. 7.39. Вспомогательный кэш между веб-браузерами и веб-серверами

Кроме отправки запросов к прокси вместо реального сервера, клиенты выполняют и собственное кэширование, используя кэш браузера. Обращение к прокси происходит только после того, как браузер попробовал удовлетворить запрос из своего собственного кэша. Таким образом прокси обеспечивает второй уровень кэширования.

Для того чтобы обеспечить дополнительные уровни кэширования, могут быть добавлены следующие прокси. Каждый прокси (или браузер) делает запрос к своему **вышестоящему прокси (upstream proxy)**. Каждый вышестоящий прокси производит кэширование для **нижестоящих прокси (downstream proxy)** или браузеров. Таким образом, возможно, что браузеры компании используют прокси компании, который использует прокси интернет-провайдера, который контактирует непосредственно с веб-серверами. Однако на практике, чтобы получить большую часть потенциальных преимуществ, часто достаточно одного уровня прокси-кэширования, который мы показали на рис. 7.39. Проблемой снова оказывается длинный хвост популярности.

Исследования веб-трафика показали, что кэш общего доступа особенно выгоден, пока число пользователей не превосходит количества сотрудников небольшой компании (скажем, 100 человек). Когда количество пользователей увеличивается, выгоды от использования общего кэша становятся незначительными из-за того, что непопулярные запросы не могут кэшироваться в связи с нехваткой пространства для хранения (Wolman и др., 1999).

Веб-прокси обеспечивают дополнительные преимущества, которые часто являются фактором в решении об их установке. Одно из преимуществ — возможность фильтровать контент. Администратор может создать на прокси черный список сайтов или фильтровать запросы, которые он делает. Например, многие администраторы неодобрительно смотрят на сотрудников, которые смотрят на рабочем месте видео на YouTube (или, еще хуже, порнографию), и устанавливают соответствующие фильтры. Еще одно преимущество наличия прокси — конфиденциальность или анонимность, когда прокси защищает идентичность пользователя от сервера.

7.5.3. Сети доставки контента

Серверные фермы и веб-прокси помогают строить большие сайты и улучшать работу сети, но они не достаточны для действительно популярных веб-сайтов, которые должны предоставлять контент в глобальном масштабе. Для этих сайтов нужен другой подход.

CDNs (Content Delivery Networks — сети доставки контента) переворачивают идею традиционного веб-кэширования с ног на голову. Вместо того чтобы клиенты искали копии страниц в ближайшем кэше, сам поставщик размещает копии страницы в наборе узлов в различных местах и направляет клиента, чтобы тот использовал в качестве сервера ближайший узел.

На рис. 7.40 показан пример пути следования данных в случае их распространения CDN. Это — дерево. Исходный сервер в CDN распространяет копию контента по другим узлам в CDN, расположенным в этом примере в Сиднее, Бостоне и Амстердаме. Этот процесс показан пунктирными линиями. Затем клиенты забирают страницы с ближайших узлов CDN. Этот процесс показан сплошными линиями. Таким образом, оба клиента в Сиднее берут копию страницы, которая расположена в Сиднее; они не достают страницу с исходного сервера, который, возможно, находится в Европе.

Использование древовидной структуры имеет три преимущества. Во-первых, распространение контента может масштабироваться до любого необходимого числа клиентов, используя большее количество узлов CDN (и больше уровней в дереве, когда распределение среди узлов CDN станет узким местом). Древовидная структура эффективна независимо от того, сколько имеется клиентов. Исходный сервер не перегружен, так как он взаимодействует с клиентами через дерево узлов CDN; ему не придется самому отвечать на каждый запрос страницы. Во-вторых, каждый клиент получает хорошую производительность за счет доставки страницы с ближайшего сервера, а не с отдаленного сервера. Это связано с тем, что время установки связи короче, из-за чего медленный старт TCP происходит быстрее, а более короткий сетевой путь с меньшей вероятностью пройдет через области затора в Интернете. Наконец, общая загруженность сети также сведена к минимуму. Если узлы CDN хорошо размещены,

каждая страница передается в каждом участке сети только один раз. Это важно, поскольку в конечном счете кто-то платит за полосу пропускания.

Рис. 7.40. Дерево распределения CDN

Идея использования дерева распределения проста. Менее простая задача — организовать использование этого дерева клиентами. Например, прокси-серверы, казалось бы, должны обеспечить решение. На рис. 7.40 видно, что если каждый клиент настроен на использование одного из узлов CDN — в Сиднее, Бостоне или Амстердаме — в качестве кэширующего веб-прокси, распределение было бы древовидным. Однако на практике эта стратегия не срабатывает по трем причинам. Первая причина состоит в том, что клиенты в данной части сети возможно относятся к различным организациям и поэтому используют различные веб-прокси. Напомним, кэширование обычно не бывает общим для разных организаций в связи с ограниченностью выгоды от кэширования при большом количестве пользователей, а также из соображений безопасности. Во-вторых, при наличие многих CDN каждый клиент может использовать только один прокси кэш. Какой из CDN он должен использовать в качестве прокси? Наконец, возможно самая практическая проблема в том, что веб-прокси настраиваются клиентами. Они могут быть настроены или не настроены на получение преимуществ от распределения контента CDN, и CDN мало что может с этим сделать.

Другой простой путь поддерживать дистрибутивное дерево с одним уровнем — использовать **зеркалирование (mirroring)**. В этом подходе, как и в предыдущих, исходный сервер повторяет контент в узлах CDN. Узлы в различных местах сети CDN называются зеркалами. Веб-страницы на исходном сервере содержат явные ссылки на различные зеркала, обычно сообщающие пользователю их местоположение. Такая схема позволяет пользователю вручную выбрать ближайшее зеркало и использовать его для загрузки контента. Пример типичного использования зеркал — размещение большого пакета программного обеспечения на серверах, расположенных, например, на восточном и западном побережье США, в Азии и в Европе. Сайты-зеркала обычно неизменны, и выбор этих сайтов остается постоянным месяцами или даже годами. Это испытанная и проверенная техника. Однако распределение в этом случае зависит от

пользователя, так как зеркала — это действительно различные веб-сайты, даже если они связаны друг с другом.

Третий подход, который преодолел трудности предыдущих двух подходов, использует DNS и называется переназначением **DNS (DNS redirection)**. Предположим, что клиент хочет получить страницу с URL *http://www.cdn.com/page.html*. Чтобы получить эту страницу, браузер будет использовать DNS, чтобы определить *www.cdn.com* по IP-адресу. Этот просмотр DNS происходит обычным образом. Используя протокол DNS, браузер узнает IP-адрес сервера имен для cdn.com, затем контактирует с сервером имен, чтобы попросить его выдать IP-адрес www.cdn.com. А теперь происходит действительно умная вещь. Сервер имен управляется CDN. Вместо того чтобы возвращать один и тот же IP-адрес для каждого запроса, он посмотрит на IP-адрес клиента, делающего запрос, и возвращает различные ответы. Ответ будет IP-адресом узла CDN, который расположен ближе всего к клиенту. Так, если клиент в Сиднее запрашивает сервер имен CDN выдать IP-адрес www.cdn.com, сервер имен вернет IP-адрес узла CDN в Сиднее, а если такой же запрос делает клиент в Амстердаме, сервер имени вернет IP-адрес узла в Амстердаме.

Эта стратегия вполне законно соответствует семантике DNS. Мы заранее видим, что серверы имен могут возвращать изменяющийся список IP-адресов. После анализа имени клиент в Сиднее получает страницу непосредственно из узела CDN в Сиднее. Дальнейшие страницы с того же «сервера» будут также взяты непосредственно с узла в Сиднее вследствие кэширования DNS. Полная последовательность шагов показана на рис. 7.41.

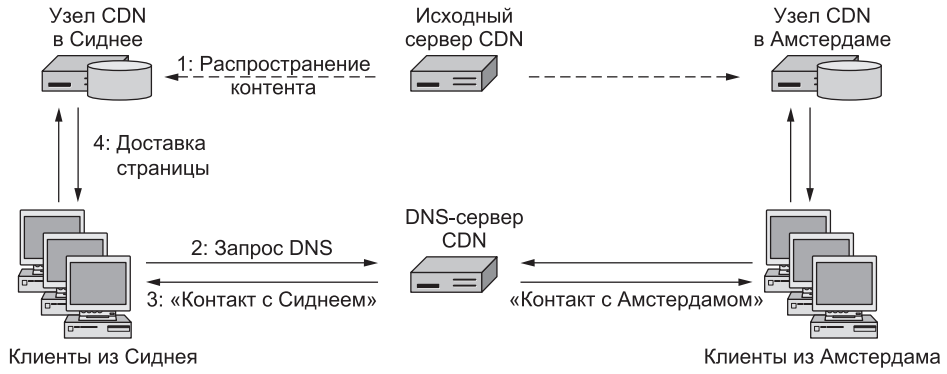


Рис. 7.41. Направление клиентов к ближайшим узлам CDN с использованием DNS

Сложный вопрос в описанном выше процессе — что означает найти самый близкий узел CDN и как это сделать. При определении понятия «ближайший» главный фактор — не география. При определении соответствия клиента и узла CDN следует принять во внимание как минимум два фактора. Первый фактор — сетевое расстояние. Клиент должен иметь короткий и большой емкости сетевой путь к узлу CDN. Это сделает загрузку более быстрой. Чтобы определить по IP-адресу клиента его сетевое местоположение, CDN используют заранее вычисленное отображение. Выбираемый узел может находиться на кратчайшем расстоянии, по прямой, а может и нет. Зна-

чение имеет некоторая комбинация длины сетевого пути и ограничений емкости на нем. Второй фактор — нагрузка, которая уже имеется на узле CDN. Если узлы CDN перегружены, они будут посылать ответы медленно, точно так же, как перегруженные веб-серверы, чего мы стремимся избежать в первую очередь. Поэтому может быть необходимо балансировать загруженность узлов CDN, отправляя часть клиентов к узлам, которые находятся несколько дальше, но менее загружены. Применение DNS для распределения контента было впервые разработано Akamai в 1998 году, когда Паутина стонала под грузом своего раннего роста. Akamai была первым большим CDN и стала лидером индустрии. Вероятно еще более умной, чем использование DNS для соединения клиентов с ближайшими узлами, была стимулирующая роль бизнеса. Компании платили Akamai за доставку их контента пользователям, так что они имели хорошо откликающиеся веб-сайты, которые были удобны клиентам. Узлы CDN должны быть размещены в местах сети с хорошей связностью, что изначально означает внутренние сети интернет-провайдеров. Для интернет-провайдеров преимущество наличия узла CDN в их сетях заключается в том, что узел CDN ограничивает необходимое количество расположенной выше сетевой пропускной способности, которая им нужна (и за которую надо платить), только прокси-кэшами. Кроме того, узел CDN улучшает отзывчивость для клиентов интернет-провайдеров, что позволяет провайдерам хорошо выглядеть в их глазах, предоставляя конкурентоспособное преимущество над провайдерами, которые не имеют узла CDN. Эти преимущества, ничего не стоящие для интернет-провайдера, делают для него установку узла CDN привлекательной. Таким образом, и поставщик контента, и интернет-провайдер, и клиенты — все выигрывают, а CDN делает деньги. Начиная с 1998 года другие компании вошли в этот бизнес, так что сейчас это конкурентная отрасль со многими поставщиками.

Как подразумевает это описание, большинство компаний не строят своих собственных CDN. Вместо этого они используют для доставки своего контента услуги поставщика CDN, например Akamai. Чтобы позволить другим компаниям использовать обслуживание CDN, нам нужно добавить к нашей картине еще один, последний шаг.

После подписания контракта владелец веб-узла предоставляет свой контент CDN. Этот контент отправляется в узлы CDN. Кроме того, владелец переписывает все ссылки из его веб-страниц, которые ведут к такому контенту. Вместо связывания с контентом на своем веб-сайте, страницы связываются с контентом через CDN. Как пример работы этой схемы, рассмотрим исходный код для веб-страницы Пушистое Видео, показанный в листинге 7.11, *а*. После предварительной обработки он преобразован в листинг 7.11, *б* и размещен на сервере Пушистое видео как страница *www.fluffyvideo.com/index.html*.

Когда пользователь печатает в своем браузере URL *www.fluffyvideo.com*, DNS возвращает IP-адрес сайта Пушистое видео, позволяя получить главную (HTML) страницу обычным способом. Когда пользователь щелкает на любой из гиперссылок, браузер просит DNS найти *www.cdn.com*. В процессе поиска происходит контакт с DNS-сервером CDN, который возвращает IP-адрес ближайшего узла CDN. Затем браузер отправляет обычный HTTP-запрос к узлу CDN, например к */fluffyvideo/koalas.mpg*. URL идентифицирует страницу для возвращения пользователю, начиная путь с *fluffyvideo* таким образом, что узел CDN может разделить запросы от различных

компаний, который он обслуживает. Наконец, видео получено, и пользователь видит милых пушистых животных.

Стратегия после разделения контента на хранящийся на CDN и у владельца страницы состоит в том, чтобы предоставить владельцу контента контроль, а на CDN переместить большую часть данных. Большинство «входных» страниц имеют небольшой размер, так как содержат только HTML-текст. Эти страницы часто связываются с большими файлами, например видео и изображения. А большие файлы поддерживаются CDN, даже если CDN целиком открыт для пользователей. Сайт выглядит точно так же, но работает быстрее.

Листинг 7.11. HTML-текст исходной страницы (*а*); та же страница после переключения на CDN (*б*)

```
(а)
<html>
<head> <title>Пушистое видео</title> </head>
<body>
<h1>Список материалов  Пушистое видео </h1>
<p> Щелкните ниже на бесплатные примеры. </p>
<a href="koalas.mpg"> Коалы сегодня </a> <br>
<a href="kangaroos.mpg"> Забавные кенгуру </a> <br>
<a href="wombats.mpg"> Милые вомбаты </a> <br>
</body>
</html>

(б)
<html>
<head> <title> Пушистое видео</title> </head>
<body>
<h1>Список материалов  Пушистое видео </h1>
<p> Щелкните ниже на бесплатные примеры. </p>
<a href="http://www.cdn.com/fluffyvideo/koalas.mpg"> Коалы сегодня </a> <br>
<a href="http://www.cdn.com/fluffyvideo/kangaroos.mpg"> Забавные кенгуру </a> <br>
<a href="http://www.cdn.com/fluffyvideo/wombats.mpg"> Милые вомбаты </a> <br>
</body>
</html>
```

Есть и другое преимущество использования сайтами общих CDN. Будущий спрос на веб-сайт может быть трудно предсказать. Часто возникают большие волны спроса, известные как «**flash crowds**» (резкое увеличение обращений на веб-сайт). Такая большая волна может возникнуть, например, когда выпущена новая версия продукта, когда прошел показ мод или другое событие, или компания появилась в новостях. Даже веб-сайт, который был ранее неизвестным, непосещаемым болотом, может внезапно стать центром Интернета, если он освещен в новостях и на него поставлены ссылки с популярных сайтов. Так как большинство сайтов не готово выдерживать большое увеличения трафика, многие из них в результате таких волн выходят из строя.

Рассмотрим один случай. Обычно веб-сайт Florida Secretary of State не очень загружен, хотя вы можете найти там информацию о корпорациях, нотариусах и культурных событиях штата Флорида, а также информацию о голосованиях и выборах. По некоторой причине, 7 ноября 2000 года (дата американских президентских выборов,

Буш против Гора), множество людей были внезапно заинтересованы в результатах выборов, опубликованных на этом сайте. Сайт внезапно стал одним из самых занятых веб-сайтов в мире и, естественно, в результате вышел из строя. Если бы там использовалась CDN, он бы, возможно, устоял.

Используя CDN, сайт получает доступ к очень большой емкости для размещения контента. Самые большие CDN имеют десятки тысяч серверов, расположенных в разных странах по всему миру. Так как в каждый момент времени только небольшое количество сайтов будет испытывать всплески количества обращений (по определению), эти сайты могут использовать емкость CDN для управления нагрузками, пока волна не схлынет. Таким образом, CDN может быстро увеличить предоставляемую сайту емкость.

Сказанное выше — это упрощенное описание работы Akamai. На практике имеют значение еще многие детали. Показанные в нашем примере узлы CDN — обычно представляют собой кластеры машин. Переадресация DNS происходит на двух уровнях: один отправляет клиента к близкой части сети, другой распределяет загрузку между узлами этой части. Имеют значения как надежность, так и скорость работы. Чтобы иметь возможность переместить клиента с одной машины в кластере на другую, ответы на втором уровне DNS предоставлены с короткими TTL, так что клиент будет повторять определение адреса через короткие промежутки времени. Наконец, до сих пор мы рассматривали распределение статических объектов, таких как изображения и видео, но CDN может также поддерживать и динамически создаваемые страницы, потоковое медиа и многое другое. Подробнее о CDN можно узнать из Dilley и др. (2002).

7.5.4. Сети одноранговых узлов (пиринговые сети)

Не каждый может установить CDN из 1000 узлов, расположенных по всему миру, для того чтобы распространять свой контент. (На самом деле арендовать 1000 виртуальных машин по всему миру не трудно, так как индустрия хостинга хорошо развита и конкурентна. Однако с получения узлов установка CDN только начинается.) К счастью, для всех остальных имеется альтернатива, которая проста в использовании и может распространять огромное количество контента. Это сеть P2P (одноранговая сеть).

Распространеие P2P-сетей началось с 1999 года. Первое широко распространенное приложение применялось для массового нарушения закона: 50 млн пользователей сети Napster обменивалось защищенными авторским правом песнями без разрешения владельцев прав, пока Napster не был закрыт решением суда, вызвав большую дискуссию. Однако технология равноправных узлов имеет много интересного и для законных целей. Другие системы продолжали развиваться с таким большим интересом со стороны пользователей, что P2P-трафик быстро превзошел веб-трафик. Сегодня BitTorrent — самый популярный протокол P2P. Он широко используется для распространения видео (лицензионного и общественного), а также и другого контента, что составляет большую долю всего интернет-трафика. Мы рассмотрим его в этом разделе.

Основная идея общего доступа к файлам в сети **P2P (Peer-to-Peer — равноранговой сети** или **пиринговой сети)** состоит в том, что множество компьютеров вместе объединяют свои ресурсы, чтобы сформировать систему распределения контента. Компьютеры здесь часто просто домашние компьютеры. Им не обязательно быть

машинами в центрах обработки данных в Интернете. Компьютеры в P2P называются пиры (peers, узлы-участники равноранговой сети), потому что каждый из них может действовать по отношению к другому и как клиент, получая его контент, и как сервер, предоставляя контент другим узлам. Системы равноправных узлов интересны отсутствием какой-либо специализированной инфраструктуры, в отличие от CDN. Каждый участвует в задаче распространения контента, часто при отсутствии какого-либо централизованного контроля.

Множество людей вдохновлены технологией P2P, поскольку им кажется, что это оказывает поддержку маленькому человеку. Причина не только в том, что для работы CDN необходима большая компания, в то время как любой обладатель компьютера может присоединиться к сети P2P. Сети P2P имеют колоссальную емкость для распространения контента, что может соответствовать самому большому из веб-сайтов.

Рассмотрим P2P сеть, состоящую из N пользователей средней величины, каждый с широкополосным подключением к сети, имеющим пропускную способность 1 Мбит/с. Общая емкость контента сети P2P, или скорость, с которой пользователи могут послать трафик в Интернет, составляет N Мбит/с. Емкость загрузки, или скорость, с которой пользователи могут получать трафик, также составляет N Мбит/с. Каждый пользователь может одновременно загружать и получать, так как он имеет связь со скоростью в 1 Мбит/с в каждом направлении.

Не очевидно, что это должно быть верно, но вся вместимость может продуктивно использоваться для распространения контента, даже для случая совместного использования одной копии файла всеми остальными пользователями. Чтобы понять, как это может быть, представьте себе, что пользователи организованы в бинарное дерево, где каждый не листовой пользователь отправляет файл двум другим пользователям. По дереву одна копия файла должна быть донесена ко всем пользователям. Чтобы всегда использовать пропускную способность загрузки для столь многих пользователей, как это только возможно (а значит, распространять большие файлы с низким временем ожидания), нам нужно конвейеризовать сетевую деятельность пользователей. Вообразите, что файл разделен на 1000 фрагментов. Каждый пользователь может получить новый фрагмент откуда-нибудь с верху дерева и одновременно послать ранее полученный фрагмент по дереву вниз. Таким образом, как только конвейер запущен, после того как послано небольшое количество фрагментов (равное глубине дерева), все не листовые пользователи будут заняты пересылкой файла другим пользователям. Так как всего не листовых пользователей приблизительно $N/2$, пропускная способность загрузки этого дерева — $N/2$ Мбит/с. Мы можем повторить этот трюк и создать другое дерево, которое использует другие $N/2$ Мбит/с пропускной способности загрузки, поменяв ролями листовые и не-листовые вершины. Вся вместе эта конструкция использует всю емкость.

Этот означает, что P2P сети самомасштабируемые. Их возможный объем загрузки растет вместе с потребностью в загрузке со стороны ее пользователей. Они всегда в некотором смысле «достаточно большие», без потребности в какой-либо специализированной инфраструктуре. Противоположная ситуация: даже большой веб-узел имеет фиксированный объем и будет или слишком большим, или слишком маленьким. Рассмотрим сайт со 100 кластерами, каждый с пропускной способностью 10 Гбит/с. Этот огромный объем не помогает даже при малом количестве пользователей. Сайт

не может доставить информацию до *N* пользователей быстрее, чем *N* Мбит/с, потому что ограничение находится на стороне пользователей, а не на веб-сайте. А когда есть более миллиона пользователей по 1 Мбит/с каждый, веб-сайт не может выкачать данные достаточно быстро, чтобы занять загрузкой всех пользователей. Это число пользователей кажется очень большим, однако большие BitTorrent сети (например, Pirate Bay) заявляют, что имеют более 10 млн пользователей. Это более 10 Тбит/с в терминах нашего примера!

Вы должны отнестись к этим небрежно вычисленным значениям с долей (или лучше с большим количеством) скептицизма, потому что они упрощают ситуацию. Существенный вызов для сетей P2P — хорошо использовать пропускную способность, при том что пользователи могут быть всех форм и размеров и иметь разную загрузку и вместимость. Однако эти цифры демонстрируют огромный потенциал P2P.

Есть другая причина, по которой сети P2P важны. CDN и другие централизованные службы вынуждают провайдеров находить персональную информацию о большом количестве пользователей, начиная с предпочтений веб-серфинга и покупок в интернет-магазинах до местоположения и адресов электронной почты. Эта информация может использоваться для лучшего, более персонализированного обслуживания, или для вторжения в частную жизнь людей. Второй вариант возможен или умышленно — как часть нового продукта, — или вследствие случайного разглашения или несанкционированно. В системах P2P не может быть одного поставщика, который способен к контролю всей системы. Это не означает, что системы P2P обязательно обеспечат конфиденциальность, так как пользователи до некоторой степени доверяют друг другу. А только означает, что они могут обеспечить форму конфиденциальности, отличную от централизованно управляемой системы. Системы P2P исследуются сейчас с целью совместного использования (например, хранения, передачи) файлов, и время покажет, существенно ли это преимущество.

Технология P2P развивалась двумя связанными путями. С более практической стороны есть системы, которые используются каждый день. Лучше всего из этих систем известны основанные на протоколе BitTorrent. С более академической стороны отмечался интенсивный интерес к алгоритмам **DHT (Distributed Hash Table — распределенная хэш-таблица)**, которые позволяют системам P2P хорошо работать как целому, вообще без зависимости от централизованных компонентов. Мы рассмотрим обе эти технологии.

BitTorrent

Протокол BitTorrent был разработан Коэном Брахмом в 2001 году, чтобы позволить набору узлов быстро и легко обеспечивать общий доступ к файлам. Существуют десятки свободно распространяемых клиентов, которые поддерживают этот протокол, точно так же, как много браузеров поддерживают протокол HTTP с веб-сервером. Протокол доступен как открытый стандарт на *www.bittorrent.org*.

В типичной системе равноправных узлов (пиров), например организованной с помощью BitTorrent, каждый из пользователей имеет некоторую информацию, которая, возможно, представляет интерес для других пользователей. Эта информация может быть свободным программным обеспечением, музыкой, видео, фотографиями и т. д.

Есть три проблемы, которые нужно решить, чтобы предоставить контент в общий доступ.

1. Как пир находит другие пиры, которые имеют контент, который он хочет загрузить?
2. Как контент дублируется пирами, чтобы обеспечить быструю загрузку для каждого?
3. Как пиры поощряют друг друга к загрузке контента другим, так же как и скачиванию для себя?

Первая проблема существует, потому что не все пиры будут иметь весь контент, по крайней мере, изначально. Подход, принятый в BitTorrent, — создание для каждого поставщика контента описания контента, названного **торрент (torrent)**. Торрент намного меньше, чем контент, и используется пиром, чтобы проверить целостность данных, которые он загружает с других пиров. Другие пользователи, которые хотят загрузить контент, должны сначала получить торрент, скажем, найти его на веб-странице, рекламирующей контент.

Торрент — это просто файл в определенном формате, который содержит два ключевых вида информация. Один вид называется **трекер** — сервер, который приводит пиры к содержимому торрента. Другой вид информации — список фрагментов одинакового размера, или **сегментов (chunks)**, из которых состоит контент. Для различных торрентов могут использоваться различные размеры сегментов, обычно от 64 до 512 Кбайт. Файл торрента содержит имя каждого сегмента, предоставленного как 160-битовый SHA-1 хэш сегмента. Мы рассмотрим криптографические хэши, такие как SHA-1 в главе 8. Пока вы можете считать что хэш — это более длинная и более безопасная контрольная сумма. Содержащий размер сегментов и хэши торрент-файл как минимум на три порядка величины меньше, чем контент, поэтому он может быть передан быстро.

Чтобы загрузить контент, описанный в торренте, пир сначала контактирует с трекером торрента. **Трекер (ttacker)** — это сервер, который поддерживает список всех остальных пиров, которые активно загружают и пересылают контент. Этот набор пиров называют рой **(swarm)**. Члены роя постоянно контактируют с трекером, чтобы сообщать, что они все еще активны, а также о том, что они покидают рой. Когда новый пир контактирует с трекером, чтобы присоединиться к рою, трекер сообщает ему об остальных пирах в рое. Получение торрента и контакт с трекером — это первые два шага для загрузки контента, как показано на рис. 7.42.

Вторая проблема — как разделить контент таким образом, чтобы обеспечивать быструю загрузку. Когда формируется начальный рой, некоторые пиры должны иметь все сегменты, составляющие контент. Эти пиры называют **сидерами (seeders — сеятелями)**. Другие пиры, которые присоединяются к рою, не будут иметь никаких сегментов; они — пиры, которые скачивают контент.

В то время как пир участвует в рое, он одновременно скачивает отсутствующие сегменты с других пиров и загружает имеющиеся у него сегменты другим пирам, которым они нужны. Этот обмен показан как последний шаг распределения контента на рис. 7.42. Через какое-то время пир собирает все больше сегментов, пока не загрузит весь контент. Пир может покинуть рой (и вернуться) в любое время. Обычно пир остается в рое в течение некоторого короткого периода после окончания своей

собственной загрузки. Из-за появляющихся и исчезающих пиров «текучесть» в рое может быть довольно большой.

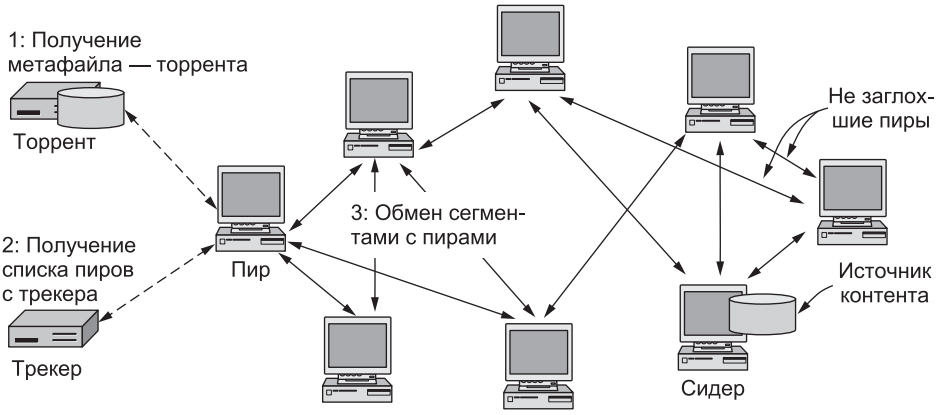


Рис. 7.42. BitTorrent

Чтобы описанный выше метод работал хорошо, каждый сегмент должен быть доступен у большого числа пиров. Если каждый должен получать сегменты в одном и том же порядке, то многие пиры зависели бы от сидеров следующего сегмента. Это создало бы узкое место. Вместо этого пиры обмениваются друг с другом списками сегментов, которые они имеют. Затем они выбирают редкие сегменты, которые трудно найти для скачивания. Идея состоит в том, что при скачивании редкого сегмента создается его копия, что делает сегмент более доступным для других пиров. Если это делают все пиры, все сегменты будут широко доступны через короткое время.

Третья проблема возможно наиболее интересная. Узлы CDN настроены исключительно для того, чтобы поставлять контент пользователям. А узлы P2P не такие. Это компьютеры пользователей, а пользователи могут быть более заинтересованы в получении кинофильма, чем в помощи в скачивании другим пользователям. Узлы, которые берут ресурсы из системы без какого-либо вклада, называются **фрирайдеры (freeriders)** или **личеры (leechers** — пиявки). Если их слишком много, система перестает хорошо функционировать. Более ранние системы P2P, как известно, работали с ними (Saroiu и др., 2003), так что BitTorrent стремится минимизировать их количество.

Подход, принятый в клиентах BitTorrent, награждает пиры, которые показывают хорошее поведение при загрузках. Каждый пир беспорядочно обращается к другим пирам, получает сегменты от них и в то же время пересылает сегменты к ним. Пир продолжает обмен сегментами только с небольшим количеством пиров, что обеспечивает самую высокую производительность скачивания, и также беспорядочно пробует другие пиры, чтобы найти хороших партнеров. Случайные обращения к пирам также позволяют вновь прибывшим получить начальные сегменты, которыми они могут обмениваться с другими пирами. Пиры, с которыми узел в настоящее время обменивается сегментами, называют **не заглохшими (unchoked)**.

Через какое-то время этот алгоритм должен сопоставить друг с другом пиры с сопоставимой возможностью загрузки и скачивания. Чем больше пир содействует другим

пирам, тем больше он может ожидать в ответ. Использование набора пиров также помогает насыщать пропускную способность пиров для высокой производительности. С другой стороны, если пир не пересылает сегментов на другие пиры, или делает это очень медленно, рано или поздно он окажется отрезанным или **заглохшим (choked)**. Эта стратегия препятствует антисоциальному поведению, такому как у личеров в рое.

Заглушающий алгоритм иногда описывается, как осуществление стратегии «**зуб-за-зуб**» **(tit-for-tat)**, которая поощряет кооперацию в повторных взаимодействиях. Однако он не защищает клиентов от обыгрывания системы в любом сильном смысле (Piatek и др., 2007). Тем не менее внимание к проблеме и механизмы, которые затрудняют фрирайд для случайных пользователей, вероятно содействовали успеху Bit-Torrent.

Как вы можете увидеть из нашего обсуждения, BitTorrent имеет свой обширный словарь терминов. Есть торренты, рои, личеры, сидеры, трекеры, а также выговоры, заглушение, прослушивание и т. д. Дополнительную информацию вы можете узнать из короткой статьи о BitTorrent (Коэн, 2003) и в Сети, начиная с сайта *www.bittorrent.org*.

DHT — распределенные хэш-таблицы

Появление файлообменных P2P сетей в 2000-х годах вызвало большой интерес в исследовательском сообществе. Сущность систем P2P в том, что они избегают централизованно управляемых структур как в CDN и других системах Это может быть существенным преимуществом. Когда системы разрастаются до очень крупных размеров, централизованно управляемые компоненты становятся узким местом и точкой сбоев. Центральные компоненты также могут быть использованы в качестве точки контроля (например, чтобы выключить сеть P2P). Однако ранние P2P системы были только частично децентрализованы, или, если они были полностью децентрализованы, они были неэффективны.

Традиционная форма BitTorrent, которую мы только что описали, использует равноранговые передачи и централизованный трекер для каждого роя. Именно трекер в системе равноправных узлов трудно децентрализовать. Ключевая проблема в том, как выяснить, какие пиры имеют определенный искомый контент. Например, каждый пользователь может иметь один или несколько элементов данных, например песен, фотографий, программ, файлов и т. д., которые другие пользователи, возможно, хотели бы получить. Как другие пользователи находят их? Создать один перечень того, у кого что есть — просто, но централизовано. Наличие у каждой пира собственного перечня не помогает. Правда, они распространяются. Однако работа для поддержания актуальности перечней контента для пиров (так как контент перемещается по системе) необходима такая большая, что это безнадежная попытка.

Вопрос, за который взялось исследовательское сообщество, — возможно ли построить для P2P такие индексы, которые были бы полностью распределенными и имели бы хорошие характеристики. Мы имеем в виду три вещи. Во-первых, каждый узел хранит только небольшой объем информации о других узлах. Данное свойство означает, что поддержание актуальности индекса не потребует больших затрат. Во-вторых, каждый узел может быстро найти записи в индексе. В противном случае, это не очень полезный индекс. В-третьих, каждый узел может использовать индекс, даже когда другие узлы

появляются и исчезают. Это свойство означает рост производительности индекса с ростом числа узлов.

Ответ на этот вопрос: «Да». Четыре различных решения были получены в 2001 году. Это Chord (Stoica и др., 2001), CAN (Ratnasamy и др., 2001), Pastry (Rowstron и Drusxel, 2001) и Tapestry (Zhao и другие., 2004). Другие решения были разработаны немного позже, в том числе Kademlia, который используется на практике (Maymounkov и Mazieres, 2002). Эти решения известны как **DHT (Distributed Hash Tables — распределенные хэш-таблицы)**, поскольку основная функциональность индекса — устанавливать соответствие между ключом и значением. Это похоже на хэш-таблицу, и решения, конечно, являются распределенными версиями.

DHT выполняют свою работу располагая регулярную структуру коммуникации между узлами, как мы увидим далее. Это поведение существенно отличается от традиционных P2P-сетей, которые используют все связи пиров. По этой причине DHT называются **структурированными P2P-сетями**. Традиционные протоколы P2P строят **неструктурированные P2P-сети**.

Опишем DHT Chord. В качестве сценария расмотрим, как заменить централизованный трекер, традиционно используемый в BitTorrent, на полностью распределенный трекер. Для решения этой задачи может использоваться Chord. В этом сценарии полный индекс — это список всех роев, к которым компьютер может присоединиться, чтобы загрузить контент. Ключ, используемый для просмотра индекса, — это описание контента торрентом. Он уникально идентифицирует рой, из которого может быть загружен контент, как хэш-функция от всех составляющих контент сегментов. Значение, сохраненное в индексе для каждого ключа, представляет собой список пиров, которые входят в рой. Эти пиры — компьютеры, с которыми можно контактировать, чтобы загрузить контент. Человек, желающий загрузить контент, например кинофильм, имеет только описание торрента. Вопрос, на который должна ответить DHT, — как при отсутствии центральной базы данных человек найдет, с каких именно пиров (из миллионов узлов BitTorrent) загрузить фильм?

Chord DHT состоит из n участвующих узлов. Это узлы, на которых запущен BitTorrent в нашем сценарии. Каждый узел имеет IP-адрес, по которому с ним можно связаться. Полный индекс распределен по узлам. Это подразумевает, что каждый узел хранит сегменты и части индекса для использования другими узлами. Ключевая роль Chord состоит в том, что он управляет этими индексами, используя идентификаторы в виртуальном пространстве, не IP-адреса узлов или имена контента, например фильма. Концепция идентификаторов состоит в том, что они являются просто m-битными числами, которые упорядочены по возрастанию в кольцо. Чтобы превратить адрес узла в идентификатор, ему с помощью хэш-функции *hash* сопоставляется m-разрядное двоичное число. Chord использует в качестве *hash* SHA-1. Это та же хэш-функция, которую мы упомянули, описывая BitTorrent. Мы рассмотрим ее, когда будем обсуждать криптографию в главе 8. Пока достаточно сказать, что это просто функция, которая отображает строку байт переменной длины в сильно случайное 160-разрядное двоичное число. Поэтому мы можем использовать ее, чтобы превратить любой IP-адрес в 160-разрядное число, называемое **идентификатор узла (node identifier)**.

На рис. 7.43, *а* показан круг идентификаторов узлов для $m = 5$. (Просто проигнорируйте пока что дуги в середине.) Некоторые из идентификаторов соответствуют узлам,

но большая часть — нет. В этом примере узлы с идентификаторами 1, 4, 7, 12, 15, 20, и 27 соответствуют фактическим узлам и заштрихованы; остальных не существует.

Теперь давайте определим функцию *successor* (*k*) как идентификатор первого реально существующего узла, следующего за *k* по кругу, по часовой стрелке. Например, *successor* (6) = 7, *successor* (8) = 12, а *successor* (22) = 27.

Ключ также строится получением 160-разрядного числа — хэшированием имени контента с помощью *hash* (то есть SHA-1). В нашем сценарии имя контента — торрент. Поэтому, для того чтобы преобразовать *torrent* (файл описания торрента) в его ключ, мы вычисляем *key* =_ *hash*(*torrent*). Это вычисление — просто локальное к процедуре *hash*.

Чтобы запустить новый рой, узлу нужно вставить в индекс новую пару ключ-значение, состоящую из (*torrent*, *my-IP-address*).

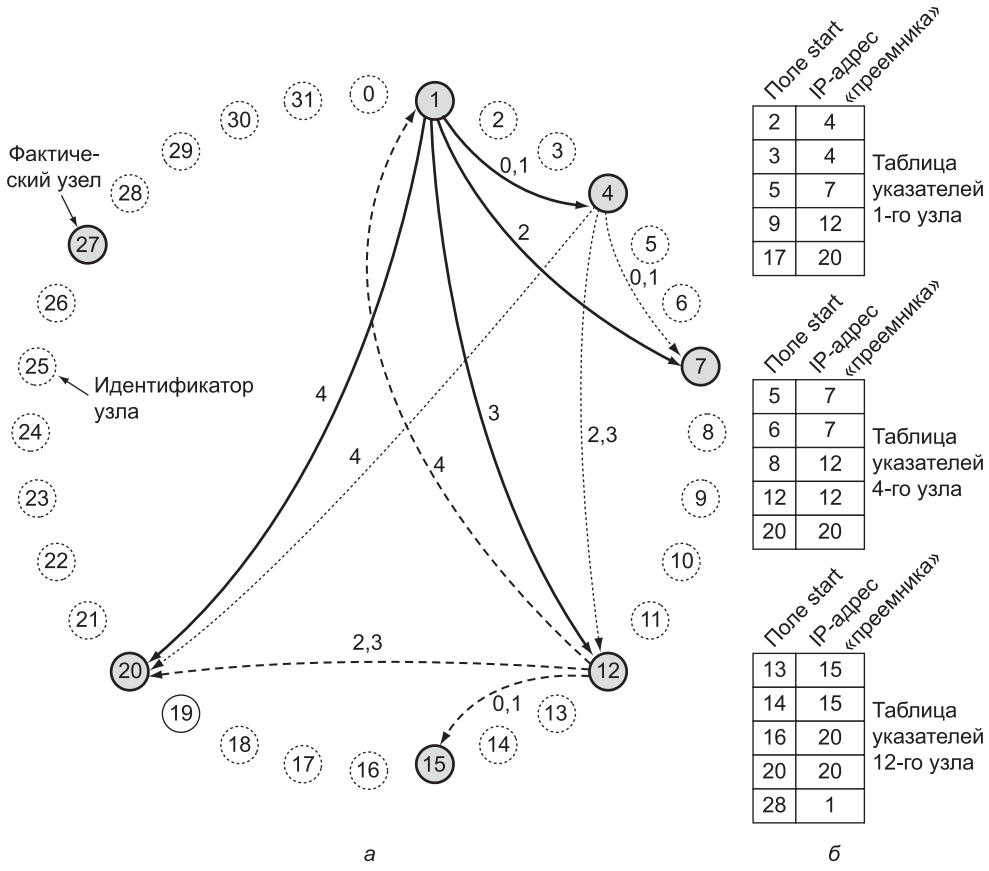


Рис. 7.43. Набор из 32 идентификаторов узлов, упорядоченных по кругу (*а*). Заштрихованные соответствуют фактическим машинам. Дуги соответствуют указателям от узлов 1, 4 и 12. Метки на дугах — индексы таблиц. Примеры таблиц указателей (*б*)

Чтобы сделать это, узел просит *successor* (*hash*(*torrent*)) сохранить *my-IP-address*. Таким образом, индекс случайным образом распространяется между узлами. Для устой-

чивости к ошибкам могут использоваться p различных хэш-функций для хранения данных на p узлах, но здесь мы не будем рассматривать тему устойчивости к ошибкам.

Через некоторое время после того, как DHT построена, другой узел хочет найти торрент, присоединиться к рою и скачать контент. Узел ищет *torrent*, сначала хэшируя его, чтобы получить *key*, а затем используя *successor* (*key*), чтобы найти IP-адрес узла, хранящего соответствующее значение. Значение — это список пиров в рое; узел может добавить свой IP-адрес к списку и контактировать с другими пирами, чтобы загрузить контент по протоколу BitTorrent.

Первый шаг прост, а второй — нет. Чтобы было возможно найти IP-адрес узла, соответствующего определенному ключу, каждому узлу необходимо поддерживать определенные административные структуры данных. Одна из них — IP-адрес его узла-преемника в круге идентификаторов узлов. Например, на рис. 7.43 *7* — это преемник для узла *4*, а *12* — преемник для узла *7*.

Поиск может происходить следующим образом. Запрашивающий узел направляет своему преемнику пакет, содержащий IP-адрес и ключ, который он ищет. Пакет продвигается по кругу, пока не обнаруживает преемника к идентификатору искомого узла. Этот узел проверяет, есть ли у него информация, соответствующая ключу, и если это так, возвращает ее непосредственно к запросившему узлу, IP-адрес которого у него есть. Однако линейный поиск всех узлов в большой равноранговой системе очень неэффективен, так как среднее число узлов, требуемых при поиске, составляет $n/2$. Чтобы повысить скорость поиска, каждый узел поддерживает то, что в Chord называется **таблица указателей (finger table)**. Таблица указателей имеет m записей, проиндексированных от 0 до $m - 1$, каждая указывающая на свой реальный узел. Каждая запись имеет два поля: *start* и IP-адрес преемника, как показано для трех узлов на рис. 7.43, *б*. Значениями полей для записи номер i в узле с идентификатором k являются:

$$start = k + 2^i \pmod{2^m},$$

$$\text{IP-адрес «преемника»} = successor\,(start[i]).$$

Заметим, что каждый узел запоминает IP-адреса относительно маленького числа узлов и большинство из них достаточно близки в терминах идентификатора узла.

С использованием таблицы указателей поиск *key* в узле k происходит следующим образом. Если *key* попадает между k и *successor* (k), то узлом, содержащим информацию о *key*, является *successor* (k), и поиск завершен. В противном случае в таблице указателей производится поиск записи, в которой поле *start* — самый близкий предшественник *key*. Непосредственно на IP-адрес из этой записи отправляется запрос продолжить поиск. Это уже ближе к *key,* но все еще до него, и есть хороший шанс, что после небольшого количества дополнительных запросов ответ будет получен. Фактически так как каждый поиск вдвое уменьшает остающееся расстояние до цели, можно показать, что среднее число поисков составит $\log_2 n$.

В качестве первого примера рассмотрим поиск *key* = 3 в узле 1. Так как узел 1 знает, что 3 лежит между ним и его преемником 4, искомый узел это 4, и поиск заканчивается возвращением IP-адреса узла 4.

В качестве второго примера рассмотрим поиск *key* = 16 в узле 1. Так как 16 не лежит между 1 и 4, происходит обращение к таблице указателей. Самый близкий

предшественник — 16 это 9, так что запрос переслан по IP-адресу 9-й записи, а именно в узел 12. Узел 12 также не знает ответа непосредственно, поэтому он ищет узел ближе всего предшествующий 16, и находит 14, который приводит IP-адрес узла 15. Запрос посылается туда. Узел 15 видит, что 16 лежит между ним и его преемником (20), и прокладывает путь обратно к узлу 1, возвращая IP-адрес 20.

Так как узлы все время присоединяются и отключаются, Chord должен иметь возможность управлять этими действиями. Мы предполагаем, что когда система начала функционировать, количество узлов было достаточно маленьким, и чтобы построить первый круг и таблицы указателей, они могли обмениваться информацией непосредственно. После этого нужна автоматизированная процедура. Когда новый узел *r* хочет присоединиться, он должен контактировать с каким-то существующим узлом и попросить его найти IP-адрес *successor* (*r*). Затем новый узел спрашивает *successor* (*r*) о его предшественнике. После этого новый узел просит их обоих вставить *r* в круг между ними. Например, если на рис. 7.43 хочет присоединиться 24, он просит любой узел найти *successor* (24), которым является 27. Затем он спрашивает 27 о его предшественнике (20). После этого сообщает им обоим о своем существовании, 20 начинает использовать 24 в качестве преемника, а 27 начинает использовать 24 в качестве предшественника. Кроме того, узел 27 передает ключи в диапазоне 21–24, который теперь принадлежит 24. Теперь 24 полностью вставлен.

Однако теперь многие таблицы указателей содержан неверную информацию. Чтобы исправить их, каждый узел поддерживает фоновый процесс, который периодически повторно вычисляет каждый указатель, запрашивая *successor*. Когда один из этих запросов указывает на новый узел, соответствующая запись указателя обновляется. Когда узел отключается корректно, он передает ключи своему преемнику и информирует его о своем отключении, так что предшественник отключающегося может соединиться с его преемником. Если же узел выходит из строя, возникает проблема, потому что его предшественник больше не имеет действительного преемника. Чтобы облегчить эту задачу, каждый узел отслеживает не только своего непосредственного преемника, но *s* последующих преемников, чтобы иметь возможность пропустить вплоть до $s - 1$ последовательно вышедших из строя узлов и снова замкнуть круг.

С тех пор как были изобретены DHT, в связи с ними было проведено большое количество исследований. Чтобы вы представили себе, насколько много, давайте зададимся вопросом: «Какие исследования в области сетей наиболее цитируемы?» Вы обнаружите, что трудно сравняться с плодовитым Chord (Stoica et al., 2001). Несмотря на эту гору исследований, приложения DHT начинают медленно появляться. Некоторые клиенты BitTorrent используют DHT для обеспечения полностью распределенного трекера, как мы описали. Большие коммерческие облачные услуги, такие, например, как Dynamo компании Amazon, также включают методы DHT (DeCandia и другие., 2007).

7.6. Резюме

Именование в ARPANET с самого начала было очень простым. В текстовом файле перечислялись имена всех хостов и соответствующие им IP-адреса. Каждую ночь на все машины подгружался этот файл. Но когда ARPANET перерос в Интернет и его

объем невероятно увеличился, потребовалась гораздо более изысканная и динамичная схема именования. Сегодня именование доменов в Интернете реализуется при помощи иерархической схемы, называемой службой имен доменов (DNS). Она организует все машины, подключенные к Интернету, в набор деревьев. В системе DNS на верхнем уровне находятся общеизвестные родовые домены, включая *com*, *edu* и около двухсот национальных доменов. DNS реализована в виде распределенной базы данных, серверы которой расположены по всему миру. Обратившись к DNS-серверу, процесс может преобразовать имя домена Интернета в IP-адрес, требующийся для общения с компьютером в этом домене.

Электронная почта — это одно из самых популярных приложений Интернета. Ею до сих пор пользуются все, начиная от детей младшего школьного возраста и заканчивая стариками преклонных годов. Большинство систем электронной почты соответствуют стандартам, описанным в RFC 5321 и 5322. Сообщения содержат ASCII-заголовки и данные разных типов, прописанных в MIME-заголовках. Почта передается агентам передачи для доставки и получается от них для представления во множестве пользовательских агентов, включая веб-приложения. Переданная почта доставляется по протоколу SMTP, устанавливающему TCP-соединение между хостом-источником и хостом-приемником.

Всемирная паутина является приложением, которое многие считают Интернетом. Изначально она являлась системой для обработки страниц, написанных на HTML и содержащих гиперссылки на другие страницы, которые загружались при помощи TCP-соединения браузера с сервером. Сегодня большая часть контента является динамической, или на стороне сервера (например, с PHP), или на стороне браузера (например, с JavaScript). Динамические страницы на сервере в сочетании с внутренними базами данных позволяют использовать такие веб-приложения, как электронная торговля и поиск. Динамические страницы браузера превращаются в полнофункциональные приложения, такие как электронная почта, которая работает внутри браузера и использует веб-протоколы для общения с удаленными серверами. Кэширование и постоянные соединения широко используются для улучшения производительности Всемирной паутины.

Использование Всемирной паутины через мобильные телефоны может быть непростой задачей, несмотря на увеличение ширины канала и мощности мобильных устройств. Веб-сайты часто высылают переработанные версии страниц с картинками меньшего размера и более простой навигацией на устройства с маленькими экранами.

Веб-протоколы все чаще используются для коммуникации между машинами. XML является более предпочтительным, чем HTML для описания контента, который легко обрабатывается компьютерами. SOAP является RPC-механизмом, который пересылает XML-сообщения при помощи HTTP.

Цифровое аудио и видео лежали в основе развития Интернета с 2000 года. Сегодня бóльшая часть интернет-трафика — это видео. Значительная его часть передается с вебсайтов при помощи смешанных протоколов (включая RTP/UDP и RTP/HTTP/TCP). Медиа в реальном времени передается многим пользователям. В него входят радио- и телестанции, передающие различные программы. Аудио и видео также используются для конференций в реальном времени. Во многих звонках используется голос поверх IP, а не традиционные телефонные сети, также для них используются видеоконференции.

Существует небольшое количество крайне популярных веб-сайтов и огромное количество не особо популярных. Для обслуживания популярных сайтов были созданы сети распределения контента. Они используют DNS, чтобы направлять клиентов на ближайший сервер. Серверы расположены в центрах обработки данных по всему миру. В качестве альтернативы P2P-сети позволяют нескольким машинам делить такой контент, как фильмы. Они обеспечивают увеличение емкости распределения контента с увеличением числа машин в такой сети, так что подобные сети могут соперничать с крупнейшими сайтами.

Вопросы

1. Многие коммерческие компьютеры имеют три разных и в то же время абсолютно уникальных идентификатора. Как они выглядят?
2. В листинге 7.1 после слова *laserjet* не поставлена точка. Почему?
3. Рассмотрите ситуацию, при которой кибертеррорист заставляет упасть все DNS-серверы в мире. Как это изменит возможность использования Интернета?
4. DNS использует UDP вместо TCP. Если DNS-пакет теряется, он автоматически не восстанавливается. Приводит ли это к возникновению проблем, и если да, то как они решаются?
5. Джон хочет заполучить оригинальное имя домена и использует программу рандомизации для генерирования вторичного имени домена. Он хочет зарегистрировать доменное имя в домене *com*. Сгенерированное доменное имя насчитывает 253 знака. Разрешит ли домен регистрацию подобного имени?
6. Может ли компьютер иметь одно имя DNS и несколько IP-адресов? Как такое может быть?
7. Число компаний, имеющих собственный веб-сайт, в последнее время сильно возросло. В результате в домене *com* существуют сайты тысяч фирм, что приводит к сильной нагрузке на сервер, обслуживающий этот домен верхнего уровня. Предложите способ решения этой проблемы без изменения схемы именования (то есть без изобретения нового домена верхнего уровня). Возможно, ваше решение потребует внесения изменений в клиентские программы.
8. Некоторые системы электронной почты поддерживают поле *Content-Return:*. В нем указывается, нужно ли возвращать содержимое письма в том случае, если оно не будет доставлено получателю. Это поле входит в состав конверта или заголовка письма?
9. Системы электронной почты хранят адресные книги e-mail, с помощью которых пользователь может найти нужный адрес. Для поиска по таким книгам имена адресатов должны быть разбиты на стандартные компоненты (например, имя, фамилия). Обсудите некоторые проблемы, которые следует решить, чтобы можно было разработать соответствующий международный стандарт.
10. Крупная юридическая фирма, в которой работает много сотрудников, предоставляет каждому из них отдельный адрес электронной почты. Адрес e-mail каждого сотрудника состоит из логина, знака @, названия фирмы и домена com. Однако фирма не определила формат логина точно. Так что некоторые сотрудники используют свои имена, некоторые — фамилии, а некоторые — инициалы. Теперь фирма хочет точно определить формат, например *имя.фамилия@название_фирмы.com*, который должен быть применен к адресам всех работников. Как можно это сделать и обойтись малой кровью.
11. Имеется двоичный файл длиной 4560 байт. Каков будет его размер после кодирования с помощью системы base64? Пара символов CR + LF вставляется через каждые 110 байт, а также в конце сообщения.

12. Назовите пять типов MIME, не указанных в тексте. Информацию можно взять из настроек браузера или из Интернета.
13. Предположим, вы хотите переслать другу MP3-файл, однако провайдер, услугами которого пользуется ваш друг, ограничивает максимальный размер входящей почты до 1 Мбайт, а файл занимает 4 Мбайт. Можно ли решить поставленную задачу, используя RFC 5322 и MIME?
14. Предположим, Джон установил механизм автоматических ответов на рабочем адресе электронной почты, на который приходят все письма, связанные с его делами, так чтобы они пересылались на его личный электронный адрес, который у него один на двоих с женой. Жена Джона об этом не знала и активировала каникулярного демона на личном ящике. Так как Джон наладил пересылку, он не установил каникулярного демона на рабочем почтовом ящике. Что произойдет, когда на этот ящик придет сообщение?
15. В любом стандарте, таком как RFC 5322, должно быть описание точной грамматики — это требуется для взаимодействия различных реализаций. Даже самые простые элементы должны быть четко определены. Например, в заголовках SMTP допустимы пробелы между символами. Приведите *два* правдоподобных альтернативных определения этих пробелов.
16. Каникулярный демон является частью пользовательского агента или агента передачи сообщений? Понятно, что он настраивается с помощью пользовательского агента, но какая часть системы занимается реальной отправкой автоматических ответов? Поясните свой ответ.
17. В простой версии алгоритма Chord для равнорангового поиска поисковики не используют таблицу указателей. Вместо этого они последовательно обходят круг в обоих направлениях. Может ли узел определить, в каком направлении нужно вести поиск. Поясните свой ответ.
18. IMAP позволяет пользователям запрашивать и загружать почту из удаленного почтового ящика. Означает ли это, что внутренний формат почтовых ящиков должен быть стандартизован, чтобы любые клиентские программы, использующие IMAP, могли обратиться к почтовому ящику на любом сервере? Аргументируйте свой ответ.
19. Рассмотрите круг Chord на рис. 7.43. Предположим, узел 18 внезапно выходит в онлайн. Какие части таблицы указателей, показанные на рисунке, это затронет?
20. Какие протоколы использует Webmail: POP3, IMAP или ни тот, ни другой? Если какой-то из этих двух, то почему выбран именно он? Если ни тот, ни другой, то к какому из них ближе по духу реально используемый протокол?
21. При пересылке веб-страницы предваряются заголовками MIME. Зачем?
22. Возможна ли ситуация, при которой щелчок пользователя на одной и той же ссылке с одним и тем же MIME-типом в Internet Explorer и в Firefox приводит к запуску совершенно разных вспомогательных приложений? Ответ поясните.
23. Хотя об этом и не было сказано в тексте, существует альтернативный вариант записи URL, использующий вместо имени DNS IP-адрес. Используйте эту информацию, чтобы объяснить, почему DNS-имя не может заканчиваться цифрой.
24. Представьте, что сотрудник факультета математики Стэнфордского университета написал новый документ, который он хочет распространить по FTP, чтобы его коллеги оставили отзывы. Он помещает документ в каталог */ftp/pub/forReview/newProof.pdf*. Как будет выглядеть URL этого документа?
25. В соответствии с табл. 7.10 *www.aportal.com* хранит список предпочтений клиента в виде cookie. Недостаток такого решения: размер cookie ограничен 4 Кбайт, и если нужно сохранить много данных о пользователе (например, о том, что он хочет видеть на странице множество биржевых сводок, новостей спортивных команд, типов новых историй, погоду сразу во многих городах, специальные предложения по разным категориям товаров, и т. д.),

то вскоре может быть достигнут порог этих описаний в 4 Кбайт. Предложите альтернативный способ хранения данных о пользователе, в котором эта проблема не возникала бы.

26. Некий «Банк для лентяев» хочет организовать специальную онлайновую банковскую систему для своих ленивых клиентов. После регистрации в системе и идентификации с помощью пароля пользователь получает cookie-файл, содержащий идентификационный номер клиента. Таким образом, ему не приходится всякий раз при входе в систему повторять ввод своих идентификационных данных. Как вам такая идея? Будет ли она работать? Насколько вообще хороша такая идея?
27. Изучите следующий HTML-тег:

    ```
    <h1 title=''this is the header''> HEADER 1 </h1>
    ```

    1) При каких условиях браузер использует атрибут *TITLE* и как?

    2) Чем атрибут *TITLE* отличается от атрибута *ALT*?
28. Как в языке HTML с изображением ассоциируется гиперссылка? Покажите на примере.
29. Напишите HTML-страницу, которая содержит ссылку на электронный адрес *имя_пользователя@имя_домена.com*. Что произойдет, когда пользователь кликнет по этой ссылке?
30. Напишите XML-страницу учета студентов университета; для каждого должно быть указано имя, адрес и средний балл.
31. Для каждого из перечисленных случаев укажите: (1) возможно ли и (2) лучше ли использовать PHP-скрипт или JavaScript и почему:

    1) Календарь на любой месяц, начиная с сентября 1752 года.

    2) Расписание рейсов из Амстердама в Нью-Йорк.

    3) Вывод полинома с коэффициентами, введенными пользователем.
32. Напишите программу на JavaScript, принимающую на входе целочисленные значения, превышающие 2, и сообщающую, является ли введенное число простым. В JavaScript синтаксис выражений `if` и `while` совпадает с аналогичными выражениями в C и Java. Оператор выполнения арифметических действий по произвольному модулю: `%`. Если понадобится найти квадратный корень числа *x*, воспользуйтесь функцией `Math.sqrt(x)`.
33. HTML-страница состоит из следующего кода:

    ```
    <html><body>
    <a href="www.info-source.com/welcome.html">Информация</a>
    </body></html>
    ```

    Когда пользователь щелкает на гиперссылке, открывается TCP-соединение, и на сервер отправляется некоторая последовательность строк. Перечислите эти строки.
34. Заголовок *If-Modified-Since* может использоваться для проверки актуальности страницы, хранящейся в кэше. Соответствующие запросы могут посылаться на страницы, содержащие изображения, звуки, видео и т. д., а также на обычные страницы на HTML. Как вы думаете, эффективность этого метода будет выше для изображений JPEG или для страниц HTML? Хорошенько подумайте над значением слова «эффективность» и после этого объясните свой ответ.
35. В день очень важного спортивного события (скажем, финала международного чемпионата по популярному виду спорта) огромное количество посетителей стремятся попасть на официальный веб-сайт мероприятия. Схожа ли эта ситуация внезапного роста трафика с выборами во Флориде в 2000 году, и почему?
36. Имеет ли смысл отдельному провайдеру функционировать в качестве сети доставки контента? Если да, то как должна работать система? Если нет, то чем плоха такая идея?

37. Предположим, что для аудио CD не используется сжатие. Сколько мегабайт данных должен содержать диск, длительность проигрывания которого равна двум часам?
38. На рис. 7.18, *в* показано, что шум квантования возникает из-за использования 4-битных отсчетов для представления 9 уровней сигнала. Первый отсчет (в нуле) является точным, а остальные — нет. Чему равна относительная погрешность для отсчетов, взятых в моменты времени, равные 1/32, 2/32 и 3/32 периода?
39. Можно ли использовать психоакустическую модель для уменьшения требуемой пропускной способности для систем интернет-телефонии? Если да, то каковы условия, при которых эта модель будет работать (если они вообще есть)? Если нет, то почему?
40. Сервер аудиопотока расположен на удалении от проигрывателя, дающем задержку 100 мс в одном направлении. Он выдает данные со скоростью 1 Мбит/с. Если проигрыватель содержит буфер объемом 2 Мбайт, то что можно сказать о расположении нижнего и верхнего пределов заполнения этого буфера?
41. Возникают ли при передаче речи поверх IP те же проблемы с межсетевыми экранами, что и при передаче потокового аудио? Ответ обсудите.
42. Какая скорость требуется для передачи несжатого цветного изображения размером 1200 × 800 пикселей и 16 битами на пиксел при 50 кадрах в секунду?
43. Может ли ошибка в одном бите в кадре MPEG повредить более одного кадра? Аргументируйте свой ответ.
44. Рассмотрим пример видеосервера, обслуживающего 50 000 клиентов. Каждый клиент смотрит три фильма в месяц. Предположим, что две трети всех фильмов начинают просматривать в 21:00. Сколько фильмов одновременно должен передавать сервер в этот период? Если для передачи каждого фильма требуется 4 Мбит/с, сколько соединений типа OC-12 нужно для соединения сервера с сетью?
45. Предположим, что закон Ципфа соблюдается для доступа к видеосерверу, на котором хранится 10 000 фильмов. Допустим, сервер хранит 1000 самых популярных фильмов в памяти, а остальные 9000 фильмов — на дисках. Какая часть запросов будет адресована памяти? Напишите небольшую программу, вычисляющую данное значение численно.
46. Некие нехорошие люди зарегистрировали имена доменов, которые незначительно отличаются от всемирно известных, таких как www.microsoft.com, и которые пользователь может посетить, просто случайно ошибившись при наборе адреса. Приведите пример по крайней мере пяти таких доменов.
47. Существует множество сайтов, зарегистрированных под именами www.слово.com, где слово — это обычное слово английского языка. Для каждой из приведенных категорий составьте список из пяти веб-сайтов и вкратце опишите их суть (например, www.cosmos.com — это сайт, посвященный проблемам космоса). Вот список категорий: животные, продукты питания, предметы быта, части тела. Что касается последней категории, просьба указывать объекты, расположенные выше талии.
48. Перепишите листинг 6.1, превратив его в реальный веб-сервер, использующий команду GET для работы с протоколом HTTP 1.1. Он должен реагировать на сообщение Host. Сервер должен кэшировать файлы, которые были недавно запрошены с диска, и обслуживать запросы, по возможности выдавая файлы из кэша.

Глава 8
Безопасность в сетях

В первые десятилетия своего существования компьютерные сети использовались, в первую очередь, университетскими исследователями для обмена электронной почтой и сотрудниками корпораций для совместного пользования принтеров. В таких условиях вопросы безопасности не привлекали большого внимания. Однако теперь, когда миллионы обычных граждан пользуются сетями для управления своими банковскими счетами, заполнения налоговых деклараций, приобретают товары в интернет-магазинах, и обнаруживаются многие недостатки, проблема сетевой безопасности становится очень серьезной. В этой главе мы рассмотрим вопросы безопасности сетей с различных точек зрения, укажем на подводные камни и обсудим различные алгоритмы и протоколы, позволяющие повысить безопасность сетей.

Тема безопасности является довольно обширной и включает в себя множество вопросов, связанных с различными человеческими грехами. В простейшем виде безопасность — это гарантия, что любопытные личности не смогут читать, или, что еще хуже, изменять сообщения, предназначающиеся другим получателям. Безопасность — это пресечение попыток получения доступа к удаленным службам теми пользователями, которые не имеют на это прав. Система безопасности должна позволять определять, написано ли сообщение «Оплатите счета до пятницы» налоговой службй , или же это фальсификация. Кроме того, системы безопасности решают проблемы, связанные с перехватом и повторным воспроизведением сообщений и с людьми, пытающимися отрицать, что они посылали данные сообщения.

Большинство проблем безопасности возникает из-за злоумышленников, пытающихся извлечь какую-либо пользу для себя или причинить вред другим. Несколько наиболее распространенных типов нарушителей перечислено в табл. 8.1. Из этого списка должно быть ясно, что задача обеспечения безопасности сетей включает в себя значительно больше, нежели просто устранение программных ошибок. Часто стоит задача перехитрить умного, убежденного и иногда хорошо финансируемого противника. Также очевидно, что меры, способные остановить случайного нарушителя, мало повлияют на серьезного преступника. Статистика, собираемая полицией, говорит о том, что большинство атак предпринимается не извне (людьми, прослушивающими линии связи), а изнутри — завистливыми или недовольными чем-либо людьми. Следовательно, системы безопасности должны учитывать и этот факт.

В первом приближении проблемы безопасности сетей могут быть разделены на четыре пересекающиеся области: секретность, аутентификация, обеспечение строгого выполнения обязательств и обеспечение целостности. Секретность (конфиденциаль-

ность) означает предотвращение попадания информации в нечистоплотные руки неавторизованных пользователей. Именно это обычно приходит в голову при упоминании безопасности сетей. Аутентификация позволяет определить, с кем вы разговариваете, прежде чем предоставить собеседнику доступ к секретной информации или вступить с ним в деловые отношения. Проблема обеспечения строгого выполнения обязательств имеет дело с подписями. Как доказать, что ваш клиент действительно прислал электронной почтой заказ на десять миллионов винтиков с левосторонней резьбой по 89 центов за штуку, если впоследствии он утверждает, что цена была 69 центов? Наконец, контроль целостности имеет дело с тем, как можно быть уверенным, что принятое вами сообщение не модифицировано по пути злоумышленником и не подделано?

Таблица 8.1. Типы нарушителей и цели их действий

Нарушитель	Цель
Студент	Прочитать из любопытства чужие письма
Хакер	Проверить на прочность чужую систему безопасности; украсть данные
Торговый агент	Притвориться представителем всей Европы, а не только Андорры
Бизнесмен	Разведать стратегические маркетинговые планы конкурента
Уволенный сотрудник	Отомстить фирме за увольнение
Бухгалтер	Украсть деньги компании
Биржевой брокер	Не выполнить обещание, данное клиенту по электронной почте
Аферист	Украсть номера кредитных карт для продажи
Шпион	Узнать военные или производственные секреты противника
Террорист	Украсть секреты производства бактериологического оружия

Все эти аспекты (секретность, аутентификация, обеспечение строгого выполнения обязательств и обеспечение целостности) встречаются и в традиционных системах, но с некоторыми существенными отличиями. Секретность и целостность достигаются с помощью заказных писем и хранения документов в несгораемых сейфах. Сегодня ограбить почтовый поезд значительно сложнее, чем во времена Джесси Джеймса.

Кроме того, людям обычно несложно отличить на глаз оригинальный бумажный документ от фотокопии. В качестве проверки попробуйте сделать фотокопию настоящего банковского чека. В понедельник попытайтесь обналичить настоящий чек в вашем банке. А во вторник попробуйте сделать то же самое с фотокопией. Проследите за разницей в поведении банковского служащего. Увы, но оригинал и копия электронных чеков неотличимы друг от друга. Понадобится некоторое время, прежде чем банки привыкнут к этому.

Люди опознают друг друга разными способами, например по лицам, голосам и почеркам. Доказательства подлинности бумажных документов обеспечиваются подписями на печатных бланках, печатями, рельефными знаками и т. д. Подделка обычно может быть обнаружена специалистами по почерку, бумаге и чернилам. При работе с электронными документами все это недоступно. Очевидно, требуются другие решения.

Прежде чем перейти к обсуждению сути самих решений, имеет смысл потратить несколько минут и попытаться определить, к какому уровню стека протоколов относится система сетевой безопасности. Вероятно, какое-то одно место для нее найти сложно. Каждый уровень должен внести свой вклад. На физическом уровне с подслушиванием можно бороться за счет помещения передающих кабелей (или, что еще лучше, оптического волокна) в герметичные трубы, наполненные инертным газом под высоким давлением. Любая попытка просверлить трубу приведет к утечке части газа из трубы, в результате давление снизится, и это послужит сигналом тревоги. Подобная техника применяется в некоторых военных системах.

На канальном уровне пакеты, передаваемые по двухточечной линии, могут зашифровываться при передаче в линию и расшифровываться при приеме. Все детали этого могут быть известны только канальному уровню, причем более высокие уровни могут даже не догадываться о том, что там происходит. Однако такое решение перестает работать в том случае, если пакету нужно преодолеть несколько маршрутизаторов, поскольку при этом пакет придется расшифровывать на каждом маршрутизаторе, что сделает его беззащитным перед атаками внутри маршрутизатора. Кроме того, такой метод не позволит защищать отдельные сеансы, требующие защиты (например, осуществление покупок в интернет-магазинах), и при этом не защищать остальные. Тем не менее этот метод, называемый **шифрованием в канале связи (link encryption)**, легко может быть добавлен к любой сети и часто бывает полезен.

На сетевом уровне могут быть установлены межсетевые экраны, позволяющие отвергать подозрительные пакеты, приходящие извне. К этому же уровню относится IP-защита.

На транспортном уровне можно зашифровать соединения целиком, от одного конца до другого. Максимальную защиту может обеспечить только такое сквозное шифрование.

Наконец, проблемы аутентификации и обеспечения строгого выполнения обязательств могут решаться только на прикладном уровне.

Итак, очевидно, что безопасность в сетях — это вопрос, охватывающий все уровни, именно поэтому ему посвящена отдельная глава.

Несмотря на то что эта глава большая, важная и содержит множество технических описаний, в настоящий момент вопрос сетевой безопасности может показаться несколько неуместным. Ведь хорошо известно, что большинство «дыр» в системах безопасности возникают из-за неумелых действий персонала, невнимательного отношения к процедурам защиты информации, недобросовестности самих сотрудников защищаемого объекта, многочисленные ошибки реализации, которые делают возможным проникновение неавторизованных пользователей и атаки с применением методов так называемой «социальной инженерии», когда клиентов обманом заставляют раскрыть свой пароль. Доля проблем, возникающих из-за преступников-интеллектуалов, прослушивающих линии связи и расшифровывающих полученные данные, сравнительно мала. Посудите сами: человек может прийти в совершенно произвольное отделение банка с найденной на улице магнитной кредитной карточкой, посетовать на то, что он забыл свой PIN-код, а бумажку, на которой он написан, съел, и ему тотчас «напомнят» секретный шифр (чего только ни сделаешь ради добрых отношений с клиентами). Ни одна криптографическая система в мире здесь не поможет. В этом

смысле книга Росса Андерсона (Anderson, 2008a) действительно ошеломляет, так как в ней приводятся сотни примеров того, как системы безопасности в самых различных производственных областях не справлялись со своей задачей. И причиной этих неудач была, мягко говоря, неряшливость в ведении дел или простое пренебрежение элементарными правилами безопасности. Тем не менее техническое основание, на котором строится электронная коммерция, когда отсутствуют проблемы с остальными факторами, это криптография.

На всех уровнях, за исключением физического, защита информации в сетях базируется на принципах криптографии. Поэтому мы начнем изучение систем безопасности с детального рассмотрения основ криптографии. В разделе 8.1 «Криптография» мы изучим базовые принципы. Далее до раздела «Управление открытыми ключами» (разделы 8.2–8.5) будут рассмотрены некоторые классические алгоритмы и структуры данных, применяемые в криптографии. После этого мы свяжем теорию с практикой и посмотрим, как все эти концепции применяются в компьютерных сетях. В конце этой главы будут приведены некоторые мысли, касающиеся технологии и социальных вопросов.

Прежде чем приступить к изложению, позвольте назвать вопросы, о которых *не* пойдет речь в этой главе. Подбирая материал, мы старались сконцентрировать внимание на вопросах, связанных именно с компьютерными сетями, а не с операционными системами или приложениями (хотя провести четкую грань зачастую бывает довольно трудно). Например, ничего не говорится об авторизации пользователя с использованием биометрии, защите паролей, атак, связанных с переполнением буферов, вирусах типа «Троянский конь», получении доступа путем обмана, внедрении кода путем межсайтового скриптинга, вирусах, червях и т. п. Обо всем этом очень много говорится в главе 9 книги «Современные операционные системы» (Tanenbaum, 2007). Все желающие узнать о вопросах обеспечения безопасности на уровне системы могут обратиться к этой книге.

Итак, в путь.

8.1. Криптография

Слово **криптография (cryptography)** происходит от греческих слов, означающих «скрытное письмо». У криптографии долгая и красочная история, насчитывающая несколько тысяч лет. В данном разделе мы всего лишь кратко упомянем некоторые отдельные моменты в качестве введения к последующей информации. Желающим ознакомиться с полной историей криптографии рекомендуется (Kahn, 1995). Для получения всестороннего представления о текущем положении дел см. (Kaufman и др., 2002). С математическими аспектами криптографии можно познакомиться, прочитав книгу (Stinson, 2002). Менее формальным (с математической точки зрения) языком ведется изложение в (Burnett и Paine, 2001).

С профессиональной точки зрения понятия «шифр» и «код» отличаются друг от друга. **Шифр (cipher)** представляет собой посимвольное или побитовое преобразование, не зависящее от лингвистической структуры сообщения. **Код (code)**, напротив, заменяет целое слово другим словом или символом. Коды в настоящее время

не используются, хотя история у них, конечно, славная. Наилучшим считается код, использовавшийся американскими войсками в Тихом океане во время Второй мировой войны. Просто-напросто для ведения секретных переговоров использовались носители языка индейцев навахо, словами из которого обозначались военные термины. Например, слово *чаи-дагахи-найл-цайди* (*chay-dagahi-nail-tsaidi* — буквально: убийца черепах) означало противотанковое оружие. Язык навахо тоновый (для различения смысла используется повышение или понижение тона), весьма сложный, не имеет письменной формы. Но самое большое его достоинство заключалось в том, что ни один японец не имел о нем ни малейшего представления.

В сентябре 1945 года газета *San Diego Union* так описывала этот код: «В течение трех лет, где бы ни высаживались военные моряки, уши японцев различали лишь странный булькающий шум, перемежающийся с другими звуками. Все это напоминало клич тибетского монаха или звук опустошаемой бутылки с горячей водой». Японцы так и не смогли взломать этот код, и многие носители языка индейцев навахо были удостоены высоких воинских наград за отличную службу и смелость. Тот факт, что США смогли расшифровать японский код, а японцы так и не узнали язык навахо, сыграл важную роль в американской победе в Тихом океане.

8.1.1. Основы криптографии

Исторически использовали и развивали искусство криптографии представители четырех профессий: военные, дипломатический корпус, люди, ведущие дневник, и любовники. Из них наиболее важную роль в развитии этой области сыграли военные. В военных организациях секретные сообщения традиционно отдавались для зашифровки и передачи плохо оплачиваемым шифровальщикам. Сам объем сообщений не позволял выполнить эту работу небольшим количеством элитных специалистов.

До появления компьютеров одним из основных сдерживающих факторов в криптографии была возможность шифровальщика выполнить необходимые преобразования, часто на поле боя, с помощью несложного оборудования. Кроме того, достаточно сложной задачей было быстрое переключение с одного криптографического метода на другой, так как для этого требовалось переобучение большого количества людей. Тем не менее опасность того, что шифровальщик может быть захвачен противником, заставила постоянно развивать способы смены криптографических методов при необходимости. Эти противоречивые требования приводят к модели процесса шифрования—дешифрации[1], показанной на рис. 8.1.

Сообщения, подлежащие зашифровке, называемые **открытым текстом (plaintext)**, преобразуются с помощью функции, вторым входным параметром которой является **ключ (key)**. Результат процесса шифрования, называемый **зашифрованным текстом (ciphertext)**, передается обычно по радио или через связного. Предполагается, что противник или **злоумышленник (intruder)** слышит и аккуратно копирует весь зашифрованный текст. Однако в отличие от получателя, которому предназначается данное

[1] Используемая автором и, соответственно, в переводе терминология несколько отличается от приводимой в российских нормативных документах. Однако она достаточно широко применяется на практике, поэтому сохранена вместе с вводимыми автором определениями. — *Примеч. ред.*

сообщение, злоумышленник не знает ключа дешифрации, и поэтому расшифровка сообщения представляет для него большие трудности, а порой она просто невозможна. Иногда злоумышленник может не только прослушивать канал связи (пассивный злоумышленник), но способен также записывать сообщения и воспроизводить их позднее, вставлять свои сообщения или модифицировать оригинальные сообщения, прежде чем они достигнут получателя (активный злоумышленник). Искусство взлома шифров называется **криптоанализом** (**cryptanalysis**). Искусства изобретать шифры (криптография) и взламывать их (криптоанализ) называются вместе **криптологией** (**cryptology**).

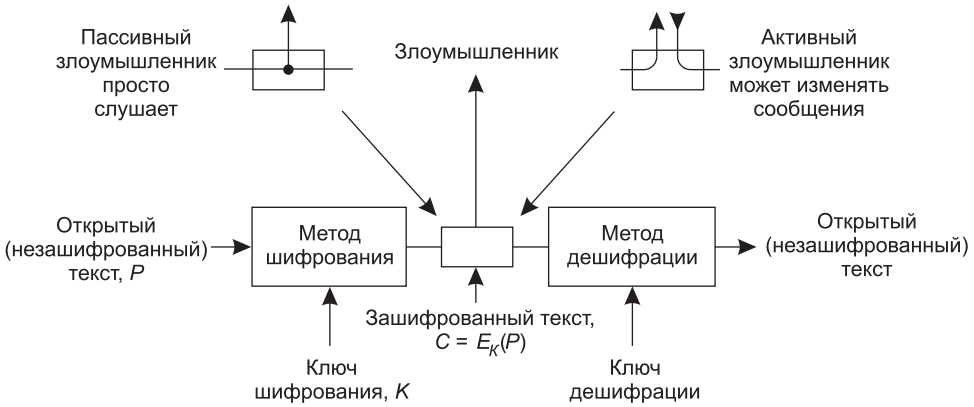


Рис. 8.1. Модель процесса шифрования—дешифрации (для шифра с симметричным ключом)

Обычно для обозначения открытого текста, зашифрованного текста и ключей полезно использовать специальную нотацию. Мы будем использовать формулу $C = EK(P)$, обозначающую, что при зашифровке открытого текста P с помощью ключа K получается зашифрованный текст C. Аналогично формула $P = DK(C)$ означает расшифровку зашифрованного текста C для восстановления открытого текста. Из этих двух формул следует, что

$$D_K(E_K(P)) = P.$$

Такая нотация предполагает, что E и D являются просто математическими функциями. Они в действительности таковыми и являются. Единственная хитрость состоит в том, что обе эти функции имеют по два параметра, один из которых (ключ) мы написали не в виде аргумента, а в виде нижнего индекса, чтобы отличать его от сообщения.

Основное правило криптографии состоит в предположении, что криптоаналитику (взломщику шифра) известен используемый метод шифрования. Другими словами, злоумышленник точно знает, как работают методы шифрования E и дешифрации D на рис. 8.1. Из-за огромных усилий, необходимых для разработки, тестирования и внедрения нового метода, каждый раз, когда старый метод оказывался или считался скомпрометированным, хранить алгоритм шифрования в секрете просто непрактично. А предположение, что метод остается секретным, когда это уже не так, могло бы причинить еще больший вред.

Здесь на помощь приходит ключ шифрования. Ключ состоит из относительно короткой строки, определяющей один из огромного количества вариантов результата шифрования. В отличие от самого метода шифрования, который может изменяться только раз в несколько лет, ключ можно менять так часто, как это нужно. Таким образом, наша базовая модель представляет собой постоянный и известный общий метод, в котором в качестве параметра используется секретный и легко изменяемый ключ. Идея, заключающаяся в предположении о том, что криптоаналитику известен метод и краеугольным камнем секретности является эксклюзивный ключ, называется **принципом Керкгофа (Kerckhoff's principle)**. Его в 1883 году впервые высказал фламандский военный криптограф Аугуст Керкгоф (Auguste Kerckhoff, 1883). Таким образом, принцип Керкгофа гласит:

Алгоритмы шифрования общедоступны; секретны только ключи.

Секретности алгоритма не стоит придавать большого значения. Попытка сохранить алгоритм в тайне, называемая в торговле **безопасностью за счет неясности (security by obscurity)**, обречена на провал. К тому же, опубликовав свой алгоритм, разработчик получает бесплатную консультацию от большого количества ученых-криптоаналитиков, горящих желанием взломать новую систему и тем самым продемонстрировать свой ум и ученость. Если никто не смог взломать алгоритм в течение долгого времени после его опубликования, то, по-видимому, этот алгоритм достаточно прочен.

Поскольку реально в тайне хранится только ключ, основной вопрос заключается в его длине. Рассмотрим простой кодовый замок. Его основной принцип состоит в том, что вы последовательно вводите десятичные цифры. Все это знают, но ключ хранится в секрете. Ключ длиной в две цифры образует 100 вариантов. Ключ длиной в три цифры означает 1000 вариантов, а при длине ключа в шесть цифр число комбинаций достигает миллиона. Чем длиннее ключ, тем выше **показатель трудозатрат (work factor)** взломщика шифра. При увеличении длины ключа показатель трудозатрат для взлома системы путем простого перебора значений ключа растет экспоненциально. Секретность передаваемого сообщения обеспечивается мощным (но все же открытым) алгоритмом и длинным ключом. Чтобы не дать прочитать свою электронную почту младшему брату, достаточно ключа длиной в 64 двоичных разряда. В коммерческих системах имеет смысл использовать ключи длиной 128 бит. Чтобы защитить ваши тексты от правительств развитых государств, потребуются ключи длиной, по меньшей мере, в 256 бит.

С точки зрения криптоаналитика задача криптоанализа имеет три принципиальных варианта. Во-первых, у криптоаналитика может быть некоторое количество зашифрованного текста без соответствующего открытого текста. Задачки, в которых в качестве исходных данных имеется в наличии **только зашифрованный текст (ciphertext-only)**, часто печатаются в различных газетах в разделе ребусов. Во-вторых, у криптоаналитика может оказаться некоторое количество зашифрованного текста и соответствующего ему открытого текста. В этом случае мы имеем дело с **проблемой известного открытого текста (known plaintext)**. Наконец, когда у криптоаналитика есть возможность зашифровать любой кусок открытого текста по своему выбору, мы получаем третий вариант проблемы дешифрации, то есть **проблему произвольного открытого текста (chosen plaintext)**. Если бы криптоаналитикам было позволено

задавать вопросы типа: «Как будет выглядеть зашифрованное ABCDEFGHJKL?», задачки из газет решались бы очень легко.

Новички в деле криптографии часто полагают, что шифр является достаточно надежным, если он может выдержать атаку первого типа (только зашифрованный текст). Такое предположение весьма наивно. Во многих случаях криптоаналитик может угадать часть зашифрованного текста. Например, первое, что говорят многие компьютеры при входе в систему, это `login:`. После того как криптоаналитик получит несколько соответствующих друг другу пар кусков зашифрованного и открытого текста, его работа становится значительно легче. Для обеспечения секретности нужна предусмотрительность криптографа, который должен гарантировать, что система не будет взломана, даже если его оппонент сможет закодировать несколько фрагментов открытого текста по своему выбору.

Исторически методы шифрования разделились на две категории: метод подстановки и метод перестановки. Мы кратко рассмотрим их в качестве введения в современную криптографию.

8.1.2. Метод подстановки

В шифрах, основанных на **методе подстановки (substitution cipher)**, каждый символ или группа символов заменяется другим символом или группой символов. Одним из древнейших шифров является приписываемый Юлию Цезарю (Julius Caesar) **шифр Цезаря (Caesar cipher)**. Этот шифр заменяет все буквы алфавита на другие с помощью циклического сдвига на три позиции. Так буква *a* становится буквой *D*, *b* становится *E*, *c* превращается в *F*, ... , а *z* — в *C*. Например, слово *attack* превращается в *DWWDFN*. В наших примерах открытый текст будет обозначаться строчными символами, а зашифрованный текст — прописными.

Некоторое обобщение шифра Цезаря представляет собой сдвиг алфавита не на три символа, а на произвольное число *k* символов. В этом случае *k* становится ключом к общему методу циклически сдвигаемых алфавитов. Шифр Цезаря, возможно, и сумел обмануть жителей Помпеи, но с тех пор ему более уже никого не удалось ввести в заблуждение.

Следующее усовершенствование состоит в установлении соответствия каждому встречающемуся в открытом тексте символу другого символа. Например,

```
открытый текст:      a b c d e f g h i j k l m n o p q r s t u v w x y z
зашифрованный текст: Q W E R T Y U I O P A S D F G H J K L Z X C V B N M
```

Такая система называется **моноалфавитным подстановочным шифром (monoalphabetic substitution cipher)**, ключом к которому является 26-символьная строка, соответствующая полному алфавиту. В нашем примере слово *attack* будет выглядеть, как *QZZQEA*.

На первый взгляд такая система может показаться надежной, так как, даже если криптоаналитику известна общая система, он не знает, какой из 26! ≈ 4 × 1026 возможных вариантов ключа применить. В отличие от шифра Цезаря применение метода простого перебора в данном случае весьма сомнительно. Даже при затратах 1 нс на проверку одного варианта ключа, чтобы перепробовать все ключи, миллиону компьютерных чипов, работающих одновременно, понадобится около 10 000 лет.

Тем не менее подобный шифр легко взламывается даже при наличии довольно небольших фрагментов зашифрованного текста. Для атаки шифра может быть использовано преимущество статистических характеристик естественных языков. Например, в английском языке буква *e* встречается в тексте чаще всего. Следом за ней по частоте использования идут буквы *t*, *o*, *a*, *n*, *i* и т. д. Наиболее часто встречающимися комбинациями из двух символов (**биграммами** — **digrams**) являются *th*, *in*, *er*, *re* и *an*. Наиболее часто встречающимися комбинациями из трех символов, или **триграммами** (**trigrams**), являются *the*, *ing*, *and* и *ion*.

Криптоаналитик, пытающийся взломать моноалфавитный шифр, начнет с того, что сосчитает относительные частоты всех символов алфавита в зашифрованном тексте. Затем он может попытаться заменить наиболее часто встречающийся символ буквой *e*, а следующий по частоте — буквой *t*. Затем он посмотрит на триграммы и попытается найти что-либо похожее на *tXe*, после чего он сможет предположить, что *X* — это *h*. Аналогично, если последовательность *thYt* встречается достаточно часто, то, вероятно, *Y* обозначает символ *a*. Обладая этой информацией, криптоаналитик может поискать часто встречающуюся триграмму вида *aZW*, что, скорее всего, означает *and*. Продолжая в том же духе, угадывая буквы, биграммы, триграммы и зная, какие последовательности символов являются наиболее вероятными, криптоаналитик побуквенно восстанавливает исходный текст.

Другой метод заключается в угадывании сразу целого слова или фразы. Например, рассмотрим следующий зашифрованный текст, полученный от бухгалтерской фирмы (разбитый на блоки по пять символов):

```
CTBMN BYCTC BTJDS QXBNS GSTJC BTSWX CTQTZ CQVUJ
QJSGS TJQZZ MNQJS VLNSX VSZJU JDSTS JQUUS JUBXJ
DSKSU JSNTK BGAQJ ZBGYQ TLCTZ BNYBN QJSW
```

В сообщении бухгалтерской фирмы, скорее всего, должно встречаться слово *financial* (финансовый). Используя тот факт, что в этом слове буква *i* встречается дважды, разделенная четырьмя другими буквами, мы будем искать в зашифрованном тексте повторяющиеся символы, отстоящие друг от друга на это расстояние. В результате мы найдем 12 таких мест в тексте в позициях 6, 15, 27, 31, 42, 48, 56, 66, 70, 71, 76 и 82. Однако только в двух случаях, в позициях 31 и 42, следующий символ (соответствующий букве *n* в открытом тексте) повторяется в соответствующем месте. Из этих двух вариантов символ *a* будет иметь правильное расположение только для позиции 31. Таким образом, теперь нам известно, что слово *financial* начинается в позиции 30. Далее можно продолжать, применяя лингвистическую статистику английского языка и угадывая целые слова.

8.1.3. Метод перестановки

Шифры, основанные на методе подстановки, сохраняют порядок символов, но подменяют их. Шифры, использующие **метод перестановки** (**transposition ciphers**), меняют порядок следования символов, но не изменяют сами символы. На рис. 8.2 показан простой перестановочный шифр с колоночной перестановкой. Ключом к шифру служит слово или фраза, не содержащая повторяющихся букв. В данном примере в качестве ключа используется слово MEGABUCK. Цель ключа — пронумеровать колонки. Пер-

вой колонкой становится колонка под буквой, расположенной ближе всего к началу алфавита и т. д. Открытый текст записывается горизонтально в строках. Шифрованный текст читается по колонкам, начиная с колонки с младшей ключевой буквой.

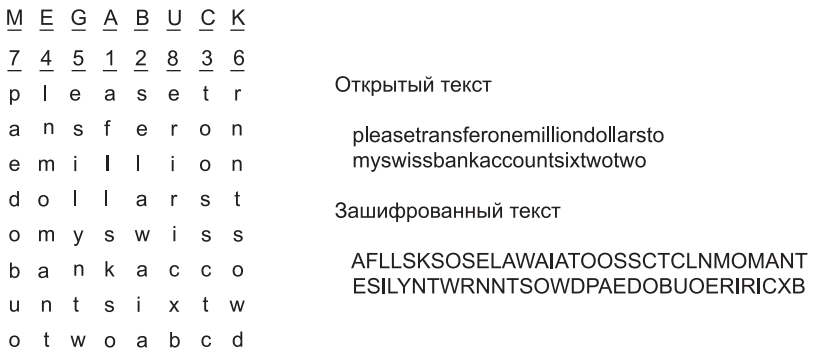


Рис. 8.2. Перестановочный шифр

Чтобы взломать перестановочный шифр, криптоаналитик должен вначале понять, что он имеет дело именно с перестановочным шифром. Если взглянуть на частоту символов *E*, *T*, *A*, *O*, *I*, *N* и т. д., легко заметить, что их частоты соответствуют нормальным частотам открытого текста. В таком случае, очевидно, что этот шифр является перестановочным, так как каждая буква в таком шифре представляет сама себя.

Затем нужно угадать число колонок. Во многих случаях по контексту сообщения можно угадать слово или фразу. Например, предположим, что криптоаналитик подозревает, что где-то в сообщении должно встретиться словосочетание *milliondollars*. Обратите внимание, что в результате того, что эти слова присутствуют в исходном тексте, в шифрованном тексте встречаются биграммы *MO*, *IL*, *LL*, *LA*, *IR* и *OS*. Символ *O* следует за символом *M* (то есть они стоят рядом по вертикали в колонке 4), так как они разделены в предполагаемой фразе дистанцией, равной длине ключа. Если бы использовался ключ длиной семь, тогда вместо перечисленных выше биграмм встречались бы следующие: *MD*, *IO*, *LL*, *LL*, *IA*, *OR* и *NS*. Таким образом, для каждой длины ключа в шифрованном тексте образуется различный набор биграмм. Перебрав различные варианты, криптоаналитик часто довольно легко может определить длину ключа.

Остается узнать только порядок колонок. Если число колонок k невелико, можно перебрать все $k(k-1)$ возможных комбинаций пар соседних колонок, сравнивая частоты образующихся биграмм со статистическими характеристиками английского языка. Пара с лучшим соответствием считается правильно позиционированной. Затем все оставшиеся колонки по очереди проверяются в сочетании с уже найденной парой. Колонка, в которой биграммы и триграммы дают максимальное совпадение со статистикой, предполагается правильной. Весь процесс повторяется, пока не будет восстановлен порядок всех колонок. Есть шанс, что на данном этапе текст уже будет распознаваемым (например, если вместо слова *million* мы увидим *milloin*, то сразу станет ясно, где сделана ошибка).

Некоторые перестановочные шифры принимают блок фиксированной длины на входе и выдают блок фиксированной длины на выходе. Такие шифры полностью опре-

деляются списком, сообщающим порядок, в котором символы попадают в выходной блок. Например, шифр на рис. 8.2 можно рассматривать в виде шифра с 64-символьным блоком. Его выход описывается последовательностью чисел 4, 12, 20, 28, 36, 44, 52, 60, 5, 13, ..., 62. Другими словами, четвертая входная буква, *a*, первой появится на выходе, за ней последует двенадцатая, *f*, и т. д.

8.1.4. Одноразовые блокноты

Разработать шифр, который невозможно взломать, на самом деле весьма просто. Методика для этого известна уже несколько десятилетий. В качестве ключа выбирается произвольная битовая строка, длина которой совпадает с длиной исходного текста. Открытый текст также преобразуется в последовательность двоичных разрядов, например, с помощью стандартной кодировки ASCII. Наконец, эти две строки поразрядно складываются по модулю 2 (операция «исключающее ИЛИ», XOR). Полученный в результате зашифрованный текст взломать невозможно, поскольку в достаточно большом отрывке любая буква, биграмма или триграмма будет равновероятной. Этот метод, известный как **одноразовый блокнот (one-time pad)**, теоретически является панацеей от любых атак, независимо от вычислительных мощностей, которыми обладает или будет когда-либо в будущем обладать взломщик. Объясняется этот невероятный факт с помощью теории информации: дело в том, что в зашифрованном сообщении не содержится никакой информации для взломщика, поскольку любой открытый текст является равновероятным кандидатом.

Пример практического использования одноразового блокнота показан на рис. 8.3. Для начала фраза «I love you» («Я люблю тебя») преобразуется в 7-битный ASCII-код. Затем выбирается одноразовая последовательность, *Последовательность 1*, которая складывается по модулю 2 с сообщением. В результате получается некий шифр. Для того чтобы его разгадать, криптоаналитику придется перебрать все возможные одноразовые последовательности, всякий раз проверяя, каким получается открытый текст. Например, если попробовать расшифровать послание с помощью *Последовательности 2* (см. рис. 8.3), получится текст «Elvis lives» («Элвис жив»). Обсуждение правдоподобности этого утверждения выходит за рамки данной книги, однако, как мы видим, исходное сообщение разгадать не удалось, но при этом получилась вполне нормальная с точки зрения английской грамматики фраза. На самом деле, для любой последовательности из 11 символов в кодировке ASCII найдется одноразовый блокнот для ее генерации. Именно это мы имеем в виду, говоря, что в зашифрованном тексте не содержится никакой информации: из него можно извлечь любое сообщение подходящей длины.

Одноразовые ключи теоретически являются очень мощным инструментом, однако у них есть ряд практических недостатков. Во-первых, такой длинный ключ невозможно запомнить, поэтому и отправитель, и получатель должны носить с собой письменную копию ключа. Если есть опасность того, что одна из этих копий может быть захвачена неприятелем, хранение письменных копий оказывается весьма нежелательным. Кроме того, полный объем данных, которые могут быть переданы, ограничен размером доступного ключа. Если шпиону повезет и он добудет большое количество информации, может выясниться, что он не сможет передать все эти сведения в центр, так как ему не хватит длины ключа. Еще одна проблема заключается в чувствитель-

ности данного метода к потерянным или вставленным символам. Если отправитель или получатель потеряют синхронизацию, все данные, начиная с этого места, будут испорчены.

Сообщение 1:
1001001 0100001 1101100 1101111 1110110 1100101 0100000 1111001 1101111 1110101 0101110
Последовательность 1:
1010010 1001011 1110010 1010101 1010010 1100011 0001011 0101010 1010111 1100110 0101011
Зашифрованный текст:
0011011 1101011 0011110 0111010 0100100 0000110 0101011 1010011 0111000 0010011 0000101

Последовательность 2:
1011110 0000111 1101000 1010011 1010111 0100110 1000111 0111010 1001110 1110110 1110110
Открытое сообщение 2
1000101 1101100 1110110 1101001 1110011 0100000 1101100 1101001 1110110 1100101 1110011

Рис. 8.3. Использование одноразового блокнота для шифрования сообщений и возможность получения произвольного открытого сообщения из зашифрованного путем подстановки другой ключевой последовательности

С появлением компьютеров метод одноразового блокнота может получить практическое применение. Ключ можно хранить на специальном диске DVD, содержащем несколько гигабит информации. Он даже не вызовет особых подозрений, если его перевозить в коробке от видеокомпакт-диска и в начале даже записать несколько минут фильма. Конечно, в сетях со скоростями передачи данных, исчисляющимися гигабитами, необходимость менять компакт-диск через каждые 30 секунд быстро утомит. Получается, что DVD с одноразовым ключом нужно вручить лично в руки будущему получателю секретного сообщения, а такой подход резко снижает вероятность практического использования метода одноразового блокнота.

Квантовая криптография

Интересно, что решение проблемы передачи по сети одноразовой последовательности пришло из довольно далекой науки — квантовой механики. Эта область остается экспериментальной, однако многообещающей. Если удастся усовершенствовать этот метод, все задачи криптографии можно будет решать с помощью одноразовых последовательностей, ведь это самый надежный способ защиты информации. Ниже мы вкратце опишем суть технологии, называемой **квантовой криптографией**. В частности, мы рассмотрим протокол **BB84**, названный так в честь его создателей и года, в котором его описание было впервые опубликовано (Bennet и Brassard, 1984).

Допустим, пользователь по имени Алиса хочет передать одноразовую последовательность другому пользователю, Бобу. Алиса и Боб называются **принципалами (principals)**, это главные герои в нашей истории. Например, Боб может быть банкиром, а Алиса — лицом, желающим вступить в деловые отношения с ним. Имена «Алиса» и «Боб» традиционно используются для обозначения принципалов практически во всех материалах, касающихся криптографии, с тех пор как Рон Ривест использовал их в первый раз много лет назад (Rivest и др., 1978). Криптографы вообще обожают разного рода традиции. Если бы мы стали описывать тут взаимоотношения Андрея и Беллы вместо Алисы и Боба, ни одному слову в этой главе никто бы не поверил

(стало бы понятно, что автор, на самом деле, далек от криптографии). А так, может быть, поверят. Поэтому пусть Алиса и Боб будут героями нашей книги.

Итак, если Алисе и Бобу удастся установить единую одноразовую последовательность, их переговоры будут полностью конфиденциальными. Задача стоит следующим образом: как им обменяться секретным ключом, не передавая друг другу DVD? Мы можем предположить, что оба пользователя находятся на разных концах одного оптоволоконного кабеля, по которому они могут передавать и принимать световые импульсы. Тем не менее нашлась бесстрашная шпионка Труди, которой удалось установить на пути этого кабеля активное подслушивающее устройство. Она может считывать сигналы, идущие в обоих направлениях. Кроме того, она может посылать как в одну, так и в другую сторону фальшивые сообщения. Ситуация для Алисы и Боба, казалось бы, безнадежная. Но тут на помощь приходит квантовая криптография, и мы сейчас увидим, что у наших героев появляется лучик надежды.

Квантовая криптография базируется на том факте, что свет передается маленькими порциями, называемыми **фотонами** и обладающими некоторыми необычными свойствами. Кроме того, пропуская свет через поляризационный фильтр, можно добиться его поляризации. Этот факт известен, прежде всего, тем, кто носит солнцезащитные очки, а также фотографам. Световой луч (то есть поток фотонов), проходя через такой фильтр, поляризуется в направлении оси фильтра (например, вертикально). Если после этого пропустить луч через второй фильтр, интенсивность света на выходе будет пропорциональна квадрату косинуса угла между осями фильтров. Если оси расположить перпендикулярно, фотоны через фильтры проникнуть не смогут. Абсолютная ориентация осей в пространстве значения не имеет — важно только их взаимное расположение.

Чтобы сгенерировать одноразовый блокнот, Алисе понадобятся два набора поляризационных фильтров. Первый набор состоит из вертикального и горизонтального фильтров. Это называется **прямолинейным базисом**. Базис — это просто система координат. Второй набор фильтров отличается от первого только тем, что он повернут на 45°, то есть один фильтр можно представить в виде линии, идущей из нижнего левого угла в верхний правый, а другой — из верхнего левого в нижний правый угол. Это называется **диагональным базисом**. Итак, у Алисы есть два набора фильтров, и она может поставить любой из них на пути светового луча. Но четыре стеклышка-поляризатора — это нечто в стиле сказки про Алису в стране чудес. В реальности имеется кристалл, одна из четырех возможных поляризаций которого изменяется электронным способом с огромной скоростью. У Боба есть такое же устройство, как у Алисы. Тот факт, что у Алисы и Боба есть по два базиса, играет важную роль в квантовой криптографии.

В каждом базисе Алиса обозначает одно из направлений нулем, а другое соответственно единицей. В примере, показанном ниже, мы предполагаем, что она выбрала вертикальное направление в качестве нулевого, а горизонтальное — в качестве единичного. Независимо от этого, направлению «нижний левый — верхний правый» присваивается значение 0, а направлению «верхний левый — нижний правый» — 1. Эта информация передается Бобу в виде открытого текста.

Теперь Алиса берет одноразовый блокнот, построенный, например, генератором случайных чисел (этот генератор сам по себе, кстати, представляет собой довольно сложный предмет), и передает его Бобу. Передача производится поразрядно, для каж-

дого бита один из двух базисов выбирается случайным образом. Чтобы передать бит, фотонная пушка испускает один фотон, поляризованный таким образом, чтобы он мог пройти через базис, выбранный для этого бита. Например, базисы могут выбираться в такой последовательности: диагональный, прямолинейный, прямолинейный, диагональный, прямолинейный и т. д. Чтобы передать одноразовый блокнот, состоящий из последовательности 1001110010100110, с помощью этих базисов, посылаются фотоны, показанные на рис. 8.4, *а*. Для данного одноразового блокнота и соответствующей ему последовательности базисов единственным образом определяется поляризация, применяемая для каждого бита. Биты, посылаемые одним фотоном за единицу времени, называются **кубитами (qubits)**.

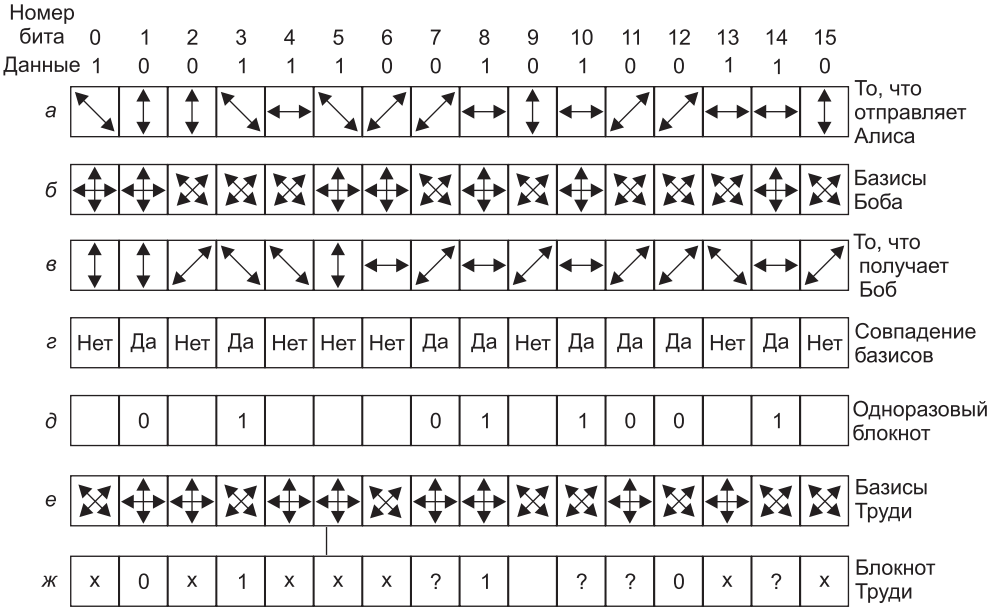


Рис. 8.4. Пример квантовой криптографии

Боб не знает, какой базис надо использовать, поэтому он использует базисы в случайном порядке для каждого прибывающего фотона, как показано на рис. 8.4, *б*. Если для данного фотона базис выбран правильно, Боб получит правильный бит. В противном случае значение бита будет случайным, так как фотон, проходя через поляризатор, повернутый на 45° относительно его собственной поляризации, с равными вероятностями попадет на направление, соответствующее единице или нулю. Это свойство фотонов является одним из основных во всей квантовой механике. Таким образом, некоторые биты будут получены правильно, некоторые — нет, но Боб не может распознать, какие из них являются правильными. Получаемая Бобом информация показана на рис. 8.4, *в*.

Так как же Боб узнает, какие базисы были подставлены правильно, а какие — нет? Для этого он открытым текстом сообщает Алисе, какие базисы он использовал при приеме каждого бита, а она в ответ сообщает (также открытым текстом), какие из

базисов были подобраны Бобом корректно. Это показано на рис. 8.4, *г*. Владея этой информацией, Алиса и Боб могут составить битовую строку корректных предположений, что показано на рис. 8.4, *д*. В среднем длина этой строки будет равна половине полной длины исходной строки, однако, так как обеим сторонам это известно, они могут использовать строку корректных предположений в качестве одноразового блокнота. Все, что Алисе надо сделать, это передать битовую строку, длина которой немного превышает удвоенную длину одноразового блокнота. Проблема решена.

Но погодите, мы же забыли про Труди! Предполагается, что ей очень хочется узнать, о чем говорит Алиса, поэтому она внедряет в линию передачи свой детектор и передатчик. К сожалению (для Труди), она тоже не знает, через какой базис пропускать каждый фотон. Лучшее, что она может сделать, это выбирать базисы случайным образом, как Боб. Пример того, как она это делает, показан на рис. 8.4, *е*. Когда Боб открытым текстом сообщает Алисе, какие базисы он использовал, а та отвечает ему, какие из них были правильные, Труди, как и Боб, узнает, какие биты она угадала, а какие — нет. Как видно из рисунка, базисы Труди совпали с базисами Алисы в позициях 0, 1, 2, 3, 4, 6, 8, 12 и 13. Однако из ответа Алисы (см. рис. 8.4, *г*) ей становится известно, что в одноразовый блокнот входят только биты 1, 3, 7, 8, 10, 11, 12 и 14. Четыре из этих битов (1, 3, 8 и 12) были угаданы правильно, Труди их запоминает. Остальные биты (7, 10, 11 и 14) не были угаданы, и их значения остаются для Труди неизвестными. Таким образом, Бобу известен одноразовый блокнот, начинающийся с последовательности 01011001 (рис. 8.4, *д*), а все, что досталось Труди, — это обрывок 01?1??0? (рис. 8.4, *ж*).

Конечно, Алиса и Боб осознают, что Труди пытается захватить часть их одноразового блокнота, поэтому они стараются уменьшить количество той информации, которая может ей достаться. Для этого они могут произвести некоторые действия над последовательностью. Например, одноразовый блокнот можно поделить на блоки по 1024 бита и возвести каждый из блоков в квадрат, получая, таким образом, числа длиной 2048 бит. Затем можно использовать конкатенацию 2048-битных чисел в качестве одноразового блокнота. Имея лишь часть битовой строки, Труди никогда не сможет проделать эти преобразования. Действия над исходным одноразовым блокнотом, уменьшающие долю информации, получаемой Труди, называются **усилением секретности (privacy amplification)**. На практике вместо поблочного возведения в квадрат применяются сложные преобразования, в которых каждый выходной бит зависит от каждого входного.

Бедная Труди. У нее не только нет идей по поводу того, как выглядит одноразовый блокнот, но она к тому же понимает, что ее присутствие не является ни для кого секретом. Как-никак, ей приходится передавать каждый принятый бит Бобу, чтобы он думал, будто разговаривает с Алисой. Беда в том, что лучшее, что может сделать Труди, это передавать каждый принятый квантобит с использованием поляризации, с помощью которой он был принят ею. При этом примерно в половине случаев поляризация будет неправильной, что приведет к появлению множества ошибок в одноразовом блокноте Боба.

Когда, наконец, Алиса начинает отправлять данные, она шифрует их, используя сложный код с упреждающей коррекцией ошибок. С точки зрения Боба, ошибка в одном бите одноразовой последовательности — это то же самое, что ошибка передачи данных, произошедшая в одном бите. В любом случае, результат состоит в полу-

чении неправильного значения бита. С помощью упреждающей коррекции ошибок, возможно, удастся восстановить потерянную информацию, но можно, кроме того, сосчитать количество ошибок. Если оно намного превышает ожидания (связанные с вероятностью возникновения ошибок оборудования), становится очевидно, что линию прослушивает Труди, и Боб может принять необходимые меры (например, сказать Алисе, чтобы она переключилась на радиоканал, вызвать полицию и т. д.). Если бы Труди могла скопировать фотон и обрабатывать свою копию, а исходный фотон пересылать Бобу в неизменном виде, у нее был бы шанс остаться незамеченной, однако на сегодняшний день отсутствуют методы клонирования фотонов. Но даже если Труди удастся получить копию фотона, значения квантовой криптографии это ни в коей мере не умалит.

Квантовая криптография может работать при наличии оптоволоконных каналов длиной 60 км, однако требуется сложное и дорогое оборудование. Несмотря на это, сама идея весьма многообещающая. Более подробно о квантовой криптографии можно узнать из книги (Mullins, 2002).

8.1.5. Два фундаментальных принципа криптографии

Хотя на следующих страницах мы рассмотрим много различных криптографических систем, в основе их всех лежат два принципа, очень важных для понимания. Обратите на них особое внимание — нарушать их очень опасно.

Избыточность

Первый принцип гласит, что все зашифрованные сообщения должны содержать определенную избыточность, то есть информацию, не требующуюся для понимания сообщения. Поясним это на примере. Представим себе компанию «Домосед», торгующую по почтовым заказам товарами 60 000 наименований. Радуясь, что им удалось так экономно распорядиться ресурсами, программисты компании «Домосед» решили, что весь бланк заказа будет состоять из 16 байт имени клиента, за которым последует поле товара из 3 байт (1 байт для обозначения количества и 2 байта для идентификатора товара). Последние три байта было решено закодировать с помощью очень длинного ключа, известного только клиенту и компании «Домосед».

На первый взгляд, такая схема может показаться надежной, в частности, потому, что пассивные злоумышленники не смогут расшифровать сообщения. К сожалению, в этой схеме имеется критический недостаток, полностью ее обесценивающий. Предположим, какой-нибудь недавно уволенный сотрудник хочет отомстить компании «Домосед» за увольнение. Перед самым уходом ему удается забрать с собой часть списка клиентов. За ночь он составляет программу, посылающую фиктивные заказы с настоящими именами клиентов. Поскольку списка ключей у него нет, он просто помещает в последние три байта случайные числа и посылает сотни заказов компании «Домосед».

Когда эти сообщения прибывают, компьютер компании «Домосед» по имени клиента находит ключ для дешифрации сообщения. К несчастью для компании «Домосед», почти все 3-байтовые сообщения могут восприниматься как достоверные,

поэтому компьютер начинает печатать заявки на доставку товаров. Хотя может показаться странным, если клиент заказывает 837 сидений для детских качелей или 540 песочниц, однако вполне возможно, что клиент собирается заняться строительством детских игровых площадок. Таким образом, активный злоумышленник (уволенный сотрудник) способен доставить очень много неприятностей, даже не вникая в смысл сообщений, посылаемых его компьютером.

Эта проблема может быть решена при помощи добавления избыточной информации к каждому сообщению. Например, если добавить к трем шифруемым байтам еще девять, например, нулевых, уволенный сотрудник уже не сможет сформировать серьезный поток достоверно выглядящих сообщений. Мораль этой истории в том, что все сообщения должны содержать достаточное количество избыточной информации, чтобы активный злоумышленник не смог выдать случайный мусор за настоящие сообщения.

Однако добавление избыточной информации облегчает работу криптоаналитика по взлому шифра. Предположим, что конкуренция в бизнесе почтовых заказов чрезвычайно высока и что главному конкуренту компании «Домосед», фирме «Лежебока», очень хочется узнать, сколько песочниц в месяц продает компания «Домосед». Для этого «лежебоки» подключились к телефонной линии «домоседов». В исходной схеме с 3-байтовыми номерами криптоанализ был почти невозможен, поскольку, предположив значение ключа, криптоаналитик не мог проверить правильность своей догадки. Как-никак, почти все сообщения были технически корректны. С введением новой 12-байтовой схемы криптоаналитик легко сможет отличить допустимое сообщение от недопустимого. Итак:

Криптографический принцип номер 1: Сообщения должны содержать избыточные данные.

Другими словами, при расшифровке сообщения получатель должен иметь возможность проверить его подлинность путем анализа и, возможно, выполнения несложных вычислений. Избыточность требуется для того, чтобы можно было противостоять попыткам активных злоумышленников обмануть получателя фальшивыми сообщениями, содержащими мусор. Вместе с тем, добавление избыточной информации облегчает пассивным злоумышленникам задачу взлома системы, так что здесь есть определенное противоречие. Кроме того, избыточные данные никогда не должны принимать форму последовательности нулей в начале или в конце сообщения, так как при зашифровке подобных сообщений некоторые алгоритмы дают более предсказуемые результаты, что облегчает работу криптоаналитиков. Многочлен циклического избыточного кода (CRC) значительно лучше подойдет для этих целей, чем просто ряд нулей, поскольку получатель сможет проверить его корректность, и это несколько усложняет работу криптоаналитика. Гораздо удобнее использовать криптографическую хэш-функцию, речь о которой пойдет ниже. Пока что просто запомните, что это более надежный вариант CRC.

Возвращаясь к квантовой криптографии, хочется сказать о том, какую роль в этой технологии играет избыточность. Из-за того, что Труди перехватывает фотоны, некоторые биты одноразового блокнота, получаемого Бобом, будут иметь неправильные значения. Бобу требуется некоторая избыточность информации, содержащейся во входящих сообщениях, для определения наличия ошибок. Одной из грубых форм

избыточности можно считать повторение передачи одного и того же сообщения. Если две пришедшие копии оказываются не идентичными, Боб узнает, что либо канал сильно зашумлен, либо кто-то перехватывает данные. Конечно, отправлять все по два раза — это слишком. Гораздо лучше использовать код Хэмминга или Рида—Соломона для определения и коррекции ошибок. Однако должно быть понятно, что для того чтобы отличать настоящие сообщения от поддельных, необходима некоторая избыточность. Это особенно важно, если присутствует активный злоумышленник.

Ограниченный срок годности

Второй принцип криптографии состоит в том, что необходимо иметь методы, позволяющие удостовериться в том, что пришедшее сообщение свежее, то есть было послано совсем недавно. Эта мера направлена на борьбу с активными злоумышленниками, воспроизводящими перехваченные ими старые сообщения. Если не принять подобных мер, наш уволенный может подсоединиться к телефонной линии компании «Домосед» и просто повторять посланные ранее настоящие сообщения. Итак, сформулируем утверждение:

Криптографический принцип номер 2: Необходим способ борьбы с повторной отправкой посланных ранее сообщений

Одной из подобных мер является включение в каждое сообщение временного штампа, действительного, скажем, только в течение 10 с. Получатель может просто хранить принятые сообщения в течение 10 секунд, отсеивая дубликаты. Сообщения возрастом более 10 с просто игнорируются как устаревшие. Другие меры защиты от дубликатов будут обсуждаться ниже.

8.2. Алгоритмы с симметричным криптографическим ключом

В современной криптографии применяются те же основные идеи, что и в традиционной криптографии, то есть перестановка и подстановка, но акценты расставляются иначе. В традиционной криптографии применялись простые алгоритмы. Сегодня верно обратное: целью является создание настолько сложного и запутанного алгоритма шифрования, что даже если криптоаналитику попадут в руки целые горы зашифрованного текста, он не сможет извлечь из этого никакой пользы без ключа.

Первым классом алгоритмов шифрования, который мы изучим, будет класс **алгоритмов с симметричным ключом (symmetric-key algorithms)**. Он получил такое название благодаря тому, что для шифрования и дешифрации сообщений применяется один и тот же ключ. На рис. 8.1 показан пример использования алгоритма с симметричным ключом. В частности, мы подробно рассмотрим **блочные шифры (block ciphers)**, которые принимают на входе n-битные блоки открытого текста и преобразуют их с использованием ключа в n-битный шифр.

Криптографические алгоритмы могут быть реализованы как аппаратно (что повышает скорость их работы), так и программно (для повышения гибкости). Несмотря

на то что большая часть наших размышлений касается алгоритмов и протоколов, не зависящих от конкретной реализации, нам кажется полезным рассмотрение принципов построения криптографической аппаратуры. Подстановки и перестановки могут быть реализованы при помощи простых электрических цепей. На рис. 8.5, *а* показано устройство, называющееся **P-блоком** (литера P означает permutation — перестановка) и используемое для перестановки восьми входных разрядов. Если пронумеровать входные биты сверху вниз (01234567), выход этого конкретного P-блока будет выглядеть как 36071245. При помощи соответствующего внутреннего устройства P-блока (распайки проводов) можно заставить его выполнять любую операцию перестановки практически со скоростью света, так как никакие вычисления в нем не производятся, а просто-напросто передается сигнал с входа на выход. Такое решение соответствует принципу Керкгофа: взломщик знает, что используется метод перестановки битов. Однако он не знает ключа, заключающегося в порядке перестановок.

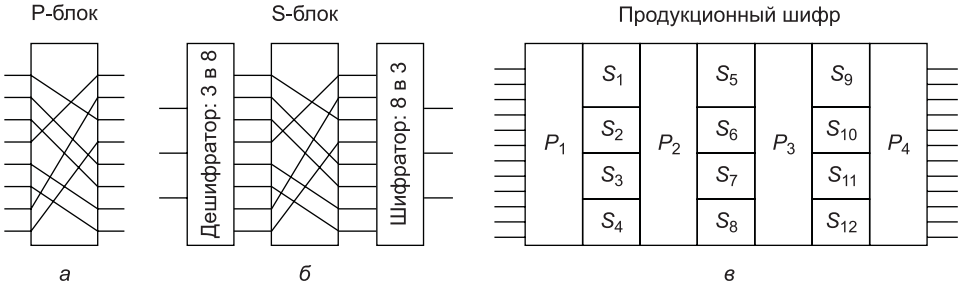


Рис. 8.5. Основные элементы продукционных шифров: P-блок (*а*); S-блок (*б*); продукционный шифр (*в*)

Подстановки (то есть замещения) выполняются **S-блоками** (S означает substitution — подстановка, замена), как показано на рис. 8.5, *б*. В данном примере на вход подается 3-битный открытый текст, а на выходе появляется 3-битный зашифрованный текст. Для каждого входного сигнала выбирается одна из восьми выходных линий декодера путем установки ее в 1. Все остальные линии устанавливаются в 0. Затем эти восемь линий проходят через P-блок, представляющий собой вторую ступень S-блока. Третья ступень производит обратное кодирование одной из восьми линий в 3-битовое двоичное число. Такое устройство заменяет восьмеричные числа 01234567 на 24506713 соответственно. То есть 0 заменяется числом 2, 1 — числом 4 и т. д. Опять же, при соответствующей распайке проводов P-блока внутри S-блока можно реализовать любой вариант подстановки. К тому же такое устройство может быть встроено в аппаратуру и работать на огромных скоростях, поскольку шифраторы и дешифраторы вносят лишь одну или две вентильных задержки (менее 1 нс), а время распространения сигнала внутри P-блока может быть менее 1 пс.

Настоящая сила этих элементов становится очевидна, если сформировать каскад из этих устройств, как показано на рис. 8.5, *в*. Получившееся в результате устройство называется **продукционным шифром (product cipher)**. В данном примере на первом этапе (P_1) 12 входных линий меняются местами. На второй ступени вход разбивается на четыре группы из трех бит, с каждой из которых операция замены выполняется не-

зависимо (S_1 до S_4). Данный метод представляет собой составление большего S-блока из нескольких меньших S-блоков. Данный способ достаточно целесообразен, так как небольшие S-блоки удобны при аппаратной реализации (то есть восьмиразрядный S-блок может быть реализован как 256-разрядная таблица перекодировки), но большие S-блоки строить неудобно (например, двенадцатиразрядный S-блок потребует как минимум 212 = 4096 перекрещенных проводов в средней стадии). Хотя такой метод представляет собой лишь частный случай общего решения, его мощь достаточно высока. Выход продукционного шифра можно сделать сложной функцией входа, используя достаточно большое количество дополнительных ступеней.

Продукционные шифры, работающие с k-битными входами и производящие k-битные последовательности, широко распространены. Обычно значение k колеблется от 64 до 256.

8.2.1. Стандарт шифрования данных DES

В январе 1977 года правительство Соединенных Штатов приняло продукционный шифр, разработанный фирмой IBM, в качестве официального стандарта для несекретных сведений. Этот шифр, получивший название **DES (Data Encryption Standard — стандарт шифрования данных)**, широко использовался в промышленности для защиты информации. В своем исходном виде он уже больше не является надежным, но в модифицированном виде все еще полезен. Сейчас мы объясним принципы его работы.

Схема шифра DES показана на рис. 8.6, *а*. Открытый текст шифруется блоками по 64 бита, в результате чего на выходе получаются 64-битные блоки зашифрованного текста. Алгоритм, использующий 56-разрядный ключ, состоит из 19 отдельных этапов. На первом этапе выполняется независимая перестановка 64 разрядов открытого текста. Последний представляет собой обратную перестановку. Предпоследний этап меняет местами левые и правые 32 разряда. Остальные 16 этапов функционально идентичны, но управляются разными функциями входного ключа. Алгоритм был разработан так, чтобы дешифрация выполнялась тем же ключом, что и шифрование. Это обеспечивает соответствие алгоритма принципу симметричных ключей. Этапы при расшифровке просто выполняются в обратном порядке.

Операция, выполняемая на одном из промежуточных этапов, показана на рис. 8.6, *б*. На каждом этапе из двух порций по 32 разряда на входе формируется две порции по 32 разряда на выходе. Правая половина входа просто копируется в левые разряды выхода. Правые 32 выходных разряда представляют собой сумму по модулю 2 левой части входа и функции правой части входа и ключа данного этапа K_i. Вся сложность алгоритма заключается в этой функции.

Функция состоит из четырех последовательно выполняемых шагов. Сначала из 32 разрядов правой части $R_i - 1$ с помощью фиксированной перестановки и дублирования формируется 48-разрядное число E. На втором шаге число E и ключ K_i складываются по модулю 2. Затем выход разделяется на восемь групп по шесть разрядов, каждая из которых преобразуется независимым S-блоком в 4-разрядные группы. Наконец, эти 8×4 разряда пропускаются через P-блок.

На каждом из 16 этапов используются различные функции исходного ключа. Перед началом работы алгоритма к ключу применяется 56-разрядная перестановка. Перед

каждым этапом ключ разделяется на две группы по 28 разрядов, каждая из которых вращается влево на число разрядов, зависящее от номера этапа. Ключ K_i получается из результата этой операции при помощи еще одной перестановки 56 разрядов. На каждом этапе из 56 разрядов ключа выбираются 48 разрядов, которые также переставляются местами.

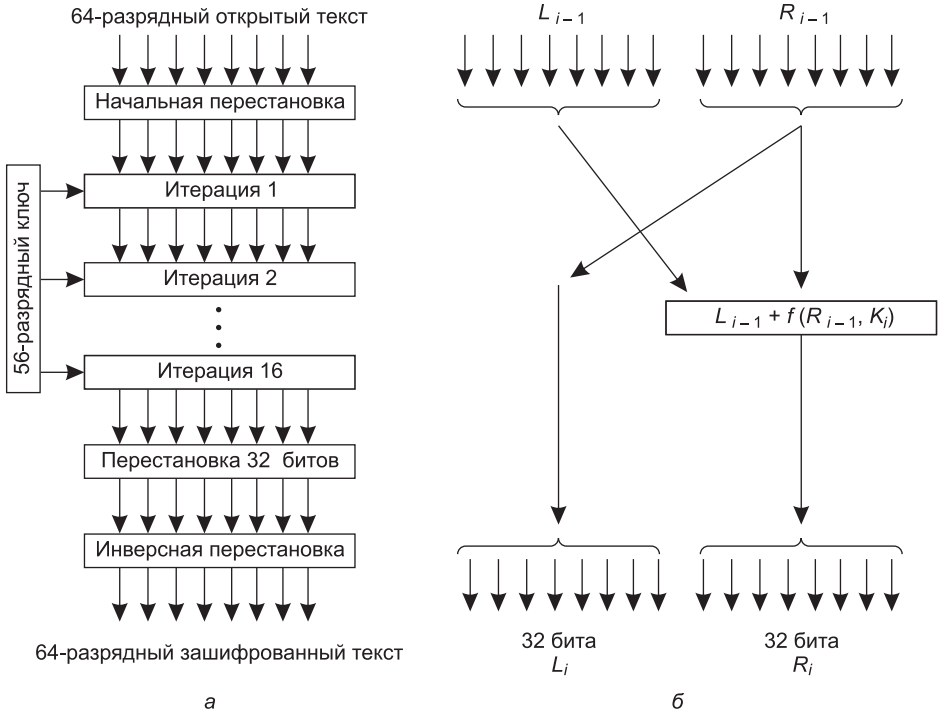


Рис. 8.6. Стандарт шифрования данных DES: *а* — общий вид; *б* — детализация одного из этапов

Иногда для повышения надежности DES используется метод, называемый **побелкой (whitening)**. Он заключается в том, что перед передачей шифра каждый блок открытого текста складывается по модулю 2 с произвольным 64-битным ключом, затем отправляется в устройство DES, после чего получившийся шифр складывается по модулю 2 со вторым 64-битным ключом. На приемном конце побелка легко устраняется путем выполнения обратных операций (это возможно, если у получателя есть два побелочных ключа). Поскольку применение этого метода увеличивает длину ключа, полный перебор в пространстве значений ключа становится еще более длительным. Обратите внимание: для побелки каждого блока применяется один и тот же ключ (то есть для каждого сообщения имеется только один побелочный ключ).

Стандарт шифрования данных DES был полон противоречий с самого момента его создания. Он основывался на шифре Люцифер (Lucifer), разработанном и запатентованном корпорацией IBM, с той разницей, что IBM использовала 128-разрядный ключ, а не 56-разрядный. Когда федеральное правительство Соединенных Штатов пожелало стандартизировать какой-нибудь шифр для несекретного применения, оно

«пригласило» IBM на «обсуждение» этого вопроса с Агентством национальной безопасности NSA (National Security Agency, известное как АНБ США), являющимся самым крупным в мире работодателем в области математики и криптологии. Агентство национальной безопасности США настолько секретно, что существует даже такая популярная шутка:

Вопрос: Что означает аббревиатура NSA?

Ответ: No Such Agency — Такого Агентства Нет.

После этих обсуждений корпорация IBM уменьшила длину ключа со 128 бит до 56 бит и решила держать в секрете процедуру разработки стандарта DES. Многие полагали, что длина ключа была уменьшена, чтобы гарантировать, что NSA сможет взломать DES, но организациям с более низким финансированием это будет не по силам. Вероятно, цель засекречивания проекта состояла в сокрытии потайного хода, позволяющего Агентству национальной безопасности еще легче взламывать шифр DES. Когда сотрудник этого управления предложил Институту инженеров по электротехнике и электронике (IEEE) отменить планирующуюся конференцию по криптографии, ощущения комфорта это не прибавило. Агентство национальной безопасности всегда препятствовало всему.

В 1977 году ученые Стэнфордского университета, занимающиеся исследованиями в области криптографии, Диффи (Diffie) и Хеллман (Hellman), разработали машину для взлома кода DES и оценили стоимость ее создания в $20 млн. По небольшому участку открытого текста и соответствующего ему зашифрованному тексту эта машина путем полного перебора 256 вариантов за один день могла найти 56-разрядный ключ. На сегодняшний день такая машина уже существует, продается и стоит менее $10 000 (Kumar et al., 2006).

Тройное шифрование с помощью DES (Triple DES)

Уже в 1979 году корпорация IBM поняла, что ключ стандарта DES слишком короток, и разработала метод, позволяющий существенно увеличить его надежность с помощью тройного шифрования (Tuchman, 1979). Выбранный метод, ставший с тех пор Международным стандартом 8732, показан на рис. 8.7. Здесь используются два ключа и три этапа. На первом этапе открытый текст зашифровывается (блок Encryption на рисунке) обычным DES ключом K_1. На втором этапе DES работает в режиме дешифрации (блок Decryption), используя ключ K_2. Наконец, выполняется еще одна операция шифрования с ключом K_1.

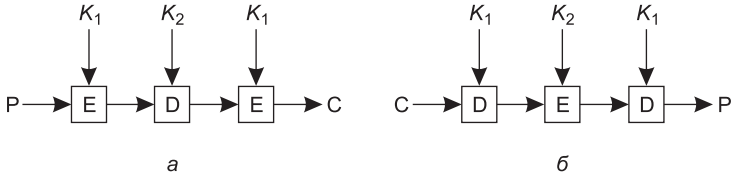


Рис. 8.7. Тройное шифрование с помощью DES (*а*); дешифрация (*б*)

Сразу возникает два вопроса. Во-первых, почему используются только два ключа, а не три? Во-вторых, почему используется последовательность операций **EDE (Encrypt**

Decrypt Encrypt — Шифрование Дешифрация Шифрование), а не **EEE (Encrypt Encrypt Encrypt — Шифрование Шифрование Шифрование**)? Причина использования всего двух ключей в том, что даже самые параноидальные криптографы в мире считают, что в настоящее время ключа длиной 112 бит вполне достаточно для коммерческих приложений (хотя в мире криптографии паранойя считается достоинством, а не болезнью). Переход на 168 разрядов повлечет лишь дополнительные расходы по хранению и транспортировке дополнительного ключа, а реальной пользы принесет мало.

Использование последовательности шифрования, дешифрации и снова шифрования объясняется обратной совместимостью с существующими DES-системами с одним ключом. Обе функции шифрования и дешифрации устанавливают соответствия между наборами 64-разрядных чисел. С точки зрения криптографии обе функции одинаково надежны. Однако использование EDE вместо EEE позволяет компьютеру, применяющему тройное шифрование, общаться с компьютером, применяющим обычное одиночное шифрование, просто установив $K_1 = K_2$. Таким образом, тройное шифрование можно легко включать в виде дополнительного режима работы, что не представляет интереса для ученых криптографов, но довольно важно для корпорации IBM и ее клиентов.

8.2.2. Улучшенный стандарт шифрования AES

В какой-то момент стало понятно, что ресурс DES (даже с тройным шифрованием) уже приближается к концу. Тогда **Национальный институт стандартов и технологий (NIST — National Institute of Standards and Technology)** — агентство Министерства торговли, занимающееся разработкой стандартов для Федерального правительства США, — решило, что правительству нужен новый криптографический стандарт для несекретных данных. NIST ясно осознавал все противоречия, связанные с DES, и прекрасно понимал, что как только будет объявлено о создании нового стандарта, все, кто хоть что-то смыслит в криптографической политике, по умолчанию будут предполагать, что и здесь имеется лазейка, с помощью которой Агентство национальной безопасности с легкостью сможет расшифровывать любую информацию. На таких условиях вряд ли кто-то согласится применять у себя новую технологию, и она, скорее всего, так и умрет в безвестности.

Исходя из этих предпосылок, NIST применил неожиданный для правительственного бюрократического аппарата подход: он решил просто спонсировать криптографический конкурс. В январе 1997 года ученые со всего мира были приглашены для представления своих разработок, касающихся нового стандарта, который назвали **AES (Advanced Encryption Standard — улучшенный стандарт шифрования)**. Требования, предъявляемые к разработкам, были таковы:

1. Алгоритм должен использовать симметричный блочный шифр.
2. Все детали разработки должны быть общедоступны.
3. Должны поддерживаться длины ключей 128, 192 и 256 бит.
4. Должна быть возможна как программная, так и аппаратная реализация.
5. Алгоритм должен быть общедоступным или базирующимся на не дискредитировавших себя понятиях.

Было рассмотрено 15 серьезных предложений. На общедоступных конференциях разработчики представляли свои проекты, а оппоненты должны были приложить максимум усилий для поиска возможных недостатков в каждом из проектов. В августе 1998 года институтом стандартов и технологий были выбраны пятеро финалистов. Выбор основывался в основном на таких аспектах, как обеспечиваемая безопасность, эффективность, простота, гибкость, а также требования к памяти (это важно для встроенных систем). Был проведен еще ряд конференций, на которых было высказано множество критических замечаний.

В октябре 2000 года NIST объявил о том, что он выбрал Rijndael Джона Домена и Винсента Раймена. Название Rijndael (произносится примерно как Райн-дол) представляет собой сокращение двух фамилий авторов: Раймен + Домен. В ноябре 2001 года Rijndael становится стандартом правительства США, опубликованным как Федеральный стандарт обработки информации, FIPS 197. Благодаря полной открытости конкурса, а также техническим возможностям Rijndael и тому факту, что выигравшая конкурс команда состояла из двух молодых бельгийских криптографов (которые вряд ли стали бы сотрудничать с NSA, предоставляя какие-то лазейки), Rijndael стал доминирующим мировым криптографическим стандартом. AES в настоящее время является частью набора инструкций для некоторых микропроцессоров (например, Intel).

Rhindael поддерживает длины ключей и размеры блоков от 128 до 256 бит с шагом в 32 бита. Длины ключей и блоков могут выбираться независимо друг от друга. Тем не менее стандарт AES говорит о том, что размер блока должен быть равен 128 битам, а длина ключа — 128, 192 или 256 бит. Однако вряд ли кто-то будет использовать 192-битные ключи, поэтому фактически AES применяется в двух вариантах: со 128-битными блоками и 128-битными ключами, а также со 128-битными блоками и 256-битными ключами.

Ниже приводится описание алгоритма, и там мы рассматриваем только один случай — 128/128, поскольку именно это, скорее всего, станет нормой для коммерческих приложений. 128-битный ключ означает, что размер пространства его значений равен $2128 \approx 3 \times 1038$. Даже если Агентству национальной безопасности удастся собрать машину из миллиона параллельных процессоров, каждый из которых будет способен вычислять один ключ в пикосекунду, на перебор всех значений потребуется около 1010 лет. К тому времени солнце уже давно потухнет, и нашим далеким потомкам придется читать распечатку со значением ключа при свете свечи.

Rijndael

С математической точки зрения, метод Rijndael основывается на теории полей Галуа, благодаря чему можно строго доказать некоторые его свойства, касающиеся секретности. Тем не менее можно рассматривать его и с точки зрения кода программы на языке C, не вдаваясь в математические подробности.

Как и в DES, в Rijndael применяются замены и перестановки. И там, и там используются несколько итераций, их число зависит от размера ключа и блока и равно 10 для 128-разрядного ключа и 128-битных блоков; для максимального размера ключа и блоков число итераций равно 14. Однако, в отличие от DES, все операции

выполняются над целыми байтами, что позволяет создавать эффективные реализации как в аппаратном, так и в программном исполнении. Схематичный алгоритм метода Rijndael приведен в листинге 8.1. Обратите внимание, что данный код используется в иллюстративных целях. Хорошие реализации кода, обеспечивающего безопасность, должны следовать определенным дополнительным практикам, таким как стирание восприимчивой памяти после использования. Для примера, см. Ferguson и др. (2010).

Листинг 8.1. Схематичный алгоритм метода Rijndael

```
#define LENGTH 16    /* Число байт в блоке данных или ключе */
#define NROWS 4      /* Число строк в массиве state */
#define NCOLS 4      /* Число столбцов в массиве state */
#define ROUNDS 10    /* Число итераций */
typedef unsigned char byte    /8-разрядное целое без знака */

rijndael(byte plaintext[LENGTH], byte ciphertext[LENGTH], byte key[LENGTH])
{
    int r;                          /* Счетчик цикла */
    byte state[NROWS][NCOLS];       /* Текущее состояние */
    struct{byte k[NROWS][NCOLS];} rk[ROUNDS + 1];   /* Ключи итерации */

    expand_key(key,rk);    /* Сформировать ключи итерации */
    copy_plaintext_to_text(state, plaintext);   /* Инициализация текущего состояния */
    xor_roundkey_into_state(state, rk[0]);      /* Сложить по модулю 2 ключ с текущим
                                                   состоянием */

    for(r=1; r<=ROUNDS; r++) {
        substitute(state);    /* Пропустить каждый байт через S-блок */
        rotate_rows(state);   /* Повернуть строку i на i байт */
        if(r < ROUNDS) mix_columns(state);        /* Смешивающая функция */
        xor_roundkey_into_state(state, rk[r]);    /* Сложить по модулю 2 ключ с текущим
                                                     состоянием */
    }
    copy_state_to_ciphertext(ciphertext, state);    /* Вернуть результат */
}
```

У функции `rijndael` три аргумента: `plaintext` — массив размером 16 байт, содержащий входные данные, `ciphertext` — массив размером 16 байт, в который будет возвращен шифр, а также **key** — массив размером 16 байт, содержащий ключ. В процессе вычислений текущее состояние данных сохраняется в байтовом массиве `state`, размер которого равен `NROWS * NCOLS`. Для 128-битных блоков данных размер этого массива равен 4 × 4 байта. В 16 байтах целиком уместится один блок.

Массив `state` изначально содержит открытый текст и модифицируется на каждом этапе вычислений. На некоторых этапах выполняется побайтовая подстановка. На других этапах — перестановка байтов внутри массива. Могут выполняться и другие преобразования. В конечном итоге содержимое `state` представляет собой зашифрованный текст, который и возвращается в качестве результата функции.

Алгоритм начинается с распространения ключа по 11 массивам одинакового размера с массивом, представляющим состояние (`state`). Эти массивы хранятся в *rk* — массиве структур, содержащих массивы состояний. Одна из этих структур будет ис-

пользована в начале вычислений, а остальные 10 — во время 10 итераций (по одному на итерацию). Вычисление ключа для каждой итерации производится довольно сложным образом, и мы не будем рассматривать детали этого процесса. Достаточно сказать, что для этого осуществляются циклические повороты и суммирования по модулю 2 различных групп разрядов ключа. Подробности можно узнать в (Daemen и Rijmen, 2002).

Следующий шаг состоит в копировании открытого текста в массив `state` для того, чтобы его можно было обрабатывать во время последующих итераций. Копируется текст в колонки по 4 байта: первые 4 байта попадают в колонку 0, вторые — в колонку 1 и т. д. И колонки, и строки нумеруются с нуля, а итерации — с единицы. Процесс создания 12 байтовых массивов размером 4 × 4 показан на рис. 8.8.

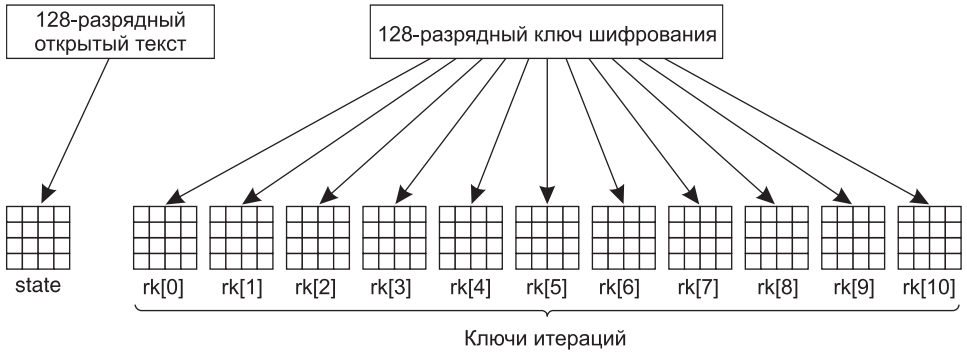


Рис. 8.8. Создание массивов state и rk

Перед началом основного цикла вычислений производится еще одно действие: `rk[0]` поразрядно складывается по модулю 2 с массивом `state`. Другими словами, каждый из 16 байт в массиве `state` заменяется суммой по модулю 2 от него самого и соответствующего байта в `rk[0]`.

Только после этого начинается главное развлечение. В цикле проводятся 10 итераций, в каждой из которых массив `state` подвергается преобразованию. Каждый раунд (итерация) состоит из четырех шагов. На шаге 1 в `state` производится посимвольная подстановка. Каждый байт по очереди используется в качестве индекса для S-блока, заменяющего его значение на соответствующую запись S-блока. На этом шаге получается прямой моноалфавитный подстановочный шифр. В отличие от DES, где используются несколько S-блоков, в Rijndael S-блок всего один.

На шаге 2 каждая из четырех строк поворачивается влево. Строка 0 поворачивается на 0 байт (то есть не изменяется), строка 1 — на 1 байт, строка 2 — на 2 байта, а строка 3 — на 3 байта. Смысл заключается в разбрасывании данных вокруг блока. Это аналогично перестановкам, показанным на рис. 8.5.

На шаге 3 происходит независимое перемешивание всех колонок. Делается это с помощью операции матричного умножения, в результате которого каждая новая колонка оказывается равной произведению старой колонки на постоянную матрицу. При этом умножение выполняется с использованием конечного поля Галуа, *GF*(28). Несмотря на то что все это кажется довольно сложным, алгоритм устроен так, что каждый эле-

мент матрицы вычисляется посредством всего лишь двух обращений к таблице и трех суммирований по модулю 2 (Daemen и Rijmen, 2002, Appendix E).

Наконец, на шаге 4 ключ данной итерации складывается по модулю 2 с массивом `state` для использования в следующем раунде.

Благодаря обратимости всех действий расшифровка может быть выполнена с помощью такого же алгоритма, но с обратным порядком следования всех шагов. Однако, есть одна хитрость, которая позволяет заниматься расшифровкой, используя алгоритм шифрования с измененными таблицами.

Данный алгоритм обладает не только очень высокой защищенностью, но и очень высокой скоростью. Хорошая программная реализация на машине с частотой 2 ГГц может шифровать данные со скоростью 700 Мбит/с. Такой скорости достаточно для шифрования видео в формате MPEG-2 в реальном масштабе времени. Аппаратные реализации работают еще быстрее.

8.2.3. Режимы шифрования

Несмотря на всю свою сложность, AES (или DES, или любой другой блочный код) представляет собой, по сути дела, моноалфавитный подстановочный шифр с большими длинами символов (в AES используются 128-битные символы, в DES — 64-битные). Для любого отрывка открытого текста шифр при прогоне через один и тот же шифрующий блок будет получаться всегда одинаковым. Скажем, если вы будете 100 раз пытаться зашифровать текст *abcdefgh*, задавая всякий раз один и тот же ключ для алгоритма DES, вы получите 100 одинаковых копий шифра. Взломщик может попытаться использовать этот недостаток при попытке расшифровки текста.

Режим электронного шифроблокнота

Чтобы понять, каким образом это свойство моноалфавитного подстановочного шифра может быть использовано для частичного взлома шифра, мы рассмотрим (тройное) кодирование по стандарту DES только лишь потому, что изображать 64-разрядные блоки проще, чем 128-разрядные. Надо иметь при этом в виду, что AES сталкивается с теми же самыми проблемами. Самый очевидный способ кодирования длинного сообщения заключается в разбиении его на отдельные блоки по 8 байт (64 бита) с последующим кодированием этих блоков по очереди одним и тем же ключом. Последний блок при необходимости можно дополнить до 64 бит. Такой метод называется **режимом электронного шифроблокнота (Electronic Code Book mode — ECB mode)**, по аналогии со старомодными шифроблокнотами, содержавшими слова и соответствующие им шифры (обычно это были пятизначные десятичные числа).

На рис. 8.9 показано начало компьютерного файла, в котором перечислены поощрительные премии сотрудникам компании. Этот файл состоит из последовательных 32-разрядных записей, по одной на сотрудника, следующего формата: 16 байт на имя, 8 байт на должность и 8 байт на премию. Каждый из шестнадцати 8-байтовых блоков (пронумерованных от 0 до 15) кодируется шифром DES.

Лесли только что поругалась с боссом и не рассчитывает на большую премию. Ким, напротив, — любимица босса, что всем известно. Лесли может получить доступ

к файлу, после того как он будет зашифрован, но до того, как он будет отослан в банк. Может ли Лесли исправить ситуацию, имея доступ только к зашифрованному файлу?

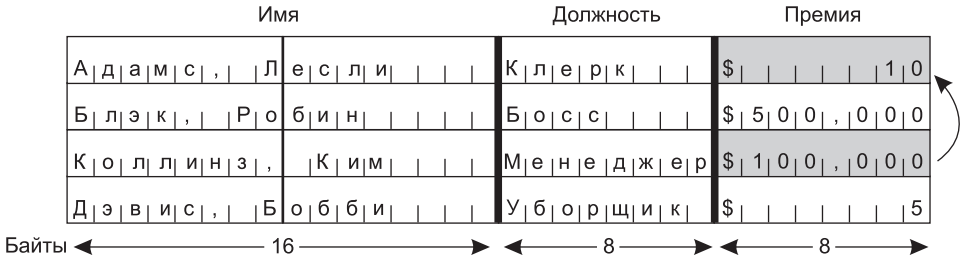


Рис. 8.9. Открытый текст файла, зашифрованного в виде 16 DES-блоков

В данном случае это очень просто. Все, что нужно сделать Лесли, — это скопировать зашифрованный блок 12 (содержащий премию Ким) и заменить им блок 4 (содержащий премию Лесли). Даже не зная содержимого блока 12, Лесли может рассчитывать на значительно более веселое Рождество. (Скопировать зашифрованный блок 8 тоже можно, но вероятность, что это будет обнаружено, выше; кроме того, Лесли, в общем-то, не жадная.)

Режим сцепления блоков шифра

Чтобы противостоять атакам подобного типа, все блочные шифры можно модернизировать таким образом, чтобы замена одного блока вызывала повреждение других блоков открытого текста после их расшифровки, превращая эти блоки (начиная с модифицированного места) в мусор. Один из таких способов — **сцепление блоков шифра (cipher block chaining)**. При этом методе, показанном на рис. 8.10, каждый блок открытого текста перед зашифровкой складывается по модулю 2 с предыдущим уже зашифрованным блоком. При этом одинаковым блокам открытого текста уже не соответствуют одинаковые блоки зашифрованного текста. Таким образом, шифр перестает быть большим моноалфавитным подстановочным шифром. Первый блок складывается по модулю 2 со случайным **вектором инициализации, IV (Initialization Vector)**, передаваемым вместе с зашифрованным текстом в виде открытого текста.

Рассмотрим работу сцепления блоков шифра на примере рис. 8.10. Начнем с вычисления $C_0 = E(P_0 \text{ XOR } IV)$. Затем мы вычислим $C_1 = E(P_1 \text{ XOR } C_0)$ и т. д. Расшифровка производится по формуле $P_0 = IV \text{ XOR } D(C_0)$ и т. д. Обратите внимание на то, что блок i является функцией всех блоков открытого текста с 0 по $i - 1$, поэтому один и тот же исходный блок текста преобразуется в разные зашифрованные блоки в зависимости от их расположения. При использовании такого способа шифрования преобразование, произведенное Лесли, приведет к появлению двух блоков чепухи, начиная с поля премии Лесли. Для сообразительного сотрудника службы безопасности эта странность может послужить подсказкой в последующем расследовании.

Сцепление блоков шифра также обладает тем достоинством, что усложняет криптоанализ, так как одни и те же блоки открытого текста преобразуются в разные зашифрованные блоки. Именно по этой причине и применяется описанный метод.

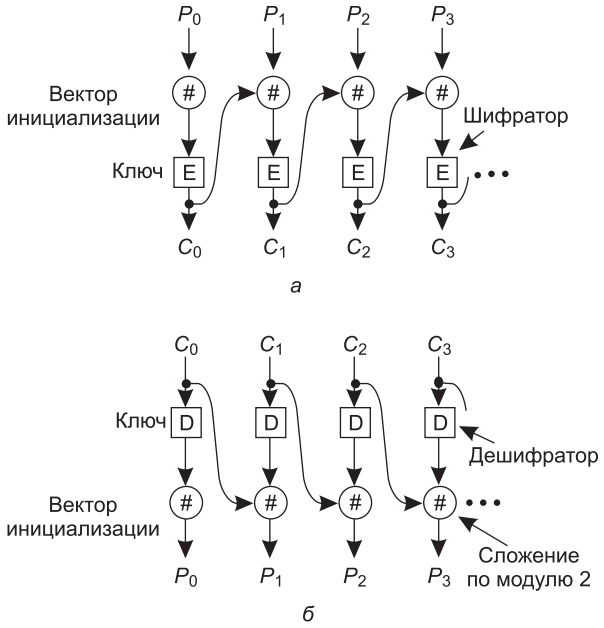


Рис. 8.10. Сцепление зашифрованных блоков: *а* — шифрование; *б* — дешифрация

Режим шифрованной обратной связи

Однако у метода сцепления блоков шифра есть и недостаток, заключающийся в том, что прежде чем может начаться шифрование или дешифрация, должен появиться целый 64-битовый блок данных. Для пользователей интерактивных терминалов, набирающих строки короче восьми символов и ждущих ответа, такой метод не подходит. Для побайтового шифрования может применяться **режим шифрованной обратной связи (cipher feedback mode)** с использованием (тройного) DES, как показано на рис. 8.11. Для стандарта AES идея остается той же самой, только используется 128-разрядный сдвиговый регистр. На рисунке мы видим состояние шифрующей машины после того, как байты с 0 по 9 уже зашифрованы и посланы. Когда прибывает десятый байт открытого текста, как показано на рис. 8.11, *а*, алгоритм DES обрабатывает 64-разрядный сдвиговый регистр, чтобы произвести 64-разрядный зашифрованный блок. Самый левый байт этого зашифрованного текста извлекается и складывается по модулю 2 с P_{10}. Этот байт передается по линии. Затем сдвиговый регистр сдвигается влево на 8 разрядов. При этом байт C_2 извлекается с левого конца регистра, а байт C_{10} вставляется в него в освободившееся место справа от C_9. Обратите внимание, что содержимое сдвигового регистра зависит от всей предыстории открытого текста, так что повторяющиеся фрагменты исходного текста будут кодироваться каждый раз по-разному. Как и для метода сцепления блоков шифра, для начала шифрования этим методом требуется вектор инициализации.

При использовании режима шифрованной обратной связи дешифрация аналогична шифрованию. В частности, содержимое сдвигового регистра *шифруется*, а не *дешиф-*

руется, поэтому байт, который складывается по модулю 2 с C_{10} для получения P_{10}, равен тому байту, который складывается по модулю 2 с P_{10} для получения C_{10}. Пока содержимое двух сдвиговых регистров идентично, дешифрация выполняется корректно. Это показано на рис. 8.11, *б*.

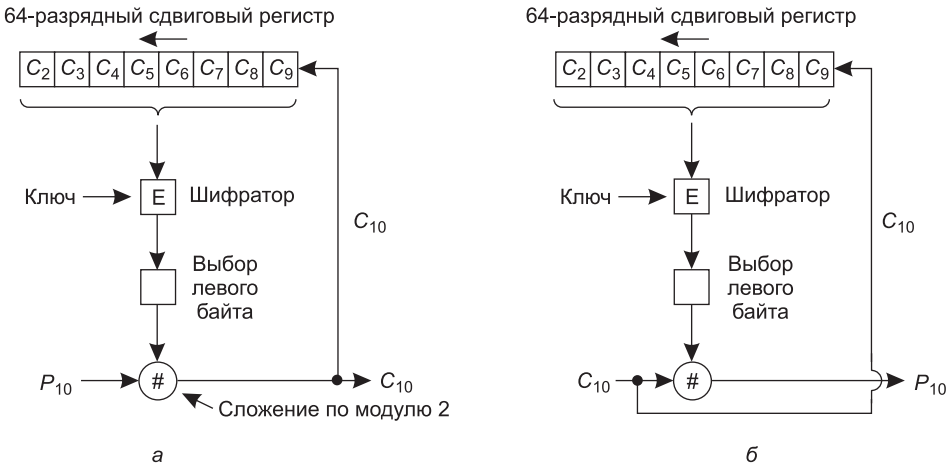


Рис. 8.11. Режим шифрованной обратной связи: а — шифрование; б — дешифрация.

Проблемы с режимом шифрованной обратной связи начинаются, когда при передаче шифрованного таким способом текста один бит случайно инвертируется. При этом испорченными окажутся 8 байтов, расшифровываемые в то время, когда поврежденный байт находится в сдвиговом регистре. Когда поврежденный байт покинет сдвиговый регистр, на выходе опять станет расшифровываться правильный открытый текст. Таким образом, результат инверсии одного бита оказывается относительно локализованным и не портит всего остатка сообщения.

Режим группового шифра

Тем не менее существуют приложения, в которых один испорченный при передаче бит приводит к порче 64 бит открытого текста, а это многовато. Для таких приложений существует четвертый вариант, называемый **режимом группового** (потокового) **шифра (stream cipher mode)**. Суть его заключается в том, что выходной блок получается шифрацией вектора инициализации с использованием ключа. Затем этот выходной блок снова шифруется с использованием ключа, в результате чего получается второй выходной блок. Для получения третьего блока шифруется второй блок и т. д. Последовательность (произвольной длины) выходных блоков, называемая **ключевым потоком (keystream)**, воспринимается как одноразовый блокнот и складывается по модулю 2 с открытым текстом. В результате получается шифрованный текст, как показано на рис. 8.12, *а*. Обратите внимание: вектор инициализации используется только на первом шаге. После этого шифруются выходные блоки. Кроме того, ключевой поток не зависит от данных, поэтому он в случае необходимости может быть вычислен заранее и совершенно не чувствителен к ошибкам передачи. Процесс дешифрации показан на рис. 8.12, *б*.

Дешифрация осуществляется путем генерации точно такого же ключевого потока на принимающей стороне. Поскольку он зависит только от вектора инициализации и ключа, ошибки передачи шифрованного текста на него не влияют. Таким образом, ошибка в одном бите передаваемого шифра приводит к ошибке только в одном бите расшифрованного текста.

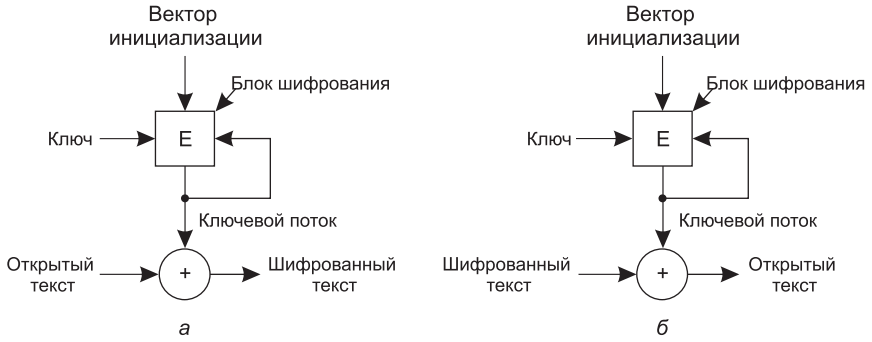


Рис. 8.12. Групповой шифр: *а* — шифрование; *б* — дешифрация

Важно никогда не использовать одну и ту же пару (ключ, вектор инициализации) в одном и том же групповом шифре, поскольку при этом всякий раз будет получаться одинаковый ключевой поток. Повторное использование ключевого потока может привести к неприятному эффекту **взлома шифра при помощи многократного использования ключевого потока (keystream reuse attack)**. Допустим, блок открытого текста P_0 шифруется с помощью ключевого потока, в результате чего получается сумма по модулю 2 P_0 и K_0. Затем берется второй блок открытого текста, Q_0, и шифруется тем же ключевым потоком (получаем Q_0 XOR K_0). Криптоаналитик, перехвативший оба блока шифрованного текста, может просто сложить их вместе по модулю 2 и получить в результате P_0 XOR Q_0, убирая тем самым ключ. Теперь у него есть сумма по модулю 2 двух блоков открытого текста. Если один из них известен (или его можно угадать), найти второй — не проблема. В любом случае, взломать сумму по модулю 2 двух блоков открытого текста можно, используя статистические свойства сообщения. Скажем, если передается английский текст, то наиболее часто встречающейся буквой в потоке будет «е» и т. д. Короче говоря, имея сумму по модулю 2 двух частей открытого текста, взломщик с высокой вероятностью сможет вычислить обе части.

Режим счетчика

Все режимы, кроме электронного шифроблокнота, обладают одним и тем же неприятным свойством: доступ к произвольной части зашифрованных данных невозможен. Допустим, например, что файл передается по сети и затем сохраняется на диске в зашифрованном виде. Так иногда делают, если принимающий компьютер представляет собой ноутбук, который может быть украден. Хранить все важные данные в зашифрованном виде действительно полезно: риск утечки секретной информации в случае попадания аппаратуры в руки «нехорошим дядям» резко снижается.

Однако доступ к файлам на диске зачастую бывает необходимо осуществлять в произвольном порядке. Особенно это касается файлов баз данных. Если файл зашифрован в режиме сцепления блоков, придется вначале дешифровать все блоки, предшествующие нужному. Согласитесь, такой способ работы несколько неудобен. По этой причине был введен еще один режим шифрования — **режим счетчика (counter mode)**. Он показан на рис. 8.13. Здесь открытый текст не шифруется напрямую. Вместо этого шифруется вектор инициализации плюс некоторая константа, а уже получающийся в результате шифр складывается по модулю 2 с открытым текстом. Сдвигаясь на единицу по вектору инициализации при шифровании каждого нового блока, можно легко получить способ дешифрации любого места файла. При этом нет необходимости расшифровывать все предшествующие блоки.

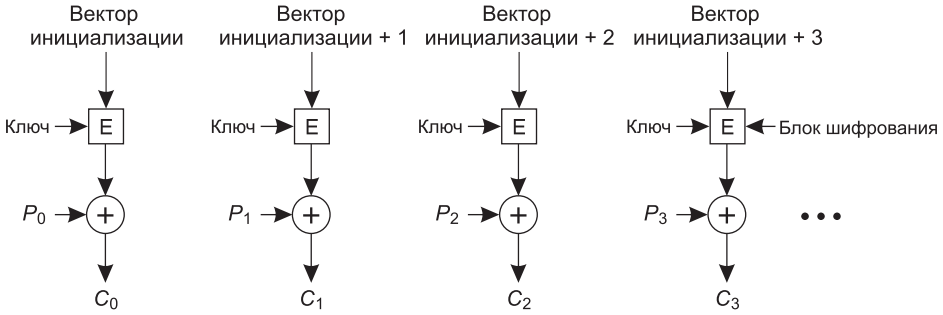


Рис. 8.13. Шифрование в режиме счетчика

Несмотря на то что режим счетчика весьма полезен, у него есть один существенный недостаток, который стоит упомянуть. Допустим, уже использовавшийся однажды ключ *K* будет использован повторно (с другим открытым текстом, но с тем же вектором инициализации), и взломщик захватит весь шифрованный текст, который был послан в обоих случаях. Ключевые потоки остаются неизменными, в итоге возникает риск взлома за счет повторного использования ключевого потока (тот же эффект, на который мы уже указывали, обсуждая режим группового шифра). Все, что криптоаналитику остается сделать, это сложить по модулю 2 два перехваченных сообщения. Тем самым он полностью снимет какую бы то ни было криптографическую защиту, и в его распоряжении окажется сумма по модулю 2 двух блоков открытого текста. Этот недостаток вовсе не означает, что режим счетчика в целом неудачен. Это говорит лишь о том, что как ключи, так и векторы инициализации должны выбираться независимо и случайным образом. Даже если один и тот же ключ случайно попадется дважды, от перехвата информацию может спасти отличающийся вектор инициализации.

8.2.4. Другие шифры

AES (Rijndael) и DES — это самые известные криптографические алгоритмы с симметричными ключами, и они являются стандартным выбором производителей хотя бы по соображениям ответственности. (Никто не станет вас винить, если вы использовали AES в своем продукте, и его взломали, но вас точно обвинят, если вы использовали нестандартный шифр, который затем был взломан). Тем не менее стоит отметить, что

в природе существует еще много других шифров с симметричными ключами. Некоторые из них встраиваются в различные программно-аппаратные продукты. Наиболее распространенные из них перечислены в табл. 8.2. Возможно использование комбинаций данных шифров, например, AES и Twofish, так чтобы для получения данных было необходимо взломать оба шифра.

Таблица 8.2. Некоторые распространенные криптографические алгоритмы с симметричными ключами

Название	Автор	Длина ключа	Комментарии
DES	IBM	56 бит	Слишком слабый для современных систем
RC4	Рональд Ривест (Ronald Rivest)	1–2048 бит	Внимание: есть слабые ключи
RC5	Рональд Ривест (Ronald Rivest)	128–256 бит	Хороший, но запатентованный
AES (Rijndael)	Домен (Daemen) и Раймен (Rijmen)	128–256 бит	Лучший
Serpent	Андерсон (Anderson), Байхэм (Biham) и Кнадсен (Knudsen)	128–256 бит	Очень сильный
Тройной DES	IBM	168 бит	Хороший, но устаревает
Twofish	Брюс Шнайер (Bruce Schneier)	128–256 бит	Очень сильный; широко распространен

8.2.5. Криптоанализ

Прежде чем закончить разговор об использовании симметричных ключей в криптографии, необходимо хотя бы упомянуть о четырех направлениях развития криптоанализа. Первый подход называется **дифференциальным криптоанализом** (Biha и Shamir, 1997). Он может использоваться для взлома любых блочных шифров. Для начала анализируется пара блоков открытого текста, отличающихся лишь небольшим числом бит. При этом внимательно анализируется происходящее при каждой внутренней итерации во время шифровании. Во многих случаях можно заметить, что одни битовые последовательности встречаются чаще других, и это соображение используется для взлома, основанного на теории вероятностей.

Второй подход, который следует обозначить, называется **линейным криптоанализом** (Matsui, 1994). С его помощью можно взломать DES только с 243 известными открытыми текстовыми блоками. Принцип работы основан на суммировании по модулю 2 некоторых бит открытого текста и изучении результатов для шаблонных последовательностей. Если повторять эту процедуру много раз, половина бит будет иметь нулевые значения, половина — единичные. Тем не менее довольно часто это соотношение изменяется в ту или иную сторону. Это отклонение, каким бы малым оно ни было, может использоваться для снижения показателя трудозатрат криптоаналитика. Более подробную информацию можно найти в материалах автора этого метода Мацуи (Matsui).

Третье направление развития связано с анализом потребляемой электроэнергии для вычисления секретного ключа. Обычно в компьютерах напряжение 3 В соот-

ветствует логической единице, а 0 В — логическому нулю. Таким образом, обработка единицы требует большего потребления электроэнергии, чем обработка нуля. Если криптографический алгоритм состоит из цикла, в котором разряды ключа обрабатываются поочередно, взломщик, заменив главный системный *n*-гигагерцовый тактовый генератор медленным (например, с частотой 100 Гц) и повесив «крокодилы» на ножки питания и заземления центрального процессора, может с большой точностью отслеживать мощность, потребляемую каждой машинной инструкцией. По этим данным узнать секретный ключ оказывается на удивление просто. Этот метод криптоанализа можно победить лишь аккуратным кодированием алгоритма на язык ассемблера таким образом, чтобы энергопотребление не зависело ни от общего ключа, ни от ключей каждой итерации.

Четвертый подход основан на временном анализе. Криптографические алгоритмы содержат большое количество условных операторов (`if`), тестирующих биты итерационных ключей. Если части `then` и `else` выполняются за различное время, то, замедлив системный тактовый генератор и измерив длительность всех шагов, можно вычислить ключи итераций. По этим ключам обычно можно вычислить и общий ключ. Анализ энергопотребления и временной анализ могут применяться и одновременно, что позволяет упростить задачу криптоанализа. Несмотря на то что анализы энергозатрат и времени выполнения операций могут показаться несколько экзотическими, на самом деле они представляют собой мощные методы, способные взломать любой шифр, если только он не имеет специальной защиты.

8.3. Алгоритмы с открытым ключом

Исторически процесс передачи ключа всегда был слабым звеном почти во всех системах шифрования. Независимо от того, насколько прочна была сама криптосистема, если нарушитель мог украсть ключ, система становилась бесполезной. До 1976 года все криптологи исходили из предпосылки, что ключ дешифрации должен быть идентичен ключу шифрования (или один может легко получиться из другого). В то же время ключи должны были быть у всех пользователей системы. Таким образом, казалось, что эта проблема неустранима: ключи должны быть защищены от кражи и в то же время их нужно распространять среди пользователей, поэтому их нельзя хранить в банковском сейфе.

В 1976 году два исследователя из Стэнфордского университета, Диффи (Diffie) и Хеллман (Hellman), предложили радикально новую криптосистему, в которой ключ шифрования и ключ дешифрации были настолько различными, что ключ дешифрации нельзя было получить из ключа шифрования. Предложенные ими алгоритм шифрования E и алгоритм дешифрации D (оба параметризованные ключом) должны были удовлетворять следующим трем требованиям.

1. $D(E(P)) = P$.
2. Крайне сложно вывести D из E.
3. E нельзя взломать при помощи произвольного открытого текста.

Первое требование состоит в том, что если применить алгоритм дешифрации D к зашифрованному сообщению $E(P)$, то мы опять получим открытый текст P. Без этого

авторизованный получатель просто не сможет расшифровать сообщение. Второе требование говорит само за себя. Третье требование необходимо, потому что, как мы скоро увидим, злоумышленники могут экспериментировать с алгоритмом столько, сколько пожелают. При таких условиях нет никаких причин, по которым ключ шифрования нельзя было бы сделать общедоступным.

Этот метод работает следующим образом. Некто, например Алиса, желая получать секретные сообщения, сначала формирует два алгоритма, удовлетворяющие перечисленным выше требованиям. Затем алгоритм шифрования и его ключ открыто объявляются, отсюда название **шифрования с открытым ключом (public-key cryptography)**. Это можно сделать, разместив открытый ключ, например, на домашней страничке Алисы. Для обозначения алгоритма шифрования, параметризованного открытым ключом Алисы, мы будем использовать запись *EA*. По аналогии (секретный) алгоритм дешифрации, параметризованный персональным ключом Алисы, мы будем обозначать *DA*. Боб делает то же самое, открыто объявляя *EB*, но храня в тайне *DB*.

Теперь посмотрим, сможем ли мы решить проблему установки надежного канала между Алисой и Бобом, которые ранее никогда не встречались. Оба ключа шифрования Алисы и Боба, *EA* и *EB*, являются открытыми. (Вообще, все пользователи сети могут, становясь пользователями, опубликовать свои ключи шифрования.) Теперь Алиса берет свое первое сообщение *P*, вычисляет *EB*(*P*) и посылает его Бобу. Боб расшифровывает его с помощью своего секретного ключа *DB*, то есть вычисляет *DB*(*EB*(*P*)) = *P*. Более никто не может прочитать это зашифрованное сообщение *EB*(*P*), так как предполагается, что система шифрования достаточно надежна, а получить ключ *DB* на основании известного ключа *EB* очень трудно. Посылая ответ, Боб передает *EA*(*R*). Таким образом, Алиса и Боб получают надежный секретный канал связи.

Обратите внимание на используемую здесь терминологию. Шифрование с открытым ключом предполагает у каждого пользователя наличие двух ключей — открытого ключа, используемого всеми для шифрования сообщений, посылаемых этому пользователю, и закрытого ключа, требующегося пользователю для дешифрации приходящих к нему сообщений. Мы будем и далее называть эти ключи *открытым* (*public*) и *закрытым* (*private*), чтобы отличать их от *секретных* (*secret*) ключей, используемых для шифрования и дешифрации в обычной криптографии с симметричным ключом.

8.3.1. Алгоритм RSA

Единственная загвоздка состоит в том, чтобы найти алгоритмы, удовлетворяющие всем трем требованиям. Поскольку преимущества шифрования с открытым ключом очевидны, многие исследователи неустанно работали над созданием подобных алгоритмов, некоторые уже опубликованы. Один хороший метод был разработан группой исследователей Массачусетского технологического института (Rivest и др., 1978). Он назван по начальным буквам фамилий трех разработчиков: **RSA** (Rivest, Shamir, Adleman). Этот метод вот уже четверть века выдерживает попытки взлома и считается очень прочным. На его основе построены многие практические системы безопасности. По этой причине Rivest, Shamir и Adleman получили в 2002 году награду ACM — Премию Тьюринга (Turing Award). Главный недостаток RSA заключается в том, что для обеспечения достаточного уровня защищенности требуется ключ длиной, по крайней

мере, 1024 бита (против 128 бит в алгоритмах с симметричными ключами). Из-за этого алгоритм работает довольно медленно.

В основе метода RSA лежат некоторые принципы теории чисел. Опишем в общих чертах, как пользоваться этим методом. Подробности можно найти в соответствующих источниках.

1. Выберем два больших простых числа p и q (обычно длиной 1024 бита).
2. Сосчитаем $n = p \times q$ и $z = (p-1) \times (q-1)$.
3. Выберем число d, являющееся взаимно простым с числом z.
4. Найдем такое число e, что остаток от деления произведения $e \times d$ на число z равен 1.

Вычислив заранее эти параметры, можно начинать шифрование. Сначала разобьем весь открытый текст (рассматриваемый в качестве битовой строки) на блоки так, чтобы каждое сообщение, P, попадало в интервал $0 \leqslant P < n$. Это несложно сделать, если разбить открытый текст на блоки по k бит, где k — максимальное целое число, для которого $2k < n$.

Чтобы зашифровать сообщение P, вычислим $C = Pe \pmod{n}$. Чтобы расшифровать C, сосчитаем $P = Cd \pmod{n}$. Можно доказать, что для всех значений P в указанном диапазоне функции шифрования и дешифрации являются взаимно обратными. Чтобы выполнить шифрование, нужны e и n. Для дешифрации требуются d и n. Таким образом, открытый ключ состоит из пары (e, n), а закрытый ключ — из пары (d, n).

Надежность метода обеспечивается сложностью нахождения множителей больших чисел. Если бы криптоаналитик мог разложить на множители (открытое) число n, он мог бы тогда найти значения p и q, а следовательно, и число z. После этого числа e и d можно найти при помощи алгоритма Евклида. К счастью, математики пытались решить проблему разложения на множители больших чисел, по меньшей мере, 300 лет, и накопленный опыт позволяет предположить, что эта проблема чрезвычайно трудна.

Ривест (Rivest) с коллегами утверждает, что для разложения на множители числа из 500 цифр необходимо 1025 лет, если применять грубую силу. Предполагается, что задействован лучший известный алгоритм и компьютер, выполняющий одну инструкцию за 1 мкс. Если миллион чипов будут работать одновременно и выполнять одну инструкцию за одну наносекунду, все равно потребуется 1016 лет. Даже при сохранении экспоненциального роста скоростей компьютеров потребуется множество лет, чтобы найти множители числа из 500 цифр, а к этому времени наши потомки могут просто выбрать еще большие p и q.

Тривиальный учебный пример работы алгоритма RSA приведен на рис. 8.14. Для этого примера мы выбрали $p = 3$, а $q = 11$, что дает значения $n = 33$, а $z = 20$. Число d можно выбрать равным 7, так как числа 20 и 7 не имеют общих делителей. При таком выборе значение e можно найти, решив уравнение $7e = 1 \pmod{20}$, откуда следует, что $e = 3$. Зашифрованный текст C получается из открытого сообщения P по формуле $C = P^3 \pmod{33}$. Получатель расшифровывает сообщение по формуле $P = C\,7 \pmod{33}$. В качестве примера на рисунке показано шифрование слова «SUZANNE».

Поскольку выбранные для данного примера простые числа так малы, P должно быть менее 33, поэтому каждый блок открытого текста может содержать лишь одну букву. В результате получается моноалфавитный подстановочный шифр, что не очень впечатляет. Если бы мы вместо этого выбрали числа p и q порядка 2^{512}, тогда число n

было бы около 21024. В этом случае каждый блок мог бы содержать до 1024 бит или 128 восьмиразрядных символов против 8 символов шифра DES или 16 AES.

Открытый текст (P)		Зашифрованный текст (C)			После дешифрации	
Символ	Число	P^3	$P^3 \pmod{33}$	C^7	$C^7 \pmod{33}$	Символ
S	19	6859	28	13492928512	19	S
U	21	9261	21	1801088541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	1	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	5	E
Вычисление отправителя				Вычисление получателя		

Рис. 8.14. Пример работы алгоритма RSA

Следует отметить, что использование алгоритма RSA в описанном выше виде аналогично использованию симметричного алгоритма в режиме **ECB (Electronic Code Book — электронный шифроблокнот)**, в котором одинаковые блоки на входе преобразуются в одинаковые блоки на выходе. Таким образом, для шифрования данных требуется сцепление блоков в каком-либо виде. Однако на практике алгоритм RSA с открытым ключом используется только для передачи одноразового секретного ключа, после чего применяется какой-нибудь алгоритм с симметричным ключом типа AES или тройного DES. Система RSA слишком медленная, чтобы шифровать большие объемы данных, однако она широко применяется для распространения ключей.

8.3.2. Другие алгоритмы с открытым ключом

Хотя алгоритм RSA получил широкое распространение, он ни в коей мере не является единственным известным алгоритмом с открытым ключом. Первым алгоритмом с открытым ключом стал «алгоритм ранца» (Merkle и Hellman, 1978). Его идея состоит в том, что имеется большое количество объектов различного веса. Владелец этих объектов кодирует сообщение, выбирая подмножество объектов и помещая их в ранец. Общий вес объектов в рюкзаке известен всем, как и список всех возможных объектов и их соответствующий вес. Список объектов, находящихся в рюкзаке, хранится в секрете. При определенных дополнительных ограничениях задача определения возможного списка объектов по известному общему весу считалась неразрешимой по вычислениям, то есть считалось, что решение можно найти только полным перебором различных сочетаний предметов списка. Поэтому она была положена в основу алгоритма с открытым ключом.

Изобретатель алгоритма Ральф Меркле (Ralph Merkle) был настолько уверен в надежности своего алгоритма, что предложил $100 любому, кто сумеет его взломать. Ади Шамир (Adi Shamir), «S» в группе RSA, мгновенно взломал его и получил награду. Это не смутило Меркле. Он усилил алгоритм и предложил за его взлом уже $1000. Рон Ривест (Ron Rivest), «R» в RSA, тут же взломал улучшенную версию алгоритма

и получил награду. Меркле не рискнул предложить $10 000 за следующую версию, поэтому «А», Леонарду Эйдлману (Leonard Adleman), не повезло. Несмотря на то что алгоритм ранца был в очередной раз исправлен, он не считается надежным и редко используется.

Другие схемы с открытым ключом основаны на сложности вычисления дискретных логарифмов. Алгоритмы, использующие этот принцип, были разработаны Эль-Гамалем (El Gamal, 1985) и Шнорром (Schnorr, 1991).

Существуют и некоторые другие методы, например, основанные на эллиптических кривых (Menezes и Vanstone, 1993). Однако основные две категории составляют алгоритмы, основанные на сложности нахождения делителей больших чисел и вычислений дискретных логарифмов. Эти задачи считаются особенно сложными, так как математики уже много лет пытались их решить без особых успехов.

8.4. Цифровые подписи

Подлинность различных бумажных документов (юридических, финансовых и др.) определяется наличием или отсутствием авторизованной рукописной подписи. Фотокопии документами не считаются. Чтобы компьютерные системы сообщений могли заменить физическое перемещение документов, состоящих из бумаги и чернил, нужно решить проблему подписи.

Проблема разработки замены рукописной подписи довольно сложна. По существу, требуется система, с помощью которой одна сторона могла бы послать другой стороне «подписанное» сообщение, так чтобы:

1) получатель мог проверить объявленную личность отправителя;
2) отправитель не мог позднее отрицать содержимое сообщения;
3) получатель не мог позднее изменить подписанное сообщение.

Первое требование существенно, например, для финансовых систем. Когда компьютер клиента заказывает компьютеру банка купить тонну золота, банковский компьютер должен быть уверен, что компьютер, пославший заказ, действительно принадлежит компании, со счета которой следует снять денежную сумму. Другими словами, банк должен иметь возможность установить подлинность клиента (а клиент — подлинность банка).

Второе требование необходимо для защиты банка от мошенничества. Предположим, что банк покупает тонну золота, и сразу после этого цена на золото резко падает. Бесчестный клиент может затем подать в суд на банк, заявляя, что он никогда не посылал заказа на покупку золота. Банк может показать сообщение в суде, но клиент будет отрицать, что посылал его. Таким образом, должно быть обеспечено требование **невозможности отречения от данных ранее обязательств (nonrepudiation)**. Методы составления цифровых подписей, которые мы изучим ниже, предназначены в том числе и для этого.

Третье требование нужно для защиты клиента в случае, если цена на золото после его покупки банком взлетает вверх, и банк пытается создать подписанное сообщение, в котором клиент просит купить не одну тонну золота, а один слиток. При таком сценарии банк, совершив мошенничество, забирает оставшуюся часть золота себе.

8.4.1. Подписи с симметричным ключом

Один из методов реализации цифровых подписей состоит в создании некоего центрального авторитетного органа, которому все доверяют, — назовем его, например, Большим Братом (Big Brother, BB). Затем каждый пользователь выбирает секретный ключ и лично относит его в офис Большого Брата. Таким образом, например, секретный ключ Алисы, K_A, известен только Алисе и Большому Брату.

Когда Алиса хочет послать открытым текстом своему банкиру Бобу подписанное сообщение P, она формирует сообщение, зашифрованное KA (ключом Алисы), $KA(B, RA, t, P)$, где B — идентификатор Боба, RA — случайное число, выбранное Алисой, t — временной штамп, подтверждающий свежесть сообщения. Затем она посылает его Большому Брату, как показано на рис. 8.15. Большой Брат видит, что это сообщение от Алисы, расшифровывает его и посылает Бобу. Сообщение, посылаемое Бобу, содержит открытый текст сообщения Алисы и подпись Большого Брата $KBB(A, t, P)$. Получив подписанное сообщение, Боб может выполнять заказ Алисы.

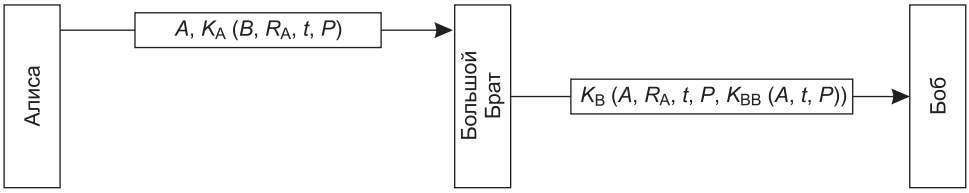


Рис 8.15. Цифровая подпись Большого Брата

Что случится, если позднее Алиса станет отрицать отправку этого сообщения? Естественно, в случае возникновения подобных конфликтов конфликтующие стороны первым делом подают друг на друга в суд (по крайней мере, в Соединенных Штатах). Наконец, когда дело попадает в суд, Алиса энергично утверждает, что она не отправляла Бобу обсуждаемое. Судья спрашивает Боба, почему он уверен, что данное сообщение пришло именно от Алисы, а не от злоумышленницы Труди. Боб сначала заявляет, что Большой Брат не принял бы сообщения от Алисы, если бы оно не было зашифровано ее ключом K_A, — у злоумышленника просто нет возможности послать Большому Брату фальшивое сообщение от Алисы.

Затем Боб элегантным жестом демонстрирует суду сообщение A: $K_{BB}(A, t, P)$. Боб заявляет, что это сообщение подписано Большим Братом, это доказывает, что Алиса послала сообщение P Бобу. Затем судья просит Большого Брата (которому все доверяют) проверить подпись под этим сообщением. Когда Большой Брат подтверждает, что Боб говорит правду, судья решает дело в пользу Боба. Дело закрыто.

Теоретически проблема с этим протоколом цифровых подписей, который показан на рис. 8.15, может возникнуть, если злоумышленник повторно воспроизведет оба сообщения. Для минимизации вероятности этого используются временные штампы. Кроме того, Боб может просмотреть все недавние сообщения, проверяя, не встречалось ли в них такое же R_A. В этом случае сообщение считается дубликатом и просто игнорируется. Очень старые сообщения, обнаруживаемые по значению временного штампа, также игнорируются. Для защиты от мгновенной атаки повторным воспроизведением

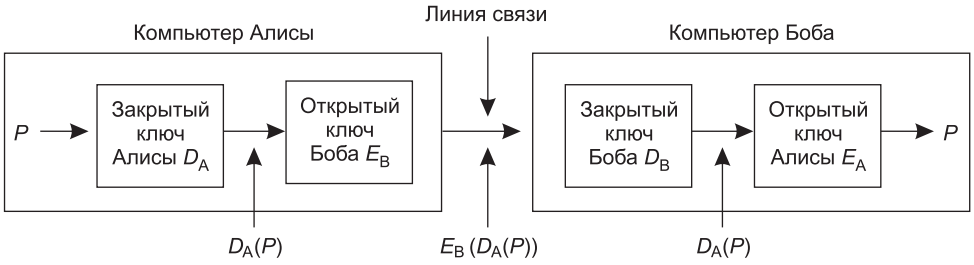
Боб просто проверяет значение случайного числа R_A, содержащегося в каждом приходящем сообщении, запоминая все такие числа, полученные за последний час. Если в течение часа такое значение получено еще не было, Боб может быть уверен, что пришедшее сообщение является новым заказом.

8.4.2. Подписи с открытым ключом

Главная проблема, связанная с применением шифрования с симметричным ключом для цифровых подписей, состоит в том, что все должны согласиться доверять Большому Брату. Кроме того, Большой Брат получает возможность читать все подписываемые им сообщения. Наиболее логичными кандидатами на управление сервером Большого Брата являются правительство, банки или нотариальные бюро. Однако эти организации вызывают доверие не у всех граждан. Таким образом, было бы гораздо лучше, если бы для получения подписи на электронном документе не требовалась авторитетная доверительная организация.

К счастью, здесь может помочь шифрование с открытым ключом. Предположим, что алгоритмы шифрования и дешифрации с открытым ключом, помимо обычного свойства $D(E(P)) = P$, обладают свойством $E(D(P)) = P$. Таким свойством, например, обладает алгоритм RSA, поэтому такое предположение не является голословным. В этом случае Алиса может послать Бобу подписанное открытое сообщение P, переслав ему $EB(DA(P))$. Обратите внимание, что Алиса знает свой собственный (закрытый) ключ дешифрации DA, так же как и открытый ключ Боба EB, так что сформировать такое сообщение ей по силам.

Получив это сообщение, Боб расшифровывает его как обычно, используя свой закрытый ключ D_B и получая в результате $D_A(P)$, как показано на рис. 8.16. Он сохраняет этот зашифрованный текст в надежном месте, после чего расшифровывает его открытым ключом шифрования Алисы E_A, получая открытый текст.



Ри. 8.16. Цифровая подпись, полученная при помощи шифрования с открытым ключом

Чтобы понять, как работает цифровая подпись в данном случае, предположим, что Алиса впоследствии отрицает, что посылала Бобу сообщение P. Когда дело доходит до суда, Боб предъявляет суду P и $DA(P)$. Судья легко может убедиться, что у Боба есть действительное сообщение, зашифрованное ключом DA, просто применив к нему ключ EA. Боб не знает закрытого ключа Алисы, следовательно, получить зашифрованное этим ключом сообщение он мог только от Алисы. Сидя в тюрьме за лжесвидетельство и мошенничество, Алиса сможет заняться разработкой новых интересных алгоритмов с открытым ключом.

Хотя схема использования шифрования с открытым ключом довольно элегантна, она обладает серьезными недостатками, связанными, правда, скорее не с самим алгоритмом, а со средой, в которой ему приходится работать. Во-первых, Боб может доказать, что это сообщение было послано Алисой, только пока ключ *DA* остается секретным. Если Алиса раскроет свой секретный ключ, этот аргумент перестанет быть убедительным, так как послать сообщение мог кто угодно, включая самого Боба.

Проблема может возникнуть, если Боб, например, является биржевым брокером Алисы. Алиса заказывает Бобу купить некоторое количество акций. Сразу после этого цена акций резко падает. Чтобы отречься от своего сообщения, посланного Бобу, Алиса заявляет в полицию, что ее дом был обворован, а компьютер вместе с секретным ключом украден. В зависимости от законов ее страны или штата, она может быть признана или не признана ответственной перед законом, особенно если она заявляет, что обнаружила, что ее квартира взломана, только после возвращения с работы, через несколько часов после того, как предполагаемый взлом случился.

Другая проблема данной схемы цифровой подписи возникает в случае, если Алиса решит сменить свой ключ. Подобное действие абсолютно законно, более того, рекомендуется периодически менять ключ, чтобы гарантировать его высокую надежность. В этом случае, если дело дойдет до судебного разбирательства, судья попытается применить к подписи *DA*(*P*) *текущий* ключ *EA* и обнаружит, что в результате не получается сообщение *P*. При этом Боб будет выглядеть довольно глупо.

В принципе для цифровых подписей можно использовать любой алгоритм с открытым ключом. Алгоритм RSA фактически стал промышленным стандартом. Он применяется во многих программах, предназначенных для обеспечения безопасности. Однако в 1991 году NIST предложил использовать для нового **Стандарта цифровой подписи DSS (Digital Signature Standard)** вариант алгоритма с открытым ключом Эль-Гамаля, основанный не на трудности разложения больших чисел на множители, а на сложности вычисления дискретных логарифмов.

Как обычно, попытка правительства навязать новые криптографические стандарты, вызвала много шума. Стандарт DSS критиковали за то, что он:

- ✦ слишком засекречен (протокол, использующий алгоритм Эль-Гамаля, разрабатывался Агентством национальной безопасности США);
- ✦ слишком медленный (от 10 до 40 раз медленнее алгоритма RSA для проверки подписей);
- ✦ слишком новый (алгоритм Эль-Гамаля еще не был достаточно тщательно проверен);
- ✦ слишком ненадежен (фиксированная длина ключа 512 бит).

При последующей переработке четвертый пункт претензий стал спорным, так как было разрешено использовать ключи длиной до 1024 бит. Однако первые два пункта актуальны и по сей день.

8.4.3. Профили сообщений

Многие методы цифровых подписей критикуются за то, что в них совмещаются две различные функции: аутентификация и секретность. Довольно часто требуется только аутентификация без секретности. К тому же, например, получить лицензию

на экспорт обычно проще, если система обеспечивает только аутентификацию, но не секретность. Ниже будет описана схема аутентификации, не требующая шифрования всего сообщения.

Эта схема основана на идее необратимой хэш-функции, которая принимает на входе участок открытого текста произвольной длины и по нему вычисляет строку битов фиксированной длины. У этой хэш-функции, часто называемой **профилем сообщения** (**message digest**, **MD**), есть четыре следующих важных свойства.

1. По заданному открытому тексту P легко сосчитать значение хэш-функции $MD(P)$.
2. По цифровой подписи $MD(P)$ практически невозможно определить значение открытого текста P.
3. Для данного P практически невозможно подобрать такой P', чтобы выполнялось равенство $MD(P') = MD(P)$.
4. Изменение даже одного бита входной последовательности приводит к очень непохожему результату.

Чтобы удовлетворять требованию 3, результат хэш-функции должен быть, по крайней мере, длиной в 128 бит, желательно даже больше. Чтобы удовлетворять требованию 4, хэш-функция должна искажать входные значения очень сильно. Этим данный метод напоминает алгоритмы с симметричными ключами, которые мы рассматривали ранее.

Профиль сообщения по части открытого текста вычисляется значительно быстрее, чем шифруется все сообщение с помощью алгоритма с открытым ключом. Поэтому профили сообщений могут использоваться для ускорения работы алгоритмов цифровых подписей. Чтобы понять, как все это работает, рассмотрим снова протокол передачи цифровой подписи, показанный на рис. 8.15. Вместо того чтобы посылать открытый текст P вместе с $K_{BB}(A, t, P)$, Большой Брат теперь вычисляет профиль сообщения $MD(P)$, применяя функцию хэширования MD к открытому тексту P. Затем он помещает $K_{BB}(A, t, MD(P))$ как пятый элемент в список, который зашифровывает ключом K_B, и отправляет его Бобу вместо $K_{BB}(A, t, P)$.

В случае возникновения спора Боб может предъявить на суде как открытый текст P, так и $K_{BB}(A, t, MD(P))$. По просьбе судьи Большой Брат расшифровывает $K_{BB}(A, t, MD(P))$, в результате чего суду предъявляется также цифровая подпись $MD(P)$, подлинность которой гарантируется Большим Братом, и сам открытый текст P, подлинность которого суд должен выяснить. Поскольку практически невозможно создать другой открытый текст, соответствующий данной цифровой подписи, суд убеждается в том, что Боб говорит правду. Использование профиля сообщения экономит время шифрования и затраты на транспортировку и хранение.

Профиль сообщения также может применяться для гарантии сохранности сообщения при передаче его по сети в системах шифрования с открытым ключом, как показано на рис. 8.17. Здесь Алиса сначала вычисляет профиль сообщения для своего открытого текста. Затем она подписывает профиль сообщения и посылает зашифрованный профиль сообщения и открытый текст Бобу. Если злоумышленник попытается подменить по дороге открытый текст P, Боб обнаружит это, сосчитав значение профиля сообщения $MD(P)$.

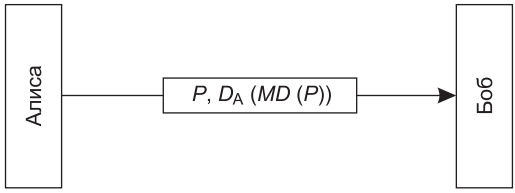


Рис. 8.17. Цифровая подпись с использованием профиля сообщения

SHA-1

Было предложено несколько вариантов функций, вычисляющих профиль сообщения. Самое широкое распространение получила функция **SHA-1** (**Secure Hash Algorithm 1** — надежный алгоритм хэширования) (NIST, 1993). Как и все профили сообщения, он перемешивает входные биты достаточно сложным образом, так что каждый выходной бит зависит от каждого входного бита. SHA-1 был разработан Агентством национальной безопасности США и получил благословение NIST (выразившееся в федеральном стандарте FIPS 180-1). Алгоритм SHA-1 обрабатывает входные данные блоками по 512 бит и формирует профиль сообщения длиной 160 бит. Типичный случай отправки Алисой несекретного, но подписанного сообщения Бобу показан на рис. 8.18. Открытый текст обрабатывается алгоритмом SHA-1, на выходе получается 160-битный хэш SHA-1. Он подписывается Алисой (закрытым ключом RSA) и отправляется вместе с открытым текстом Бобу.

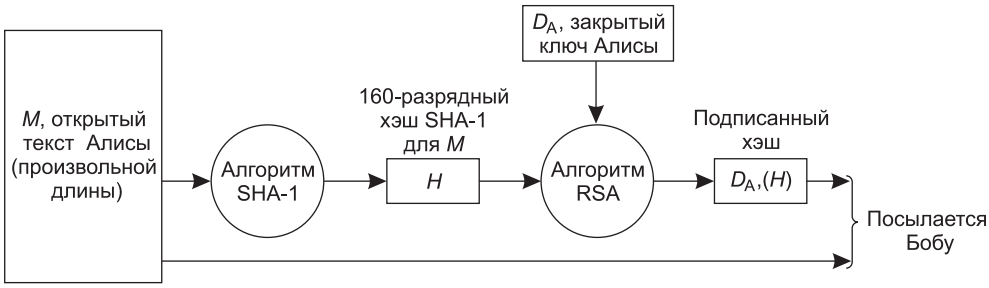


Рис. 8.18. Применение SHA-1 и RSA для создания подписей несекретных сообщений

При получении сообщения Боб сам вычисляет хэш-функцию с помощью алгоритма SHA-1 и применяет открытый ключ Алисы к подписанному хэшу для того, чтобы получить исходный хэш, *H*. Если они совпадают, сообщение считается корректным. Так как Труди не может, перехватив сообщение, изменить его таким образом, чтобы значение *H* совпадало с контрольным, Боб легко узнает обо всех подменах, которые совершила Труди. Для сообщений, чья неприкосновенность существенна, а секретность не имеет значения, часто применяется схема, показанная на рис. 8.18. При относительно небольших затратах на вычисления она гарантирует, что любые изменения, внесенные на пути следования сообщения, будут с высокой вероятностью выявлены.

Давайте теперь вкратце рассмотрим, как работает SHA-1. Для начала алгоритм SHA-1 также дополняет сообщение единичным битом в конце, за которым следует такое количество нулевых бит (равное как минимум 64), чтобы в итоге получилось общее число бит, кратное 512. Затем 64-разрядное число, содержащее длину сообщения (до битового дополнения), логически складывается (операция ИЛИ) с 64 младшими битами. На рис. 8.19, *а* показано сообщение с дополнением, расположенным справа, потому что английский текст и рисунки читаются слева направо (то есть правая граница рисунка воспринимается как его конец, а левая — как начало). Применительно к вычислительной технике такое расположение соответствует обратному порядку хранения байтов (сначала передается самый значимый, старший бит). Такая реализация присуща, например, SPARC или IBM 360 и его последователям. Однако, вне зависимости от используемой техники, SHA-1 вставляет битовое дополнение в конец сообщения.

Во время выполнения вычислений SHA-1 работает с пятью 32-битными переменными ($H_0 \ldots H_4$), в которых накапливается значение хэш-функции. Они показаны на рис. 8.19, *б*. Их начальные значения — это постоянные величины, определенные стандартом.

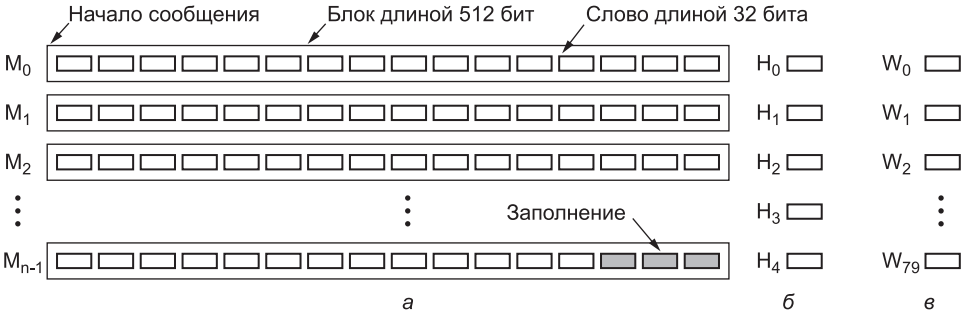


Рис. 8.19. Сообщение, дополненное до размера, кратного 512 битам (*а*); выходные переменные (*б*); массив слов (*в*)

Затем поочередно обрабатываются блоки с M_0 по M_{n-1}. Для текущего блока 16 слов сначала копируются в начало вспомогательного массива W размером 80 слов, как показано на рис. 8.19, *в*. Оставшиеся 64 слова вычисляются с использованием следующей формулы:

$$W_i = \mathrm{S}^1(W_{i-3} \text{ XOR } W_{i-8} \text{ XOR } W_{i-14} \text{ XOR } W_{i-16}) \; (16 \le i \le 79),$$

где $S^b(W)$ представляет собой поворот 32-разрядного слова W на b бит. Теперь по значениям $H_0 \ldots H_4$ инициализируются переменные от A до E.

Сами вычисления на псевдо-С можно записать таким образом:

```
for(i=0;i<80;i++){
    temp = S5(A) +fi(B, C, D) + E + Wi + Ki;
    E=D; D=C; C=S30(B); B=A; A=temp;
}
```

где постоянные K_i определяются стандартом. Смешивающие функции f_i задаются следующим образом:

$f_i(B,C,D) = (B \text{ AND } C) \text{ OR } (\text{NOT } B \text{ AND } D)$ $(0 \le i < 19)$

$f_i(B,C,D) = B \text{ XOR } C \text{ XOR } D$ $(20 < i < 39)$

$f_i(B,C,D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D)$ $(40 < i < 59)$

$f_i(B,C,D) = B \text{ XOR } C \text{ XOR } D$ $(60 < i < 79)$

После 80 итераций цикла значения переменных A ... E добавляются к H_0 ... H_4 соответственно.

После обработки первого 512-битного блока начинается обработка следующего. Массив W инициализируется заново с помощью нового блока, однако H сохраняется неизменным. По окончании этого блока обрабатывается следующий, и т. д., пока все 512-разрядные блоки сообщения не попадут в эту кастрюлю. После обработки последнего блока пять 32-разрядных слов в массиве H выводятся в качестве 160-битного значения криптографической хэш-функции. Полный код SHA-1 приведен в RFC 3174.

В настоящее время идет работа над новыми версиями SHA-1 с 224, 256-, 384- и 512-разрядными значениями хэш-функций. Эти версии вместе называются SHA-2. Не только их хэши длиннее, чем хэши SHA-1, профиль был изменен еще и для того, чтобы избавиться от некоторых потенциально слабых сторон SHA-1. SHA-2 пока еще широко не используется, но это вероятно случится в будущем.

MD5

Для полноты картины мы отметим еще один популярный профиль. Алгоритм **MD5 (Message Digest 5 — профиль сообщения 5)** представляет собой пятую версию хэш-функций, разработанных Рональдом Ривестом (Ronald Rivest). Сначала сообщение дополняется до длины 448 бит по модулю 512. Затем к нему добавляется исходная длина сообщения, рассматриваемая как 64-разрядное число, в результате чего получается блок битов длиной кратной 512. Последний шаг подготовки к вычислениям инициализирует 128-разрядный буфер, задавая его содержимое равным некоему фиксированному значению.

Затем начинаются вычисления. На каждом этапе берется 512-разрядный блок входного текста и тщательно перемешивается со 128-разрядным буфером. Для пущей навористости в кастрюлю также кидается содержимое таблицы синусов. Именно синусы используются не потому, что их результат более случаен, чем результат других генераторов случайных чисел (в которых часто также применяются тригонометрические функции), а чтобы избежать каких бы то ни было подозрений в создании потайной лазейки, через которую потом разработчик (или заказчик) мог бы войти. Процесс продолжается, пока не будут обработаны все входные блоки. Содержимое 128-разрядного буфера и образует профиль сообщения.

После более десяти лет использования и изучения MD5 были обнаружены совпадения, то есть разные сообщения с одинаковым хэшем (Sotirov et al., 2008). Это страшное предзнаменование для функции профилей, так как оно означает, что функция не может безопасно использоваться для представления сообщения. Таким образом, сообщество по безопасности считает MD5 сломанным; его нужно заменить там, где это возможно, и новые системы не должны использовать его как часть своей конструкции. Несмотря на это, на момент написания книги этот алгоритм еще держится на плаву.

8.4.4. Задача о днях рождения

В мире шифров многое оказывается совсем не таким, каким кажется на первый взгляд. Можно, например, предполагать, что для ниспровержения профиля сообщения, состоящего из m разрядов, потребуется порядка 2^m операций. На самом деле, часто оказывается достаточно $2^{m/2}$ операций, если использовать метод, основанный на **задаче о днях рождения (birthday attack)**, опубликованный в ставшей классической книге (Yuval, 1979).

В основе идеи этого метода лежит задача, часто приводимая в качестве примера профессорами математики на курсах по теории вероятности. Вопрос: сколько студентов должно находиться в классе, чтобы вероятность появления двух человек с совпадающими днями рождения превысила 1/2? Большинство студентов обычно ожидают, что ответ будет значительно больше 100. На самом же деле, теория вероятности утверждает, что это число равно 23. Не вдаваясь в тонкости анализа этой проблемы, дадим интуитивно понятное объяснение: из 23 человек мы можем сформировать $(23 \times 22)/2 = 253$ различных пары, у каждой из которой дни рождения могут совпасть с вероятностью 1/365. Теперь этот ответ уже не кажется таким удивительным.

В более общем случае, если имеется некое соответствие между n входами (людьми, сообщениями и т. д.) и k возможными выходами (днями рождения, профилями сообщений и т. д.), мы имеем $n(n-1)/2$ входных пар. Если $n(n-1)/2 > k$, то вероятность того, что будет хотя бы одно совпадение выхода при различных входах, довольно велика. Таким образом, вероятность существования двух сообщений с одинаковыми профилями велика уже при $n > \sqrt{k}$ Это означает, что 64-разрядный профиль сообщения можно с большой вероятностью взломать (то есть найти два различных сообщения с одинаковым профилем), перебрав 232 сообщений.

Рассмотрим практический пример. На кафедре компьютерных наук Государственного университета появились вакансия и два кандидата на эту должность, Том и Дик. Том работает на факультете на два года дольше Дика, поэтому его кандидатура будет рассматриваться первой. Если он получит эту должность, значит, Дику не повезло. Том знает, что заведующая кафедрой Мэрилин высоко ценит его работу, поэтому он просит ее написать для него рекомендательное письмо декану факультета, который будет решать дело Тома. После отправки все письма становятся конфиденциальными.

Мэрилин просит написать это письмо декану своего секретаря Элен, подчеркивая, что она хотела бы видеть в этом письме. Когда письмо готово, Мэрилин просматривает его, вычисляет и подписывает 64-разрядный профиль письма и посылает этот подписанный профиль декану. Позднее Элен может послать само письмо электронной почтой.

К несчастью для Тома, у Элен роман с Диком, и она хочет обмануть Тома. Поэтому она пишет следующее письмо с 32 вариантами в квадратных скобках.

Уважаемый господин декан,

Это [*письмо* | *обращение*] отражает мое [*искреннее* | *откровенное*] мнение о проф. Томе Уилсоне, являющемся [*кандидатом* | *претендентом*] на профессорскую должность [*в настоящее время* | *в этом году*]. Я [*знакома* | *работала*] с проф. Уилсоном в течение [*почти* | *около*] шести лет. Он является [*выдающимся* | *блестящим*] исследователем, обладающим [*большим талантом* | *большими возможностями*] и известным

[*во всем мире* | *не только в нашей стране*] своим [*серьезным* | *созидательным*] подходом к [*большому числу* | *широкому спектру*] [*сложных* | *перспективных*] вопросов.

Он также является [*высоко* | *весьма*] [*уважаемым* | *ценимым*] [*преподавателем* | *педагогом*]. Его студенты дают его [*занятиям* | *лекциям*] [*самые высокие* | *высочайшие*] оценки. Он самый [*популярный* | *любимый*] [*преподаватель* | *учитель*] [*нашей кафедры* | *нашего Университета*].

[*Кроме* | *Помимо*] того, [*гранты* | *контракты*] проф. Уилсона [*существенно* | *значительно*] пополнили [*фонды* | *финансовые запасы*] нашей кафедры. Эти [*денежные* | *финансовые*] средства [*позволили нам* | *дали возможность*] [*выполнить* | *осуществить*] [*много* | *ряд*] [*важных* | *специальных*] программ, [*таких как* | *среди которых*], государственная программа 2000 года. Без этих средств было бы невозможным продолжение этой программы, такой [*важной* | *значительной*] для нас. Я настойчиво рекомендую вам предоставить ему эту должность.

К несчастью для Тома, закончив печатать это письмо, Элен тут же принимается за второе:

Уважаемый господин декан,

Это [*письмо* | *обращение*] отражает мое [*искреннее* | *откровенное*] [*мнение* | *суждение*] о проф. Томе Уилсоне, являющемся [*кандидатом* | *претендентом*] на профессорскую должность в [*настоящее время* | *этом году*]. Я [*знакома* | *работала*] с проф. Уилсоном в течение [*почти* | *около*] шести лет. Он является [*слабым* | *недостаточно талантливым*] [*исследователем* | *ученым*], почти не известным в той области науки, которой он занимается. В его работах практически не заметно понимания [*ключевых* | *главных*] [*проблем* | *вопросов*] современности.

[*Более* | *Кроме*] того, он также не является сколько-нибудь [*уважаемым* | *ценимым*] [*преподавателем* | *педагогом*]. Его студенты дают его [*занятиям* | *лекциям*] [*самые низкие* | *негативные*] оценки. Он самый непопулярный [*преподаватель* | *учитель*] нашей кафедры, [*славящийся* | *печально известный*] своей [*привычкой* | *склонностью*] [*высмеивать* | *ставить в неудобное положение*] студентов, осмелившихся задавать вопросы на его [*лекциях* | *занятиях*].

[*Кроме* | *Помимо*] того, [*гранты* | *контракты*] проф. Уилсона [*почти* | *практически*] не пополняют [*фондов* | *финансовых запасов*] нашей кафедры. Если не удастся быстро найти новый источник финансирования, [*мы будем вынуждены* | *нам придется*] [*закрыть* | *прекратить*] [*много* | *ряд*] [*важных* | *специальных*] программ, [*таких как* | *среди которых*] государственная программа 2000 года. К сожалению, при таких [*условиях* | *обстоятельствах*] я не могу [*предлагать* | *рекомендовать*] его вам на эту должность.

Затем Элен заставляет свой компьютер сосчитать 232 профиля сообщения для каждого варианта обоих писем, что занимает всю ночь. Есть шансы, что один профиль первого письма совпадет с одним из профилей второго письма. Если нет, она может добавить еще несколько вариантов слов и выражений в каждое письмо и попытаться еще раз за выходные. Предположим, что ей удалось найти такое совпадение. Назовем положительный отзыв письмом *A*, а отрицательный — письмом *B*.

Элен отправляет письмо *A* электронной почтой Мэрилин на утверждение. Мэрилин, конечно, утверждает письмо, вычисляет 64-разрядный профиль сообщения, подписывает профиль и посылает по почте подписанный профиль декану. Независимо Элен посылает декану письмо *B* (вместо письма *A*, которое следует отправить на самом деле).

Получив письмо и подписанный профиль, декан запускает алгоритм вычисления профиля сообщений для письма *B*, видит, что его профиль совпадает с тем, что ему прислала Мэрилин, и увольняет Тома. Он даже не может себе представить, что Элен удалось создать два письма с одинаковыми профилями сообщений и отправить ему совсем не тот вариант, который читала и подписала Мэрилин. (Вариант с хэппи-эндом: Элен сообщает Дику о своих проделках. Потрясенный, Дик порывает с ней. Элен в ярости бежит сознаваться во всем Мэрилин. Мэрилин звонит декану. В конце концов, Том получает профессуру.) При использовании алгоритма SHA-1 подобная атака шифра является достаточно трудной для исполнения, так как даже если компьютер сможет вычислять по 1 трлн профилей в секунду, потребуется более 32 00 лет, чтобы перебрать по 280 варианта для обоих писем, что все равно не даст 100%-й гарантии совпадения. Конечно, если заставить параллельно работать 1 млн, вместо 32 000 лет потребуется 2 недели.

8.5. Управление открытыми ключами

Криптография с использованием открытых ключей позволяет передавать секретные данные, не обладая общим ключом заранее, а также создавать электронные подписи сообщений без необходимости привлечения третьей, доверительной стороны. Наконец, подписанные профили сообщений позволяют получателю легко и надежно проверять целостность и аутентичность полученных данных.

Однако мы как-то слишком ловко обошли один вопрос: если Алиса и Боб друг друга не знают, как они смогут обменяться открытыми ключами перед началом общения? Вот, казалось бы, очевидное решение: выложить открытый ключ на веб-сайте. Однако так делать нельзя, и вот почему: представим себе, что Алиса хочет найти открытый ключ Боба на его веб-сайте. Каким образом она это делает? Набирает URL сайта. Браузер ищет DNS-адрес домашней страницы Боба и посылает запрос *GET*, как показано на рис. 8.20. К сожалению, Труди в этот момент перехватывает запрос и посылает Алисе фальшивую страницу. В качестве таковой может выступать, к примеру, копия страницы Боба, на которой вместо его открытого ключа выложен открытый ключ Труди. После того как Алиса зашифрует свое первое сообщение с помощью E_T, Труди расшифрует его, прочтет, зашифрует с помощью открытого ключа Боба и перешлет сообщение Бобу, который даже не подозревает обо всех этих перипетиях. Но гораздо хуже то, что Труди может изменять сообщения перед повторным шифрованием и отправкой Бобу. Очевидно, нужен некий механизм секретного обмена открытыми ключами.

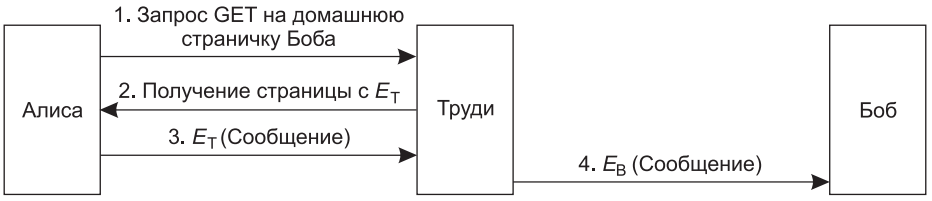


Рис. 8.20. Способ вторжения в систему с открытыми ключами

8.5.1. Сертификаты

Первая попытка организации защищенного обмена ключами может состоять в создании круглосуточного интернет-центра распространения ключей по требованию. Один из множества недостатков такого решения заключается в том, что данную систему не удастся масштабировать, и сам этот центр очень скоро станет узким местом. А если он не выдержит нагрузки, вся интернет-безопасность в один момент сойдет на нет.

По этой причине было придумано новое решение, не требующее, чтобы центр распространения ключей был доступен постоянно. В общем-то, даже не требуется, чтобы он вообще работал в подключенном (онлайновом) режиме. Вместо этого на него возлагается обязанность сертификации открытых ключей, принадлежащих как физическим, так и юридическим лицам. Организация, занимающаяся сертификацией открытых ключей, в настоящее время называется **Управлением сертификации** (**CA — Certification Authority**).

В качестве примера рассмотрим такую ситуацию: Боб хочет разрешить Алисе и другим лицам, с которыми он не знаком, устанавливать с ним защищенные соединения. Он приходит в Управление сертификации со своим открытым ключом, а также паспортом или другим удостоверением личности и просит зарегистрировать ключ. Управление выдает ему сертификат, напоминающий тот, что показан на рис. 8.21, и подписывает хэш SHA-1 своим закрытым ключом. Затем Боб оплачивает услуги Управления и получает дискету с сертификатом и подписанным хэшем.

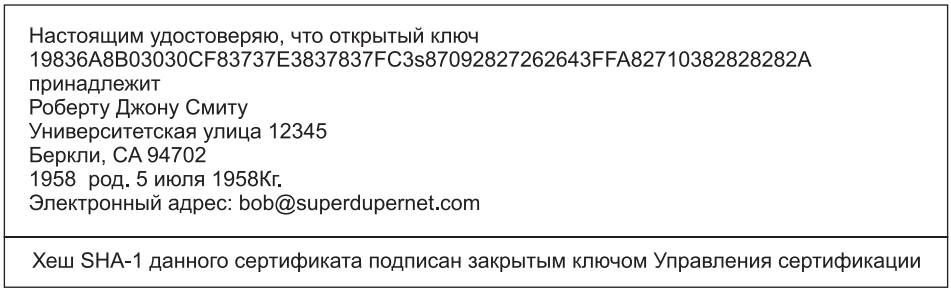


Рис. 8.21. Пример сертификата и подписанного хэша

Основная задача сертификата состоит в связывании открытого ключа с именем принципала (физического или юридического лица). Сертификаты сами по себе никак не защищаются и не хранятся в тайне. Например, Боб может выложить его на свой сайт и поставить ссылку: «Здесь можно посмотреть на сертификат моего открытого ключа». Перейдя по этой ссылке, пользователь увидит и сертификат, и блок с подписью (подписанный хэш SHA-1 сертификата).

Давайте теперь снова взглянем на сценарий, показанный на рис. 8.20. Что может сделать Труди, перехватив запрос страницы Боба, посланный Алисой? Она может выложить на странице свой собственный сертификат и блок с подписью на подложной странице, однако Алиса, читая этот сертификат, сразу догадается, что она разговаривает не с Бобом: в нем его имени просто нет. Труди может изменить домашнюю страницу Боба «на лету», заменив его открытый ключ своим собственным. И все же,

проверив сертификат алгоритмом SHA-1, она получит значение хэш-функции, не согласующееся с тем, которое будет вычислено по открытому ключу Управления сертификации и блоку подписи. Так как Труди не имеет доступа к закрытому ключу Управления сертификации, она никак не может сгенерировать блок подписи, содержащий хэш модифицированной страницы с выложенным на ней открытым ключом Труди. Таким образом, Алиса может не беспокоиться о подлинности ключа Боба. И вот, как мы и обещали, при такой схеме не требуется, чтобы Управление сертификации постоянно работало в подключенном режиме. Тем самым ликвидируется потенциально узкое место системы.

Сертификат может связывать открытый ключ не только с принципалом, но и с **атрибутом**. Например, сертификат может содержать такую информацию: «Данный открытый ключ принадлежит лицу старше 18 лет». Этим можно подтвердить статус принципала или убедиться в том, что ему разрешен доступ к некоторым специфическим данным, на которые накладываются возрастные ограничения. При этом имя принципала может и не раскрываться. Обычно владелец сертификата отсылает его на веб-сайт, принципалу или тому процессу, который обеспокоен возрастом клиента. В ответ генерируется случайное число, шифруемое с помощью открытого ключа (считываемого с сертификата). Если клиент сможет расшифровать его и отослать обратно, это будет служить подтверждением тому, что он действительно обладает указанными в сертификате характеристиками. Еще это случайное число может быть использовано в качестве сеансового ключа для будущего соединения.

Сертификат может содержать атрибут еще в одном случае: если речь идет об объектно-ориентированной распределенной системе. Каждый объект обычно обладает некоторым набором методов. Владелец объекта может предоставлять каждому клиенту сертификат с указанием тех методов, которыми он может пользоваться. Этот список в виде поразрядной карты отображения информации (битовой карты) можно связать с открытым ключом, используя подписанный сертификат. Опять же, если владелец сертификата сможет подтвердить факт обладания соответствующим закрытым ключом, он сможет воспользоваться методами, указанными в списке. Особенностью такого использования сертификатов является отсутствие необходимости указывать имя владельца. Это бывает полезно в ситуациях, когда важна конфиденциальность.

8.5.2. X.509

Если бы все желающие подписать что-нибудь обращались в Управление идентификации с различными видами сертификатов, обслуживание всех различных форматов вскоре стало бы проблемой. Во избежание этого был разработан и утвержден организацией ITU специальный стандарт сертификатов. Он называется **X.509** и широко применяется в Интернете. В свет, начиная с 1988 года, вышло уже три версии стандарта, и мы будем рассматривать новейшую из них — третью.

На стандарт X.509 сильное влияние оказал мир OSI, в связи с чем в нем появились некоторые неприятные вещи, как, например, определенный принцип кодирования и именования. Удивительно, что проблемная группа по развитию Интернета, IETF, согласилась с данной концепцией, особенно учитывая то, что практически во всех

областях, начиная с машинных адресов и заканчивая транспортными протоколами и форматами электронных писем, IETF всегда игнорировала OSI и пыталась сделать что-нибудь более толковое. IETF-версия X.509 описана в RFC 5280.

По сути, X.509 — это способ описания сертификатов. Основные поля сертификата перечислены в табл. 8.3. Из описаний, приведенных в правой колонке, должно быть понятно, для чего служит поле. За дополнительной информацией обращайтесь к RFC 2459.

Например, если Боб работает в отделе ссуд банка Money Bank, его X.500-адрес будет выглядеть так:

```
/C=US/O=MoneyBank/OU=Loan/CN=Bob/
```

где `C` — страна, `O` — организация, `OU` — отдел организации, `CN` — имя. Управление сертификации и другие сущности именуются похожим образом. Существенная проблема с системой именования X.500 заключается в том, что если Алиса пытается соединиться с *bob@moneybank.com* и имеет данные сертификата с именем в этом формате, для нее вовсе не очевидно, что этот сертификат имеет отношение именно к тому Бобу, который ей нужен. К счастью, начиная с третьей версии, разрешено использование имен DNS вместо X.500, поэтому данная проблема может счастливо разрешиться.

Сертификаты кодируются с использованием **OSI ASN.1 (Abstract Syntax Notation 1 — системы записи абстрактного синтаксиса 1)**. Ее можно рассматривать как нечто подобное структуре в языке C, за тем исключением, что эта запись очень странная и многословная. Более подробную информацию о X.509 можно найти в (Ford и Baum, 2000).

Таблица 8.3. Основные поля сертификата в стандарте X.509

Поле	Значение
Version	Версия X.509
Serial number	Это число вместе с названием Управления сертификации однозначно идентифицирует сертификат
Signature algorithm	Алгоритм генерации подписи сертификата
Issuer	X.500-имя Управления
Validity period	Начало и конец периода годности
Subject name	Сущность, ключ которой сертифицируется
Public key	Открытый ключ сущности и идентификатор использующего его алгоритма
Issuer ID	Необязательный идентификатор, единственным образом определяющий эмитента (создателя) сертификата
Subject ID	Необязательный идентификатор, единственным образом определяющий владельца сертификата
Extensions	Различные возможные расширения
Signature	Подпись сертификата (генерируется с помощью закрытого ключа Управления сертификации)

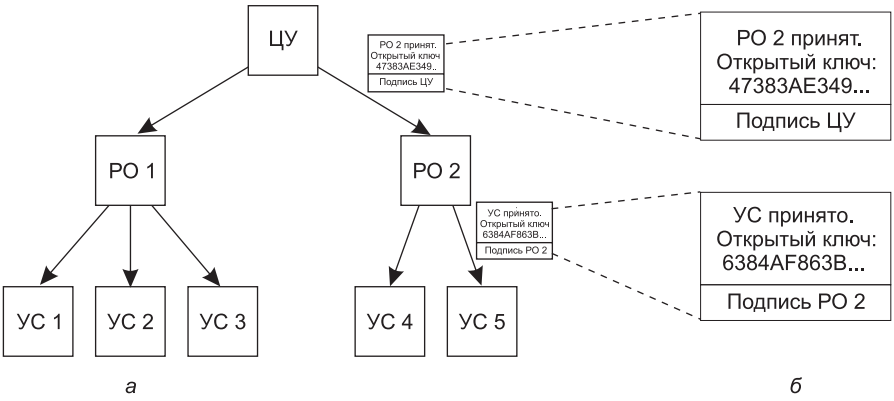
8.5.3. Инфраструктуры систем с открытыми ключами

Понятно, что одного Управления сертификации на весь мир недостаточно. Оно бы быстро перестало функционировать из-за огромной нагрузки, да еще и стало бы эпицентром всех проблем, связанных с безопасностью сетей. Возможно, следует создать целый набор таких Управлений, использующих один и тот же закрытый ключ для подписания сертификатов, под покровительством одной и той же организации. Однако, хотя это и решит проблему распределения нагрузки, возникнет новый вопрос, связанный с *утечкой ключа*. Если по всему миру будут разбросаны десятки серверов, хранящих закрытый ключ Управления сертификации, велик шанс, что рано или поздно этот ключ будет украден или пропадет каким-то иным образом. Если ключ будет рассекречен, всю мировую инфраструктуру электронной безопасности можно будет похоронить. Вместе с тем, наличие всего одного центрального Управления сертификации — это тоже риск.

Далее, какая организация будет заведовать Управлением? Довольно трудно представить себе какую-либо законную структуру с большим кредитом доверия мирового масштаба. В некоторых странах предпочтительно, чтобы это было какое-нибудь правительственное учреждение, а где-то — наоборот, чтобы это было чем угодно, но не правительством.

По этим причинам был разработан альтернативный способ сертификации открытых ключей. Он известен под общим названием **PKI (Public Key Infrastructure — инфраструктура систем с открытыми ключами)**. В этом разделе мы рассмотрим только общие принципы действия PKI; поскольку было внесено множество предложений по ее модификации и некоторые детали могут со временем измениться.

PKI состоит из множества компонентов, среди которых пользователи, Управления сертификации, сами сертификаты, а также каталоги. Инфраструктура систем с открытыми ключами предоставляет возможность структурной организации этих компонентов и определяет стандарты, касающиеся различных документов и протоколов. Одним из простейших видов PKI является иерархия Управлений, представленная на рис. 8.22. На рисунке представлены три уровня, однако реально их может быть как больше, так и меньше. Управление сертификации верхнего уровня (root) мы будем называть Центральным управлением (ЦУ). Центральное управление сертифицирует управления второго уровня — назовем их **Региональными отделами (РО — Regional Authorities, RA)**, — так как они могут обслуживать некоторый географический регион, например страну или континент. Этот термин не стандартизован. Названия для уровней иерархии вообще не оговариваются стандартом. Региональные отделы, в свою очередь, занимаются легализацией реальных Управлений сертификации (УС), эмитирующих сертификаты стандарта X.509 для физических и юридических лиц. При легализации Центральным управлением нового Регионального отдела последнему выдается сертификат, подтверждающий его признание. Он содержит открытый ключ нового РО и подпись ЦУ. Аналогичным образом РО взаимодействуют с Управлениями сертификации: выдают и подписывают сертификаты, содержащие открытые ключи УС и признающие легальность их деятельности.



Ри. 8.22. Иерархия PKI (*а*); цепочка сертификатов (*б*)

Итак, наша PKI работает следующим образом. Допустим, Алисе нужен открытый ключ Боба, чтобы она могла с ним пообщаться. Она ищет и находит содержащий его сертификат, подписанный УС 5. Однако Алиса никогда ничего не слышала про УС 5. Этим «Управлением» может оказаться, на самом деле, десятилетняя дочка Боба. Алиса может отправиться в УС 5 и попросить подтвердить легитимность. Управление в ответ может показать сертификат, полученный от РО 2 и содержащий открытый ключ УС 5. Теперь, вооружившись открытым ключом УС 5, Алиса может удостовериться в том, что сертификат Боба действительно подписан УС 5, а значит, является легальным.

Если только РО 2 не является двенадцатилетним сыном Боба. Если Алисе вдруг придет в голову такая мысль, она может запросить подтверждение легитимности РО 2. Ответом будет служить сертификат, подписанный Центральным управлением и содержащий открытый ключ РО 2. Вот теперь Алиса может не сомневаться, что она получила открытый ключ Боба, а не кого-то другого.

А если Алиса хочет узнать открытый ключ ЦУ? Как это сделать? Загадка. Предполагается, что открытый ключ ЦУ знают все. Например, он может быть «зашит» внутрь ее браузера.

Но наш Боб — добряк, он не хочет доставлять Алисе лишние хлопоты. Он понимает, что она захочет проверить легитимность УС 5 и РО 2, поэтому он сам собирает соответствующие сертификаты и отправляет их ей вместе со своим. Теперь, зная открытый ключ ЦУ, Алиса может проверить по цепочке все интересующие ее организации. Ей не придется никого беспокоить для подтверждения. Поскольку все сертификаты подписаны, она может запросто уличить любые попытки подлога. Цепочка сертификатов, восходящая к ЦУ, иногда называется **доверительной цепочкой (chain of trust)** или **путем сертификата (certification path)**. Описанный метод широко применяется на практике.

Конечно, остается проблема определения владельца ЦУ. Следует иметь не одно Центральное управление, а несколько, причем связать с каждым из них свою иерархию региональных отделов и управлений сертификации. На самом деле, в современные браузеры действительно «зашиваются» открытые ключи более 100 центральных

управлений, иногда называемые **доверительными якорями (trust anchors)**. Как видите, можно избежать проблемы одного всемирного учреждения, занимающегося сертификацией.

Однако встает вопрос, какие доверительные якоря производители браузеров могут считать надежными, а какие — нет. Все, на самом деле, сводится к тому, насколько конечный пользователь доверяет разработчику браузера, насколько он уверен в том, что решения генерируются грамотно и доверительные якоря не принимаются от всех желающих за умеренную плату. Большинство браузеров обеспечивают возможность проверки ключей ЦУ (обычно это делается с помощью сертификатов, подписанных им) и удаления подозрительных ключей.

Каталоги

Инфраструктура систем с открытыми ключами должна решать еще один вопрос. Он касается места хранения сертификатов (и цепочек, ведущих к какому-нибудь доверительному якорю). Можно заставить всех пользователей хранить свои сертификаты у себя. Это безопасно (так как невозможно подделать подписанные сертификаты незаметно), но не слишком удобно. В качестве каталога для сертификатов было предложено использовать DNS. Прежде чем соединиться с Бобом, Алисе, видимо, все равно придется узнать с помощью службы имен доменов (DNS) его IP-адрес. Так почему бы не заставить DNS возвращать вместе с IP-адресом всю цепочку сертификатов?

Кому-то это кажется выходом из положения, однако некоторые считают, что лучше иметь специализированные серверы с каталогами для хранения сертификатов X.509. Такие каталоги могли бы с помощью имен X.500 обеспечивать возможность поиска. Например, теоретически можно представить себе услугу сервера каталогов, позволяющую получать ответы на запросы типа «Дайте мне полный список всех людей по имени Алиса, работающих в отделе продаж в любом месте США или Канады».

Аннулирование

Реальный мир полон разного рода сертификатов, среди которых, например, паспорта, водительские удостоверения. Иногда эти сертификаты необходимо аннулировать (например, водительское удостоверение надо аннулировать за езду в нетрезвом состоянии). Та же проблема возникает и в мире цифровых технологий: лицо, предоставившее сертификат, может отозвать его за нарушение противоположной стороной каких-либо условий. Это необходимо делать и тогда, когда закрытый ключ сущности перестал быть защищенным, или, что еще хуже, ключ УС потерял кредит доверия. Таким образом, инфраструктура систем с открытыми ключами должна как-то обеспечивать процедуру аннулирования. Возможность аннулирования все усложняет.

Первым шагом в этом направлении является принуждение всех УС к периодическому выпуску **списка аннулированных сертификатов (CRL — Certificate Revocation List)**. В нем перечисляются порядковые номера всех аннулированных сертификатов. Поскольку в сертификатах содержится дата окончания срока годности, в CRL следует включать номера только тех из них, срок годности которых еще *не* истек. По истечении

срока годности сертификаты перестают быть действительными автоматически, поэтому нужно различать случаи аннулирования «по старости» и по другим причинам. В любом случае, их использование необходимо запрещать.

К сожалению, возникновение списков аннулированных сертификатов означает, что лицо, собирающееся использовать сертификат, должно вначале убедиться в том, что сертификата нет в этом списке. В противном случае от использования надо отказаться. Тем не менее сертификат мог быть аннулирован тотчас же после выпуска самого свежего варианта черного списка. Получается, что единственный надежный способ — это узнать о состоянии сертификата непосредственно у УС. Причем эти запросы придется посылать при каждом использовании сертификата, так как нет никакой гарантии, что его аннулирование не произошло несколько секунд назад.

Еще больше усложняет ситуацию то, что аннулированный сертификат иногда требуется восстанавливать. Например, если причиной отзыва была неуплата каких-нибудь взносов, после внесения необходимой суммы не остается никаких причин, которые не позволяли бы восстановить сертификат. Обработка ситуаций аннулирования и восстановления сводит на нет такое ценное свойство сертификатов, как возможность их использования без помощи УС.

Где хранить списки аннулированных сертификатов? Было бы здорово хранить их там же, где и сами сертификаты. Одна из стратегий подразумевает, что УС периодически выпускает черные списки и заставляет вносить обновления в каталоги (удаляя отозванные сертификаты). Если для хранения сертификатов каталоги не используются, можно кэшировать их в разных местах в сети. Поскольку черный список сам по себе является подписанным документом, любые попытки подлога тотчас будут замечены.

Если сертификаты имеют большие сроки годности, списки аннулированных сертификатов также будут довольно длинными. Например, количество отозванных кредитных карточек со сроком годности 5 лет будет гораздо больше списка отозванных трехмесячных карточек. Стандартным способом борьбы с длинными списками является достаточно редкий выпуск самих списков и частый — обновлений к ним. Кроме всего прочего, это помогает снизить необходимую для распространения списков пропускную способность.

8.6. Защита соединений

Мы закончили изучение прикладных инструментов. Было описано большинство применяемых методов и протоколов. Оставшаяся часть главы будет посвящена применению этих методов на практике для обеспечения безопасности сетей. Кроме того, будут высказаны некоторые мысли относительно социального аспекта этого вопроса.

Следующие четыре раздела посвящены безопасности соединений, то есть тому, как секретно и без риска подмены данных передавать биты от пункта отправления до пункта назначения, а также тому, как не пускать на линию посторонние биты. Это ни в коем случае не полный список проблем сетевой безопасности, однако перечисленные вопросы являются одними из самых важных.

8.6.1. IPsec

Проблемная группа IETF в течение многих лет мирилась с отсутствием безопасности в Интернете. Обеспечить ее было действительно непросто и прежде всего потому, что разгорелись жаркие споры вокруг того, какую часть Интернета следует, собственно, защищать. Большинство экспертов по вопросам безопасности уверены в том, что по-настоящему надежная система должна выполнять сквозную шифрацию и сквозное обеспечение целостности данных (то есть все это должно быть сделано на прикладном уровне). Это означает, что процесс-источник шифрует и/или ставит защиту целостности данных и отправляет их процессу-приемнику, который, соответственно, дешифрует данные и проверяет их целостность. Тогда можно будет заметить любые попытки взлома (даже на уровне операционной системы на любой из сторон). Беда такого подхода в том, что для обеспечения безопасности требуется вносить изменения во все приложения. Это означает, что необходимо «спустить» шифрацию на транспортный уровень или организовать новый специализированный подуровень между прикладным и транспортным. Он должен быть сквозным, но в то же время не требующим внесения изменений в приложения.

Противоположная точка зрения состоит в том, что пользователи все равно не осознают необходимости применения мер безопасности и просто не способны корректно использовать все предоставленные возможности. При этом никто не захочет каким-либо образом изменять существующие программы, поэтому сетевой уровень должен выполнять проверку подлинности и/или шифровать сообщения незаметно для пользователя. Долгие годы сражений привели к победе этой точки зрения: был разработан стандарт безопасности, ориентированный на сетевой уровень. Одним из аргументов было то, что шифрация на сетевом уровне, с одной стороны, не помешает тем пользователям, которые серьезно относятся к безопасности, а с другой — в некоторой степени убережет беспечных пользователей.

Результатом всех этих дискуссий было создание стандарта **IPsec (IP security — IP-безопасность)**, описанного в RFC 2401, 2402, 2406 и др. Не всем пользователям требуется шифрование соединений (выполнение соответствующих процедур может занимать существенную часть вычислительных ресурсов). Однако, вместо того чтобы делать шифрование необязательным, пользователю предлагается в случае необходимости выбирать пустой алгоритм. В RFC 2410 расписываются такие достоинства пустого алгоритма, как простота, легкость реализации и высокая скорость.

IPsec служит основой для множества услуг, алгоритмов и модулей. Причиной наличия множества услуг является то, что далеко не все хотят постоянно платить за все возможные услуги, поэтому нужные сервисы предоставляются порционно. Основные услуги таковы: секретность, целостность данных, защита от взлома методом повторения сообщений (когда жулик повторяет подслушанный разговор). Все это основано на криптографии с симметричными ключами, поскольку здесь критична высокая производительность.

Для чего нужен целый набор алгоритмов? Дело в том, что считающийся сегодня надежным алгоритм завтра может быть сломан. Если сделать IPsec независимым от конкретного алгоритма, стандарт выживет даже в случае взлома одного из алгоритмов.

Для чего нужны разные модули? Для того чтобы можно было защищать и одно TCP-соединение, и весь трафик между парой хостов, и весь трафик между парой защищенных маршрутизаторов и т. д.

Несколько удивительно, что IPsec ориентирован на соединение, несмотря на его присутствие на уровне IP. На самом деле, это не так странно, как кажется. Ведь безопасность можно обеспечить только созданием ключа и использованием его в течение какого-то времени. А это, по сути дела, разновидность соединения. К тому же все соединения погашают расходы на их установление за счет передачи большого количества пакетов. «Соединение» в контексте IPsec называется **защищающей связью (Security Aconnection — SA)**. Защищающая связь — это симплексное соединение между двумя конечными точками, с которым связан специальный идентификатор защиты. Если требуется передача защищенных данных в обоих направлениях, понадобятся две защищающие связи. Идентификаторы защиты передаются в пакетах, следующих по этим надежным соединениям, и используются по прибытии защищенных пакетов для поиска ключей и другой важной информации.

Технически IPsec состоит из двух основных частей. Первая описывает два новых заголовка, которые можно добавлять к пакету для передачи идентификатора защиты, данных контроля целостности и другой информации. Вторая часть, **ISAKMP (Internet Security and Key Management Protocol — интернет-безопасность и протокол управления ключами)**, предназначена для создания ключей. ISAKMP является средой. Основным протоколом для выполнения работы является **IKE (Internet Key Exchange — обмен ключами в Интернете)**. Должна использоваться версия 2 IKE, так как более ранняя версия имела серьезные недостатки, как указали Perlman и Kaufmn, (2000).

IPsec может работать в двух режимах. В **транспортном режиме** заголовок IPsec вставляется сразу за заголовком IP. Поле *Protocol* заголовка IP изменяется таким образом, чтобы было понятно, что далее следует заголовок IPsec (перед заголовком TCP). В заголовке IPsec содержится информация, касающаяся безопасности, — в частности, идентификатор защищающей связи, новый порядковый номер и, возможно, проверка целостности поля полезной нагрузки.

В **режиме туннелирования** весь IP-пакет вместе с заголовком вставляется внутрь нового IP-пакета с совершенно новым заголовком. Этот режим хорош тогда, когда туннель заканчивается где-нибудь вне конечного пункта. В некоторых случаях концом туннеля является шлюз, обеспечивающий безопасность, например, корпоративный межсетевой экран. Он обычно используется для VPN (Virtual Private Network — виртуальная частная сеть). В этом режиме межсетевой экран вставляет и извлекает пакеты, проходящие через него в разные стороны. При такой организации машины ЛВС компании гарантированно будут обслужены по стандарту IPsec. Об этом совершенно не приходится беспокоиться: все заботы берет на себя межсетевой экран.

Еще режим туннелирования полезен, когда несколько TCP-соединений объединяются вместе и обрабатываются в виде единого шифрованного потока, поскольку в данном случае взломщик не может узнать, кто и кому передает пакеты, а также в каком количестве. А ведь иногда даже объем трафика, передаваемого одним лицом другому, является ценной информацией. Например, если во время военного кризиса трафик между Пентагоном и Белым домом резко снижается и при этом так же резко растет

трафик между Пентагоном и какой-нибудь военной базой в Колорадо, перехватчик может сделать из этого далеко идущие выводы.

Изучение структуры потока по проходящим пакетам называется **анализом трафика**. Если используется туннелирование, такой анализ становится задачей весьма сложной. Недостаток режима туннелирования заключается в том, что приходится расширять заголовок IP-пакетов, за счет чего заметно возрастает суммарный размер пакетов. В транспортном режиме размер пакетов изменяется незначительно.

Первый из новых заголовков называется **заголовком идентификации (AH — Authentication Header)**. С его помощью проверяется целостность данных и выполняется защита от взлома путем повторной передачи. Однако он не имеет никакого отношения к секретности (шифрованию данных). Применение AH в транспортном режиме показано на рис. 8.23. В стандарте IPv4 он располагается между заголовком IP (вместе со всеми необязательными полями) и заголовком TCP. В IPv6 это просто еще один дополнительный заголовок. Так он и воспринимается. Формат AH действительно очень близок к формату дополнительного заголовка IPv6. К полезной нагрузке иногда добавляют заполнение, чтобы достичь определенной длины, необходимой алгоритму идентификации, что тоже показано на рис. 8.23.

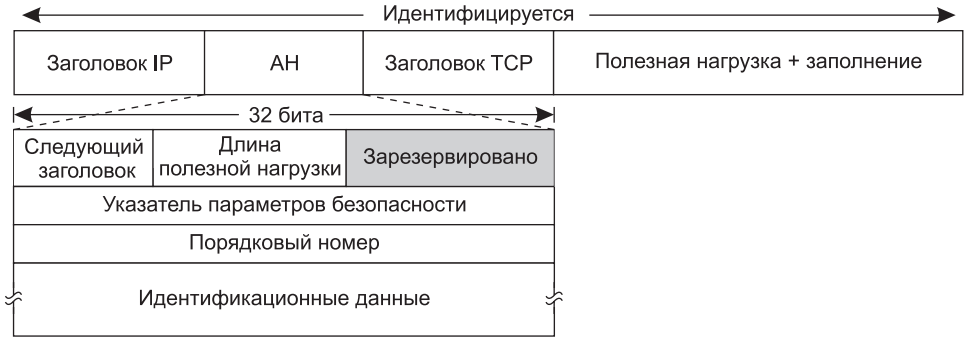


Рис. 8.23. Заголовок идентификации IPsec в транспортном режиме для IPv4

Рассмотрим заголовок AH. Поле *Следующий заголовок* хранит значение, которое в поле *Протокол* заголовка IP ранее было заменено на 51, чтобы показать, что далее следует заголовок AH. Обычно здесь встречается код для TCP (6). Поле *Длина полезной нагрузки* хранит количество 32-разрядных слов заголовка AH минус 2.

Поле *Указатель параметров безопасности* — это идентификатор соединения. Он вставляется отправителем и ссылается на конкретную запись в базе данных у получателя. В этой записи содержится общий ключ и другая информация данного соединения. Если бы этот протокол был придуман ITU, а не IETF, это поле, скорее всего, называлось бы *Номером виртуального канала*.

Поле *Порядковый номер* применяется для нумерации всех пакетов, посылаемых по защищенной связи. Все пакеты получают уникальные номера, даже если они посылаются повторно. Имеется в виду, что повторно передаваемый пакет имеет номер, отличный от номера оригинального пакета (даже если порядковый номер TCP тот же самый). Это поле служит для предотвращения взлома путем повторной

передачи. Порядковые номера никогда не повторяются. Если же окажутся использованными все 2^{32} номера, для продолжения общения устанавливается новая защищающая связь.

Наконец, поле переменной длины *Идентификационные данные* содержит цифровую подпись, вычисляемую относительно полезной нагрузки. При установке защищающей связи стороны договариваются об используемом алгоритме генерации подписей. Чаще всего здесь не применяется шифрование с открытыми ключами, так как все известные алгоритмы этого типа работают слишком медленно, а пакеты необходимо обрабатывать с очень большой скоростью. Протокол IPsec основан на шифровании с симметричными ключами, поэтому перед установкой защищающей связи отправитель и получатель должны договориться о значении общего ключа, применяемого при вычислении подписи. Один из простейших способов заключается в вычислении хэш-функции для пакета и общего ключа. Отдельно общий ключ, конечно, не передается. Подобная схема называется **HMAC (Hashed Message Authentication Code — код идентификации хэшированного сообщения)**. Вычисление этого кода выполняется гораздо быстрее, чем последовательный запуск SHA-1 и RSA.

Заголовок AH не позволяет шифровать данные. Его основная польза выявляется, когда важна проверка целостности, но не нужна секретность. Стоит отметить, что при проверке целостности при помощи AH охватываются некоторые поля заголовка IP, в частности, те из них, которые не изменяются при прохождении пакета от маршрутизатора к маршрутизатору. Поле *Время жизни*, например, меняется при каждой пересылке через маршрутизатор, поэтому его нельзя охватить при проверке целостности. Однако IP-адрес источника охватывается, тем самым предотвращается возможность его подмены взломщиком.

Альтернативой заголовку IPsec служит заголовок **ESP (Encapsulating Security Payload — инкапсулированная защищенная полезная нагрузка)**. Как показано на рис. 8.24, этот заголовок может применяться как в транспортном режиме, так и в режиме туннелировaия.

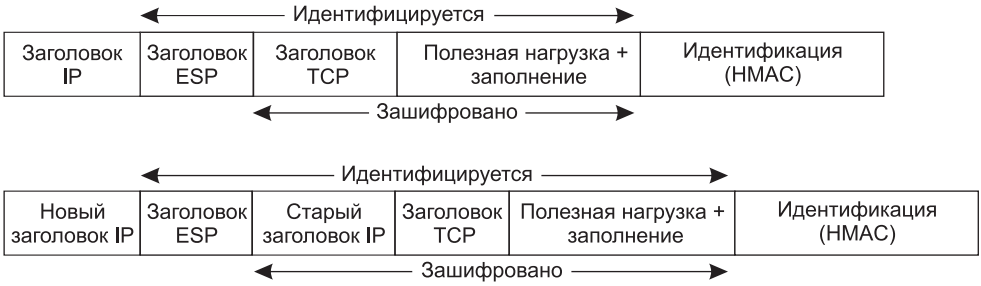


Рис. 8.24. ESP: *а* — в транспортном режиме; *б* — ESP в режиме туннелирования

Заголовок ESP состоит из двух 32-разрядных слов: *Указателя параметров безопасности* и *Порядкового номера*. Мы их уже встречали в заголовке AH. Третье слово, которое обычно следует за ними, однако технически не является частью заголовка, это *Вектор инициализации* (если только не применяется пустой алгоритм шифрования — тогда это поле опускается).

ESP, как и AH, обеспечивает проверку целостности при помощи HMAC, однако вместо того, чтобы включать хэш в заголовок, он вставляется после поля полезной нагрузки. Это видно на рис. 8.24. Такое расположение полей дает преимущество при аппаратной реализации метода. Оно заключается в том, что HMAC может подсчитываться во время передачи битов полезной нагрузки по сети и добавляться к ним в конец. Именно поэтому в Ethernet и других стандартах локальных сетей циклический контроль избыточности вставляется в концевик, а не в заголовок. При применении заголовка AH пакет приходится буферизировать и вычислять подпись, только после его можно отправлять. Это потенциально приводит к уменьшению числа пакетов, которые можно передать за единицу времени.

Казалось бы, если ESP умеет делать все то же самое, что и AH, и даже больше, причем он еще и гораздо эффективнее, тогда зачем мучиться с AH? Причины этого в основном исторические. Изначально заголовок AH обеспечивал только проверку целостности, а ESP — только секретность. Позднее ESP научили использовать для проверки целостности, но разработчики AH не хотели, чтобы он канул в Лету после всей той работы, которую они проделали. Единственный аргумент в пользу AH заключается в том, что с его помощью можно частично проверять заголовок IP, чего не умеет ESP. И все же это аргумент довольно слабый. Еще один сомнительный аргумент состоит в том, что система, поддерживающая AH, но не поддерживающая ESP, возможно, будет иметь меньше проблем при получении лицензии на экспорт, поскольку этот заголовок не имеет отношения к секретности и шифрованию. Похоже, что AH со временем все-таки исчезнет с горизонта.

8.6.2. Брандмауэры

Возможность соединять любые компьютеры друг с другом в некоторых случаях является достоинством, а в других, наоборот, недостатком. Возможность бродить по Интернету доставляет много радости домашним пользователям. Для менеджеров отделов безопасности корпораций эта возможность кажется кошмаром. Большинство компаний располагает огромными объемами конфиденциальной информации, размещенной на компьютерах, подключенных к сети, — коммерческие тайны, планы развития производства, рыночные стратегии, аналитические отчеты финансового состояния и т. д. Раскрытие этих сведений перед конкурентами может иметь ужасные последствия.

Помимо опасности утечки информации наружу, имеется опасность проникновения вредной информации, такой как вирусы, черви и прочей цифровой заразы, способной взламывать секреты, уничтожать ценные данные, на борьбу с которой уходит масса времени системных администраторов. Часто эту инфекцию заносят беззаботные сотрудники, желающие поиграть в новую модную компьютерную игру.

Таким образом, требуются специальные средства, удерживающие «доброкачественную» информацию внутри, а «вредную» — снаружи. Один из способов состоит в применении IPsec. Этот метод защищает данные при их пересылке. Однако шифрование не спасает от вирусов и хакеров, способных проникнуть в локальную сеть. Помочь защитить сети от нежелательного проникновения снаружи может установка межсетевых экранов, к рассмотрению которых мы сейчас обратимся.

Межсетевые экраны (firewalls, также называемые **брандмауэрами)** представляют собой современную реализацию средневекового принципа обеспечения безопасности. Они напоминают ров, вырытый вокруг замка. Суть конструкции заключается в том, что все входящие и выходящие из замка должны проходить по одному подъемному мосту, где полиция ввода/вывода сможет проверить их личность. Тот же принцип может быть применен и в сетях: у компании может быть несколько локальных сетей, соединенных произвольным образом, но весь внешний трафик должен проходить через электронный подъемный мост (межсетевой экран), как показано на рис. 8.25. Не существует никакого другого пути.

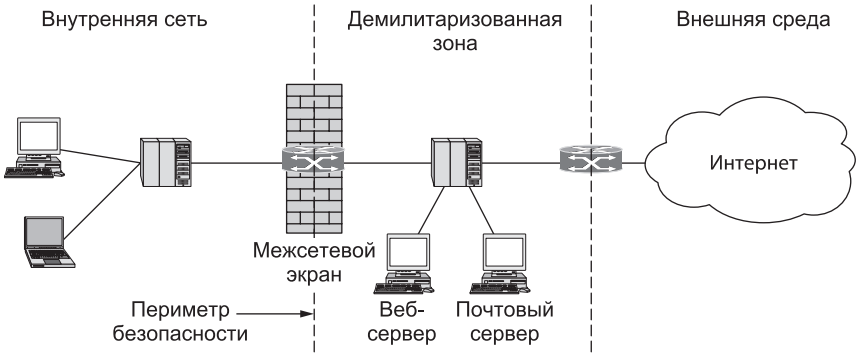


Рис. 8.25. Межсетевой экран, защищающий внутреннюю сеть

Брандмауэр работает как **пакетный фильтр (packet filter)**. Он исследует каждый входящий и исходящий пакеты. Пакеты, удовлетворяющие определенным критериям, пропускаются сквозь фильтр. Не сумевшие пройти проверку пакеты удаляются.

Критерии фильтрации обычно устанавливаются правилами или таблицами, где перечислены допустимые и блокируемые отправители и получатели, а также правила, описывающие действия над исходящими и входящими пакетами.

В общем случае настроек TCP/IP информация о получателе или отправителе состоит из IP-адреса и номера порта. Номера портов определяют требуемую службу. Например, порт 25 используется для почты, порт 80 — для HTTP. Некоторые порты просто могут быть заблокированы. Например, компания может заблокировать все входящие пакеты для всех IP-адресов, в сочетании TCP и совместимых с портом 79. Он раньше широко использовался службой Finger для поиска электронных адресов, но мало используется в настоящее время.

Остальные порты блокируются не так просто. Сложность состоит в том, что сетевые администраторы мечтают о безопасности, но не могут блокировать коммуникацию с внешним миром. Второй вариант был бы проще и лучше для безопасности, но поток жалоб от пользователей шел бы без перерыва. Здесь помогают такие решения, как **DMZ (DeMilitarized Zone — демилитаризированная зона)**, которая показана на рис. 8.25. DMZ — это часть сети компании, которая не угрожает ее безопасности. Сюда включается все. Если разместить веб-сервер в DMZ, то компьютеры через Интернет могут войти с ним в контакт, чтобы зайти на сайт компании. Межсетевой экран может быть настроен так, чтобы блокировать входящий трафик TCP к порту 80, так что

компьютеры, подключенные к Интернету, не смогут использовать этот порт, для того чтобы атаковать компьютеры во внутренней сети. Для того чтобы веб-сервером можно было управлять, межсетевой экран может разрешать соединения между машинами во внутренней сети и веб-сервером. Межсетевые экраны стали все более сложными в процессе гонки вооружений с преступниками. Изначально они применяли набор правил независимо для каждого пакета, затем оказалось сложно писать правила, которые бы предоставляли необходимую функциональность, но блокировали весь нежелательный трафик. **Межсетевые экраны с контекстной фильтрацией (statefull firewalls)** привязывали пакеты к соединениям и заголовкам TCP/IP, чтобы отслеживать соединения. Благодаря этому создавались такие правила, что, например, внешний веб-сервер мог посылать пакеты внутреннему хосту, но только если внутренний хост сначала устанавливал соединение с внешним сервером. Такое правило было невозможно при простой пакетной фильтрации, когда пропускаются или не пропускаются все пакеты с внешнего сервера.

Следующий уровень усложнения для учета контекста — это **шлюз прикладного уровня (application-level gateway)**. Во время этой обработки межсетевой экран просматривает пакеты даже за пределами заголовка TCP, чтобы проверить, что делает приложение. При этом возможно разделить HTTP-трафик, используемый для того, чтобы зайти на сайты в сети Интернет и HTTP-трафик, используемый для обмена файлами. Администратор может написать такие правила, чтобы сотрудники компании не обменивались файлами, но позволить заходить на сайты, которые необходимы для бизнеса. С помощью этих методов исходящий трафик может исследоваться примерно так же, как входящий трафик, например, для того, чтобы важные документы не уходили по электронной почте за пределы компании.

Как ясно из обсуждения выше, межсетевые экраны нарушают стандартные иерархические расположения слоев протоколов. Они являются устройствами сетевого уровня, но также контролируют транспортный уровень и уровень приложений, чтобы производить фильтрацию. Это делает их довольно хрупкими. Например, межсетевые экраны ориентируются на стандартную нумерацию портов, чтобы определить, какой трафик находится в пакете. Стандартные порты часто используются, но все же не всеми компьютерами и не всеми приложениями. Некоторые пиринговые приложения выбирают динамические порты, чтобы их было трудно вычислить (и заблокировать). Шифрование с помощью IPSEC или других схем скрывает информацию более высокого уровня от межсетевого экрана. Наконец, межсетевой экран не может быстро сказать компьютерам, которые связываются с помощью него, почему соединение не удалось. Он должен просто притвориться разрезанным кабелем. По всем этим причинам сетевые пуристы считают межсетевые экраны недостатком архитектуры Интернета. Тем не менее Интернет может оказаться опасным местом. Межсетевые экраны помогают решать эту проблему, так что они, скорее всего, будут существовать и дальше.

Даже в случае идеально настроенного межсетевого экрана остается множество проблем, связанных с безопасностью. Например, если входящие пакеты пропускаются только со стороны конкретных сетей (например, со стороны ЛВС дочерней фирмы компании), взломщик, находящийся вне зоны действия брандмауэра, может просто фальсифицировать адрес отправителя и тем самым преодолеть барьер. Если же нечестный сотрудник компании решит переслать секретную документацию, он может

зашифровать ее или вообще сфотографировать, и тогда эти данные смогут проникнуть через любые почтовые анализаторы. Мы даже не обсуждаем тот факт, что хотя 75 % атак происходит вне зоны действия межсетевого экрана, атаки, которые происходят в зоне действия межсетевого экрана (например, производятся недовольными сотрудниками), являются самыми опасными (Verizon, 2009).

Еще один недостаток межсетевых экранов состоит в том, что они обеспечивают единый периметр или единую защиту. Если эта защита ломается — все пропало. По этой причине межсетевые экраны используются для многоуровневой защиты. Например, межсетевой экран может охранять вход во внутреннюю сеть, и на каждом компьютере также может быть свой межсетевой экран. Читатели, считающие, что достаточно одного контрольно-пропускного пункта, видимо, давно не летали международными авиалиниями.

К тому же существует целый класс атак, с которыми не способны справиться никакие межсетевые экраны. Идея, лежащая в основе межсетевых экранов, заключается в том, чтобы не давать взломщикам проникнуть в систему, а секретным данным — уходить наружу. К сожалению, в мире есть много людей, которые не могут найти себе лучшего занятия, нежели препятствовать работоспособности сайтов. Они отправляют вполне легитимные сообщения до тех пор, пока сайт не перестанет функционировать из-за чрезмерной нагрузки. Например, такое хулиганство может заключаться в рассылке пакетов *SYN* для установки соединений. Сайт выделит часть таблицы под это соединение и пошлет в ответ пакеты *SYN + ACK*. Если взломщик не ответит, табличная запись будет продолжать оставаться зарезервированной в течение нескольких секунд до наступления тайм-аута. Если одновременно посылаются тысячи запросов на соединение, никакие запросы от честных граждан просто не пробьются к серверу, так как все ячейки таблицы окажутся заняты. Атаки, целью которых является нарушение деятельности объекта, а не получение секретных данных, называются **атаками типа DoS (Denial of Service — отказ в обслуживании (запроса))**. Обычно адрес отправителя в пакетах с запросами фальсифицирован, поэтому найти вандала не так просто. DoS атаки против серьезных сайтов являются обычным делом.

Существует и более жестокий вариант такой атаки. Если сетевому хулигану уже удалось взломать несколько сотен компьютеров, расположенных по всему миру, он может приказать им всем забивать запросами один и тот же сервер. Тем самым не только повышается «убойная сила», но и уменьшаются шансы на обнаружение негодяя, так как пакеты приходят с самых разных компьютеров, ничем плохим себя ранее не зарекомендовавших. Этот тип атаки носит название **DDoS (Distributed Denial of Service — распределенный отказ в обслуживании)**. С этой напастью бороться трудно. Даже если атакуемая машина сможет быстро распознать поддельный запрос, на его обработку и отвержение потребуется некоторое время, в течение которого придут другие запросы, и в итоге центральный процессор будет постоянно занят их обработкой.

8.6.3. Виртуальные частные сети

Многие компании владеют множеством подразделений, расположенных в разных городах, иногда даже в разных странах. До появления общедоступных сетей передачи данных обычным делом было арендовать выделенную телефонную линию для

организации связи между некоторыми или всеми парами подразделений. В некоторых компаниях такой подход применяется до сих пор. Сеть, состоящая из компьютеров, принадлежащих компании, и выделенных телефонных линий, называется **частной сетью**.

Частные сети работают хорошо и обладают высокой защищенностью. Если бы были доступны только выделенные линии, то отсутствовала бы проблема утечки трафика, и взломщикам пришлось бы физически подключаться к линиям, чтобы перехватить данные, а это не так просто. Беда в том, что стоимость аренды одного выделенного канала T1 между двумя точками составляет тысячи долларов (в месяц!), а аренда линии T3 во много раз дороже. Когда появились общедоступные сети передачи данных, а позднее и Интернет, у многих компаний возникло естественное желание воспользоваться ими для передачи данных (а может, и голоса). При этом, правда, не хотелось терять свойства защищенности, присущие частной сети.

Это соображение вскоре привело к изобретению **виртуальных частных сетей (VPN — Virtual Private Networks)**, которые являются оверлейными сетями, работающими поверх обычных общедоступных сетей, но обладающими свойствами частных сетей. Они называются «виртуальными», потому что такие сети — это почти иллюзия; аналогичным образом, виртуальные каналы — это не реальные каналы, а виртуальная память — это не реальная память.

Все более популярным становится организация VPN прямо в Интернете. В общем случае, каждый офис оборудуется межсетевым экраном, и создаются туннели через Интернет между всеми парами офисов, как показано на рис. 8.26, *а*. Еще одно преимущество использования Интернета для связи состоит в том, что туннели могут быть созданы по требованию и включать, например, компьютер работника, который находится дома или путешествует, пока он имеет соединение с Интернетом. Гибкость выше, чем у выделенных линий, но с точки зрения компьютеров с VPN данная топология выглядит как частная сеть, что показано на рис. 8.26, *б*. При запуске системы каждая пара межсетевых экранов должна договориться о параметрах защищающей связи, таких как набор услуг, режимов, алгоритмов и ключей. Если используется IPsec в режиме туннелирования, можно собрать весь трафик между любыми двумя парами офисов в один надежный поток и установить защищающую связь, обеспечив тем самым контроль целостности, секретности и даже определенный иммунитет против анализа трафика. Во многие межсетевые экраны встроен специальный инструментарий для работы с виртуальными частными сетями. Можно построить систему и на некоторых обычных маршрутизаторах, но поскольку межсетевые экраны — это почти неотъемлемая часть систем сетевой безопасности, вполне естественно начинать и заканчивать туннели именно на межсетевых экранах, проводя четкую границу между компанией и Интернетом. Таким образом, естественная и наиболее распространенная комбинация — это межсетевые экраны, виртуальные частные сети и IPsec с ESP в режиме туннелирования.

После установки защищающей связи начинается передача данных. С точки зрения маршрутизатора, работающего в Интернете, пакет, проходящий по туннелю VPN, — это самый обычный пакет. Единственное, что его отличает от остальных, это наличие заголовка IPsec после заголовка IP. Но поскольку дополнительные заголовки на процесс пересылки никак не влияют, маршрутизаторы не сильно беспокоит заголовок IPsec.

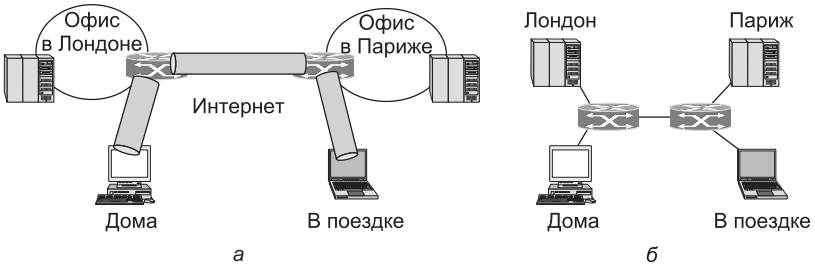


Рис. 8.26. *а*. Виртуальная частная сеть. *б*. Топология, видимая изнутри сети

Другая возможность, приобретающая популярность, — это реализация VPN с помощью интернет-провайдера. За счет использования MPLS (как обсуждалось в главе 5), пути для трафика VPN между офисами компании могут быть установлены через сеть интернет-провайдера. Эти пути отделяют трафик VPN от другого интернет-трафика и могут гарантировать определенную пропускную способность или другое качество обслуживания.

Основное преимущество VPN состоит в том, что она совершенно прозрачна для всего пользовательского ПО. Установкой и управлением защищающих связей занимаются межсетевые экраны. Единственный человек, которому есть дело до настройки сети, — это системный администратор, который обязан сконфигурировать и поддерживать сетевые шлюзы, или администратор интернет-провайдера, который поддерживает пути MPLS. Для всех остальных виртуальная частная сеть мало чем отличается от частной сети на основе выделенной линии. Более подробно про VPN написано в (Lewis, 2006).

8.6.4. Безопасность в беспроводных сетях

Оказывается, удивительно просто создать систему, которая логически полностью надежна, то есть состоит из VPN и межсетевых экранов, и при этом на практике протекает, как решето. Такая ситуация может возникнуть, если в сети есть беспроводные машины, передающие данные с помощью радиосигнала, проходящего прямо над межсетевым экраном в обе стороны. Радиус действия сетей типа 802.11 может составлять несколько сотен метров, поэтому шпион, желающий перехватить информацию, может просто приехать на автостоянку перед зданием фирмы, оставить в машине ноутбук с приемопередатчиком 802.11, записывающим все, что слышно в эфире, и пойти гулять по городу. К вечеру на жестком диске он обнаружит массу ценной информации. Теоретически такого происходить не должно. Правда, теоретически ограбления банков тоже не должны происходить.

За многие проблемы безопасности стоит сказать спасибо производителям беспроводных базовых станций (точек доступа), пытающихся сделать свою продукцию дружественной по отношению к пользователю. Обычно, если пользователь вынимает свое устройство из сумки и вставляет в розетку, оно сразу начинает работать, и практически всегда все окружающие в зоне действия радиопередатчика смогут услышать любые секреты, о которых он проболтается. Если же затем это устройство подключить

к Ethernet, весь трафик, проходящий по локальной сети, может быть перехвачен ноутбуком, стоящим в припаркованной неподалеку машине. Беспроводная связь — это мечта шпиона, ставшая реальностью: информация сама идет в руки, только успевай ее ловить. Очевидно, что вопрос безопасности в беспроводных сетях стоит куда острее, чем в проводных. В этом разделе мы рассмотрим некоторые методы, позволяющие в какой-то мере обезопасить системы такого рода. Дополнительную информацию можно найти в (Nichols и Lekkas, 2002).

Безопасность в сетях 802.11

Часть стандарта 802.11, называемая 802.11i, описывает протокол безопасности канального уровня, не позволяющий беспроводному узлу читать или другим образом вмешиваться в сообщения, посланные другой парой беспроводных узлов. Он проходит под торговым названием **WPA2 (WiFi Protected Access 2 — защищенный доступ к WiFi, версия 2)**. Простой WPA — это промежуточная схема, которая реализует подмножество стандарта 802.11i. Ей следует предпочитать WPA2. Мы коротко опишем 802.11i, но сначала отметим, что он заменяет **WEP (Wired Equivalent Privacy — секретность, эквивалентная проводным сетям)**, первое поколение протоколов безопасности 802.11. WEP был разработан комитетом по сетевым стандартам, что отличается от того способа, которым, предположим, NIST выбрал AES. Результаты были разрушительны. Так что же было не так? Как оказалось, практически все с точки зрения безопасности. Например, WEP зашифровывал конфиденциальные данные с помощью XOR с выходом поточного шифра. К сожалению, слабая работа с ключом означала, что эти выходные данные использовались несколько раз. Таким образом, их было просто расшифровать. В качестве еще одного примера можно сказать, что проверка целостности основывалась на 32-битном CRC. Это эффективный код для определения ошибок передачи, но он не является криптографически сильным механизмом для определения злоумышленников.

Эти и другие недостатки разработки сделали WEP очень легким для компрометации. Первая практическая демонстрация того, что WEP был сломан, состоялась, когда Адам Стабблфилд был молодым сотрудником компании AT&T (Stubblefield и др., 2002). Он мог реализовать и проверить атаку, описанную Fluhrer и др. (2001) за одну неделю, большая часть которой была потрачена на убеждение менеджеров купить ему WiFi карту для использования в эксперименте. Программы, взламывающие пароли WEP за одну минуту, теперь свободно доступны, поэтому использование WEP не поощряется. Он запрещает обычный доступ, но не обеспечивает никакой реальной формы безопасности. Группа 802.11i спешно собралась, когда стало ясно, что WEP был сломан. В результате был создан формальный стандарт от июня 2004 года.

Теперь мы опишем 802.11i, который обеспечивает реальную безопасность, если правильно настроен и правильно используется. Существует два обычных сценария, в которых используется WPA2. Первый — это корпоративное использование, когда у компании есть отдельный сервер для аутентификации, хранящий имена пользователей и пароли, которые используются, чтобы определить, имеет ли право клиент получить доступ к сети. В этом случае клиенты используют стандартные протоколы для того, чтобы аутентифицировать себя и войти в сеть. Основные стандарты — это **802.1X**,

где точка доступа позволяет клиенту вести диалог с сервером аутентификации и наблюдать результат, и **EAP (Extendable Authentication Protocol — расширенный протокол аутентификации)** (RFC 3748), который описывает, как взаимодействуют клиент и аутентификационный сервер. EAP является средой, а другие стандарты определяют сообщения протокола. Мы не будем подробно рассказывать о деталях данного обмена, в кратком обзоре они не имеют значения.

Второй сценарий — домашнее использование в условиях, где нет аутентификационного сервера. Вместо него есть единый общий пароль, который используется клиентами для доступа в беспроводную сеть. Эта система менее сложная, чем в случае с аутентификационным сервером, именно поэтому она используется в домашних условиях и в маленьких фирмах, но она и менее надежная. Основная разница состоит в том, что при наличии аутентификационного сервера каждый клиент получает ключ для шифрования трафика, неизвестный другим клиентам. При едином общем пароле для каждого клиента создается свой ключ, но у всех клиентов одинаковый пароль, и они могут узнать ключи друг друга, если захотят.

Ключи, использующиеся для шифрования трафика, рассчитываются как часть аутентификационного опознавания. Опознование происходит сразу после того, как клиент связывается с беспроводной сетью и проходит аутентификацию на аутентификационном сервере, если он есть. В начале опознавания у клиента есть либо общий пароль сети, либо пароль для аутентификационного сервера. Пароль используется для получения основного ключа. Но основной ключ не используется прямым образом для шифрования пакетов. Существует стандартная криптографическая практика создавать новый ключ для каждого периода использования, менять ключ для разных сеансов и держать основной ключ в секретности. При опознавании рассчитывается именно ключ сеанса.

Ключ сеанса рассчитывается при четырехпакетном опознавании, показанном на рис. 8.27. Во-первых, **AP (Access Point — точка доступа)** посылает случайный номер для идентификации. Случайные номера, использующиеся только один раз в протоколах безопасности, таких как этот, называются **nonces (нонсы — временные значения)**, это сокращение выражения «number used once» — «номер, использующийся только один раз». Клиент также выбирает свой собственный временный номер. Он использует временный номер, адрес MAC и адрес AP, а также основной ключ, чтобы вычислить ключ сеанса, Ks. Ключ сеанса разбивается на отрезки, каждый из которых используется для различных целей, но мы опустили эту деталь. Теперь у клиента есть ключи сеанса, а у AP нет. Клиент посылает свой временный номер AP, AP производит тот же самый расчет, чтобы получить ключ сеанса. Временные номера могут быть посланы открытым способом, так как на основании них невозможно рассчитать ключи без дополнительной, секретной информации. Сообщение от клиента защищено проверкой целостности, которая называется **MIC (Message Integrity Check — проверка целостности сообщения)**, данная проверка основывается на ключе сеанса. AP может установить, что MIC верный и сообщение действительно пришло от клиента, после того как она рассчитает сессионные ключи. MIC это просто другое название кода аутентификации сообщения, как в HMAC. Название MIC часто используется вместо протоколов безопасности, чтобы не путать с адресами **MAC (Medium Access Contol — контроль доступа к среде)**.

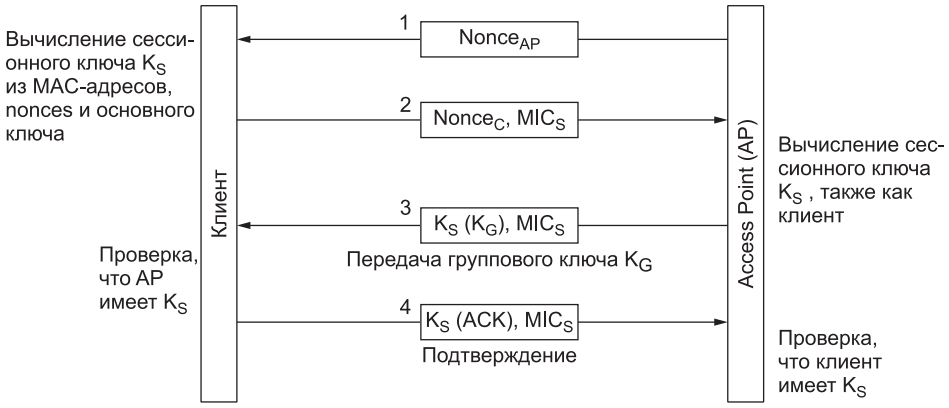


Рис. 8.27. Четырехпакетное опознавание и генерация сеансового ключа в 802.11i

В последнем из двух сообщений AP выдает клиенту общий ключ K_G, и клиент подтверждает подлинность сообщения. Получение данных сообщений позволяет клиенту удостовериться, что у AP есть верные ключи сеанса, и наоборот. Общий ключ используется для передачи трафика по 802.11 LAN. Так как в результате опознавания оказывается, что у каждого клиента есть свои ключи шифрования, ни один из этих ключей не может быть использован AP для передачи пакетов всем клиентам беспроводной сети; надо будет послать отдельную копию со своим ключом каждому клиенту. Вместо этого используется общий ключ, так что широковещательный трафик может быть послан один раз и получен всеми клиентами. Этот ключ должен обновляться по мере того, как клиенты уходят из сети и присоединяются к ней.

Наконец, мы подходим к той части, где ключи используются для обеспечения безопасности. Два протокола могут быть использованы в 802.11i для обеспечения конфиденциальности, цельности и аутентификации. Как и WPA, один из протоколов, **TKIP (Temporary Key Integrity Protocol — временный протокол целостности ключа)** был временным решением. Он был разработан для того, чтобы увеличить безопасность старых и медленных карт 802.11, так что безопасность у него, по крайней мере, выше, чем у WEP. Однако его сейчас тоже можно взломать, так что лучше использовать другой рекомендованный протокол — CCMP. Что означает **CCMP**? Это сокращение от Counter mode with Cipher block chaining message authentification code protocol — режим счетчика с протоколом аутентификации в режиме сцепления обратной связи. Мы будем называть его CCMP. Вы можете называть его как хотите.

CCMP работает довольно прямым путем. Он использует шифрование AES с помощью ключа и блоков размером 128 бит. Ключ выводится из ключа сеанса. Чтобы обеспечить конфиденциальность, сообщения зашифровываются с помощью AES в режиме счетчика. Мы обсуждали режимы шифров в разделе 8.2.3. Эти режимы предотвращают шифрование каждый раз одинаковых сообщений в одинаковые наборы бит. Режим счетчика подмешивает счетчик в процесс шифрования сообщения. Чтобы обеспечить целостность, сообщение, включая поля заголовков, кодируется шифром в режиме обратной связи, и последний блок из 128 бит сохраняется как MIC. Затем

и сообщение (закодированное в режиме счетчика) и MIC высылаются. И клиент, и AP могут осуществлять данную кодировку или проверить ее при получении беспроводного пакета. Для широковещательных или групповых сообщений данная процедура применяется с использованием группового ключа.

Безопасность в системах Bluetooth

Радиус действия систем Bluetooth значительно меньше, чем сетей 802.11, поэтому взломщику не удастся произвести атаку, оставив ноутбук в припаркованной рядом со зданием машине, однако вопрос безопасности важен и тут. Например, предположим, что компьютер Алисы оборудован беспроводной клавиатурой стандарта Bluetooth. Если не установить систему защиты, то Труди, находясь за стенкой, в соседнем офисе, сможет без труда прочесть все, что набирает Алиса, включая исходящую почту. Можно захватить все, что передается на беспроводной принтер, если расположиться неподалеку от него (включая входящую почту и конфиденциальные бумаги). К счастью, в Bluetooth есть рабочая схема защиты, нарушающая планы всевозможных личностей типа Труди. Ниже мы опишем основные черты этой схемы.

Система защиты Bluetooth (версии 2.1 и более поздние) может работать в четырех режимах, начиная от полного бездействия и заканчивая тотальным шифрованием данных и контролем целостности. Как и в случае с 802.11, если система защиты отключена (по умолчанию для старых устройств это именно так), о какой-либо безопасности говорить не приходится. Большинство пользователей не включают защиту до тех пор, пока не грянет гром. Можно привести сельскохозяйственный пример такого подхода: ворота конюшни закрывают только после исчезновения лошади.

Bluetooth обеспечивает безопасность на нескольких уровнях. На физическом уровне для этого применяются скачкообразные изменения частот, но поскольку любое устройство, появляющееся в микросети, должно узнать последовательность скачков частоты, эта последовательность, очевидно, не является секретной. Настоящая защита информации начинает проявляться тогда, когда вновь прибывшее подчиненное устройство пытается запросить канал для связи с управляющим устройством. До появления Bluetooth 2.1 предполагалось, что оба устройства совместно использовали предварительно установленный закрытый ключ. В некоторых случаях он прошивается в обоих устройствах (например, в гарнитуре и мобильном телефоне, продающихся вместе). В других случаях в одном из устройств (например, в гарнитуре) ключ прошит, а в сопряженное устройство (например, мобильный телефон) пользователь должен ввести ключ вручную в виде десятичного числа. Общие ключи такого типа называются **отмычками (passkeys)**. К сожалению, отмычки очень часто жестко кодируются как «1234» или другие предсказуемые значения, и в любом случае это будет четырехзначное число, имеющее только 104 вариантов. При Bluetooth 2.1 устройства выбирают код из шестизначного диапазона, что делает отмычку менее предсказуемой, но все же недостаточно надежной.

Перед установкой канала подчиненное и управляющее устройства должны выяснить, владеют ли они отмычками. В случае положительного ответа им необходимо договориться о том, каким будет канал: шифрованным, с контролем целостности или

и таким, и таким. Затем выбирается ключ сеанса длиной 128 бит, некоторые биты которого могут быть сделаны общедоступными. Такое послабление сделано в целях соответствия системы ограничениям, введенным правительствами разных стран и запрещающим экспорт или использование ключей, длина которых больше той, что способно взломать правительство.

Шифрование выполняется с применением потокового шифра E0, контроль целостности — с применением **SAFER+**. И тот и другой представляют собой традиционные блочные шифры с симметричными ключами. SAFER+ пытались использовать в AES, однако очень быстро отказались от этой мысли, так как он работал гораздо медленнее других. Работа над Bluetooth завершилась еще до того, как был выбран шифр AES; в противном случае, вероятно, использовался бы алгоритм Rijndael.

Процесс шифрования с использованием потокового (группового) шифра показан на рис. 8.12. На нем видно, что открытый текст суммируется по модулю 2 с ключевым потоком. В результате получается шифрованный текст. К сожалению, алгоритм E0, как и RC4, чрезвычайно слаб (Jacobsson и Wetzel, 2001). Несмотря на то что на момент написания книги он еще не взломан, его сходство с шифром A5/1, чей провал угрожает безопасности всего GSM-трафика, наводит на грустные мысли (Biryukov и др., 2000). Многим (в том числе и авторам) кажется удивительным тот факт, что в игре «кошки–мышки» между криптографами и криптоаналитиками так часто побеждают последние.

Еще одна проблема безопасности, связанная с Bluetooth, состоит в том, что система идентифицирует только устройства, а не пользователей. Это приводит к тому, что вор, укравший устройство Bluetooth, получит доступ к финансовым и другим счетам жертвы. Тем не менее система безопасности в Bluetooth реализована и на верхних уровнях, поэтому даже в случае взлома защиты на канальном уровне некоторые шансы еще остаются, особенно если приложение для выполнения транзакции требует ввода PIN-кода вручную с помощью какой-нибудь разновидности клавиатуры.

8.7. Протоколы аутентификации

Аутентификация (или идентификация) — это метод, с помощью которого процесс удостоверяется в том, что его собеседник является именно тем, за кого он себя выдает. Проверка подлинности удаленного процесса при активных злонамеренных попытках проникновения представляет собой удивительно сложную задачу и требует сложных протоколов, основанных на криптографии. В данном разделе мы познакомимся с несколькими протоколами аутентификации, применяемыми в незащищенных компьютерных сетях.

Следует отметить, что понятия аутентификации и авторизации иногда путают. Аутентификация связана с вопросом подлинности вашего собеседника. Авторизация имеет дело с разрешениями. Например, клиентский процесс обращается к файловому серверу и говорит: «Я процесс Скотта, и я хочу удалить файл *cookbook.old*». Файл-сервер должен решить следующие два вопроса.

1. Действительно ли это процесс Скотта (аутентификация)?
2. Имеет ли Скотт право удалять файл *cookbook.old* (авторизация)?

Только после того, как на оба вопроса будут получены недвусмысленные утвердительные ответы, может быть выполнено запрашиваемое действие. Ключевым является первый вопрос. После того как сервер узнает, с кем он разговаривает, для проверки прав доступа потребуется лишь просмотреть содержимое локальных таблиц или баз данных. По этой причине в данном разделе мы уделим особое внимание вопросу аутентификации.

Общая схема, используемая практически всеми протоколами аутентификации, состоит из следующих действий. Алиса желает установить защищенное соединение с Бобом или считающимся надежным **Центром распространения ключей (KDC — Key Distribution Center)**. Затем в разных направлениях посылаются еще несколько сообщений. По мере их передачи хулиганка по имени Труди может перехватить, изменить и снова воспроизвести эти сообщения, чтобы обмануть Алису и Боба или просто сорвать сделку.

Тем не менее, когда протокол завершит свою работу, Алиса должна быть уверена, что разговаривает с Бобом, а Боб — что разговаривает с Алисой. Кроме того, в большинстве протоколов собеседники также установят секретный **ключ сеанса (session key)**, которым будут пользоваться для последующего обмена информацией. На практике весь обмен данными шифруется с помощью одного из алгоритмов с секретным ключом (AES или тройной DES), так как их производительность намного выше производительности алгоритмов с открытым ключом. Тем не менее алгоритмы с открытым ключом широко применяются в протоколах аутентификации и для определения ключа сеанса.

Цель использования нового, случайно выбираемого ключа сеанса для каждого нового соединения состоит в минимизации трафика, посылаемого с использованием закрытых и открытых ключей пользователя, уменьшении количества шифрованного текста, который может достаться злоумышленнику, а также минимизации вреда, причиняемого в случае, если процесс даст сбой и дамп ядра попадет в чужие руки. Поэтому после установления соединения в процессе должен храниться только один временный ключ сеанса. Все постоянные ключи должны быть тщательно стерты.

8.7.1. Аутентификация, основанная на общем секретном ключе

В нашем первом протоколе аутентификации мы предположим, что у Алисы и Боба есть общий секретный ключ K_{AB}. Об этом секретном ключе можно договориться при личной встрече или по телефону, но в любом случае не по (незащищенной) сети.

В основе этого протокола лежит принцип, применяемый во многих протоколах аутентификации: одна сторона посылает другой случайное число, которое другая сторона преобразует особым образом и возвращает результат. Такие протоколы называются протоколами типа **оклик-отзыв (challenge-response)**. В этом и последующих протоколах аутентификации будут использоваться следующие условные обозначения:

A и B — Алиса и Боб;

R_i — оклик, где индекс означает его отправителя;

K_i — ключи, где индекс означает владельца ключа;

K_S — ключ сеанса.

Последовательность сообщений нашего первого протокола аутентификации с общим ключом показана на рис. 8.28. В первом сообщении Алиса посылает свое удостоверение личности, A, Бобу тем способом, который ему понятен. Боб, конечно, не знает, пришло ли это сообщение от Алисы или от злоумышленника, поэтому он выбирает большое случайное число R_B и посылает его в качестве оклика «Алисе» открытым текстом (сообщение 2). Затем Алиса шифрует это сообщение секретным ключом, общим для нее и Боба, и отправляет шифрованный текст $K_{AB}(R_B)$ в сообщении 3. Когда Боб видит это сообщение, он понимает, что оно пришло от Алисы, так как злоумышленник не должен знать ключа K_{AB}, и поэтому он не смог бы сформировать такое сообщение. Более того, поскольку оклик R_B выбирался случайно в большом пространстве чисел (например, 128-битных случайных чисел), очень маловероятно, чтобы злоумышленник мог уже видеть этот оклик и ответ на него в предыдущих сеансах.

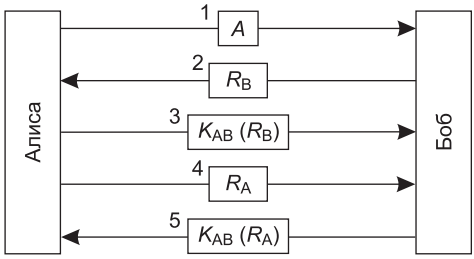


Рис. 8.28. Двусторонняя аутентификация при помощи протокола оклик-отзыв

К этому моменту Боб уверен, что говорит с Алисой, однако Алиса еще пока не уверена ни в чем. Злоумышленник мог перехватить сообщение 1 и послать обратно оклик R_B. Возможно, Боба уже нет в живых. Далее протокол работает симметрично: Алиса посылает оклик, а Боб отвечает на него. Теперь уже обе стороны уверены, что говорят именно с тем, с кем собирались. После этого они могут установить временный ключ сеанса K_S, который можно переслать друг другу, закодировав его все тем же общим ключом K_{AB}.

Количество сообщений в этом протоколе можно сократить, объединив в каждом сообщении ответ на предыдущее сообщение с новым окликом, как показано на рис. 8.29. Здесь Алиса сама в первом же сообщении посылает Бобу оклик. Отвечая на него, Боб помещает в то же сообщение свой оклик. Таким образом, вместо пяти сообщений понадобилось всего три.

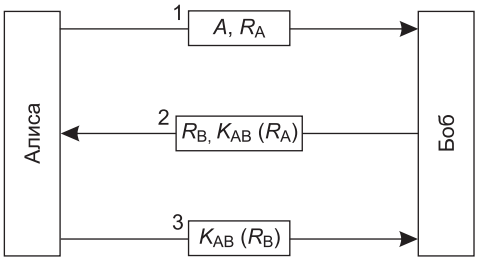


Рис. 8.29. Укороченный двусторонний протокол аутентификации

Лучше ли этот протокол, чем предыдущий? С одной стороны, да: он короче. Но, к сожалению, пользоваться таким протоколом не рекомендуется. При некоторых обстоятельствах злоумышленник может атаковать этот протокол способом, известным под названием **зеркальная атака (reflection attack)**. В частности, Труди может взломать его, если ей будет позволено одновременно открыть несколько сеансов связи с Бобом. Такое вполне возможно, если, скажем, Боб — это банк, позволяющий устанавливать несколько одновременных соединений с банкоматами.

Схема зеркальной атаки показана на рис. 8.30. Она начинается с того, что Труди, объявляя себя Алисой, посылает оклик R_T. Боб, как обычно, отвечает своим собственным окликом R_B. Теперь, казалось бы, Труди в тупике. Что ей делать? Она ведь не знает $K_{AB}(R_B)$.

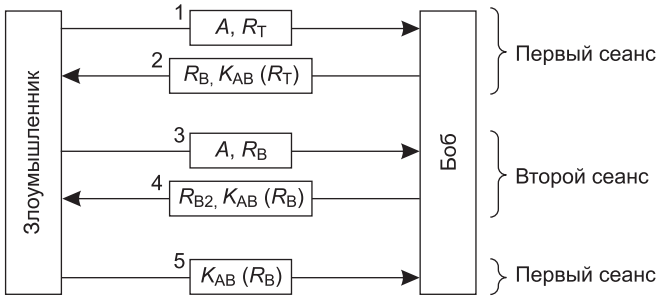


Рис. 8.30. Зеркальная атака

Злоумышленник может открыть второй сеанс сообщением 3 и подать в качестве оклика Бобу оклик самого Боба, взятый из второго сообщения. Боб спокойно шифрует его и посылает обратно $K_{AB}(R_B)$ в сообщении 4. Теперь у Труди есть необходимая информация, поэтому она завершает первый сеанс и прерывает второй. Боб теперь уверен, что злоумышленник — это Алиса, поэтому предоставляет Труди доступ к банковским счетам Алисы и позволяет перевести деньги с ее текущего счета на секретный счет в швейцарском банке без каких-либо колебаний.

Мораль этой истории такова:

Разработать корректный протокол аутентификации сложнее, чем это может показаться.

Приведем четыре общих правила, которые часто оказываются полезными и помогают избежать распространенных ошибок.

1. Инициатор сеанса должен подтверждать свою личность прежде, чем это сделает отвечающая сторона. Это помешает злоумышленнику получить ценную для него информацию, прежде чем он подтвердит свою личность.
2. Следует использовать два раздельных общих секретных ключа: один для инициатора сеанса, а другой для отвечающего, K_{AB} и K'_{AB}.
3. Инициатор и отвечающий должны выбирать оклики из различных непересекающихся наборов. Например, инициатор должен пользоваться четными номерами, а отвечающий — нечетными.

4. Протокол должен уметь противостоять атакам, при которых запускается второй параллельный сеанс, информация для которого извлекается при помощи первого сеанса (или наоборот).

Если нарушается хотя бы одно из этих правил, протокол оказывается уязвимым. В приведенном примере были нарушены все четыре правила, что привело к разрушительным последствиям.

Вернемся к ситуации, показанной на рис. 8.28. Можно ли с уверенностью сказать, что этот протокол не подвержен зеркальным атакам? Может быть. Ситуация с этим очень шаткая. Труди удалось справиться с нашим протоколом, используя зеркальную атаку, потому что он позволял запустить параллельный сеанс с Бобом и ввести его в заблуждение, передав ему его собственный оклик. А что произойдет, если вместо живой Алисы, сидящей за компьютером, стоит обычный компьютер общего назначения, принимающий параллельные сеансы связи? Посмотрим, что Труди сможет сделать.

Чтобы понять, каким образом Труди взламывает протокол, обратимся к рис. 8.31. Алиса объявляет свои идентификационные данные в сообщении 1. Труди это сообщение перехватывает и запускает собственный сеанс, посылая сообщение 2 и прикидываясь Бобом. Здесь мы, как и раньше, изобразили серыми прямоугольниками сообщения второго сеанса. Алиса отвечает на сообщение 2, говоря в сообщении 3 следующее: «Ты представляешься Бобом? Это необходимо подтвердить». Здесь Труди заходит в тупик: она не может подтвердить, будто она — это Боб.

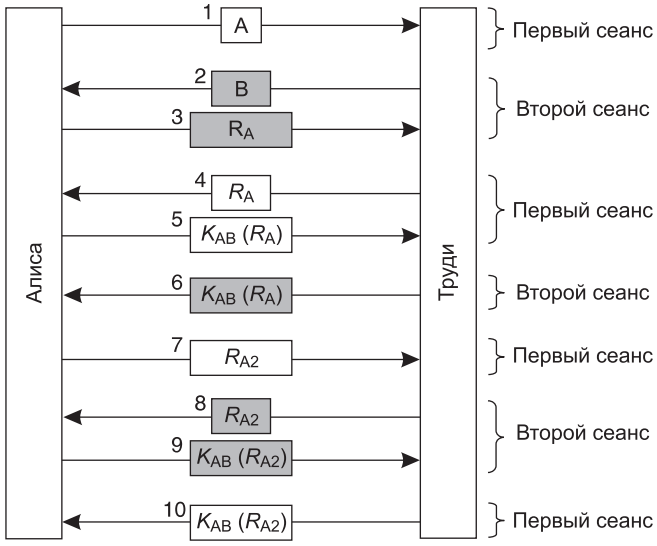


Рис. 8.31. Зеркальная атака протокола, показанного на рис. 8.28

Что же теперь Труди может сделать? Она возвращается к первому сеансу, где как раз наступает ее очередь отправки оклика. При этом отправляется R_A, полученный в сообщении 3. Алиса любезно отвечает на это в сообщении 5, предоставляя тем самым Труди информацию, необходимую ей для создания сообщения 6 в сеансе 2. Труди может теперь выбирать сеанс, так как она корректно ответила на оклик Алисы

во втором сеансе. Сеанс 1 можно закрыть, переправить в сеансе 2 какое-нибудь старое число и получить в итоге заверенный сеанс связи с Алисой.

Однако Труди просто невыносима, и она доказывает это своим дальнейшим поведением. Вместо того чтобы отправить какое-нибудь старое число для завершения регистрации сеанса 2, она ждет, пока Алиса пошлет сообщение 7 (ее оклик для сеанса 1). Конечно, Труди понятия не имеет, как ответить на это, поэтому она вновь проводит зеркальную атаку, отправляя R_{A2} в качестве сообщения 8. Алиса очень кстати шифрует R_{A2} в сообщении 9. Труди переключается на сеанс 1 и отправляет Алисе то число, какое ей хочется, в сообщении 10. Откуда она берет это число? Очевидно, из сообщения 9, пришедшего от Алисы. С этого момента Труди может гордиться тем, что у нее есть два полностью заверенных сеанса связи с Алисой.

Эта атака приводит к несколько иному результату, нежели протокол с тремя сообщениями, который мы видели на рис. 8.30. На этот раз Труди удается установить сразу два заверенных соединения с Алисой. В предыдущем примере одно заверенное соединение было установлено с Бобом. Опять же, если бы протокол удовлетворял всем четырем перечисленным требованиям, атака успеха бы не имела. Детальное обсуждение различных типов атак и методов противодействия им приведено в (Bird и др., 1993). Там также описана методика построения протоколов, корректность которых можно строго доказать. Однако даже простейший из таких протоколов достаточно сложен, поэтому сейчас мы обратимся к другому классу (вполне корректных) протоколов.

Итак, новый протокол аутентификации показан на рис. 8.32 (Bird и др., 1993). Тут мы видим тот же самый HMAC, который мы уже обсуждали при изучении IPsec. Для начала Алиса посылает Бобу *nonce* R_A в виде сообщения 1. Боб при ответе выбирает собственную *nonce*, R_B, и высылает ее вместе с HMAC. HMAC формируется построением структуры данных, состоящую из *nonce* Алисы и Боба, их идентификаторов, а также общего закрытого ключа K_{AB}. Затем вся эта структура с помощью хэш-функции (например, SHA-1) помещается в HMAC. После приема сообщения 2 Алиса становится счастливым обладателем R_A (это значение выбрано ею же), R_B, полученного в виде открытого текста, двух идентификаторов и закрытого ключа K_{AB}, известного и так. Имея все эти данные, она может вычислить HMAC самостоятельно. Если он согласуется с HMAC, содержащимся в сообщении, она убеждается, что говорит с Бобом, поскольку Труди не знает K_{AB} и, следовательно, не может угадать HMAC, который следует отослать. В ответе Алисы Бобу содержится HMAC, состоящий из двух *nonce*.

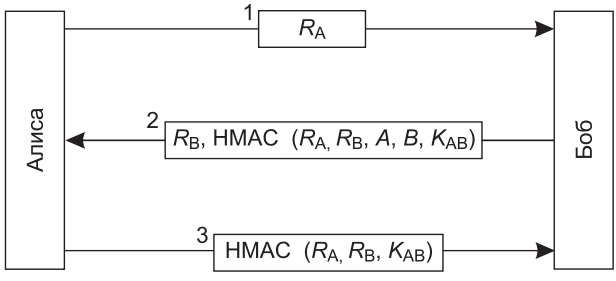


Рис. 8.32. Аутентификация с применением HMAC

Вопрос: может ли Труди как-нибудь взломать такой протокол? Нет, потому что она не может заставить ни одну из сторон шифровать выбранное ей самой значение или применять к нему хэш-функцию, как это было в ситуации на рис. 8.30. Оба HMAC включают в себя значения, выбранные отправителем. Труди не способна их контролировать каким-либо образом.

Использование HMAC — это далеко не единственное, что можно сделать. Альтернативная схема, которая применяется довольно часто, заключается в шифровании элементов данных последовательно с помощью сцепления блоков шифра.

8.7.2. Установка общего ключа: протокол обмена ключами Диффи—Хеллмана

Итак, мы предположили, что у Алисы и Боба есть общий секретный ключ. Предположим теперь, что у них его нет (поскольку до сих пор не разработана универсальная инфраструктура PKI создания подписей и распространения сертификатов). Как им получить такой ключ? Алиса может позвонить Бобу и передать ему ключ по телефону, но он, возможно, спросит: «Как вы докажете, что вы — Алиса, а не злоумышленник?» Они могут попытаться организовать встречу, на которую каждый придет с паспортом, водительскими правами и тремя кредитными картами, но, будучи занятыми людьми, они, возможно, не смогут найти устраивающую обоих дату встречи в течение нескольких месяцев. К счастью, существует способ для совершенно незнакомых людей установить общий секретный ключ среди бела дня, даже если злоумышленник старательно записывает каждое сообщение.

Протокол, позволяющий не встречавшимся ранее людям устанавливать общий секретный ключ, называется **протоколом обмена ключами Диффи—Хеллмана (Diffie—Hellman key exchange)** (Diffie и Hellman, 1976) и работает следующим образом. Алиса и Боб договариваются о двух больших простых числах, n и g, где $(n-1)/2$ — также является простым числом, кроме того, на число g накладываются некоторые дополнительные условия. Эти числа могут быть открытыми, поэтому каждый из них может просто выбрать n и g и открыто сообщить о них другому. Затем Алиса выбирает большое (например, двоичное 1024-разрядное) число x и держит его в секрете. Аналогично, Боб выбирает большое секретное число y.

Алиса начинает протокол обмена ключами с того, что посылает Бобу сообщение, содержащее $(n, g, g^x \bmod n)$, как показано на рис. 8.33. Боб отвечает Алисе сообщением, содержащим $gy \bmod n$. Теперь Алиса берет число, присланное ей Бобом, и возводит его в степень x, получая $(gy \bmod n)^x \bmod n$. Боб выполняет подобные вычисления, и получает $(g^x \bmod n)y \bmod n$. В соответствии с законами модульной арифметики оба вычисления должны быть равны $g^{xy} \bmod n$. Таким образом, как по мановению волшебной палочки, у Алисы и Боба есть общий секретный ключ $g^{xy} \bmod n$.

Конечно, злоумышленник видел оба сообщения. Ему известны значения n и g из первого сообщения. Если бы ему удалось вычислить значения x и y, ему бы удалось получить секретный ключ. Беда в том, что, зная $g^x \bmod n$ и n, найти значение x очень трудно. На сегодняшний день неизвестен алгоритм вычисления дискретного логарифма модуля очень большого простого числа.

Для примера возьмем (совершенно нереальные) значения $n = 47$ и $g = 3$. Алиса выбирает значение $x = 8$, а Боб выбирает $y = 10$. Оба эти числа хранятся в секрете. Со-

общение Алисы Бобу содержит числа (47, 3, 28), так как $3^8 \bmod 47 = 28$. Боб отвечает Алисе числом 17. Алиса вычисляет $17^8 \bmod 47$ и получает 4. Боб вычисляет $28^{10} \bmod 47$ и получает также 4. Таким образом, независимо друг от друга, Алиса и Боб определили, что значение секретного ключа равно 4. Чтобы найти ключ, злоумышленнику придется решить уравнение $3^x \bmod 47 = 28$, что можно сделать путем полного перебора для таких небольших чисел, но только не для чисел длиной в несколько сотен бит. Все известные в настоящее время алгоритмы просто очень долго работают, даже на работающих параллельно наибыстрейших суперкомпьютерах.

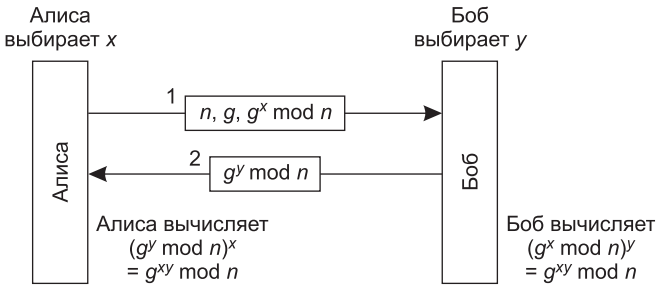


Рис. 8.33. Протокол обмена ключами Диффи—Хеллмана

Несмотря на всю элегантность алгоритма Диффи—Хеллмана, имеется одна проблема: когда Боб получит три числа (47, 3, 28), как он сможет удостовериться в том, что они посланы Алисой, а не злоумышленником (Труди)? Способа узнать это не существует. К сожалению, Труди может воспользоваться этим, чтобы обмануть Алису и Боба, как показано на рис. 8.34. Здесь, пока Алиса с Бобом выбирают значения x и y, Труди выбирает свое случайное число z. Алиса посылает Бобу сообщение 1. Труди перехватывает его и отправляет вместо него Бобу сообщение 2, используя правильные значения n и g (которые посылались открытым текстом), но со своим значением z вместо x. Она также посылает обратно Алисе сообщение 3. Позднее Боб отправляет Алисе сообщение 4, которое Труди снова перехватывает и хранит у себя.

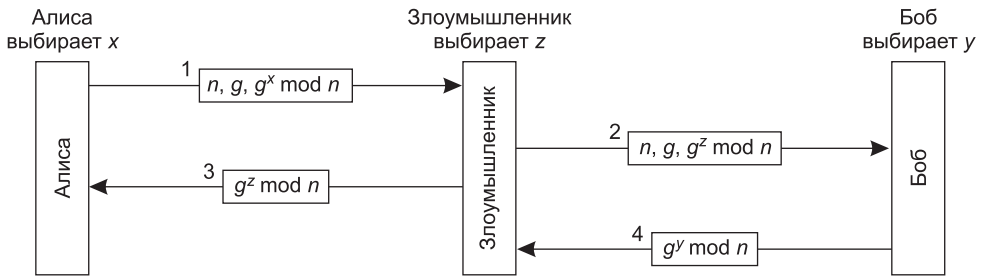


Рис 8.34. Атака типа «человек посередине»

Теперь все занимаются вычислением остатков от деления. Алиса вычисляет значение секретного ключа: $g^{xz} \bmod n$. Те же самые вычисления производит Труди (для общения с Алисой). Боб вычисляет $g^{yz} \bmod n$, что также делает и Труди (для общения с Бобом). Каждое сообщение, посылаемое Алисой в шифрованном сеансе, перехватывается Труди, сохраняется, изменяется, если это нужно, и отправляется (по желанию

Труди) Бобу. То же самое происходит и в обратном направлении. Труди видит все сообщения и может изменять их по своему усмотрению, в то время как Алиса и Боб полагают, что у них имеется защищенный канал для связи друг с другом. Подобные действия злоумышленника называются атакой типа **«человек посередине» (man-in-the-middle attack)**. Еще одно название этой атаки — **«пожарная цепочка» (bucket brigade attack)**, поскольку слегка напоминает старинных пожарных, передававших друг другу по цепочке ведра с водой.

8.7.3. Аутентификация с помощью центра распространения ключей

Итак, установка общего секретного ключа с незнакомцем почти удалась. С другой стороны, вероятно, не следовало вообще этим заниматься. Чтобы общаться с n людьми, вам понадобится хранить n ключей. Для людей, чей круг общения широк, хранение ключей может превратиться в серьезную проблему, особенно если все эти ключи придется хранить на отдельных пластиковых картах.

Другой подход состоит в организации доверительного центра распространения ключей (**KDC**, **key distribution center**). При такой схеме у каждого пользователя всего один ключ, общий с KDC. Операции с ключами аутентификации и сеансовыми ключами проходят через KDC. Простейший протокол аутентификации с помощью центра распространения ключей, включающий две стороны и доверенный KDC, изображен на рис. 8.35.

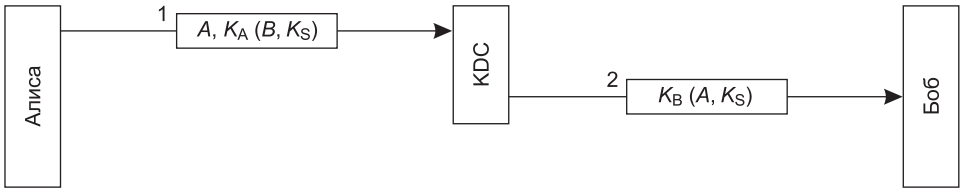


Рис. 8.35. Первая попытка протокола аутентификации с помощью KDC-центра

Идея, лежащая в основе протокола, проста: Алиса выбирает ключ сеанса, K_S, и заявляет KDC, что она желает поговорить с Бобом при помощи ключа K_S. Это сообщение шифруется секретным ключом K_A, которым совместно владеют только Алиса и центр распространения ключей. Центр распространения ключей расшифровывает это сообщение и извлекает из него идентификатор личности Боба и ключ сеанса. Затем он формирует новое сообщение, содержащее идентификатор личности Алисы и ключ сеанса, и посылает его Бобу. Это сообщение зашифровывается ключом K_B — секретным ключом, общим для Боба и центра распространения ключей. Расшифровав это сообщение, Боб узнает, что Алиса желает с ним поговорить и какой ключ она хочет использовать.

Аутентификация в данном случае происходит сама собой. KDC знает, что сообщение 1 пришло от Алисы, так как больше никто не может зашифровать его секретным ключом Алисы. Аналогично, Боб знает, что сообщение 2 пришло от KDC, так как кроме него их общий секретный ключ никому не известен.

К сожалению, этот протокол содержит серьезную ошибку. Труди нужны деньги, поэтому она придумывает некую легальную услугу, которую она могла бы выполнить для Алисы. Затем Труди делает Алисе заманчивое предложение и получает эту работу. Выполнив ее, Труди вежливо предлагает Алисе оплатить услуги, переведя деньги на ее банковский счет. Чтобы оплатить работу, Алиса устанавливает ключ сеанса со своим банкиром Бобом. Затем она посылает Бобу сообщение с просьбой перевести деньги на счет Труди.

Тем временем Труди возвращается к своим темным делам. Она копирует сообщение 2 (см. рис. 8.35) и запрос на перевод денег, следующий за ним. Затем она воспроизводит оба сообщения для Боба. Боб получает их и думает: «Должно быть, Алиса снова наняла Труди. Похоже, она хорошо справляется с работой». Боб перечисляет еще столько же денег со счета Алисы на счет Труди. Получив пятидесятый запрос на перевод денег, Боб выбегает из офиса, чтобы найти Труди и предложить ей большую ссуду, чтобы она могла расширить свой чрезвычайно успешный бизнес. Подобная проблема получила название **атаки повторным воспроизведением (replay attack)**.

Существует несколько решений этой проблемы. Первое решение состоит в помещении в каждое сообщение временного штампа. Все устаревшие сообщения просто игнорируются. Беда здесь в том, что системные часы в сети синхронизировать с большой степенью точности невозможно, поэтому должен существовать какой-то срок годности временного штампа. Труди может обмануть протокол, послав повторное сообщение во время этого интервала.

Второе решение заключается в помещении в сообщение уникального номера, обычно называемого **нонсом** (см. nonce в разделе 8.6.4.). Каждая сторона должна запоминать все предыдущие нонсы и отвергать любое сообщение, содержащее использованный ранее нонс. Однако нонсы должны храниться вечно, иначе Труди попытается воспроизвести сообщение пятилетней давности. Кроме того, если машина потеряет список нонсов в результате сбоя, она снова станет уязвимой к атакам повторным воспроизведением. Можно комбинировать временные штампы и нонсы, чтобы ограничить срок хранения нонсов, но так или иначе, протокол должен быть значительно усложнен.

Более сложный метод аутентификации состоит в использовании многостороннего протокола оклик-отзыв. Хорошо известным примером такого протокола является **протокол аутентификации Нидхэма—Шредера (Needham—Schroeder authentication protocol)** (Needham-Schroeder, 1978), один из вариантов которого показан на рис. 8.36.

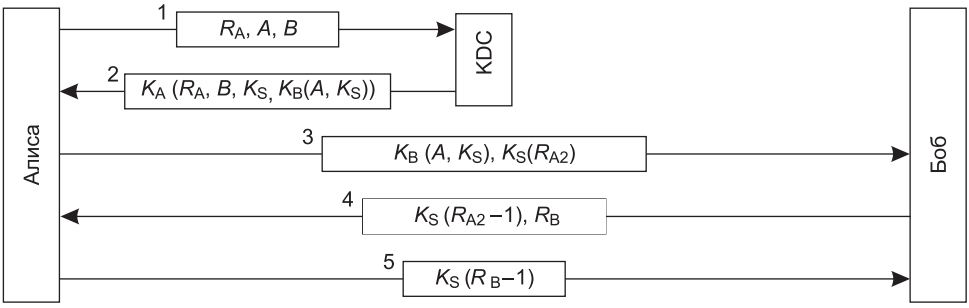


Рис. 8.36. Протокол аутентификации Нидхэма—Шредера

Работа протокола начинается с того, что Алиса сообщает KDC, что она желает поговорить с Бобом. Это сообщение содержит в качестве нонса большое случайное число R_A. Центр распространения ключей посылает обратно сообщение 2, содержащее случайное число Алисы, ключ сеанса и так называемый билет, который она может послать Бобу. Цель посылки случайного числа R_A состоит в том, чтобы убедить Алису, что сообщение 2 является свежим, а не повторно воспроизведенным. Идентификатор Боба также помещается в сообщение 2, на случай, если злоумышленник (Труди) вздумает заменить его идентификатор на свой в сообщении 1, так чтобы KDC зашифровал билет в конце сообщения 2 ключом K_T (ключ Труди), вместо K_B. Билет, зашифрованный ключом K_B, помещается внутри зашифрованного сообщения, чтобы злоумышленник не смог заменить его чем-либо другим, пока сообщение 2 добирается до Алисы.

Затем Алиса посылает билет Бобу вместе с новым случайным числом R_{A2}, зашифрованным ключом сеанса K_S. В сообщении 4 Боб посылает обратно $K_S(R_{A2} - 1)$, чтобы доказать Алисе, что она разговаривает с настоящим Бобом. Отсылать обратно просто $K_S(R_{A2})$ бессмысленно, так как это число могло быть украдено злоумышленником из сообщения 3.

Получив сообщение 4, Алиса убеждается, что разговаривает с Бобом и что до сих пор не было использовано повторных сообщений. Между отправкой случайного числа R_{A2} и получением ответа на него в виде $K_S(R_{A2} - 1)$ проходит довольно короткий промежуток времени. Цель сообщения 5 — убедить Боба, что он действительно разговаривает с Алисой, и что в этом сеансе связи также отсутствуют повторно воспроизведенные данные. Возможность атаки с помощью повторного воспроизведения ранее записанной информации исключается этим протоколом благодаря тому, что каждая сторона формирует оклик другой стороны и получает на него отзыв.

Несмотря на всю кажущуюся солидность протокола, в нем, тем не менее, имеется небольшое слабое место. Если злоумышленнику удастся каким-либо способом раздобыть старый ключ сеанса K_S, он сможет инициировать новый сеанс с Бобом, повторно воспроизведя сообщение 3 с использованием скомпрометированного ключа, и выдать себя за Алису (Denning и Sacco, 1981). На этот раз злоумышленник может украсть деньги со счета Алисы, даже не выполнив никаких услуг.

Позднее Нидхэм и Шредер опубликовали протокол, решающий эту проблему (Needham и Shroeder, 1987). В том же выпуске того же журнала Отуэй (Otway) и Рис (Rees) также опубликовали протокол, решающий эту проблему более коротким путем. На рис. 8.37 показан слегка видоизмененный протокол Отуэя—Риса.

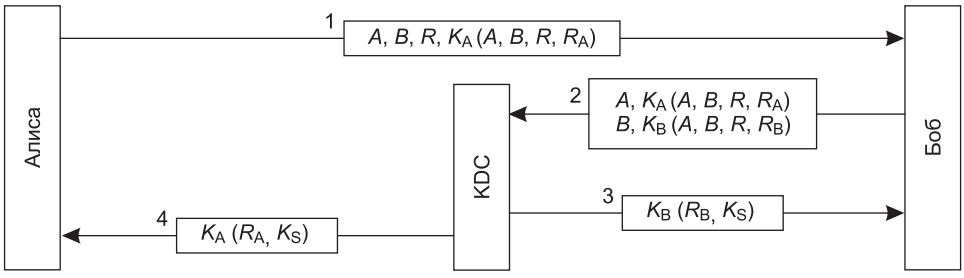


Рис. 8.37. Протокол аутентификации Отуэя—Риса (слегка упрощенный)

В протоколе Отуэя—Риса Алиса начинает с формирования пары случайных номеров: R, который будет использоваться в качестве общего идентификатора, и R_A, который Алиса будет использовать в качестве оклика Боба. Получив это сообщение, Боб формирует новое сообщение из зашифрованной части сообщения Алисы и аналогичной собственной части. Обе части сообщения, зашифрованные ключами K_A и K_B, идентифицируют Алису и Боба, содержат общий идентификатор и оклики.

Центр распространения ключей проверяет, совпадают ли общие идентификаторы R в обеих частях сообщения. Они могут не совпадать, если злоумышленник подменил R в сообщении 1 или заменил часть сообщения 2. Если оба общих идентификатора R совпадают, KDC считает сообщение, полученное от Боба, достоверным. Затем он формирует ключ сеанса K_S и отправляет его Алисе и Бобу, зашифровав ключ сеанса ключами Алисы и Боба. Каждое сообщение также содержит случайное число получателя, в доказательство того, что эти сообщения посланы KDC, а не злоумышленником. К этому моменту Алиса и Боб обладают одним и тем же ключом сеанса и могут начать обмен информацией. После первого же обмена данными они увидят, что обладают одинаковыми копиями ключа сеанса K_S, на чем процесс аутентификации можно будет считать завершенным.

8.7.4. Аутентификация при помощи протокола Kerberos

Во многих реально работающих системах (включая Windows 2000 и более поздние) применяется протокол аутентификации **Kerberos**, основанный на одном из вариантов протокола Нидхэма—Шредера. Он назван по имени трехглавого пса греческих мифов Кербера[1], охранявшего выход из Аида. Кербер пропускал в Аид всякого, но не выпускал оттуда никого. Протокол Kerberos был разработан в Массачусетском технологическом институте для обеспечения пользователям рабочих станций надежного доступа к сетевым ресурсам. Его основное отличие от протокола Нидхэма—Шредера состоит в предположении о довольно хорошей синхронизации всех часов в сети. Было разработано несколько последовательных версий протокола. Версия V5 наиболее широко применяется в промышленности и описана в RFC 4120. От более ранней версии, V4, в конец концов отказались, после того как в ней были найдены серьезные недостатки (Yu и др., 2004). V5 улучшена по сравнению с V4 — есть много небольших изменений по отношению к протоколу и несколько улучшенных характеристик, например, она больше не опирается на устаревший DES. Дополнительную информацию см. в (Neumann и Ts'o (1994).

В работе протокола Kerberos, помимо рабочей (клиентской) станции Алисы, принимают участие еще три сервера:

1. Сервер аутентификации (AS, Authentication Server): проверяет личность пользователей при входе в сеть.
2. Сервер выдачи билетов (TGS, Ticket Granting Server): выдает «билеты, подтверждающие подлинность».
3. Боб, то есть сервер, предоставляющий услуги Алисе.

[1] Чаще называемого Цербером благодаря латинскому написанию. — *Примеч. перев.*

Сервер аутентификации AS аналогичен центру распространения ключей KDC в том, что у него есть общий секретный пароль для каждого пользователя. Работа сервера выдачи билетов TGS состоит в выдаче свидетельств, убеждающих другие серверы в том, что владелец билета действительно является тем, за кого он себя выдает.

Чтобы начать сеанс, Алиса усаживается за клавиатуру произвольной общедоступной рабочей станции и вводит свое имя и название TGS. Рабочая станция посылает введенное имя открытым текстом на сервер аутентификации, как показано на сообщении 1 рис. 8.38. Сервер аутентификации AS возвращает рабочей станции Алисы ключ сеанса и билет $K_{TGS}(A, K_S, t)$ для сервера выдачи билетов TGS. Ключ сеанса зашифровывается секретным ключом Алисы, так чтобы только Алиса могла его расшифровать. Только после получения сообщения 2 рабочая станция запрашивает пароль Алисы, и никак не раньше. С помощью этого пароля формируется ключ K_A, которым расшифровывается сообщение 2 и из него извлекается ключ сеанса. После расшифровки рабочая станция сразу же уничтожает хранящийся в ее памяти пароль. Если вместо Алисы на рабочей станции попытается зарегистрироваться Труди, введенный ею пароль окажется неверным, что будет обнаружено рабочей станцией, так как стандартная часть сообщения 2 окажется неверной.

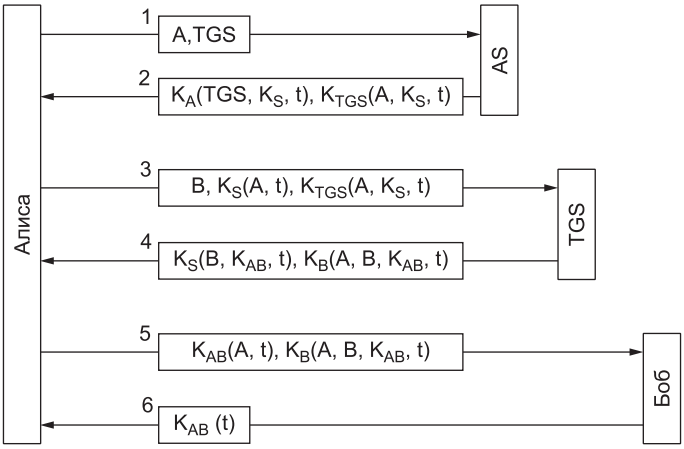


Рис. 8.38. Работа протокола Kerberos V5

После регистрации в сети Алиса может сообщить рабочей станции, что она хочет вступить в контакт с файловым сервером, то есть Бобом. При этом рабочая станция посылает серверу выдачи билетов сообщение 3 с просьбой выдать билет для общения с Бобом. Ключевым элементом этого запроса является билет $K_{TGS}(A, K_S, t)$, который зашифрован секретным ключом TGS-сервера и используется для подтверждения личности отправителя. Сервер выдачи билетов отвечает созданием ключа сеанса K_{AB}, которым будут пользоваться Алиса и Боб. Он отправляет Алисе две версии этого ключа. Один ключ зашифрован ключом сеанса K_S, поэтому Алиса может его прочитать. Второй ключ представляет собой еще один билет, который шифруется ключом Боба K_B, что позволяет Бобу его прочитать.

Злоумышленник может скопировать сообщение 3 и попытаться использовать его снова, но ему помешает временной штамп t, отправляемый вместе с этим сообщением.

Злоумышленник не может заменить этот временной штамп на более новый, так как не знает ключа сеанса K_S, которым пользуется Алиса для общения с сервером выдачи билетов. Даже если злоумышленник очень быстро повторит сообщение 3, все равно, единственное, что он получит в ответ, это сообщение 4, которое он не смог расшифровать в первый раз и не сможет расшифровать и во второй раз.

После этого Алиса через новый билет может послать Бобу ключ K_{AB} для установки сеанса с Бобом. Эти сообщения также содержат временные штампы. Сообщение 6, получаемое в ответ (в качестве возможной проверки), подтверждает, что Алиса говорит именно с Бобом, а не со злоумышленником.

Наконец, после этой серии обмена сообщениями Алиса сможет обмениваться с Бобом данными, используя ключ сеанса K_{AB}. Если после этого Алиса решит, что ей необходим другой сервер, например Кэрол (Carol, C), она может просто послать серверу выдачи билетов сообщение, аналогичное третьему, заменив в нем B на C (то есть идентификатор Боба на идентификатор Кэрол). TGS мгновенно ответит сообщением, содержащим билет, зашифрованный ключом K_C. Этот билет Алиса пошлет Кэрол, для которой он будет служить гарантией подлинности Алисы.

Достоинство этого протокола состоит в том, что теперь Алиса может получать защищенный доступ к любому серверу сети, и в то же время ее пароль ни разу не передавался по сети. В действительности, он только на несколько миллисекунд появлялся в ее рабочей станции. Однако обратите внимание, что каждый сервер выполняет свою собственную процедуру авторизации. Когда Алиса предъявляет свой билет Бобу, это всего лишь подтверждает Бобу подлинность предъявителя билета. К чему же Алиса может получить доступ на сервере, решает Боб.

Поскольку разработчики системы Kerberos не рассчитывали, что весь мир станет доверять одному-единственному серверу аутентификации, они обеспечили существование нескольких **областей** (**realms**), каждая из которых имеет свой собственный сервер аутентификации (AS) и сервер выдачи билетов (TGS). Чтобы получить билет для сервера, расположенного в удаленной области, Алиса должна запросить у своего TGa билет, который будет принят TGм удаленной области. Если удаленный TGp зарегистрировался на локальном TGe (так же, как это делают локальные серверы), локальный TGp выдаст Алисе билет, действительный на удаленном TGe. После этого она может получить у удаленного TGa билеты к серверам данной удаленной области. Обратите внимание, что для того, чтобы две стороны, расположенные в различных областях, могли установить друг с другом защищенный сеанс связи, каждая из сторон должна доверять TGy другой стороны. Иначе они просто не смогут работать вместе.

8.7.5. Аутентификация с помощью шифрования с открытым ключом

Взаимная аутентификация также может выполняться с помощью шифрования с открытым ключом. Для начала Алисе нужно получить открытый ключ Боба. Если инфраструктура PKI реализована на основе сервера каталогов, выдающего сертификаты на открытые ключи, Алиса может потребовать сертификат Боба, что показано в виде сообщения 1 на рис. 8.39. Ответ, содержащийся в сообщении 2, — это сертификат X.509

с открытым ключом Боба. Проверив корректность подписи, Алиса может отправить Бобу сообщение со своим идентификатором и нонсом.

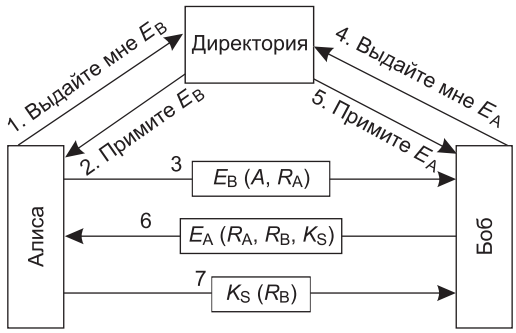


Рис. 8.39. Взаимная идентификация с помощью открытого ключа

Когда Боб получает это сообщение, он не знает, пришло ли оно от Алисы или от Труди (злоумышленника), но он делает вид, что все в порядке и просит сервер каталогов выдать ему открытый ключ Алисы (сообщение 4). Вскоре он его получает (в сообщении 5). Затем он отсылает Алисе сообщение 6, содержащее случайное число Алисы R_A, свой собственный нонс R_B и предлагаемый ключ сеанса K_S. Алиса расшифровывает полученное сообщение 6 своим закрытым ключом. Она видит в нем свое случайное число R_A и очень этому рада: это подтверждает, что сообщение пришло от Боба, так как у Труди не должно быть способа определить значение этого числа. Кроме того, случайное число R_A свидетельствует о свежести этого сообщения. Алиса соглашается на установку сеанса, отправляя сообщение 7. Когда Боб видит свое случайное число R_B, зашифрованное ключом сеанса, который он сам же сформировал, он понимает, что Алиса получила сообщение 6 и проверила значение R_A. Боб счастлив и доволен.

Может ли злоумышленник каким-либо образом обмануть этот протокол? Он может сфабриковать сообщение 3 и спровоцировать Боба на проверку Алисы, но Алиса увидит число R_A, которого она не посылала, и не станет продолжать. Злоумышленник не сможет убедительно подделать сообщение 7, так как ему не известны значения оклика R_B или ключа K_S, и он не может определить их, не имея закрытого ключа Алисы. Так что ему не везет.

8.8. Конфиденциальность электронной переписки

При пересылке между двумя удаленными пользователями сообщение обычно преодолевает по пути десяток других машин. Любая из них может читать и записывать проходящую через нее почту. Конфиденциальности не существует, что бы ни думали об этом многие пользователи. Тем не менее многие пользователи желали бы иметь возможность посылать электронную почту так, чтобы ее мог прочитать только тот,

для кого она предназначается, и никто другой: ни шеф, ни даже правительство. Эта потребность стимулировала применение некоторыми группами и отдельными разработчиками криптографических принципов к электронной почте. В следующих разделах мы познакомимся с широко распространенной системой защиты электронной почты PGP, а также дадим общее представление о еще одной — S/MIME. Дополнительную информацию см. в (Kaufman и др., 2002; Schneier, 1995).

8.8.1. PGP

Наш первый пример, система **PGP (Pretty Good Privacy — довольно хорошая конфиденциальность)**, создана всего одним человеком, Филом Циммерманом (Phil Zimmermann, 1995a, 1995b). Циммерман является сторонником безопасности в сетях и его девиз таков: «Если конфиденциальность нарушается, значит, она доступна только нарушителям закона». Выпущенная в 1991 году система PGP представляет собой полный пакет для электронной почты, обеспечивающий конфиденциальность, аутентификацию, цифровые подписи и сжатие. Все это делается в легкой и удобной форме. Более того, полный пакет, включающий все исходные тексты программ, свободно распространяется через Интернет. Благодаря своему качеству, цене (нулевой) и простоте установки на различных платформах, включая UNIX, Linux, Windows и Mac OS, в настоящее время система PGP получила широкое распространение.

PGP кодирует данные с помощью блочного шифра **IDEA (International Data Encryption Algorithm — международный алгоритм шифрования данных)**, использующего ключи длиной 128 бит. Он был изобретен в Швейцарии в те времена, когда DES уже считался устаревшим, а AES еще не был придуман. Концептуально IDEA похож на DES/AES: производится смешивание разрядов в серии, однако детали реализации функций отличаются от DES и AES. Управление ключами происходит с помощью RSA, а для задач обеспечения целостности данных применяется MD5. Все эти методы мы обсуждали ранее.

История создания системы PGP весьма запутана с первого дня (Levy, 1993). Поскольку она свободно распространялась через Интернет, правительство США объявило, что Циммерман нарушил закон, запрещающий экспорт военного имущества. Следствие по этому делу длилось пять лет, однако в один прекрасный день прекратилось по двум основным причинам. Во-первых, Циммерман собственноручно не выкладывал PGP в Интернете, и адвокат аргументировал позицию защиты тем, что обвиняемый *сам* никогда не занимался экспортом чего бы то ни было (кроме того, еще надо доказать, что создание сайта равносильно экспорту). Во-вторых, правительство вдруг осознало, что выигрыш дела означал бы, что любой веб-сайт, содержащий загружаемые программы, связанные с секретностью, подпадает под действие закона о торговле такими предметами, как танки, подводные лодки, военные самолеты и ядерное оружие.

Честно говоря, законы, касающиеся экспорта, кажутся несколько диковатыми. Правительство решило, что размещение программы на веб-странице можно приравнять к нелегальному экспорту, и надоедало Циммерману по этому поводу целых 5 лет. С другой стороны, если кто-то опубликует в книге полный исходный код PGP на язы-

ке C (крупным шрифтом, да еще и с контрольной суммой в конце каждой страницы, что облегчит сканирование) и затем займется экспортом этой книги, правительство и глазом не моргнет: книги по закону не являются военным имуществом.

Еще одна проблема, с которой внезапно столкнулась система PGP, была связана с посягательством на патентные права. Владелец патента на RSA корпорация RSA Security сослалась на то, что использование метода RSA в PGP является посягательством на патент. Эта проблема разрешилась в версиях начиная с 2.6. Кроме того, в PGP используется другой запатентованный алгоритм, IDEA, что поначалу тоже вызывало некоторые вопросы.

Так как PGP — это система с открытым исходным кодом, появилось множество модификаций, созданных различными группами и отдельными заинтересованными лицами. Некоторые из них пытались каким-то образом обойти законы об экспорте оружия, другие старались избежать применения запатентованных алгоритмов, а третьи работали над превращением PGP в коммерческий продукт с закрытым исходным кодом. Несмотря на то что законы об экспорте оружия несколько смягчились (тем не менее продукцию, использующую AES, до сих пор нельзя экспортировать за пределы США), а срок действия патента RSA закончился в сентябре 2000 года, следствием всех этих проблем стало появление и распространение нескольких несовместимых версий PGP, имеющих разные названия. Ниже обсуждается классический вариант PGP, он же является самым старым и простым. Еще одна популярная версия, Open PGP, описана в RFC 2440. Можно отметить еще GNU Privacy Guard.

В системе PGP намеренно используются уже существующие криптографические алгоритмы, а не изобретаются новые. Все они прошли тщательную проверку ведущими криптоаналитиками мира, и история создания этих алгоритмов не запятнана участием каких-либо государственных организаций, пытающихся их ослабить. Последнее качество является особенно большим преимуществом для всех, кто склонен не доверять правительству.

Система PGP поддерживает сжатие текста, секретность и цифровые подписи, а также предоставляет исчерпывающие средства управления ключами. Как ни странно, не поддерживаются средства электронной почты. Она больше всего похожа на препроцессор, берущий на входе открытый текст и создающий на выходе подписанный шифротекст, представленный в формате base64. Разумеется, эти выходные данные можно отправить по электронной почте. Некоторые реализации на последнем шаге обращаются к пользовательскому агенту, чтобы упростить задачу реальной отправки сообщения.

Чтобы понять, как работает система PGP, рассмотрим пример на рис. 8.40. Алиса хочет надежным способом послать Бобу открытым текстом подписанное сообщение *P*. У Алисы и у Боба есть закрытый (*DX*) и открытый (*EX*) RSA-ключи. Предположим, что каждому из них известен открытый ключ другого. Способы передачи ключей мы рассмотрим позднее.

Алиса начинает с того, что запускает на своем компьютере программу PGP. Программа PGP сначала хэширует ее сообщение *P* с помощью алгоритма MD5, а затем шифрует полученный хэш при помощи ее закрытого RSA-ключа D_A. Получив это сообщение, Боб может расшифровать хэш открытым ключом Алисы и убедиться в его правильности. Даже если какой-либо злоумышленник (например, Труди) мог полу-

чить хэш на этой стадии и расшифровать его известным открытым ключом Алисы, сила алгоритма MD5 гарантирует невозможность создания другого сообщения с тем же хэшем (из-за трудоемкости вычислений).

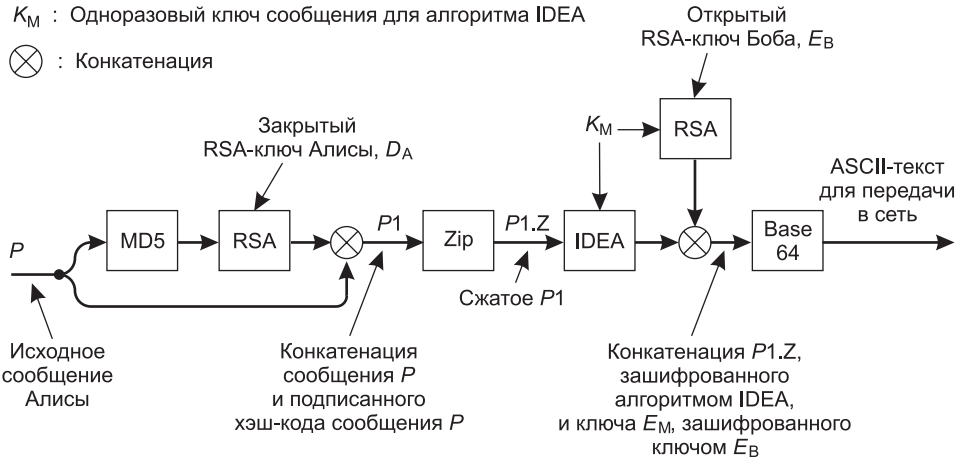


Рис. 8.40. Использование системы PGP для передачи сообщения

Затем зашифрованный хэш-код и оригинальное сообщение объединяются в единое сообщение *P1*, которое сжимается с помощью программы ZIP, использующей алгоритм Лемпеля—Зива (Ziv и Lempel, 1977). Будем называть результат этого этапа *P1.Z*.

Затем программа PGP предлагает Алисе ввести случайную текстовую строку. При формировании 128-разрядного ключа сообщения K_M для алгоритма IDEA учитываются как содержимое, так и скорость ввода. (В литературе по PGP этот ключ назван сеансовым, что является неправильным употреблением термина, так как никакого сеанса нет.) Затем *P1.Z* шифруется алгоритмом IDEA с помощью ключа K_M в режиме шифрованной обратной связи. Кроме того, ключ K_M шифруется открытым ключом Боба, E_B. Эти два компонента объединяются и преобразуются в кодировку base64, о которой уже рассказывалось в главе 7, когда мы говорили о стандартах MIME. Получающееся в результате сообщение содержит только буквы, цифры и символы +, * и =, что означает — это сообщение может быть помещено в тело письма стандарта RFC 822 и можно надеяться, что оно прибудет к получателю без изменений.

Получив сообщение, Боб выполняет обратное преобразование base64 и расшифровывает IDEA-ключ своим закрытым RSA-ключом. С помощью IDEA-ключа он расшифровывает сообщение и получает *P1.Z*. Распаковав zip-файл, Боб отделяет зашифрованный хэш-код от открытого текста и расшифровывает его открытым ключом Алисы. Если в результате обработки открытого текста алгоритмом MD5 получается тот же самый хэш-код, это означает, что сообщение *P* действительно пришло от Алисы.

Следует отметить, что алгоритм RSA используется здесь только в двух местах: для зашифровки 128-разрядного MD5-хэша и 128-разрядного IDEA-ключа. Алгоритм RSA медленный, но ему нужно зашифровать всего лишь 256 бит, что совсем немного. Более того, все эти 256 бит в высшей степени случайны, поэтому злоумышленнику придется очень сильно попотеть, чтобы угадать ключ. Основное шифрование вы-

полняется алгоритмом IDEA, который на порядок быстрее, чем RSA. Итак, система PGP обеспечивает секретность, сжатие и цифровую подпись и делает это намного эффективнее, чем схема, показанная на рис. 8.16.

Система PGP поддерживает четыре длины ключа RSA. Пользователь может самостоятельно выбирать нужную длину. Предлагаются следующие варианты длины.

1. Несерьезная (384 бит): шифр может быть взломан сегодня же организациями с большим бюджетом.
2. Коммерческая (512 бит): возможно, шифр смогут взломать организации из трех букв.
3. Военная (1024 бит): никто на Земле не сможет взломать этот шифр.
4. Межпланетная (2048 бит): никто во всей Вселенной не сможет взломать шифр.

Поскольку алгоритм RSA используется только для двух небольших вычислений, всем следует всегда применять ключи межпланетного варианта, длиной 2048 бит.

Формат классического PGP-сообщения показан на рис. 8.41. Также используются многочисленные другие форматы. Сообщение состоит из трех частей: области ключа, области подписи и области сообщения. Область ключа, помимо самого IDEA-ключа, содержит также идентификатор открытого ключа, так как пользователям разрешено иметь несколько открытых ключей.

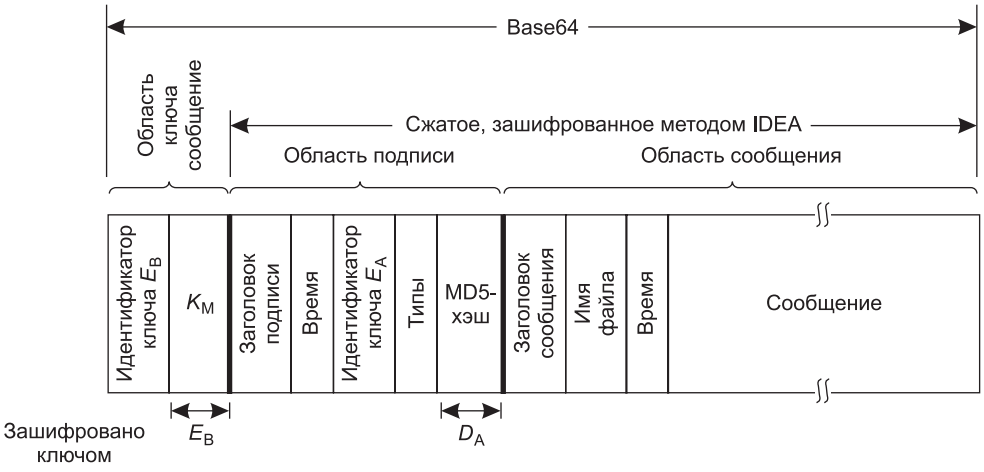


Рис. 8.41. PGP-сообщение

Область подписи содержит заголовок, который нас сейчас не интересует. За заголовком следует временной штамп, идентификатор открытого ключа отправителя, с помощью которого получатель сможет расшифровать хэш-код, используемый в качестве подписи. Следом идет идентификатор использованных алгоритмов шифрования и хэширования (чтобы можно было пользоваться, например, MD6 или RSA2, когда они будут разработаны). Последним в области подписи располагается сам зашифрованный хэш-код.

Часть сообщения также содержит заголовок, имя файла по умолчанию, на случай, если получатель будет сохранять принятое сообщение на диске, временной штамп создания сообщения и, наконец, само сообщение.

Работе с ключами в системе PGP было уделено особое внимание, так как это ахиллесова пята всех систем защиты. У каждого пользователя локально находится две структуры данных: набор закрытых ключей и набор открытых ключей (эти наборы иногда называют связками). **Связка закрытых ключей (private key ring)** содержит одну или несколько личных пар ключей, состоящих из закрытого и открытого ключа. Несколько пар ключей поддерживаются, чтобы позволить пользователям периодически их менять, когда возникают опасения, что тот или иной ключ скомпрометирован. При этом для смены ключа не требуется принимать каких-либо экстренных мер по передаче нового ключа. У каждой пары ключей есть связанный с ней идентификатор, так что отправителю нужно всего лишь сообщить получателю, которым открытым ключом был зашифрован ключ сообщения. Идентификатор сообщения состоит из младших 64 разрядов открытого ключа. За отсутствие конфликтов между идентификаторами ключей отвечают сами пользователи. Закрытые ключи на диске зашифрованы специальным паролем (произвольной длины), защищающем их от кражи.

Связка открытых ключей (public key ring) содержит открытые ключи корреспондентов пользователей. Они нужны для шифрования ключей сообщений, связанных с каждым сообщением. Каждая запись набора открытых ключей содержит не только ключ, но также его 64-разрядный идентификатор и отметку, указывающую степень доверия пользователя этому ключу.

Степень доверия ключу зависит, например, от способа его получения. Предположим, что открытые ключи расположены на электронных досках объявлений (BBS). Труди может атаковать доску объявлений и подменить размещенный там открытый ключ Боба своим ключом. Когда Алиса попытается воспользоваться подмененным ключом, Труди сможет применить к Бобу атаку типа «человек посередине».

Чтобы предотвратить подобные атаки или хотя бы минимизировать их ущерб, Алисе необходимо знать, насколько она может доверять открытому ключу Боба, хранящемуся в ее наборе открытых ключей. Если Боб лично дал ей диск с ключом, она может поставить такому ключу максимальную степень доверия. Вот в этом и заключается децентрализованный, контролируемый пользователем подход к управлению открытыми ключами, отличающий PGP от централизованной схемы PKI.

Однако на практике открытые ключи часто получают, опрашивая доверенный сервер ключей. По этой причине после стандартизации X.509 система PGP стала поддерживать сертификаты наряду с традиционным для PGP механизмом связки открытых ключей. Все современные версии PGP имеют поддержку X.509.

8.8.2. S/MIME

Следующим изобретением IETF в области обеспечения конфиденциальности электронной почты стала система под названием **S/MIME (Secure/MIME — защищенный MIME)**. Она описывается в RFC с 2632 по 2643. Она обеспечивает аутентификацию, целостность данных, секретность и проверку подлинности информации. Обладает неплохой гибкостью, поддерживает разнообразные криптографические алгоритмы.

По названию можно догадаться, что S/MIME тесно связана с MIME в том смысле, что позволяет защищать любые типы сообщений. Определено множество новых заголовков MIME, например, для цифровых подписей.

В S/MIME нет жесткой иерархии сертификатов, отсутствует единый центр управления, что составляло проблему для более ранней системы **PEM (Privacy Enhanced Mail — почта повышенной секретности)**. Вместо этого пользователи могут работать с набором доверительных якорей. До тех пор пока сертификат может быть проверен по доверительному якорю, он считается корректным. Система S/MIME использует стандартные алгоритмы и протоколы, которые мы уже рассматривали, поэтому на этом мы закончим ее обсуждение. Более подробную информацию вы найдете в RFC.

8.9. Защита информации во Всемирой паутине

Мы только что закончили изучение двух важных областей, в которых требуется защита информации, — соединения и электронная почта. Можно сказать, что это были аперитив и суп. Теперь же мы приступаем к главному блюду: защите информации во Всемирной паутине. Именно в WWW работает большинство злоумышленников, делая свое грязное дело. В следующих разделах будут рассмотрены некоторые проблемы, относящиеся к безопасности во Всемирной паутине.

Эту тему можно разделить на три части. Первая связана с безопасным именованием объектов и ресурсов. Вторая — с установлением аутентифицированных соединений. Третья — с тем, что случается, когда веб-сайт отправляет клиенту исполняемый код. После перечисления возможных опасностей мы рассмотрим все эти вопросы.

8.9.1. Возможные опасности

Практически каждую неделю газеты публикуют статьи о проблемах безопасности во Всемирной паутине. Ситуация складывается действительно довольно мрачная. Посмотрим на некоторые примеры того, что уже имело место. Во-первых, мы помним, как домашние страницы многочисленных организаций самых разных масштабов подвергались атакам хакеров и заменялись подложными страницами. (Термин «хакер» (hacker) приобрел значение «взломщик» благодаря журналистам, которые мало что понимали в компьютерном мире, но попытались воспользоваться профессиональным жаргоном программистов. На самом же деле, изначально «хакерами» называли великих программистов. Взломщиков же мы и называем взломщиками («cracker»).) В списке сайтов, которые удалось взломать, находятся такие как Yahoo!, сайт вооруженных сил США, ЦРУ, НАСА, а также *New York Times*. В большинстве случаев взломщики просто заменяли оригиналы на свои странички с каким-нибудь смешным (обычно издевательским) текстом, и уже через несколько часов сайты удавалось восстановить.

Однако были гораздо более серьезные атаки. Многие сайты были выведены из строя за счет искусственно созданной чрезмерной нагрузки (атака типа «отказ в обслуживании», DoS), с которой заведомо не может справиться сервер. Зачастую такие нападения совершались сразу с нескольких машин, которые взломщику уже удалось

взломать и заставить против воли участвовать в преступлении («распределенный DoS», DDoS). Такие атаки настолько распространены, что уже перестали быть новостью. Тем не менее ущерб от них исчисляется тысячами долларов.

В 1999 году шведский взломщик проник на сайт Hotmail (корпорации Microsoft) и создал зеркало, на котором все желающие могли ввести имя любого пользователя этого сайта и прочесть всю его текущую почту и почтовые архивы.

А один русский 19-летний взломщик по имени Максим смог украсть с сайта, занимающегося электронной коммерцией, номера 300 000 кредитных карт. Затем он обратился к их владельцам и сообщил, что если они не заплатят ему 100 000 долларов, он опубликует номера кредиток в Интернете. Они не поддались на провокацию, и тогда он действительно опубликовал номера кредитных карт, что нанесло серьезный ущерб невинным жертвам.

Двадцатитрехлетний студент из Калифорнии послал по электронной почте в агентство новостей фальшивый пресс-релиз, в котором сообщалось об огромных убытках корпорации Emulex и об уходе в отставку ее генерального директора. Спустя несколько часов биржевые цены на акции Emulex снизились на 60 %, в результате чего их держатели лишились более $2 млрд. Злоумышленник заработал около четверти миллиона долларов, продав акции незадолго до своего ложного заявления. Хотя в данном случае взлом не произошел непосредственно во Всемирной паутине, понятно, что объявление подобного рода, размещенное на сайте компании, привело бы к такому же эффекту.

К сожалению, одно перечисление таких примеров могло бы занять несколько страниц. И теперь нам пора обратиться к технической стороне дела. Более подробную информацию, касающуюся проблем безопасности всех видов, см. в (Anderson, 2008a; Stuttard и Pinto, 2007; Schneier, 2000). Поиск в Интернете также даст неплохие результаты.

8.9.2. Безопасное именование ресурсов

Начнем с чего-нибудь очень простого. Допустим, Алиса хочет посетить веб-сайт Боба. Она набирает в браузере URL, и через несколько секунд появляется страничка. Но в самом ли деле эта страничка создана Бобом? Может, да, а может, нет. Не исключено, что Труди снова принялась за свои шуточки. Например, она могла перехватить исходящие от Алисы пакеты и изучить их. Найдя запрос GET на получение страницы Боба, Труди могла сама зайти на эту страницу, изменить ее и отослать Алисе. Алиса не заметила бы ровным счетом ничего. Хуже того, Труди могла изменить цены в электронном магазине Боба на более низкие, сделав тем самым предложение Боба очень привлекательным. Вероятность того, что теперь Алиса вышлет номер своей кредитной карты «Бобу» (с целью приобрести чего-нибудь по выгодной цене), резко повысилась.

Одним из недостатков схемы «человек посередине» является то, что Труди должна быть в состоянии перехватывать исходящий трафик Алисы и подделывать свой исходящий трафик. На практике она должна прослушивать телефонную линию либо Боба, либо Алисы (поскольку прослушивание оптоволоконного кабеля — задача непростая). Это, конечно, возможно, но Труди не только умна и хитра, но и ленива. Она знает более простые способы обмануть Алису.

Обман DNS

Допустим, Труди может взломать систему DNS (например, ту ее часть, которая хранится в кэше DNS у интернет-провайдера Алисы) и заменить IP-адрес Боба (например, 36.1.2.3) своим IP-адресом (например, 42.9.9.9). Тогда можно провести атаку. То, как все должно работать в нормальной ситуации, показано на рис. 8.42, *а*: (1) Алиса запрашивает у службы DNS IP-адрес Боба и (2) получает его. (3) Она запрашивает домашнюю страничку Боба и (4) получает ее. После того как Труди заменяет IP-адрес Боба на свой собственный, мы получаем ситуацию, показанную на рис. 8.42, *б*. Алиса ищет IP-адрес Боба, а получает вместо него IP-адрес злоумышленницы Труди, поэтому весь трафик Алисы, предназначенный для Боба, приходит, на самом деле, Труди. Та может организовать атаку типа «человек посередине», не мучаясь с установкой «крокодилов» на телефонной линии Алисы. Вместо этого она может заменить всего одну запись на сервере имен DNS. Это, согласитесь, куда проще.

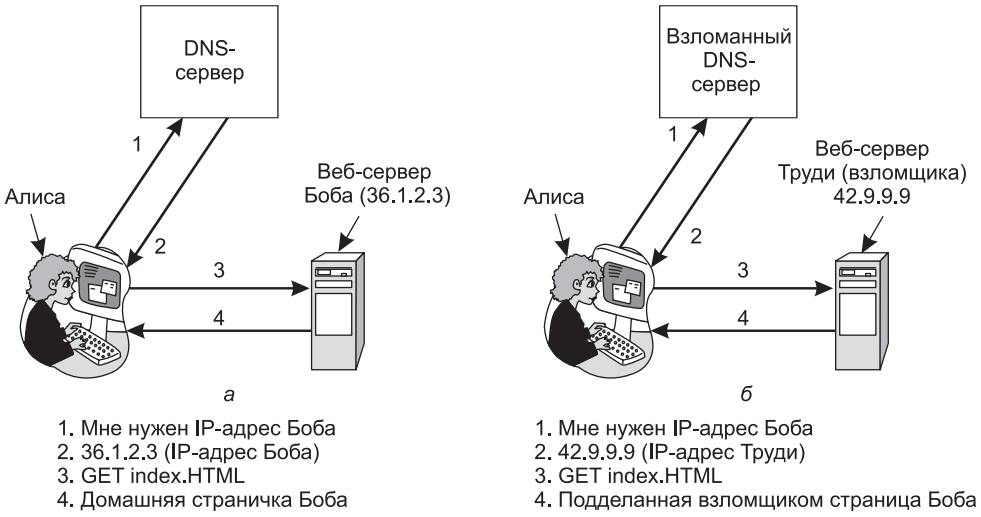


Рис. 8.42. Нормальная ситуация (*а*); атака со взломом DNS и изменением записи, относящейся к Бобу (*б*)

Как Труди удалось обмануть DNS? А это оказалось не таким уж сложным делом. Если не вдаваться в подробности, можно описать процесс так: Труди обманным путем заставляет DNS-сервер провайдера Алисы послать запрос для поиска адреса Боба. К несчастью, так как DNS использует UDP, сервер не может узнать, кто является реальным отправителем ответа. Труди использует это свойство, фальсифицируя ожидаемый ответ и тем самым занося неверные сведения об IP-адресе Боба в кэш DNS-сервера. Для простоты мы будем предполагать, что провайдер Алисы изначально не имеет сведений о веб-сайте Боба, *bob.com*. Если же такие сведения есть, злоумышленник может выждать, пока срок действия записи истечет, и попробовать еще раз (либо применить другие хитрости).

Труди начинает свою атаку с того, что посылает провайдеру Алисы запрос на поиск IP-адреса *bob.com*. Так как соответствующая запись отсутствует, сервер, в свою

очередь, опрашивает сервер домена верхнего уровня (*.com*). Но Труди опережает этот сервер и посылает ложный ответ, в котором сообщается, что IP-адрес *bob.com* якобы 42.9.9.9. Как мы знаем, в реальности это адрес Труди. Так как этот ответ приходит первым, данные из него заносятся в кэш сервера провайдера, а настоящий ответ, если он приходит позже, отвергается. Установка ложного IP-адреса называется **обманом DNS (DNS spoofing)**. А кэш, в котором хранится заведомо ложный IP-адрес, называется **отравленным кэшем (poisoned cache)**.

Надо сказать, что на практике все не так просто. Во-первых, провайдер Алисы все-таки проверяет наличие в ответе правильного адреса сервера верхнего уровня. Но Труди может написать в соответствующем поле что угодно и преодолеть эту преграду. Учитывая то, что адреса серверов верхнего уровня общедоступны, сделать это несложно.

Во-вторых, для того чтобы DNS-сервер мог понять, какому запросу соответствует ответ, во все запросы добавляются порядковые номера. Чтобы обмануть провайдера Алисы, Труди должна знать текущий порядковый номер. Самый простой способ узнать его — это зарегистрировать собственный домен, например *trudy-the-intruder.com*.

Предположим, что IP-адрес этого домена также 42.9.9.9. Труди создает DNS-сервер для этого домена: *dns.trudy-the-intruder.com*. Его IP-адрес тот же самый (42.9.9.9), поскольку оба домена расположены на одном и том же компьютере. Теперь надо заставить провайдера Алисы поинтересоваться DNS-сервером Труди. Сделать это несложно. Требуется лишь запросить, например, *foobar.trudy-the-intruder.com*, и серверу провайдера Алисы придется опросить сервер верхнего уровня, *.com*, и узнать у него, кто обслуживает новый домен Труди.

И вот теперь, когда запись *dns.trudy-the-intruder.com* занесена в кэш провайдера, можно спокойно начинать атаку. Труди запрашивает у провайдера Алисы *www.trudy-the-intruder.com*, а тот в ответ посылает на DNS-сервер Труди соответствующий запрос. Вот в этом-то запросе и содержится нужный злоумышленнице порядковый номер. Теперь Труди должна действовать без промедления: она ищет с помощью провайдера Алисы Боба и тут же отвечает на собственный вопрос, посылая фальшивку: «Адрес *bob.com*: 42.9.9.9». Этот подделанный ответ несет в себе порядковый номер, на единицу больше только что полученного. За время атаки она может послать еще одну фальшивку, с номером, на два больше полученного, а также еще около дюжины таких «ответов» с увеличивающимися номерами. Задача одного из них нам уже ясна. Остальные никому не нужны, их просто выкинут. После прибытия фальшивого ответа на запрос Алисы он будет помещен в кэш; к тому времени, когда доберется настоящий ответ, он будет отвергнут, так как сервер уже ничего не ожидает.

И вот Алиса ищет IP-адрес *bob.com* и узнает, что он равен 42.9.9.9. Как мы знаем, это адрес Труди, которая провела успешную атаку типа «человек посередине», не выходя из своей комнаты. Последовательность предпринятых ею шагов, пронумерованных в соответствии с рассмотренным выше вариантом, показана на рис. 8.43. Избежать атак описанного типа можно, заставив DNS-серверы использовать случайные идентификаторы в своих запросах вместо того, чтобы просто инкрементировать их. К сожалению, заткнув одну «дыру», мы обнаруживаем другую. Например, ID всего лишь шестнадцатибитовые, так что работать со всеми ними легко, когда компьютер совершает подстановки.

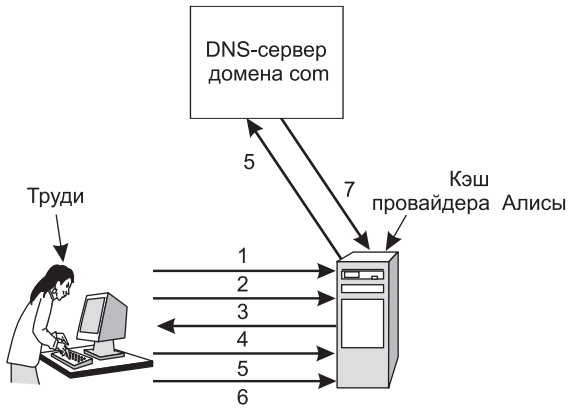


Рис. 8.43. Обман провайдера Алисы

Защита DNS

Настоящая проблема в том, что служба DNS разрабатывалась в те времена, когда Интернет был чисто исследовательской сетью, работавшей в нескольких сотнях университетов, и ни Алиса, ни Боб, ни Труди на этот праздник жизни приглашены не были. Вопрос защиты информации тогда еще не стоял; задача максимум была заставить Интернет функционировать. Но с годами среда, в которой приходилось выживать Интернету, сильно изменилась, поэтому в 1994 году IETF основала рабочую группу, задачей которой было защитить DNS. Этот (продолжающийся) проект известен под названием **DNSsec (DNS Security — защита DNS)**; первые результаты работы группы опубликованы в RFC 2535. К сожалению, DNSsec до сих пор не удается развернуть в больших масштабах, поэтому многие DNS-серверы продолжают подвергаться нападениям злоумышленников.

Концептуально система DNSsec очень проста. Она основана на шифровании с открытыми ключами. Каждая зона DNS (в терминах рис. 7.2) обладает парой ключей, а именно открытым и закрытым. Вся информация, отправляемая DNS-сервером, подписывается с помощью закрытого ключа зоны отправителя, поэтому аутентичность может быть запросто проверена принимающей стороной.

DNSsec предоставляет три основные услуги.

1. Подтверждение места отправления данных.
2. Распространение открытых ключей.
3. Аутентификацию транзакций и запросов.

Самой главной является первая услуга, с ее помощью проверяется то, что пришедшие данные были подтверждены их отправителем. Вторая услуга полезна для безопасного хранения и извлечения открытых ключей. Третья позволяет защититься от атак повторного воспроизведения и обмана сервера. Обратите внимание: секретность здесь не обеспечивается, этой задачи нет, поскольку вся информация DNS считается открытой. Так как процесс введения DNSsec в строй, скорее всего, будет продолжаться в течение нескольких лет, важно предоставить возможность общения между собой сер-

верам, снабженным системой защиты и не снабженным ею. Это неявно подразумевает, что протокол изменять нельзя. Рассмотрим теперь эту систему чуть более детально.

Записи DNS группируются в наборы, называемые **RRSet (Resource Record Set — набор записей ресурсов)**. В набор входят все записи с одинаковыми именами, классами и типами. Скажем, в наборе может быть несколько записей *A*, если имя DNS соответствует первичному и вторичному IP-адресам. Наборы расширяются за счет некоторых новых типов записей (обсуждаются ниже). Каждый RRSet хэшируется (например, с использованием SHA-1). Хэш подписывается при помощи закрытого ключа зоны (например, по алгоритму RSA). Единицей передаваемой клиентам информации является подписанный RRSet. Получив его, клиент может проверить, действительно ли для генерации подписи был взят закрытый ключ зоны отправителя. Если подпись корректна, данные принимаются. Так как каждый RRSet содержит собственную подпись, наборы можно кэшировать где угодно, даже на не слишком надежных серверах, не опасаясь за их судьбу.

Система DNSec вводит несколько новых типов записей. Первая из них — это запись *KEY*. В ней хранятся открытый ключ зоны, пользователя, хоста или другого принципала, криптографический алгоритм генерации подписи, наименование протокола передачи и еще несколько бит. Открытый ключ хранится в незащищенном виде. Сертификаты X.509 не используются из-за их громоздкости. В поле алгоритма рекомендуемое значение, соответствующее MD5/RSA, равно 1, для других комбинаций используются другие значения. Поле протокола может указывать на использование IPsec или другого протокола защиты соединений (если таковой вообще применяется).

Второй новый тип записей — *SIG*. В такой записи содержится подписанный хэш, сформированный в соответствии с алгоритмом, указанным в *KEY*. Подпись охватывает все записи RRSet, включая все записи *KEY*, однако не включая саму себя. Здесь также содержатся время начала и конца действия подписи, имя владельца подписи и некоторая дополнительная информация.

Система DNSsec устроена так, что закрытый ключ зоны может храниться в автономном режиме. Один или два раза в день содержимое базы данных зоны можно вручную переносить (например, записав компакт-диск) на машину, работающую в автономном режиме и хранящую закрытый ключ. Там можно сгенерировать подписи для всех наборов, и полученные таким образом записи *SIG* можно снова записать на компакт-диск и перенести на главный сервер. Таким образом, закрытый ключ можно хранить на компакт-диске, запертом в сейфе и вынимаемом только для того, чтобы подписать на автономной машине ежедневное обновление наборов типа RRSet. По окончании генерации подписей все копии ключа удаляются из памяти, а диск и компакт-диск возвращаются в сейф. Эта процедура превращает электронную защиту информации в физическую, что гораздо понятнее пользователям.

Метод предварительного подписания наборов значительно ускоряет процесс обработки запросов, так как отпадает необходимость в шифровании «на лету». Платой за это является большой объем дискового пространства, необходимого для хранения всех ключей и подписей в базах данных DNS. Из-за этого некоторые записи увеличиваются в размере десятикратно.

Получив подписанный RRSet, клиент должен применить открытый ключ зоны для расшифровки хэша, затем вычислить хэш самостоятельно и сравнить два значения.

В случае их соответствия данные считаются корректными. Тем не менее эта процедура не решает вопрос получения клиентом открытого ключа зоны. Одним из способов является запрос ключа у надежного сервера и передача его по защищенному соединению (например, при помощи IPsec).

Однако на практике предполагается, что у клиентов уже есть открытые ключи всех доменов верхнего уровня. Если Алиса пожелает посетить сайт Боба, она запросит у службы DNS набор RRSet для *bob.com*, в котором будет содержаться IP-адрес и запись *KEY* с открытым ключом Боба. RRSet будет подписан доменом верхнего уровня (*com*), поэтому Алиса запросто сможет проверить подлинность набора. Пример содержимого набора RRSet приведен в табл. 8.4.

Таблица 8.4. Пример набора RRSet для *bob.com*. Запись *KEY* содержит открытый ключ Боба. Запись *SIG* — это хэш *A* и *KEY*, подписанный сервером домена верхнего уровня (com) для проверки их аутентичности

Имя домена	Время жизни	Класс	Тип	Значение
bob.com.	86400	IN	A	36.1.2.3
bob.com.	86400	IN	KEY	3682793A7B73F731029CE2737D…
bob.com.	86400	IN	SIG	86947503A8B848F5272E53930C…

Теперь, вооружившись заверенной копией открытого ключа Боба, Алиса может узнать у DNS-сервера Боба IP-адрес *www.bob.com*. Этот RRSet будет подписан закрытым ключом Боба, поэтому Алиса сможет проверить подлинность подписи возвращенного Бобом набора. Если злоумышленнику каким-то образом удастся внедрить фальшивый RRSet в один из кэшей, Алиса заметит это, так как запись *SIG* будет неправильной.

Тем не менее DNSsec также предоставляет криптографический механизм для связывания ответов с соответствующими запросами, предотвращающий атаки типа той, что показана на рис. 8.43. Эта (необязательная) мера заключается в добавлении к ответу хэша запроса, подписанного закрытым ключом опрашиваемого. Поскольку Труди неизвестен закрытый ключ сервера верхнего уровня (домена *com*), она не сможет подделать ответ этого сервера на запрос провайдера Алисы. Она, конечно, может опередить настоящий ответ, но фальшивка будет замечена по неправильной подписи хэшированного запроса.

DNSsec поддерживает и некоторые другие типы записей. Так, для хранения сертификатов (например, стандарта X.509) можно использовать запись *CERT*. Зачем нужна такая запись? Дело в том, что есть желающие превратить DNS в инфраструктуру PKI. Случится это на самом деле или нет, пока еще неизвестно. На этом мы заканчиваем обсуждение DNSsec. Более подробную информацию вы найдете в RFC 2535.

8.9.3. SSL — протокол защищенных сокетов

Ну, что ж, защита имен ресурсов — это неплохое начало, однако этим не обеспечивается в полной мере безопасность во Всемирной паутине. Следующий шаг состоит в установлении безопасных соединений. Сейчас мы рассмотрим, как это делается. Все, что связано с безопасностью, — сложно, и эта тема не исключение.

Когда веб-технологии были впервые представлены широкой публике, они использовались для распространения статических страниц. Однако еще давным-давно некоторые компании задумались об использовании Всемирной паутины для выполнения финансовых транзакций, таких как покупка товаров по кредитным картам, онлайновые банковские операции, электронная торговля ценными бумагами. Для таких приложений требовалась организация защищенных соединений. В 1995 году тогдашний лидер среди производителей браузеров, корпорация Netscape Communications, в ответ на это представила систему безопасности под названием **SSL (Secure Sockets Layer — протокол защищенных сокетов)**. Соответствующее программное обеспечение, как и сам протокол, в наше время используется очень широко (в том числе программами Firefox, Safari, Internet Explorer), поэтому стоит рассмотреть SSL более детально.

Итак, SSL создает защищенное соединение между двумя сокетами, позволяющее:

1) клиенту и серверу договориться об используемых параметрах;
2) провести аутентификацию сервера клиентом;
3) организовать тайное общение;
4) обеспечить защиту целостности данных.

Все перечисленные пункты нам уже знакомы, поэтому мы не будем их комментировать.

Расположение SSL в структуре обычного стека протоколов показано на рис. 8.44. По сути дела, между прикладным и транспортным уровнями появляется новый уровень, принимающий запросы от браузера и отсылающий их по TCP для передачи серверу. После установки защищенного соединения основная задача SSL заключается в поддержке сжатия и шифрования. Если поверх SSL используется HTTP, этот вариант называется **HTTPS (Secure HTTP — защищенный HTTP)**, несмотря на то что это обычный протокол HTTP. Впрочем, возможно и отличие: скажем, доступ может осуществляться через новый порт (443) вместо стандартного (80). Кстати говоря, область применения SSL не ограничивается исключительно веб-браузерами, но это наиболее распространенное применение. Он также может обеспечивать взаимную аутентификацию.

Существует несколько версий протокола SSL. Ниже мы будем обсуждать только версию 3, так она распространена наиболее широко. SSL может обладать разными дополнительными функциями, среди которых наличие или отсутствие сжатия, тот или иной алгоритм шифрования, а также некоторые вещи, связанные с ограничениями экспорта в криптографии. Последнее в основном предназначено для того, чтобы можно было удостовериться, что серьезная криптография используется лишь тогда, когда оба конца соединения находятся в США. В других случаях длину ключа ограничивают 40 битами, что криптографы воспринимают как своего рода шутку. Однако Netscape должна была ввести это ограничение, чтобы получить лицензию на экспорт от правительства США.

SSL состоит из двух субпротоколов, один из которых предназначен для установления защищенного соединения, а второй — для использования этого соединения. Начнем с рассмотрения вопроса установления соединения. Работа субпротокола, занимающегося этим, показана на рис. 8.44. Все начинается с сообщения 1, в котором Алиса посылает Бобу запрос на установку соединения. В нем указывается версия SSL,

а также предпочтения Алисы относительно сжатия и алгоритмов шифрования. Также в нем содержится нонс R_A, который будет использован впоследствии.

Рис 8.44. Уровни (и протоколы), используемые обычным домашним браузером с SSL

Теперь наступает очередь Боба. В сообщении 2 он выбирает один из алгоритмов, поддерживаемых Алисой, и посылает собственный нонс R_B. В сообщении 3 он отсылает сертификат со своим открытым ключом. Если сертификат не подписан какой-нибудь уважаемой организацией, он также отправляет цепочку сертификатов, по которым Алиса может удостовериться в том, что сертификату Боба действительно можно доверять. Все браузеры, включая тот, что установлен у Алисы, изначально снабжаются примерно сотней открытых ключей, поэтому если среди присланных Бобом сертификатов встретится один из этих ключей, Алиса сможет по нему восстановить ключ Боба и проверить его. В этот момент Боб может прислать и другие сообщения (например, запрос на получение сертификата Алисы с ее открытым ключом). После окончания выполнения своей части протокола Боб посылает сообщение 4, в котором говорит, что настала очередь Алисы.

Алиса в ответ выбирает 384-разрядный **подготовительный ключ (premaster key)** и посылает его Бобу, зашифровав предварительно своим открытым ключом (сообщение 5). Настоящий ключ сеанса вычисляется при помощи подготовительного ключа и нонсов обеих сторон. Это достаточно сложная процедура. После получения сообщения 5 и Алиса, и Боб могут вычислить ключ сеанса. Для этого Алиса просит Боба переключиться на новый шифр (сообщение 6), а также сообщает о том, что она считает субпротокол установления соединения оконченным (сообщение 7). Боб соглашается с ней (сообщения 8 и 9).

Однако, несмотря на то что Алиса знает, кто такой Боб, последний Алису не знает (если только у нее нет открытого ключа и сертификата к нему, что довольно необычно для обычного физического лица). Поэтому первым сообщением для Алисы запросто может оказаться просьба пройти регистрацию, используя полученные ранее имя пользователя и пароль. Впрочем, протокол регистрации в системе выходит за область полномочий SSL. Так или иначе, по окончании этой серии запросов-подтверждений может начинаться передача данных.

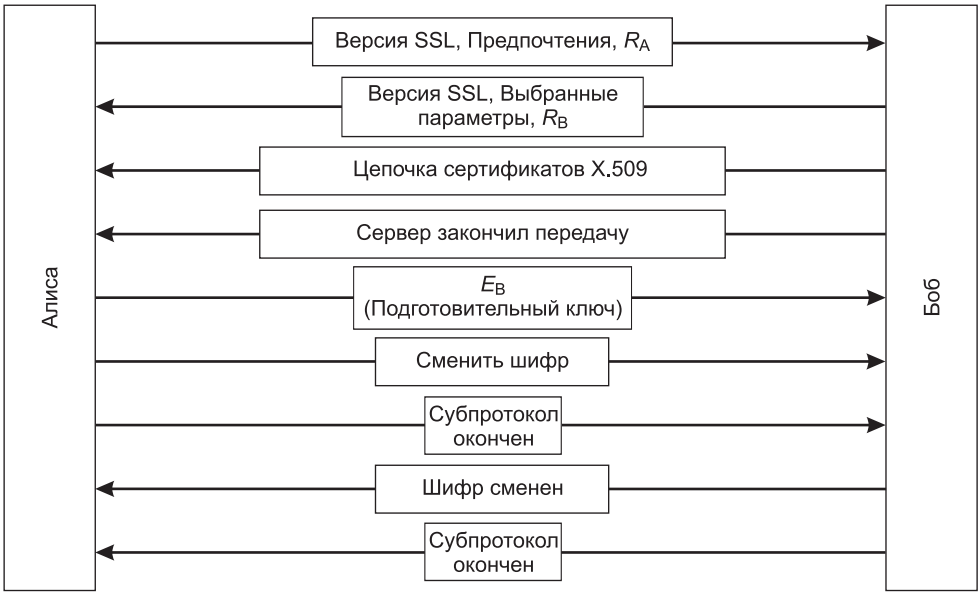


Рис. 8.45. Упрощенный вариант субпротокола SSL установления соединения

Как уже говорилось, SSL поддерживает разнообразные криптографические алгоритмы. Наиболее сильный из них использует для шифрования тройной DES с тремя отдельными ключами и SHA-1 для обеспечения целостности данных. Такое сочетание алгоритмов работает довольно медленно, поэтому применяется в основном при выполнении банковских операций и в других приложениях, в которых требуется высокий уровень защиты. В обычных приложениях электронной коммерции для шифрования применяется RC4 с 128-разрядным ключом, а для аутентификации — MD5. В качестве исходных данных RC4 передается 128-разрядный ключ, который разрастается во много раз при работе алгоритма. Это внутреннее число используется для создания ключевого потока. Последний суммируется по модулю 2 с открытым текстом, в результате чего получается обычный потоковый шифр, как было показано на рис. 8.12. Экспортные версии алгоритма также работают с алгоритмом RC4 и 128-разрядным ключом, однако 88 из этих разрядов делаются открытыми, что позволяет довольно быстро взломать шифр.

Для реальной передачи данных используется второй субпротокол, показанный на рис. 8.46. Сообщения, поступающие от браузера, разбиваются на единицы данных размером до 16 Кбайт. Если сжатие данных включено, каждая из этих единиц независимо сжимается. Затем по двум нонсам и подготовительному ключу вычисляется закрытый ключ, который объединяется со сжатым текстом, и результат хэшируется по согласованному алгоритму (чаще всего MD5). Хэш добавляется к каждому фрагменту в виде **MAC (Message Authetication Code — код аутентификации сообщения)**. Этот сжатый фрагмент вместе с MAC кодируется согласованным алгоритмом с симметричным ключом (обычно это суммирование по модулю 2 с ключевым потоком RC4). Наконец, присоединяется заголовок фрагмента, и фрагмент передается по TCP-соединению.

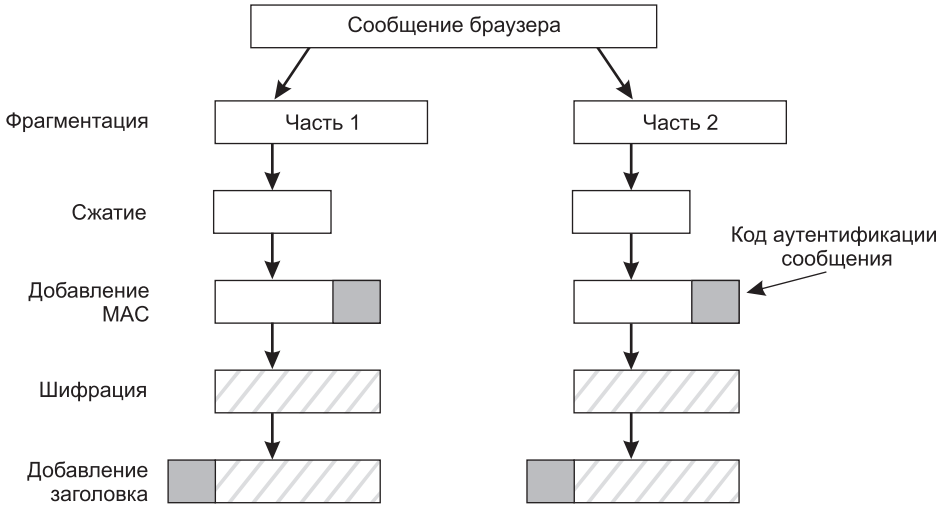


Рис. 8.46. Передача данных при использовании SSL

Следует остерегаться следующего подводного камня: уже говорилось о том, что RC4 имеет некоторые слабые ключи, которые довольно просто взламываются, поэтому SSL с RC4 — это довольно шаткая основа (Fluhrer и др., 2001). Браузеры, позволяющие пользователю выбирать тот или иной шифр, лучше всего настраивать на постоянное использование тройного алгоритма DES со 168-разрядными ключами и SHA-1, невзирая на то, что такая комбинация работает еще медленнее, чем RC4 + MD5. Или можно использовать браузеры, поддерживающие преемника SSL, которого мы коротко опишем.

С SSL связана еще одна проблема: у принципалов может не быть сертификатов, а даже если они есть, далеко не всегда производится проверка соответствия ключей и сертификатов.

В 1996 году корпорация Netscape Communications направила SSL на стандартизацию в IETF. Результатом стал стандарт **TLS (Transport Layer Security — защита транспортного уровня)**. Он описан вRFC 5246.

TLS был построен на SSL, версия 3. В SSL при создании стандарта TLS было внесено не так уж много изменений, однако их оказалось достаточно для того, чтобы SSL версии 3 и TLS стали несовместимыми. Например, в целях усиления ключа был изменен способ вычисления ключа сеанса по подготовительному ключу и нонсам. Из-за этой несовместимости большинство браузеров применяют оба протокола, и TLS превращается обратно в SSL, если это необходимо. Это называется SSL/TLS. Впервые TLS был применен в 1999 году, версия 1.2 появилась в августе 2008 года. Она включает поддержку более сильных наборов шифров (в том числе AES). SSL продолжает удерживать сильные рыночные позиции, хотя, вероятно, TLS постепенно его заменит.

8.9.4. Безопасность переносимых программ

Именование ресурсов и соединения — это две области, которые, несомненно, тесно связаны с защитой информации во Всемирной паутине. Однако существуют и другие,

не менее важные вопросы, связанные с той же темой. Поначалу веб-страницы представляли собой полностью статические HTML-файлы и не содержали исполняемый код. Теперь же на веб-страницах очень часто встречаются небольшие программы: Java-апплеты, управляющие элементы ActiveX, скрипты JavaScript. Загрузка и выполнение таких **переносимых программ (mobile code)**, очевидно, связана с большим риском для безопасности. Были разработаны различные методы, направленные на минимизацию этого риска. Ниже мы обозначим некоторые вопросы, связанные с проблемами, возникающими из-за переносимых программ, и некоторые подходы к обращению с ними.

Безопасность Java-апплетов

Java-апплеты — это небольшие программы на языке Java, откомпилированные в машинный язык со стековой организацией под названием **JVM (Java Virtual Machine — виртуальная машина Java)**. Такие программы могут размещаться на веб-странице и загружаться вместе с ней. После загрузки страницы апплеты обрабатываются интерпретатором JVM в браузере, как показано на рис. 8.47.

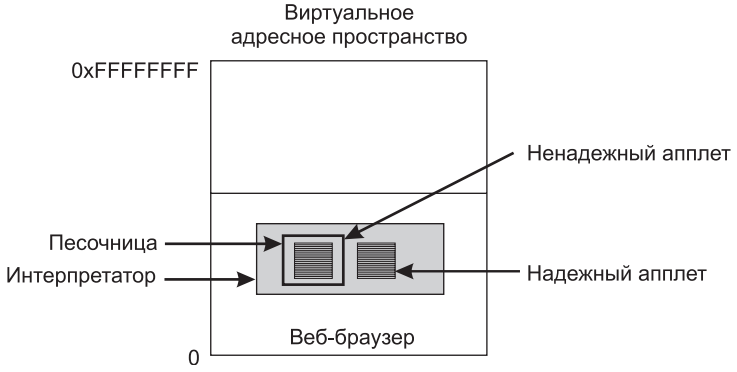


Рис. 8.47. Апплеты могут интерпретироваться веб-браузером

Преимущество интерпретируемого кода перед компилируемым состоит в том, что перед исполнением изучается каждая инструкция. Это дает интерпретатору возможность проверить состоятельность адреса инструкции. Кроме того, системные вызовы также перехватываются и интерпретируются. Как именно они обрабатываются, зависит от политики защиты информации. Например, если апплет надежный (например, он был создан на локальном диске), его системные вызовы могут обрабатываться без дополнительных проверок. Если же апплет не может считаться надежным (например, он был загружен из Интернета), его можно поместить в так называемую **песочницу (sandbox)**, регулирующую его поведение и пресекающую его попытки использовать системные ресурсы.

Если апплет пытается захватить системный ресурс, вызов передается монитору безопасности, который может разрешить или запретить данное действие. Монитор исследует вызов с точки зрения локальной политики защиты информации и затем принимает нужное решение. Таким образом, можно предоставить апплетам доступ

к некоторым (но не ко всем) ресурсам. К сожалению, в реальной жизни такая модель работает плохо, в ней постоянно возникают ошибки.

ActiveX

Управляющие элементы ActiveX — это двоичные программы, рассчитанные на процессор x86, которые можно внедрять в веб-страницы. Когда на странице встречается такая программа, производится проверка необходимости ее выполнения, и в случае положительного ответа она запускается. Эти программы не интерпретируются и не помещаются в песочницы, поэтому они обладают такими же возможностями, как обычные пользовательские программы, и в принципе могут нанести большой вред. Таким образом, вся защита информации в данном случае сводится к вопросу о том, стоит ли запускать управляющий элемент. Можно сделать вывод, что вся эта конструкция — одна большая дыра в системе безопасности.

Для принятия таких решений корпорацией Microsoft был выбран метод, базирующийся на **подписях кода (code signing)**. Суть в том, что каждый элемент ActiveX снабжается цифровой подписью, а именно, хэшем кода, подписанным его создателем с использованием открытого ключа. Когда браузер встречает управляющий элемент, он вначале проверяет правильность подписи, убеждаясь в том, что код не был заменен по дороге. Если подпись корректна, браузер проверяет по своим внутренним таблицам, можно ли доверять создателю программы. Возможно, про самого создателя ничего не известно, но существует цепочка заверений, ведущая к какому-либо известному своей надежностью разработчику. Если создатель надежный, программа выполняется, в противном случае игнорируется. Система, созданная Microsoft для проверки управляющих элементов ActiveX, называется **Authenticode**.

Полезно противопоставлять друг другу подходы Java и ActiveX. В первом случае не производятся никакие попытки установить авторство апплета.

Вместо этого используется интерпретатор, который запрещает апплету совершать определенные нежелательные действия. Что касается метода подписания кода, то в этом случае, напротив, поведение программы во время ее выполнения никак не отслеживается. Если она была получена из проверенного источника и по дороге не была изменена, она просто запускается. Проверка самого кода не осуществляется. Если программист намеренно написал код, форматирующий жесткий диск и стирающий флэш-память компьютера и при этом он считается проверенным и надежным программистом, то код выполнится и выведет из строя компьютер (если только в браузере не отключены управляющие элементы ActiveX).

Многие считают, что доверять неизвестным производителям программного обеспечения несколько легкомысленно. Чтобы доказать это, один программист из Сиэтла основал свою компанию и добился получения сертификата надежности, что не так уж сложно. После этого он написал управляющий элемент ActiveX, который всего-навсего выключал компьютер. Он распространил свою программу весьма широко, и она выключила не одну тысячу компьютеров. Впрочем, машины после этого можно было запросто включить заново, поэтому никакого ущерба такой управляющий элемент нанести не мог. Цель проекта состояла в том, чтобы указать миру на наличие проблемы. Официальная реакция выразилась в отзыве сертификата для данного конкретного

управляющего элемента, и на этом инцидент был исчерпан. Но проблема-то решена не была, и нечистые на руку программисты могли продолжать использовать эту дыру в защите Garfinkel и Spafford, 2002. Поскольку нет никакой возможности проследить за деятельностью всех компаний, пишущих переносимые программы, вскоре метод подписания кода может представлять собой довольно серьезную угрозу.

JavaScript

В JavaScript вообще отсутствует какая-либо официальная модель системы защиты информации, зато существует длинная история неудачных попыток ее внедрения. Каждый производитель пытается придумать что-нибудь свое. Например, в Netscape Nvigator версии 2 было реализовано нечто подобное Java-модели, а уже в четвертой версии прослеживаются черты модели подписей кода.

Суть проблемы в том, что чужеродной программе разрешается выполнять какие-то действия. Это может привести к непредсказуемым последствиям. С точки зрения безопасности это то же самое, что позвать в гости вора и пытаться внимательно следить за тем, чтобы он не проник из кухни в гостиную. Если произойдет что-нибудь неожиданное, а вы в этот момент отвлечетесь, может случиться что угодно. Аргументом в защиту переносимых программ служит то, что с их помощью легко реализуется флэш-графика и быстрое взаимодействие с пользователем. Создатели веб-сайтов обычно считают, что это гораздо важнее, чем защита информации, особенно когда дело касается какого-нибудь чужого компьютера.

Расширения и дополнения к браузерам

Так же как в веб-страницы можно добавить код, точно так же можно сделать дополнение к браузерам (**browser extension** — расширение браузера; **add-on** — дополнение к браузеру, «аддон»; **plug-in** — подключаемый модуль, «плагин»), и рынок этих дополнений растет. Эти дополнения представляют собой компьютерные программы, которые расширяют функциональность веб-браузеров. Подключаемые модули часто предлагают какую-то определенную функцию по интерпретации или показу определенного контента, например документов PDF или флэш-анимации. Расширения и дополнения обеспечивают новые функции браузера, такие как лучшее управление паролями, новые способы управления страницами (например, помечая их) или более удобную интернет-торговлю.

Установить дополнение, расширение или подключаемый модуль очень просто: достаточно просто найти нужную вам программу в сети и пойти по ссылке, чтобы ее установить. Все эти программы написаны в разных средах, в зависимости от того, к какому браузеру они будут применяться. Однако в первом приближении они становятся частью надежной вычислительной базы браузера. То есть если устанавливается код с ошибками, сам браузер может начать работать с ошибками.

Здесь существует две возможные ловушки. Первая — программа может начать наносить урон, например собирать личную информацию и отсылать ее на удаленный сервер. Браузер только будет знать, что пользователь установил расширение именно для этой цели. Вторая проблема состоит в том, что модули расширения предоставляют

браузеру возможность считывать новые типы информации. Эта информация может быть написана на самостоятельном языке программирования. PDF и Flash — хорошие примеры. Когда пользователи просматривают страницы с PDF и Flash, модули расширения в их браузере выполняют коды PDF и Flash. Лучше бы эти коды были безопасными, но существуют некоторые уязвимости. По всем этим причинам дополнения и модули расширения лучше ставить по мере необходимости и приобретать только у надежных продавцов.

Вирусы

Вирусы — это своеобразная форма переносимого кода. Только в отличие от приведенных выше примеров, запускать такие программы никто не хочет. Основное отличие вирусов от обычных переносимых программ заключается в том, что они воспроизводят сами себя. Когда в систему проникает вирус (с веб-страницы, во вложении электронного письма или как-то еще), для начала он заражает исполняемые программы, хранящиеся на диске. При запуске какой-либо из этих программ управление передается вирусу, а тот обычно пытается распространить свое действие еще и на другие машины, например, рассылая самого себя по электронной почте всем адресатам из адресной книги жертвы. Некоторые вирусы заражают загрузочный сектор жесткого диска, поэтому вирус активируется при загрузке машин.

Вирусы в какой-то момент стали представлять собой крупномасштабную проблему для Интернета и принесли многомиллиардные убытки. Какого-либо простого решения проблемы не существует. Возможно, положение может спасти создание нового поколения операционных систем, базирующихся на защищенных микроядрах, и сплоченность пользователей, процессов и ресурсов.

8.10. Социальный аспект

Интернет и технологии защиты информации — это те области, в которых очень тесно сплелись социальные вопросы, государственная политика и технологии. Ниже мы кратко рассмотрим три проблемы: конфиденциальность, свободу слова и авторские права. Совершенно очевидно, что в рамках этой книги мы сможем дать лишь поверхностное описание этой темы. Более подробную информацию следует искать в (Anderson, 208a; Garfinkel и Spafford, 2002; Schneier, 2004). Многие материалы можно прочитать в Интернете. Достаточно лишь набрать в поисковой машине «конфиденциальность» (privacy — для получения информации на английском языке), «цензура» (censorship) или «авторские права» (copyright). Также можно посмотреть сайт этой книги: *www.pearsonhighered.com/tannenbaum*.

8.10.1. Конфиденциальность

Имеют ли люди право на секреты? Хороший вопрос. Четвертая поправка к конституции США запрещает правительственным организациям без особой нужды интересоваться намерениями граждан, их жильем и личными бумагами. Ограничен перечень

обстоятельств, при которых этот запрет может быть нарушен. Таким образом, вопрос конфиденциальности стоит на повестке дня уже более 200 лет, по крайней мере, в США.

Что изменилось за последнее десятилетие? Правительство получило возможность с невиданной легкостью шпионить за гражданами, а граждане — с не меньшей легкостью предотвращать шионаж. В XVIII веке для получения доступа к личным бумагам гражданина требовалось выслать к нему в имение полицейского, которому нужно было в любую непогоду доскакать на коне, претерпевая всевозможные лишения, которые нередки в долгом пути, — и все это для того, чтобы прочесть один чужой листок бумаги. В наши дни телефонные компании и интенет-провайдеры обеспечивают всех, кто может предъявить соответствующий ордер, подслушивающими устройствами. С их помощью задача полицейских сильно облегчается, к тому же нет риска выпасть из седла, заснув в пути.

Тем не менее использование криптографии в значительной степени меняет дело. Любой желающий может озаботиться загрузкой и установкой PGP, генерированием хорошо защищенного, надежного ключа, и в результате он получит уверенность в том, что никто во Вселенной не сможет прочесть его электронную почту, независимо от наличия у него ордера на обыск. Правительства прекрасно это понимают и им, разумеется, это сильно не нравится. В реальности конфиденциальность означает, что уполномоченным органам очень трудно следить за преступниками всех мастей, а также за журналистами и политическими оппонентами. Неудивительно, что многие правительства запрещают использование и экспорт криптографии. Во Франции, к примеру, до 1999 года любая негосударственная криптография была просто запрещена, если только государству не предоставлялись все используемые ключи.

Франция в этом деле не была одинока. В апреле 1993 года правительство США объявило о своем желании создать аппаратный криптопроцессор (**clipper chip**) и сделать его стандартным для применения в любых сетевых коммуникациях. Таким образом, как было заявлено, граждане получат гарантированную конфиденциальность. Вместе с тем, упоминалось о том, что правительство будет иметь возможность расшифровывать весь трафик таких криптопроцессоров при помощи специальной технологии (**key escrow**), позволяющей правительству получать доступ ко всем ключам. Однако были даны обещания использовать эту возможность только при наличии соответствующей санкции. Понятно, что такое заявление вызвало большой фурор: сторонники конфиденциальности осуждали весь план от начала до конца, а чиновники, выступающие в поддержку этого начинания, были восхищены предложением правительства. Тем не менее правительство почему-то сдало позиции и отказалось от собственной идеи.

Огромное количество материалов, посвященных конфиденциальности цифровой информации, доступно на веб-сайте Electronic Frontier Foundation (*www.eff.org*).

Анонимные рассылки

PGP, SSL и другие технологии позволяют устанавливать между двумя сторонами защищенные, аутентифицированные соединения, не подверженные вмешательству третьих сторон. Однако иногда конфиденциальность лучше всего обеспечивается как раз *отсутствием* аутентификации, то есть, по сути дела, установлением анонимных

соединений. Анонимность востребована как при передаче сообщений между двумя пользователями, так и в сетевых телеконференциях.

Рассмотрим некоторые примеры. Во-первых, политические диссиденты, живущие при авторитарном режиме, могут захотеть во избежание репрессий общаться анонимно. Во-вторых, различные нарушения во многих коммерческих, образовательных, правительственных и других организациях зачастую выявляются не без помощи доносчиков, которые желают оставаться неизвестными. В-третьих, приверженцы нетрадиционных (а значит, как правило, порицаемых) социальных, политических или религиозных убеждений видят одну из немногих возможностей общения в телеконференциях (или электронной почте), где они могут скрывать свои истинные имена. В-четвертых, многие предпочитают обсуждать алкоголизм, душевные заболевания, сексуальные проблемы, проблемы жестокого обращения с детьми или отношение к преследуемым меньшинствам в телеконференциях, где они могут оставаться анонимными. Кроме того, конечно, существует масса иных примеров.

Рассмотрим один конкретный пример. В 1990-х годах некоторые критики одной нетрадиционной религиозной секты опубликовали свои взгляды в конференции USENET с помощью **анонимной рассылки (anonymous remailer)**. Сервер позволял пользователям создавать псевдонимы и посылать на него электронные письма, которые затем рассылались от имени выбранного псевдонима. В итоге не было возможности понять, кто является настоящим автором писем. Некоторые из этих статей были разоблачениями, в состав которых, по мнению представителей секты, входили коммерческие тайны и документы, защищенные авторским правом. В ответ на эти разоблачения секта подала в суд, жалуясь на раскрытие коммерческих тайн и нарушение закона об авторском праве. И то и другое в том округе, где находился сервер, считалось преступлением. Последовал суд, и владельцев сервера вынудили раскрыть истинные имена тех, кто скрывался под псевдонимами и писал разоблачения (кстати, это был не первый прецедент, связанный с недовольством церкви раскрытием ее тайн: Уильям Тиндэйл (William Tyndale был в 1536 году сожжен за перевод Библии на английский).

Значительная часть интернет-сообщества была сильно возмущена таким грубым нарушением принципов конфиденциальности. Все были согласны и с тем, что владелец анонимной рассылки, хранившей таблицу соответствия настоящих электронных адресов и псевдонимов (это было названо анонимной рассылкой первого типа), был не прав. Этот случай стимулировал развитие анонимных рассылок, которые могли бы противостоять таким атакам со стороны суда.

Рассылки нового типа, часто называемые **шифрованными панковскими рассылками (cypherpunk remailer)**, работают следующим образом. Пользователь создает электронное письмо с обычными заголовками RFC 822 (разумеется, отсутствует *From*), шифрует его открытым ключом рассылки и отправляет на сервер. Там от него отрезаются заголовки RFC 822, содержимое расшифровывается, и сообщение рассылается подписчикам. В рассылке нет никаких учетных записей, не ведутся никакие журналы, поэтому даже в случае конфискации сервера никаких следов прошедших через него писем обнаружено не будет.

Многие пользователи, которые особенно сильно озабочены проблемой собственной анонимности, прогоняют свои сообщения через цепочки анонимных рассылок, как показано на рис. 8.48. В данном примере Алиса хочет послать Бобу действительно

очень-очень анонимное поздравление с Днем св. Валентина. Для этого она использует три анонимные рассылки. Она сочиняет письмо *M* и вставляет заголовок, содержащий адрес электронной почты Боба. Затем все это сообщение шифруется открытым ключом рассылки 3, E_3 (показано горизонтальной штриховкой). К этому прибавляется заголовок с электронным адресом рассылки 3 (передается открытым текстом). В итоге получается сообщение, показанное между рассылками 2 и 3 на рисунке.

На этом история сообщения не заканчивается. Оно шифруется открытым ключом рассылки 2, E_2 (показано вертикальной штриховкой), и дополняется открытым заголовком, содержащим электронный адрес рассылки 2. Получившееся в итоге сообщение показано на рисунке между рассылками 1 и 2. Затем Алиса шифрует свое сообщение открытым ключом рассылки 1, E_1, добавляет адрес этой рассылки и, наконец, отправляет его. Конечное состояние сообщения показано на рисунке справа от Алисы.

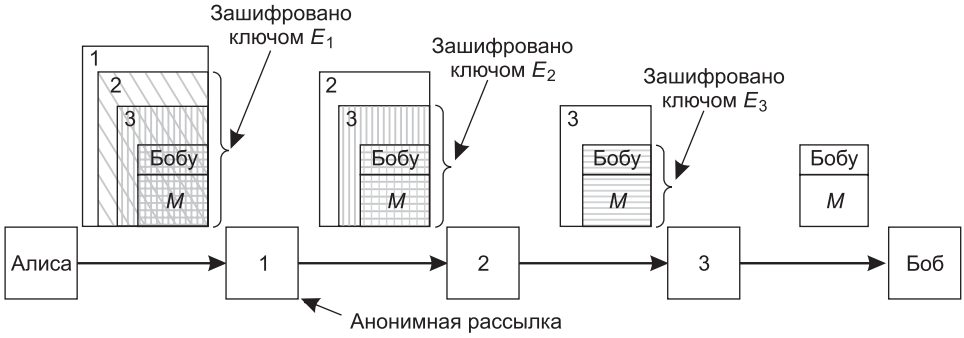


Рис. 8.48. Использование трех анонимных рассылок для передачи письма Алисы Бобу

Когда письмо Алисы достигает рассылки 1, от него отрезается внешний заголовок. Тело сообщения расшифровывается и пересылается в рассылку 2. Аналогичные шаги производятся и на двух других серверах.

Несмотря на то что восстановить путь конечного сообщения до Алисы и так очень сложно, многие рассылки принимают еще и дополнительные меры предосторожности. Например, они могут задерживать сообщения на какой-то случайный промежуток времени, добавлять или удалять всякий мусор в конце сообщения, переставлять сообщения местами, в общем, делать все возможное для того, чтобы запутать тех, кто отслеживает трафик и пытается понять, кто является автором того или иного сообщения, прошедшего через анонимную рассылку. Описание системы, реализующей в настоящее время описанные выше идеи, можно найти в (Mazières и Kaashoek, 1998).

Сфера применения принципов анонимности не ограничивается одной электронной почтой. Существуют также услуги, позволяющие анонимно просматривать интернет-материалы, используя ту ж самую форму пути со слоями, где один узел знает только следующий узел в цепочке. Такой метод называется **луковой маршрутизацией (onion routing)**, так как каждый узел снимает определенный слой с луковицы, чтобы определить, куда дальше передавать пакет. Tor — хороший пример такой системы (Dingledine и др., 2004). Пользователь может настроить свой браузер на использование такой услуги в качестве прокси. После этого все HTTP-запросы направляются по адресу, принадлежащему этому сервису, который производит запрос страницы за пользовате-

ля. Если на сервере, обеспечивающем анонимность, не хранятся журналы активности, никто не сможет определить, кто на самом деле запрашивал страницу.

8.10.2. Свобода слова

Конфиденциальность связана с проблемой сокрытия от посторонних глаз информации, не подлежащей обнародованию. Вторым ключевым социальным аспектом является, несомненно, свобода слова и ее противоположность — цензура. В этом случае правящие органы пытаются ограничить спектр информации, которую граждане могут читать и публиковать. Всемирная паутина с ее миллионами страниц — это настоящий рай для цензуры. В зависимости от типа и идеологии режима, в список запрещенных к просмотру материалов могут помещаться страницы, содержащие что-либо из перечисленного ниже.

1. Материалы, которые нельзя показывать детям и подросткам.
2. Материалы, пропагандирующие ненависть к каким-либо этническим, религиозным, сексуальным или другим группам.
3. Информация о демократии и демократических ценностях.
4. Описания исторических событий, не совпадающие с официальной версией.
5. Руководства по взлому разного рода замков, созданию оружия, шифрованию сообщений и т. д.

Плохой, негодный сайт проще всего запретить к просмотру.

Иногда результаты такой политики оказываются неожиданными. Например, некоторые публичные библиотеки установили у себя веб-фильтры, не пропускающие порнографические сайты и таким образом делающие содержимое Паутины безопасным для просмотра детьми. Фильтры перед выводом страниц сверяют адреса со своими «черными списками», к тому же проверяют их на наличие бранных слов. Однажды в округе Лаудаун, штат Вирджиния, фильтр заблокировал поиск информации по раку молочной железы, так как запрос содержал слово «breast» (женская грудь, молочная железа). Клиент возбудил дело против правительства округа. В то же время был и другой случай: в Ливермор, штат Калифорния, когда один родитель подал в суд на публичную библиотеку за то, что там *не был* установлен фильтр, и он застал своего 12-летнего сына за просмотром порнографического сайта. Так что же библиотеке делать?

Многие никак не могут понять, что Всемирная паутина — действительно всемирная. Она охватывает весь земной шар. В разных странах существуют разные взгляды на то, что должно, а чего не должно быть в Сети. Например, в ноябре 2000 года французский суд постановил, что Yahoo!, корпорация, находящаяся в Калифорнии, должна запретить доступ к аукциону памятных вещей нацистов для французских пользователей, так как обладание такой информацией идет вразрез с французским законодательством. Yahoo! апеллировала к суду США, который решил спор в пользу корпорации, однако в целом проблема того, законы какой страны должны применяться в Интернете, остается актуальной.

Попробуйте представить себе, что случится, если какой-нибудь суд штата Юта вынесет решение о том, что Франция должна заблокировать сайты, посвященные винам,

так как распространение подобной информации нарушает строгие законы штата, касающиеся алкогольной продукции? Точно так же Китай может сделать табуированными все сайты, на которых рассказывается про демократию, так как это не в интересах Поднебесной. Должны ли иранские законы о религии применяться в либеральной Швеции? Может ли Саудовская Аравия заблокировать сайты, защищающие права женщин? Становится понятно, что эта проблема подобна ящику Пандоры и способна породить множество вопросов.

Ценное замечание высказывает Джон Гилмор (John Gilmore): «Сеть воспринимает цензуру как разрушенный участок дороги и идет в обход». Конкретная реализация этой мысли называется **службой вечности (eternity service)** (Anderson, 1996). Ее цель — гарантировать, что однажды опубликованные материалы не исчезнут и не будут переписаны заново, как было принято в Советском Союзе во времена Сталина. Пользователь службы вечности должен лишь указать, в течение какого срока следует обеспечивать сохранность информации, заплатить пропорциональную сроку и объемам информации сумму и загрузить данные на сервер. После этого никто, включая самого пользователя, не сможет удалить или отредактировать размещенные на сервере службы вечности материалы.

Как такую услугу реализовать на практике? Проще всего организовать равноранговую (пиринговую) систему, в которой документы будут размещаться на десятках серверов участников проекта, каждый из которых будет получать свою долю вознаграждения, что послужит стимулом для их вступления в проект. Серверы должны располагаться в самых разных местах и под разной юрисдикцией, что обеспечит максимальную устойчивость системы. Списки 10 выбранных случайным образом серверов следует хранить в тайне в разных местах, чтобы в случае неудачи, произошедшей с одним из них, могли выжить другие. Любые государственные органы, помешанные на уничтожении неугодной информации, никогда не смогут быть до конца уверенными в том, что они нашли все копии. Кроме того, систему можно сделать самовосстанавливающейся, в том смысле, что в случае прихода известия об уничтожении каких-то экземпляров документов, держатели остальных копий попытаются найти новые места хранения на замену выбывшим из строя.

Служба вечности была первой попыткой противостояния цензуре в Сети. С тех пор было высказано много различных идей на эту тему, и некоторые из них даже нашли свое воплощение. Были добавлены новые возможности, такие как шифрование, анонимность, отказоустойчивость. Зачастую документы разбивают на несколько фрагментов и хранят их на нескольких серверах. Среди таких систем можно упомянуть Freenet (Clarke и др., 2002), PASIS (Wylie и др., 2000) и Publius (Waldman и др., 2000). Еще одна разработка описана в (Serjantov, 2002).

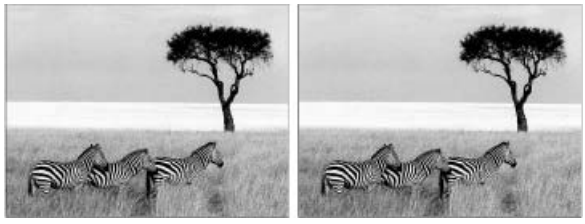
Все больше и больше стран пытаются контролировать экспорт таких неосязаемых вещей, как веб-сайты, программное обеспечение, научные документы, электронная почта, телефонные службы помощи и т. п. Даже в Великобритании, славящейся своими вековыми традициями поддержки свободы слова, появляются строгие законы, которые, к примеру, определяют техническую дискуссию между британским профессором и иностранным студентом в Кембриджском университете как предмет экспорта, подлежащий государственному лицензированию (Andreson, 2002). Очевидно, что многие люди считают, что такая политика крайне противоречива.

Стеганография

В странах, где цензура применяется особенно широко, всегда существуют диссиденты, использующие свои методы обхода этой цензуры. Криптография, конечно, позволяет (не всегда вполне законно) посылать секретные сообщения так, чтобы никто не смог узнать их смысл, однако, если государство считает Алису своим врагом, один тот факт, что она общается с Бобом, может и его поставить в положение врага государства. Таким образом политики, обычно не сильно владеющие математикой, понимают и применяют принцип транзитивности. Выручить могут анонимные рассылки, но и их местное правительство может запретить, и тогда для отправки сообщения за границу понадобится экспортная лицензия. Таким образом, анонимные рассылки — это тоже не панацея. Однако Всемирная паутина всегда найдет выход из положения.

Люди, которым требуется секретное общение, зачастую пытаются скрыть сам факт общения. Наука, занимающаяся сокрытием сообщений, называется **стеганографией** (**steganography**, не путать со стенографией!), от греческого слова, которое можно перевести как «защищенное письмо». Сами древние греки первыми и начали использовать этот метод. Геродот описывал своеобразный метод секретного общения военачальников: посыльному брили волосы на голове, на затылке рисовали татуировку с секретным сообщением и ждали, пока волосы снова отрастут, после чего отправляли посыльного в путь. Современные технологии базируются на той же концепции, разве что пропускная способность стала выше, а задержки — ниже.

В качестве примера рассмотрим рис. 8.49, *а*. На этой фотографии, сделанной одним из авторов в Кении, изображены три зебры и акация. Однако она привлекательна не только с эстетической точки зрения. Дело в том, что рис. 8.49, *б* включает в себя внедренный полный текст пяти самых известных пьес Шекспира: *Гамлет*, *Король Лир*, *Макбет*, *Венецианский купец* и *Юлий Цезарь* — все тексты вместе занимают более 700 Кбайт.



а *б*

Рис. 8.49. Три зебры и дерево (*а*). Три зебры, дерево и полный текст пяти пьес Вильяма Шекспира (*б*)

Как же это сделано? Стеганографический канал работает следующим образом. Размер исходного изображения равен 1024 × 768 пикселей. Каждый пиксель представлен тремя 8-битными числами, каждое из которых отвечает за свою составляющую цвета (существуют красный, зеленый и синий каналы интенсивности цвета). Результирующий цвет получается линейной суперпозицией интенсивностей трех цветов. При методе стеганографического шифрования младший бит каждого из трех указанных чисел заменяется нужным значением секретного канала. Таким образом, в каждом

пикселе помещаются три бита секретной информации: один в канале интенсивности красного цвета, один — зеленого и один — синего. В нашем изображении поместится $1024 \times 768 \times 3$ бит или 294 912 байт секретной информации.

Полный текст пяти пьес Шекспира с небольшим предисловием занимает 734 891 байт. Текст можно сжать до 274 Кб с помощью любого архиватора. Затем зашифровать эти сжатые данные с использованием алгоритма IDEA, после чего разместить данные в младших битах значений интенсивностей цветов. Как видите (вернее, это как раз остается невидимым), присутствие текстовой информации совершенно незаметно. На большой, полноцветной фотографии это также незаметно. Глаз человека просто не в состоянии отличить оттенки, выраженные 21-разрядным числом, от 24-разрядных оттенков.

Черно-белые фотографии, приведенные на рис. 8.49, дают неполное представление о возможностях стеганографии. Более эффектный вариант рис. 8.49, *б* продемонстрирован в полноцветной версии, которую можно найти на веб-сайте книги.

Для нужд незаметного обмена информацией диссиденты могут создать веб-сайт, заполненный политкорректными фотографиями (например, Великого Вождя, местных спортивных команд, теле- и кинозвезд и т. п.). Разумеется, эти фотографии могут содержать стеганографические сообщения. Учитывая то, что эти сообщения перед помещением в графический файл сжимаются и шифруются, даже если кто-то и заподозрит их присутствие «внутри» фотографий, отличить их от белого шума, а тем более расшифровать их смысл будет очень и очень тяжело. Конечно, фотографии для стеганографической обработки должны быть отсканированы или сняты на цифровую камеру самостоятельно: если просто взять картинку из Интернета и записать в нее секретное сообщение, выявить «второе дно» простым побитовым сравнением будет гораздо проще.

Понятно, что стеганографические сообщения можно записывать не только в графические файлы. Очень хорошо для этих целей подходят звуковые файлы. Скрытая информация может содержаться в VoIP-звонке благодаря манипуляции с задержками пакетов, искажению звука или даже в полях заголовка пакетов (Lubacz и др., 2010). Надо сказать, что даже форматирование текста и расположение тэгов в HTML-файле может нести определенную скрытую информацию.

Мы рассмотрели стеганографию в контексте проблемы свободы слова, однако у этой технологии есть масса иных применений. Очень часто авторы электронных изображений вшивают в файлы секретные сообщения, которые в случае необходимости смогут подтвердить их авторские права. Если кто-нибудь захочет украсть удачную фотографию и разместит ее на своем сайте без указания авторства, законный владелец с помощью стеганографической подписи сможет доказать в суде свою правоту. Такие подписи иногда называют **водяными знаками**. Эта тема обсуждается в (Piva и др., 2002).

Подробное рассмотрение проблем стеганографии можно найти в (Wayner, 2008).

8.10.3. Защита авторских прав

Конфиденциальность и цензура — это те области, в которых сталкиваются технологические аспекты и общественные интересы. Третьей такой областью является защита авторских прав. **Авторское право (copyright)** гарантирует свободу распоряжения

интеллектуальной собственностью (Intellectual Property — IP) ее *создателям*, которыми могут быть, например, писатели, поэты, художники, композиторы, музыканты, фотографы, кинорежиссеры, хореографы и т. д. Авторское право выдается на определенный срок, который обычно равен сроку жизни автора, плюс 50 лет (75 лет — в случае корпоративного авторского права). По окончании этого срока интеллектуальная собственность становится достоянием общества, и каждый волен распоряжаться ею, как хочет. Так, например, проект Gutenberg (*www.promo.net/pg*) разместил в Сети тысячи произведений, давно уже ставших общественным достоянием (работы Шекспира, Диккенса, Марка Твена). По просьбе Голливуда в 1998 года Конгресс США разрешил продлить срок действия авторских прав еще на 20 лет, утверждая, что без принятия этой меры больше никто ничего не станет создавать. В то же время патенты на изобретения действуют в течение всего лишь 20 лет, и никто не жалуется — люди продолжают совершать открытия и делать изобретения.

Вопрос о защите авторских прав вышел на первый план, когда у службы Napster, сервиса обмена музыкой внезапно оказалось 50 млн пользователей. Несмотря на то что записи системой Napster никуда не копировались, судом было предъявлено обвинение в хранении центральной базы данных с информацией о том, какие у кого из пользователей имеются записи. Таким образом, система побуждала пользователей совершать преступные действия, нарушающие закон об авторских правах. В общем-то, никто не жалуется на то, что идея авторских прав плоха (хотя многим не нравится то, что компании обладают намного большими привилегиями в этом плане, чем простые граждане), однако следующее поколение технологий распространения музыкальных записей уже поднимает вопросы этического характера.

Например, рассмотрим равноранговую сеть, члены которой занимаются вполне законным обменом файлами (записями, являющимися общественным достоянием, домашними видеозаписями, религиозными трактатами (не составляющими коммерческую тайну церквей) и т. д.). Возможно, какая-то небольшая часть этих файлов защищена законом об авторских правах. Допустим, все участники проекта находятся на постоянном соединении (пользуются ADSL или кабельным Интернетом). На каждой машине хранится список того, что есть на жестком диске, а также список всех машин сети. В поисках какого-либо файла можно обратиться к произвольно выбранной машине и запросить у нее список имеющихся материалов. Если нужная информация не находится, можно попробовать опросить все остальные машины из списка, а также машины из списков, хранящихся у других. Компьютеры здесь сильно облегчают задачу поиска некоторых редких материалов. Найдя нужный файл, пользователь просто копирует его себе.

Если найденная работа оказывается защищенной законом об авторских правах, пользователь, сам того не желая, становится нарушителем этого закона (тут еще, конечно же, играет роль то, в какой стране происходит дело и, соответственно, какие законы следует применять в каждом конкретном случае, так как в некоторых случаях загружать файлы — это незаконно, а скачивать — законно). А виноват ли поставщик информации? Является ли преступлением хранение у себя на жестком диске записей, за которые были заплачены деньги и которые были вполне легально загружены из Интернета, при условии, что к диску могут иметь доступ посторонние лица? Если в вашу лачугу, никогда не знавшую замков и засовов, проникает вор с ноутбуком и ска-

нером, снимает копию с книги, защищенной законом об авторских правах, и уходит восвояси, разве *вы* виноваты в этом преступлении, разве *вы* должны защищать чужие авторские права?

Однако есть еще одна проблема, связанная с авторскими правами. Ведется ожесточенная борьба между Голливудом и компьютерной индустрией. Голливуд требует усилить защиту интеллектуальной собственности, а компьютерщики говорят, что они не обязаны сторожить голливудские ценности. В октябре 1998 года Конгресс принял **Акт об авторских правах в цифровом тысячелетии (DCMA — Digital Millenium Copyright Act)**, в котором говорится о том, что взлом механизма защиты, присутствующего в работе, защищенной законом об авторских правах, а также сообщение другим технологии взлома является преступлением. Аналогичный законодательный акт был принят и в Евросоюзе. С одной стороны, все как-то забыли подумать о том, что для пиратов с Дальнего Востока такие акты указом не являются, а с другой стороны, многие считают, что новый закон нарушил баланс между интересами владельцев авторских прав и общественными интересами.

Взять хотя бы такой пример. В сентябре 2000 года консорциум, связанный с музыкальной индустрией, озабоченный созданием надежной онлайновой системы продажи аудиозаписей, организовал конкурс, пригласив всех желающих попробовать взломать систему (это действительно очень важный этап, необходимый при создании любой новой системы защиты). Группа ученых из различных университетов под руководством профессора Эдварда Фельтена (Edward Felten) из Принстона, специализирующихся в области защиты информации, приняли вызов и сломали систему. Затем была написана статья, описывающая выводы, сделанные в ходе исследования. Она была направлена на конференцию USENIX, посвященную проблемам защиты информации. Доклад был рассмотрен и принят на соответствующем уровне. Однако незадолго до конференции Фельтен получил уведомление от Ассоциации звукозаписывающей индустрии Америки (**RIAA — Recording Industry Association of America**) о том, что эта организация в случае опубликования статьи подаст на авторов в суд за нарушение Акта DCMA.

Ученым ничего не оставалось делать, как послать запрос в федеральный судебный орган, пытаясь выяснить, является ли еще легальным опубликование научных статей, касающихся защиты информации. Опасаясь, что дальнейшее развитие событий будет отнюдь не в пользу звукозаписывающей индустрии, ее представители сняли свои претензии к Фельтену, и на этом инцидент был исчерпан. Несомненно, причиной недовольства звукозаписывающей индустрии была слабость предложенной ими системы. Получилось, что сначала пригласили людей, чтобы те попытались взломать защиту, а когда некоторым это реально удалось, на них чуть было не подали за это в суд. После того как все конфликты были улажены, статья все-таки была напечатана (Craver и др., 2001). Почти очевидно, что такие конфронтации впереди еще будут многократно.

Между тем, пиратская музыка и пиратские фильмы вызвали огромный рост файлообменных пиринговых сетей. Это не радует правообладателей, которые использовали DMCA в качестве ответного действия. Также существуют автоматические системы, которые производят поиск по файлообменным сетям, а затем отсылают предупреждения операторам сети и пользователям, которые подозреваются в нарушении авторского права. В Соединенных Штатах эти предупреждения называются **DMCA takedown**

notices — предупреждение DMCA об удалении. Этот поиск напоминает гонку вооружений, потому что действительно очень трудно поймать нарушителей авторских прав. Даже ваш принтер можно принять за нарушителя (Piatek и др., 2008).

С обсуждаемой темой тесно связана **доктрина законного использования (fair use doctrine)**, ставшая результатом судебных решений во многих странах. Эта доктрина состоит в том, что покупатели продукции, защищенной законом об авторских правах, имеют определенные ограниченные права на копирование этой продукции, включая даже использование ее частей для научных целей (например, в качестве обучающего материала в школах и колледжах) и создание архивных резервных копий на тот случай, если что-нибудь случится с исходным носителем. Как проверить, является ли использование продукции законным? Показатели таковы: (1) коммерческое использование, (2) количество процентов скопированных данных, (3) влияние копирования на объем продаж. Так как DMCA и аналогичные законы, принятые Евросоюзом, запрещают взлом систем защиты от копирования, такие законы заодно запрещают легальное добросовестное использование. По сути, DMCA отбирает у потребителей историческое право активно поддерживать продавцов, у которых они приобрели продукцию. Провал этой идеи неизбежен.

Еще одно явление, затмевающее собой по уровню смещения баланса между обладателями авторских прав и потребителями даже DMCA, это **доверенные вычисления (trusted computing)**, защищаемые такими представителями индустрии, как **TGC (Trusted Computing Group — группа доверенных вычислений)**. Разрабатывается этот проект совместными усилиями Intel и Microsoft. Идея состоит в том, чтобы обеспечить поддержку наблюдению за действиями пользователя (например, за воспроизведением скопированной незаконным образом звукозаписи) на уровне операционной системы и ниже и запрещать совершать нежелательные действия. Система дополняется небольшим чипом **TPM (Trusted Platform Module — модуль доверенной платформы)**, в работу которого сложно вмешаться. Большинство персональных компьютеров, продаваемых сегодня, продаются вместе TPM. Такая система могла бы даже позволить программам, написанным обладателями авторских прав, манипулировать персональными компьютерами пользователей, так что пользователь не сможет ничего поменять. Вследствие этого возникает вопрос, кто является доверенным в доверенных вычислениях. Очевидно, не пользователь. Несомненно, общественный резонанс будет огромен. Конечно, это очень здорово, что индустрия наконец обратила внимание на проблемы защиты информации, однако нельзя не заметить с прискорбием, что большинство усилий направлено на усиление законов об авторских правах, а не на борьбу с вирусами, взломщиками, мошенниками и другими проблемами, волнующими большинство пользователей.

Короче говоря, авторам законов и юристам теперь предстоит в течение долгих лет пытаться урегулировать взаимоотношения владельцев авторских прав и потребителей. Киберпространство ничем не отличается от социума: и там, и там постоянно сталкиваются интересы различных групп, что приводит к ожесточенной борьбе и судебным разбирательствам, результатом которых рано или поздно становится нахождение какого-то компромисса. По крайней мере, стоит на это надеяться, как стоит надеяться на относительное затишье до появления новой противоречивой технологии.

8.11. Резюме

Криптография представляет собой инструмент, используемый для обеспечения конфиденциальности информации, проверки ее целостности и аутентичности. Все современные криптографические системы основаны на принципе Керкгофа, гласящем, что алгоритмы должны быть доступны всем желающим, а ключи — храниться в тайне. Многие алгоритмы при шифровании текста выполняют сложные преобразования, включающие в себя замены и перестановки. Тем не менее если удастся реализовать на практике принципы квантовой криптографии, то с помощью одноразовых блокнотов можно будет создать действительно надежные криптосистемы.

Все криптографические алгоритмы можно разделить на два типа: с симметричными ключами и с открытыми ключами. Алгоритмы с симметричными ключами при шифровании искажают значения битов в последовательности итераций, параметризованных ключом. Наиболее популярные алгоритмы этого типа — тройной DES и AES (Rijndael). Алгоритмы с симметричным ключом могут работать в режиме электронного шифроблокнота, сцепления блоков шифра, потокового шифра и др.

Алгоритмы с открытыми ключами отличаются тем, что для шифрования и дешифрации используются разные ключи, причем ключ дешифрации невозможно вычислить по ключу шифрования. Эти свойства позволяют делать ключ открытым. Чаще всего применяется алгоритм RSA, основанный на сложности разложения больших чисел на простые сомножители.

Официальные, коммерческие и другие документы необходимо подписывать. Существуют различные методы генерации цифровых подписей, основанные на алгоритмах как с симметричными ключами, так и с открытыми. Обычно при помощи алгоритмов типа MD5 или SHA-1 вычисляется хэш-функция сообщения, которое необходимо подписать, и подпись ставится не на само сообщение, а на значение хэш-функции.

Управление открытыми ключами можно реализовать при помощи сертификатов, представляющих собой документы, связывающие между собой открытые ключи и обладающих ими принципалов. Сертификаты подписываются надежной организацией или кем-либо, способным предъявить цепочку сертификатов, ведущих в итоге к этой надежной организации. Начальное звено этой цепочки (центральное управление) должно быть известно заранее, для этого в браузеры вшиваются номера многих сертификатов центральных управлений.

Эти криптографические методы позволяют защитить сетевой трафик. На сетевом уровне для этого работает система IPsec, занимающаяся шифрованием пакетов, передающихся между хостами. Межсетевые экраны могут фильтровать как входящий, так и исходящий трафик, анализируя используемые протоколы и порты. Виртуальные частные сети могут имитировать старые сети на основе выделенных линий для предоставления соответствующих повышенных свойств защиты. Наконец, в беспроводных сетях требуется довольно серьезная защита информации, чтобы сообщения не читали все подряд, и такие протоколы как 802.11i ее обеспечивают.

При установлении сеанса связи между двумя сторонами они должны идентифицировать себя и, возможно, определить общий ключ сеанса. Для этого существуют различные протоколы аутентификации, среди которых есть те, которые подразумевают

наличие третьей, надежной стороны, а также протоколы Диффи—Хеллмана, Kerberos и протоколы с использованием открытых ключей.

Защита информации в электронной почте может быть достигнута применением различных комбинаций тех методов, которые мы изучили в этой главе. PGP, например, сжимает сообщение, а потом шифрует его с помощью секретного ключа. Секретный ключ шифруется при помощи открытого ключа получателя. Кроме того, вычисляется хэш-функция сообщения. Проверка целостности может быть выполнена за счет того, что на хэш перед отправкой ставится подпись.

Безопасность во Всемирной паутине — это тоже важная проблема, которая начинается с безопасного именования ресурсов. DNSsec позволяет предотвращать обманы DNS-серверов. Большинство сайтов, посвященных электронной коммерции, для установления надежных, аутентифицированных сеансов между клиентом и сервером используют SSL/TLS. Для борьбы с возможными проблемами, связанными с переносимыми программами, разработаны разные методы, среди которых помещение загружаемого кода в изолированную среду («песочницу») и подписание кодов.

В Интернете возникает множество вопросов, в которых технологические проблемы переплетаются с государственной политикой. Мы рассмотрели лишь некоторые из них: конфиденциальность, свободу слова и авторские права.

Вопросы

1. Расшифруйте следующее сообщение, составленное с помощью моноалфавитного подстановочного шифра. Открытый текст, состоящий только из букв, представляет собой хорошо известный отрывок из поэмы Льюиса Кэрролла.

   ```
   mvyy bek mnyx n yvjjyr snijrh invq n muvjvdt je n idnvy jurhri n fehfevir pyeir oruvdq
   ki ndq jur jkhjyri nyy nqlndpr
   Jurb nhr mnvjvdt ed jur iuvdtyr mvyy bek pezr ndq wevd jur qndpr
   mvyy bek, medj bek, mvyy bek, medj bek, mvyy bek wevd jur qndpr
   mvyy bek, medj bek, mvyy bek, medj bek, medj bek wevd jur qndpr
   ```

2. Гомогенный шифр является вариантом моноалфавитного подстановочного шифра, в котором буквы алфавита размера m сначала отображаются на целые числа в промежутке от 0 до $m - 1$. Затем целочисленное представление открытого текста преобразуется в целочисленное представление шифротекста. Функция шифрования для отдельной буквы это $E(x) = (ax + b) \bmod m$, где m является размером алфавита, а a и b — ключи шифра, одинаково важные. Злоумышленник обнаруживает, что Боб создал зашифрованный текст с использованием гомогенного шифра. Он получает копию зашифрованного текста и обнаруживает, что самая частая буква в зашифрованном тексте это «R», а следующая по частоте «K». Определите, как злоумышленник может взломать код и получить текст.

3. Взломайте следующий колоночный перестановочный шифр. Открытый текст взят из популярной книги о компьютерах, поэтому слово «computer» может встретиться с большой вероятностью. Открытый текст состоит из одних букв (без пробелов). Зашифрованный текст разбит на блоки по пять символов для удобства чтения.

   ```
   aauan cvlre rurnn dltme aeepb ytust iceat npmey iicgo gorch crsoc nntii imiha oofpa
   gsivt tpsit lbolr otoex
   ```

4. Алиса использовала перестановочный шифр для того, чтобы зашифровать свои сообщения для Боба. В целях повышенной безопасности она зашифровала подстановочный шифр пере-

становочным ключом и сохранила шифр в своем компьютере. Труди получила перестановочный ключ. Может ли она расшифровать сообщения Алисы? Почему да или почему нет?

5. Подберите 77-битный одноразовый блокнот, с помощью которого из шифра, показанного на рис. 8.3, можно получить текст «Hello World».
6. Вы шпион и, к вашему счастью, в вашем распоряжении есть библиотека с бесконечным количеством книг. У вашего оператора тоже есть такая библиотека. Вы договорились использовать книгу «Властелин колец» как одноразовый блокнот. Объясните, как вы будете использовать все имеющееся для создания бесконечного одноразового блокнота.
7. При использовании квантовой криптографии требуется наличие фотонной пушки, способной при необходимости испускать одиночный фотон, соответствующий одному биту. Подсчитайте, сколько фотонов переносит 1 бит по оптоволоконной линии связи с пропускной способностью 250 Гбит/с. Длина фотона предполагается равной длине волны, то есть 1 мкм (для нужд данной задачи). Скорость света в оптоволокне равна 20 см/нс.
8. Если злоумышленнику удастся перехватить и регенерировать фотоны при использовании квантовой криптографии, некоторые значения будут приняты неверно, а значит, появятся ошибки и в одноразовом блокноте, принимаемом Бобом. Какая (в среднем) часть блокнота, принятого Бобом, будет содержать ошибки?
9. Фундаментальный принцип криптографии гласит, что все сообщения должны быть избыточными. Но мы знаем, что избыточность позволяет злоумышленнику понять, правильно ли он угадал секретный ключ. Рассмотрите два типа избыточности. В первом типе начальные n бит открытого текста содержат известную последовательность. Во втором конечные n бит сообщения содержат хэш. Эквивалентны ли эти типы избыточности с точки зрения безопасности? Ответ поясните.
10. На рис. 8.5 S-блоки и P-блоки сменяют друг друга. Сделано ли это из эстетических соображений, или такая организация шифратора дает более надежный код, чем в случае, когда сначала идут все P-блоки, а затем все S-блоки? Обсудите свой ответ.
11. Разработайте атаку шифра DES, основываясь на информации, что открытый текст состоит исключительно из прописных ASCII-символов, пробелов, запятых, двоеточий, символов точка с запятой, переводов строк и возврата каретки. О битах четности, содержащихся в открытом тексте, ничего не известно.
12. В тексте было подсчитано, что машине для взлома шифра, снабженной миллионом процессоров, способных обрабатывать 1 ключ за 1 нс, понадобится всего лишь 1016 лет для взлома 128-битной версии AES. Давайте сосчитаем, как можно сократить это время до одного года. Для этого компьютеры должны работать в 1016 раз быстрее. Если закон Мура, утверждающий, что вычислительные мощности компьютеров удваиваются каждые 18 месяцев, будет продолжать соблюдаться всегда, сколько лет пройдет до того, как параллельный компьютер сможет сократить время взлома шифра до одного года?
13. Ключи AES могут иметь длину 256 бит. Сколько вариантов ключей доступно в таком режиме? Сможете ли вы найти в какой-нибудь науке — например, физике, химии или астрономии — понятия, оперирующие подобными величинами? Найдите в Интернете материалы, посвященные большим числам. Сделайте выводы из этого исследования.
14. Предположим, что сообщение было зашифровано с помощью шифра DES в режиме сцепления блоков. В одном из блоков зашифрованного текста при передаче один бит изменился с 0 на 1. Какая часть текста будет при этом повреждена?
15. Снова рассмотрим шифрование со сцеплением блоков. На этот раз между блоками зашифрованного текста при передаче вставился один нулевой бит. Какая часть текста будет повреждена в этом случае?

16. Сравните режим шифрования со сцеплением блоков с режимом шифрованной обратной связи с точки зрения количества операций шифрования, необходимых для передачи большого файла. Какой режим более эффективен и насколько?
17. Используя алгоритм открытого ключа RSA, при следующих кодах символов: $a = 1$, $b = 2$ и т. д., $y = 25$, $z = 26$.

    1) Если $p = 5$ и $q = 13$, найдите пять допустимых значений для d.

    2) Если $p = 5$, $q = 31$, а $d = 37$, найдите e.

    3) Используя $p = 3$, $q = 11$ и $d = 9$, найдите e и зашифруйте «hello».
18. Алиса и Боб используют открытые ключи RSA для шифрования при коммуникации. Труди обнаруживает, что они используют простое число для определения числа n пар открытого ключа. Другими словами, Труди обнаруживает, что $n_a = p_a \times q$ и $n_b = p_b \times q$. Как Труди может использовать эту информацию для того, чтобы взломать код Алисы?
19. Рассмотрите режим счетчика, показанный на рис. 8.13, но со значением вектора инициализации, равным 0. Угрожает ли использование нуля надежности шифра в целом?
20. На рис. 8.17 мы видим, как Алиса передает Бобу подписанное сообщение. Если Труди заменит P, Боб заметит это. А что будет, если Труди заменит и P, и подпись?
21. Цифровые подписи имеют один недостаток: их могут использовать лентяи. После составления договора в электронной коммерции обычно требуется, чтобы клиент подписал его своим хэшем SHA-1. Если пользователь не озаботится проверкой соответствия хэша и контракта, он имеет шанс случайно подписать другой договор. Допустим, вездесущая и бессмертная мафия решила сделать на этом деньги. Бандиты создают платный веб-сайт (содержащий, например, порнографию, азартные игры и т. п.) и спрашивают у новых клиентов номера кредитных карт. Затем пользователю отсылается договор, в котором говорится, что он желает воспользоваться услугами и оплатить их с помощью кредитной карты. Договор следует подписать, однако владельцы сайта знают, что вряд ли кто-то станет сверять хэш с контрактом. Покажите, каким образом злоумышленники смогут купить бриллианты в легальном ювелирном интернет-магазине за счет ничего не подозревающих клиентов.
22. В математическом классе 25 учащихся. Предположим, все студенты родились в первой половине года — между первым января и тридцатым июня. Какова вероятность того, что, по крайней мере, у двух из них совпадают дни рождения? Допустим, что ни у кого из них день рождения не приходится на 29 февраля, поэтому общее число рассматриваемых вариантов дней рождения равно 181.
23. После того как Элен созналась Мэрилин в своем обмане в деле с предоставлением должности Тому, Мэрилин решила устранить проблему, записывая содержимое своих сообщений на диктофон, с тем чтобы ее новый секретарь просто набирала соответствующий текст. Затем Мэрилин собирается изучать набранные сообщения на своем терминале, чтобы проверить, что они содержат точные ее слова. Может ли новый секретарь по-прежнему воспользоваться атакой, основанной на задаче о днях рождения, чтобы фальсифицировать сообщение, и если да, то как? *Подсказка*: может.
24. Рассмотрите пример неудачной попытки получения Алисой открытого ключа Боба (см. рис. 8.20). Допустим, они уже установили общий закрытый ключ, однако Алиса все же хочет узнать открытый ключ Боба. Существует ли в данной ситуации безопасный способ его получения? Если да, то какой?
25. Алиса желает общаться с Бобом, используя шифрование с открытым ключом. Она устанавливает соединение с кем-то, кто по ее предположению является Бобом. Она спрашивает

у него открытый ключ, и тот отсылает его вместе с сертификатом X.509, подписанным Центральным управлением. У Алисы уже есть открытый ключ ЦУ. Что должна теперь сделать Алиса, чтобы удостовериться, что она действительно общается с Бобом? Допустим, Бобу безразлично, с кем он общается (то есть под именем Боба мы здесь понимаем, например, какую-нибудь общедоступную службу).

26. Допустим, система использует PKI, базирующийся на древовидной иерархии Управления сертификации. Алиса желает пообщаться с Бобом и после установления соединения получает от него сертификат, подписанный УС *X*. Допустим, Алиса никогда не слышала про *X*. Что ей нужно сделать, чтобы удостовериться, что на той стороне канала связи находится именно Боб?
27. Может ли IPsec с заголовком AH использоваться в транспортном режиме, если одна из машин находится за NAT-блоком? Ответ поясните.
28. Алиса хочет послать Бобу сообщение, используя хэши SHA-1. Она консультируется с вами по поводу использования подходящего алгоритма. Что вы посоветуете?
29. Назовите одну причину, по которой межсетевые экраны следует настраивать на анализ входящего трафика. Теперь назовите одну причину, по которой межсетевые экраны следует настраивать на анализ исходящего трафика. Как вы думаете, насколько успешен такой анализ?
30. Допустим, организация установила виртуальную частную сеть для обеспечения безопасной связи между своими сайтами в Интернете. Джим, сотрудник этой организации, использует виртуальную частную сеть для общения со своим боссом, Мэри. Опишите тип коммуникации между Джимом и Мэри, который бы не требовал шифрования или других механизмов защиты, и другой тип коммуникации, для которого потребовалось бы шифрование или другие механизмы защиты. Объясните свой ответ.
31. Измените одно сообщение в протоколе, изображенном на рис. 8.30, так чтобы протокол стал устойчивым к зеркальной атаке. Объясните суть ваших изменений.
32. Для установки секретного ключа между Алисой и Бобом используется алгоритм Диффи—Хеллмана. Алиса посылает Бобу (227, 5, 82). Боб отвечает (125). Секретное число Алисы $x = 12$, а секретное число Боба $y = 3$. Покажите, как Алиса и Боб могут рассчитать секретный ключ.
33. Даже если два пользователя не пользовались общим ключом и не имеют сертификатов, они все равно могут установить общий закрытый ключ при помощи алгоритма Диффи—Хеллмана.

    1) Объясните, каким образом этот алгоритм подвергается атаке типа «человек посередине».

    2) Как эта подверженность изменится, если n или g секретны?
34. Почему в протоколе, изображенном на рис. 8.35, *A* посылается открытым текстом вместе с зашифрованным ключом сеанса?
35. В протоколе Нидхэма—Шредера (Needham—Schroeder) Алиса формирует два оклика, *RA* и *RA*2. Не достаточно ли будет использовать один оклик?
36. Предположим, что организация для аутентификации применяет метод Kerberos. Как в терминах безопасности и работоспособности повлияет на систему выход из строя сервера аутентификации или сервера выдачи билетов?
37. Алиса использует протокол аутентификации с открытым ключом, показанный на рис. 8.39, чтобы аутентифицировать коммуникацию с Бобом. Однако, посылая сообщение 7, Алиса забыла зашифровать *RB*. Таким образом, Труди знает значение *RB*. Нужно ли Алисе и Бобу повторять процедуру аутентификации с новыми параметрами, для того чтобы удостовериться, что коммуникация безопасна? Ответ поясните.

38. В протоколе аутентификации с открытым ключом, показанном на рис. 8.39, в сообщении 7 случайное число *RB* зашифровано ключом *KS*. Необходимо ли это шифрование или допустимо отправить это число обратно открытым текстом? Ответ поясните.
39. У кассовых аппаратов, использующих магнитные карты и PIN-коды, есть существенный недостаток: кассир-злоумышленник может модифицировать считывающее устройство своего кассового аппарата, чтобы считать и сохранить всю информацию, хранящуюся на карте, включая PIN-код, с целью послать дополнительные (поддельные) транзакции в будущем. В следующем поколении кассовых терминалов будут использоваться карты с полноценным центральным процессором, клавиатурой и небольшим дисплеем. Разработайте для этой системы протокол, который не сможет взломать кассир-злоумышленник.
40. Можно ли передать сообщение PGP с помощью многоабонентной рассылки? Какие ограничения будут применяться?
41. Предположим, что повсеместно в Интернете используется PGP. Можно ли в этом случае послать PGP-сообщение по произвольному интернет-адресу и быть полностью уверенным в том, что на приемной стороне оно будет корректно расшифровано? Обсудите свой ответ.
42. На атаке, показанной на рис. 8.43, пропущен один шаг. Он не является необходимым для того, чтобы атака была удачной, однако его наличие может помочь уменьшить возможные подозрения. Итак, что же пропущено?
43. Протокол передачи данных SSL подразумевает наличие двух нонсов и подготовительного ключа. Какую роль играют нонсы (если они вообще играют какую-либо роль)?
44. Представьте себе изображение размером 2048 на 512 пикселей. Вы хотите зашифровать файл размером 2,5 Мбайт. Какую часть файла вы сможете зашифровать в этом рисунке? Какую часть вы бы смогли зашифровать, если бы вы сжали файл до четверти его оригинального размера? Продемонстрируйте ваши расчеты.
45. Изображение на рис. 8.49, *б* содержит ASCII-текст пяти пьес Шекспира (хотя по фотографии этого не скажешь). Можно ли спрятать музыку, а не текст, между зебрами? Если да, то как и сколько? Если нет, то почему?
46. Вам выдали текстовый файл размером 60 Мбайт для шифрования, используя стеганографию в младших битах каждого цвета в файле рисунка. Какого размера рисунок потребуется для шифрования целого файла? Какой размер потребуется, если сначала файл был сжат до трети своего оригинального размера? Выдайте ответ в пикселях и продемонстрируйте расчеты. Предположим, что изображения имеют соотношение 3:2, например 3000 × 2000 пикселей.
47. Алиса была постоянным пользователем анонимной рассылки первого типа. Она могла помещать сколько угодно сообщений в свою любимую конференцию *alt.fanclub.alice*, но все знали, что их автором является Алиса, так как все письма были подписаны одним и тем же псевдонимом. Если предполагать, что рассылка работает правильно, у Труди нет возможности писать от имени Алисы. После того как все анонимные рассылки первого типа были закрыты, Алиса перешла на шифропанковские рассылки и начала обсуждать новую тему в своей конференции. Предложите способ защиты в новых условиях от Труди, пытающейся писать письма от чужого имени.
48. Найдите в Интернете описание какого-нибудь интересного случая, касающегося конфиденциальности, и напишите небольшой отчет на эту тему.
49. Найдите в Интернете описание очередного случая противостояния закона об авторских правах добросовестному использованию продукции и напишите небольшой отчет о том, что вы узнали.
50. Напишите программу, шифрующую входные данные путем суммирования по модулю 2 с ключевым потоком. Найдите или напишите сами хороший генератор случайных чисел

для создания ключевого потока. Программа должна работать подобно фильтру, принимая на входе открытый текст и выдавая на выходе на стандартное устройство вывода шифр (и наоборот). У программы должен быть один параметр: ключ, запускающий генератор случайных чисел.

51. Напишите процедуру для вычисления хэша SHA-1 блока данных. У процедуры должно быть два параметра: указатель на входной буфер и указатель на 20-байтовый выходной буфер. Чтобы найти спецификацию SHA-1, поищите в Интернете FIPS 180-1, в этом документе содержится полное описание.
52. Напишите функцию, которая принимает поток ASCII-символов и шифрует его с помощью режима сцепления блоков. Размер блока должен быть 8 байт. Программа должна получать открытый текст из стандартного устройства ввода и печатать зашифрованный текст на стандартном устройстве вывода. Для решения этой задачи вы можете воспользоваться любой подходящей системой, чтобы определить, что достигнут конец входного потока и/или когда следует делать вставку, чтобы завершить блок. Вы можете выбрать формат вывода данных, главное, чтобы он не был двусмысленным. Программа должна получать два параметра:

    1) Указатель на вектор инициализации.

    2) Номер, k, представляющий смещение шифра подстановки, так что значение ASCII будет зашифровано k-м значением впереди него в алфавите.

    Например, если $x = 3$, тогда A шифруется как D, B шифруется как E и т. д.

    Сделайте разумные допущения по отношению к последнему значению в наборе ASCII. Убедитесь, что вы четко документируете в своем коде все допущения, касающиеся входного потока и алгоритма шифрования.
53. Цель этой задачи — дать вам лучшее представление о механизме работы RSA. Напишите функцию, которая получает в качестве параметров простые числа p и q, рассчитывает открытый и секретный RSA-ключи с использованием этих параметров и выводит n, z, d и e в качестве выходных данных. Функция также должна принимать поток значений ASCII и шифровать этот входящий поток с помощью рассчитанных ключей RSA. Программа должна получать открытый текст со стандартного устройства ввода и распечатывать зашифрованный текст на стандартном устройстве вывода. Шифрование должно производиться по каждому значению, то есть должно браться каждое значение из входных данных и зашифровываться независимо от других значений. Для решения этой задачи можно воспользоваться любой подходящей системой, чтобы определить, что достигнут конец входного потока. Вы можете выбрать формат вывода данных, главное, чтобы он не был двусмысленным. Убедитесь, что вы четко документируете в своем коде все допущения, касающиеся входного потока и алгоритма шифрования.

Глава 9

Рекомендации для чтения и библиография

Мы закончили изучение компьютерных сетей, но все это — только начало. Многие интересные темы не были рассмотрены во всей полноте и со всеми подробностями, а многие вопросы были вообще опущены из-за отсутствия места. Эта глава содержит список дополнительной литературы для читателей, желающих продолжить изучение компьютерных сетей.

9.1. Литература для дальнейшего чтения

Существует большое количество книг, касающихся всех аспектов компьютерных сетей и распределенных систем. Среди журналов, часто публикующих статьи по этой теме, стоит выделить следующие два: *IEEE/ACM Transactions on Networking* и *IEEE Journal on Selected Areas in Communications*.

В периодических изданиях ACM Special Interest Groups on Data Communications (SIGCOMM) и Mobility of Systems, Users, Data, and Computing (SIGMOBILE) публикуется много интересных статей, особенно по темам, по которым происходит развитие. Это издания *Computer Communication Review u Mobile Computing and Communications Review*.

Кроме того, Институт инженеров по электротехнике и электронике IEEE выпускает еще три журнала: *IEEE Internet Computing, IEEE Network Magazine* и *IEEE Communications Magazine* — в них содержатся обзоры, учебные статьи и информация об исследованиях, связанные с компьютерными сетями. Первые два в основном посвящены архитектуре, стандартам и программному обеспечению, тогда как журнал *IEEE Communications Magazine* большей частью занят освещением технологий коммуникаций (оптоволоконной, спутниковой связи и т. д.).

Ежегодно или раз в два года проводятся несколько конференций, которые освещаются в многочисленных статьях, посвященных сетям. В частности, обратите внимание на конференции *SIGCOMM, NSDI (Symposium on Networked Systems Design and Implementation), MobiSys (Conference on Mobile Systems, Applications, and Services), SOSP (Symposium on Operating Systems Principles)* и *OSDI (Symposium on Operating Systems Design and Implementation)*.

Ниже мы перечислим дополнительную литературу, сгруппированную по главам этой книги. Большая часть рекомендуемых книг или отдельных глав представляют собой самоучители либо обзоры. Полные ссылки даны в разделе 9.2.

9.1.1. Введение и неспециализированная литература

Comer, *The Internet Book*, 4-е издание.

Сюда стоит заглянуть всем, кто ищет простое и понятное описание Интернета. В этой книге в доступной даже для новичка форме рассказывается об истории, развитии, технологиях, протоколах и службах Интернета. Тем не менее эта книга будет интересна и более подготовленным читателям благодаря большому количеству содержащегося в ней материала.

Computer Communication Review, 25th Anniversary Issue, Jan. 1995

Crovella and Krishnamurthy, *Internet Measurement*

Как узнать, хорошо ли работает Интернет? Ответ на этот вопрос не тривиален, потому что за Интернет никто не отвечает. Эта книга описывает методы, которые были развиты, чтобы измерить работу Интернета, от сетевой инфраструктуры до приложений.

IEEE Internet Computing, Jan. – Feb. 2000

В первом выпуске нового тысячелетия журнал *IEEE Internet Computing* представил то, что и ожидалось: размышления людей, участвовавших в создании Интернета, о том, каким он будет в новом веке. В обсуждении участвуют такие эксперты, как Поль Бэрэн (Paul Baran), Лоуренс Робертс (Lawrence Roberts), Леонард Кляйнрок (Leonard Kleinrock), Стэфан Крокер (Stephen Crocker), Дэнни Коэн (Danny Cohen), Боб Мэткалф (Bob Metcalfe), Билл Гейтс (Bill Gates), Билли Джой (Billy Joy) и др. Посмотрите, насколько исполнились их предсказания десятилетие спустя.

Kipnis, *Beating the System: Abuses of the Standards Adoption Process*

Комитеты по стандартизации пытаются работать максимально добросовестно и независимо от разработчиков, но, к сожалению, некоторые компании пытаются нарушить эту систему. Например, уже не раз случалось так, что компания помогает в разработке стандарта, а после его утверждения заявляет, что стандарт основывается на принадлежащем ей патенте и что вопросы выдачи лицензий и цен на них компания будет решать сама. Данный материал будет полезен тем, кто хочет узнать о нелицеприятной стороне стандартизации.

Hafner and Lyon, *Where Wizards Stay Up Late*

Naughton, *A Brief History of the Future*

Кто же все-таки изобрел Интернет? Многие хотят, чтобы их включили в число изобретателей. И некоторые из них по праву этого заслуживают. Пол Баран (Paul Baran), который написал отчет, описывающий коммутацию пакетов, люди в различных университетах, которые разрабатывали архитектуру ARPANET, сотрудники BBN, которые запрограммировали первые IMP, Боб Кан (Bob Kahn) и Винт Серф (Vint Cerf), которые изобрели TCP/IP и т. д. Эти книги рассказывают историю Интернета, по крайней мере, до 2000 года и содержат множество анекдотов.

9.1.2. Физический уровень

Bellamy, *Digital Telephony*, 3-е издание.

В этом солидном издании содержится все, что вы когда-либо хотели узнать о телефонной системе, и даже более того. Особый интерес представляют главы, посвященные передаче данных и мультиплексированию, цифровой коммутации, волоконной оптике, мобильной телефонии и DSL.

Hu and Li, *Sattelite-Based Internet: A Tutorial*

Доступ в Интернет через спутник отличается от использования наземных линий связи. Здесь должны учитываться не только задержки, но и маршрутизация, а также коммутация. Авторы рассматривают проблемы применения спутниковых систем для доступа в Интернет.

Joel, *Telecommunications and the IEEE Communications Society*

Здесь в очень сжатой, но удивительно понятной форме описывается история телекоммуникаций, начиная с телеграфа и заканчивая сетями стандарта 802.11. Вы найдете разделы, посвященные радио, телефонии, аналоговой и цифровой коммутации, подводным кабелям, цифровой передаче данных, телевизионному вещанию, спутникам, кабельному ТВ, оптическим линиям связи, мобильным телефонам, коммутации пакетов, ARPANET и, конечно же, Интернету.

Palais, *Fiber Optic Communications*, 5-е издание.

Обычно книги по оптоволоконной технологии предназначаются для специалистов, но эта книга написана более доступным языком. В этой книге рассказывается про волноводы, источники света, детекторы света, соединительные муфты, модуляцию, шум и др.

Su, *The UMTS Air Interface in RF Engineering*

В этой книге дается подробный обзор одной из основный сотовых систем поколения 3G. Основное внимание уделяется радиоинтерфейсу, то есть беспроводным протоколам, которые используются между мобильными телефонами и сетевой инфраструктурой.

Want, *RFID Explained*

Эта книга — легкий для чтения учебник для начинающих о том, как работает необычная технология физического уровня RFID. В книге рассматриваются все аспекты RFID, включая его возможное применение. Приводятся и некоторые примеры реализации RFID и полученного при этом опыта.

9.1.3. Канальный уровень

Kasim, *Delivering Carrier Ethernet*

В настоящее время Ethernet является не только технологией местной связи. Новой формой является использование Ethernet в качестве канала для передачи на большие расстояния с надежностью Ethernet. Книга содержит эссе, подробно затрагивающие эту тему.

Lin and Costello, *Error Control Coding,* 2-е издание.

Коды, обнаруживающие и исправляющие ошибки, — основа надежности компьютерных сетей. Этот популярный учебник объясняет некоторые из самых важных кодов,

от простых линейных кодов Хэмминга до более сложных, имеющих малую плотность кодов проверки четности. Автор пытается использовать минимум необходимой алгебры, но тем не менее ее довольно много.

Stallings, *Data and Computer Communications,* 9-е издание.

Во второй части рассказывается о цифровой передаче данных и различных каналах, в том числе об обнаружении ошибок, контроле ошибок с повторными передачами и управлении потоками.

9.1.4. Подуровень управления доступом к среде

Andrews и др., *Fundamentals of WiMAX*

Эта всесторонняя книга дает полное представление о технологии WiMAX, от идеи широкополосной беспроводной передачи до беспроводных методов с использованием OFDM и нескольких антенн, включая систему мультидоступа. Учебный стиль книги дает самое доступное изложение этого трудного материала.

Gast, *802.11 Wireless Networks,* 2-е издание.

Это хорошее введение для начала знакомства с технологией и протоколами 802.11. Оно начинается с подуровня MAC, а затем представляет материал о других физических уровнях, а также безопасности. Однако, второе издание не достаточно новое, чтобы содержать много информации о 802.11n.

Perlman, *Interconnections,* 2-е издание.

Это заслуживающая доверия и в то же время увлекательно написанная книга о мостах, маршрутизаторах и маршрутизации в целом. Автор книги участвовала в разработке алгоритмов, применяемых в мосте связующего дерева в сетях стандарта IEEE 802, и она, несомненно, является одним из ведущих мировых экспертов в области различных аспектов сетевых технологий.

9.1.5. Сетевой уровень

Comer, *Internetworking with TCP/IP*, том 1, 5-е издание.

Пятое издание наиболее полного труда о наборе протоколов TCP/IP. Первая половина книги посвящена преимущественно протоколу IP и взаимосвязанным протоколам сетевого уровня. В остальных главах, также заслуживающих внимания, описываются в основном более высокие уровни.

Grayson и др., *IP Design for Mobile Networks*

Традиционные телефонные сети и Интернет находятся на противоположных позициях с сетями мобильных телефонов, осуществляемыми на основе IP. Эта книга рассказывает о том, как разработать сеть, которая поддерживает службу мобильного телефона, используя IP-протоколы.

Huitema, *Routing in the Internet*, 2-е издание.

Если вы хотите получить глубокое понимание о протоколах маршрутизации, то эта книга для вас. В ней детально описываются алгоритмы как с произносимыми названиями (например, RIP и CIDR), так и с непроизносимыми (OSPF, IGRP и BGP).

Так как книга достаточно старая, в ней не рассказывается о новейших разработках, но то, что представлено — изложено очень хорошо.

Koodli and Perkins, *Mobile Inter-networking with IPv6*

В этой книге представлены две важные разработки сетевого уровня: IPv6 и Mobile IP. Обе темы затронуты хорошо. Перкинс был одной из движущих сил Mobile IP.

Nucci and Papagiannaki, *Design, Measurement and Management of Large-Scale IP Networks*

Мы много говорили о том, как сети работают, но не о том, как бы вы разрабатывали их, устанавливали и осуществляли управление, если бы были интернет-провайдером. Книга заполняет этот промежуток, представляя современные методы для организации трафика и то, как интернет-провайдеры оказывают услуги, используя сети.

Perlman, *Interconnections,* 2-е издание.

В главах с 12 по 15 автор описывает многочисленные аспекты разработки алгоритмов одноадресной и групповой рассылки как для глобальных, так и для локальных сетей и их реализацию в различных протоколах. Однако безусловно интереснее всего читать главу 18, в которой автор делится своими личными впечатлениями о многолетней работе с сетевыми протоколами. Эта глава и информативна, и весьма забавна, ее следует прочесть всем разработчикам протоколов.

Stevens, *TCP/IP Illustrated*, Том 1

Главы с 3 по 10 содержат доступное описание протокола IP и взаимосвязанных с ним протоколов (ARP, RARP и ICMP), дополненное примерами.

Varghese, *Network Algorithmics*

Мы провели много времени, говоря о том, как маршрутизаторы и другие сетевые элементы взаимодействуют друг с другом. Эта книга другая: она о том, как реально разработаны маршрутизаторы для отправки пакетов на потрясающих скоростях. Книгу необходимо прочесть для внутреннего понимания этого и связанных вопросов. Автор — признанный авторитет в хитрых алгоритмах, которые используются на практике для создания высокоскоростных сетевых элементов в программном и аппаратном обеспечении.

9.1.6. Транспортный уровень

Comer, *Internetworking with TCP/IP*, том 1, 5-е издание.

Как уже говорилось выше, автор написал наиболее полный труд о наборе протоколов TCP/IP. Вторая половина книги посвящена протоколам UDP и TCP.

Farrell and Cahill, *Delay- and Disruption-Tolerant Networking*

Эта короткая книга, которую стоит прочитать для более глубокого понимания архитектуры, протоколов и приложений «неполноценных сетей», которые должны работать при жестких условиях связи. Авторы участвовали в развитии DTN в составе IETF DTN Research Group.

Stevens, *TCP/IP Illustrated*, том 1

Главы с 17 по 24 содержат доступное описание протокола TCP, дополненное примерами.

9.1.7. Прикладной уровень

Berners-Lee и др., *The World Wide Web*

Взгляд из прошлого на Всемирную паутину и на пути ее развития со стороны человека, который ее придумал, и его коллег по CERN. Статья посвящена архитектуре Всемирной паутины, унифицированным указателям (URL), протоколу HTTP, языку HTML, а также перспективам на будущее. Приводится сравнение с другими распределенными информационными системами.

Held, *A Practical Guide to Content Delivery Networks,* 2-е издание.

Эта книга дает практичное представление о том, как действуют CDN, подчеркивая практические соображения о разработке и функционировании CDN, которые работают хорошо.

Hunter и др., *Beginning XML*, 4-е издание.

Есть много, много книг об HTML, XML и веб-сервисах. Эта 1000-страничная книга содержит большую часть того, что Вы, вероятно, захотите узнать. В ней объясняется не только, как написать XML и XHTML, но то, как разработать веб-сервисы, которые создают и манипуллируют XML, используя Ajax, SOAP и другие обычно применяемые методы.

Krishnamurthy и Redfox, *Web Protocols and Practice*

Тяжело найти более понятную книгу, которая при этом охватывала бы все аспекты Всемирной паутины. Здесь, разумеется, рассказывается о клиентах, серверах, прокси и кэшировании. Однако вряд ли вы могли бы предположить, что есть в этой книге разделы, посвященные трафику и его измерению во Всемирной паутине, а также текущим исследованиям и направлениям развития web.

Simpson, *Video Over IP*, 2-е издание.

Автор бросает широкий взгляд на то, как может использоваться IP-технология, чтобы передавать видео по сетям как в Интернете, так и в разработанных для этого частных сетях. Примечательно, что эта книга ориентируется на профессионалов в сфере видео, интересующихся сетями, а не наоборот.

Wittenburg, *Understanding Voice Over IP Technology*

В этой книге рассматривается работа IP-телефонии, от переноса аудиоданных с применением IP-протоколов и проблем качества обслуживания (QoS), до SIP и набора протоколов H.323. Материал обладает необходимой степенью детализации, но при этом доступен и разбит на удобоваримые блоки.

9.1.8. Безопасность в сетях

Anderson, *Security Engineering,* 2-е издание.

Эта книга представляет замечательную смесь методов безопасности, выраженных в понимании того, как люди их используют (и используют неправильно). В ней описываются более технические вещи, нежели в *Secrets and Lies*, однако менее технические, чем в *Network Security* (см. ниже). Вначале даются основы методов защиты информации, затем следуют целые главы, посвященные разным приложениям, включая

банковские системы, системы управления атомной энергетикой, безопасную печать, биометрию, физическую защиту, электронные войны, защиту в телекоммуникациях, электронную коммерцию и защиту авторских прав.

Ferguson и др., *Cryptography Engineering*

Много книг посвящено тому, как работают популярные шифровальные алгоритмы. Эта книга говорит вам, как использовать криптографию — почему шифровальные протоколы разработаны таким образом и как соединить их в систему, которая удовлетворит вашим целям безопасности. Это довольно компактная книга, прочесть которую необходимо каждому, кто разрабатывает системы, которые зависят от криптографии.

Fridrich, *Steganography in Digital Media*

Стеганография применялась еще в древней Греции, где с пустых дощечек удаляли воск, записывали секретное сообщение и снова покрывали табличку воском. В настоящее время видео, аудио и другой контент в Интернете создают различные каналы для секретных сообщений. В книге обсуждаются современные методы для сокрытия и обнаружения информации в изображениях.

Kaufman и др., *Network Security*, 2-е издание.

Эту заслуживающую доверия и часто остроумную книгу следует читать в первую очередь, если вас интересует дополнительная информация о техническом аспекте алгоритмов и протоколов безопасности сетей. В ней подробно освещаются алгоритмы и протоколы с секретным и открытым ключом, хэширование сообщений, аутентификация, протокол Kerberos, PKI, IPsec, SSL/TLS и обеспечение безопасности электронной почты. Все темы снабжены примерами. Глава 26, посвященная фольклору на тему защиты информации, — это настоящий шедевр. В деле обеспечения безопасности важны все детали. Если вы собираетесь разработать действительно полезную систему защиты, эта глава подскажет вам в виде примеров из реальной жизни много интересного.

Schneier, *Secrets and Lies*

Прочитав *Cryptography Engineering* от корки до корки, вы станете знатоком криптографических алгоритмов. Если же вы после этого не менее внимательно прочтете *Secrets and Lies* (что займет уже гораздо меньше времени), вы поймете, что одними алгоритмами дело не ограничивается. Слабость большинства систем защиты связана не с плохими алгоритмами или слишком короткими ключами, а с пороками в окружающей эти системы среде. Эта книга является прекрасным нетехническим описанием систем безопасности, рассматривающим проблему в самом широком смысле.

Skoudis and Liston, *Counter Hack Reloaded, 2*-е издание.

Как остановить взломщика? Надо думать так же, как он. В этой книге показан взгляд на сеть со стороны взломщика; автор утверждает, что защита информации должна быть одной из функций всей сетевой системы в целом, она не должна додумываться и прикручиваться в виде специальной технологии к уже существующим сетям. Рассматриваются почти все типы наиболее распространенных атак, включая «социальный инжиниринг» — тип атаки, рассчитанный на незнание пользователем систем электронной безопасности.

9.2. Алфавитный список литературы

ABRAMSON, N.: "Internet Access Using VSATs," IEEE Commun. Magazine, vol. 38, pp. 60–68, July 2000.

AHMADI, S.: "An Overview of Next-Generation Mobile WiMAX Technology," IEEE Commun. Magazine, vol. 47, pp. 84–88, June 2009.

ALLMAN, M., and PAXSON, V.: "On Estimating End-to-End Network Path Properties," Proc. SIGCOMM '99 Conf., ACM, pp. 263–274, 1999.

ANDERSON, C.: The Long Tail: Why the Future of Business is Selling Less of More, rev. upd. ed., New York: Hyperion, 2008a.

ANDERSON, R.J.: Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd ed., New York: John Wiley & Sons, 2008b.

ANDERSON, R.J.: "Free Speech Online and Office," Computer, vol. 25, pp. 28–30, June 2002.

ANDERSON, R.J.: "The Eternity Service," Proc. Pragocrypt Conf., CTU Publishing House, pp. 242–252, 1996.

ANDREWS, J., GHOSH, A., and MUHAMED, R.: Fundamentals of WiMAX: Understanding Broadband Wireless Networking, Upper Saddle River, NJ: Pearson Education, 2007.

ASTELY, D., DAHLMAN, E., FURUSKAR, A., JADING, Y., LINDSTROM, M., and PARKVALL, S.: "LTE: The Evolution of Mobile Broadband," IEEE Commun. Magazine, vol. 47, pp. 44–51, Apr. 2009.

BALLARDIE, T., FRANCIS, P., and CROWCROFT, J.: "Core Based Trees (CBT)," Proc. SIGCOMM '93 Conf., ACM, pp. 85-95, 1993.

BARAN, P.: "On Distributed Communications: I. Introduction to Distributed Communication Networks," Memorandum RM-420-PR, Rand Corporation, Aug. 1964.

BELLAMY, J.: Digital Telephony, 3rd ed., New Ytfrk: Wiley, 2000.

BELLMAN, R.E.: Dynamic Programming, Princeton, NJ: Princeton University Press, 1957.

BELLOVIN, S.: "The Security Flag in the IPv4 Header," RFC 3514, Apr. 2003.

BELSNES, D.: "Flow Control in the Packet Switching Networks," Communications Networks, Uxbridge, England: Online, pp. 349–361, 1975.

BENNET, C.H., and BRASSARD, G.: "Quantum Cryptography: Public Key Distribution and Coin Tossing," Int'l Conf. on Computer Systems and Signal Processing, pp. 175–179, 1984.

BERESFORD, A., and STAJANO, F.: "Location Privacy in Pervasive Computing," IEEE Pervasive Computing, vol. 2, pp. 46–55, Jan. 2003.

BERGHEL, H.L.: "Cyber Privacy in the New Millennium," Computer, vol. 34, pp. 132–134, Jan. 2001.

BERNERS-LEE, T., CAILLIAU, A., LOUTONEN, A., NIELSEN, H.F., and SECRET, A.: "The World Wide Web," Commun. of the ACM, vol. 37, pp. 76–82, Aug. 1994.

BERTSEKAS, D., and GALLAGER, R.: Data Networks, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1992.

BHATTI, S.N., and CROWCROFT, J.: "QoS Sensitive Flows: Issues in IP Packet Handling," IEEE Internet Computing, vol. 4, pp. 48–57, July–Aug. 2000.

BIHAM, E., and SHAMIR, A.: "Differential Cryptanalysis of the Data Encryption Standard," Proc. 17th Ann. Int'l Cryptology Conf., Berlin: Springer-Verlag LNCS 1294, pp. 513–525, 1997.

BIRD, R., GOPAL, I., HERZBERG, A., JANSON, P.A., KUTTEN, S., MOLVA, R., and YUNG, M.: "Systematic Design of a Family of Attack-Resistant Authentication Protocols," IEEE J. on Selected Areas in Commun., vol. 11, pp. 679–693, June 1993.

BIRRELL, A.D., and NELSON, B.J.: "Implementing Remote Procedure Calls," ACM Trans, on Computer Systems, vol. 2, pp. 39–59, Feb. 1984.

BIRYUKOV, A., SHAMIR, A., and WAGNER, D.: "Real Time Cryptanalysis of A5/1 on a PC," Proc. Seventh Int'l Workshop on Fast Software Encryption, Berlin: Springer-Verlag LNCS 1978, p. 1, 2000.

BLAZE, M., and BELLOVIN, S.: "Tapping on My Network Door," Commun. of the ACM, vol. 43, p. 136, Oct. 2000.

BOGGS, D., MOGUL, J., and KENT, C.: "Measured Capacity of an Ethernet: Myths and Reality," Proc. SIGCOMM '88 Conf., ACM, pp. 222–234, 1988.

BORISOV, N., GOLDBERG, I., and WAGNER, D.: "Intercepting Mobile Communications: The Insecurity of 802.11," Seventh Int'l Conf. on Mobile Computing and Networking, ACM, pp. 180–188, 2001.

BRADEN, R.: "Requirements for Internet Hosts—Communication Layers," RFC 1122, Oct. 1989.

BRADEN, R., BORMAN, D., and PARTRIDGE, C.: "Computing the Internet Checksum," RFC 1071, Sept. 1988.

BRANDENBURG, K.: "MP3 and AAC Explained," Proc. 17th Intl. Conf.: High-Quality Audio Coding, Audio Engineering Society, pp. 99–110, Aug. 1999.

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C., MALER, E., YERGEAU, F., and COWAN, J.: "Extensible Markup Language (XML) 1.1 (Second Edition)," W3C Recommendation, Sept. 2006.

BRESLAU, L., CAO, P., FAN, L., PHILLIPS, G., and SHENKER, S.: "Web Caching and Zipf-like Distributions: Evidence and Implications," Proc. INFOCOM Conf., IEEE, pp. 126–134, 1999.

BURLEIGH, S., HOOKE, A., TORGERSON, L., FALL, K., CERF, V., DURST, B., SCOTT, K., and WEISS, H.: "Delay-Tolerant Networking: An Approach to Interplanetary Internet," IEEE Commun. Magazine, vol. 41, pp. 128–136, June 2003.

BURNETT, S., and PAINE, S.: RSA Security's Official Guide to Cryptography, Berkeley, CA: Osborne/McGraw-Hill, 2001.

BUSH, V.: "As We May Think," Atlantic Monthly, vol. 176, pp. 101–108, July 1945.

CAPETANAKIS, J.I.: "Tree Algorithms for Packet Broadcast Channels," IEEE Trans, on Information Theory, vol. IT-25, pp. 505–515, Sept. 1979.

CASTAGNOLI, G., BRAUER, S., and HERRMANN, M.: "Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits," IEEE Trans. on Commun., vol. 41, pp. 883–892, June 1993.

CERF, V., and KAHN, R.: "A Protocol for Packet Network Interconnection," IEEE Trans, on Commun., vol. COM-22, pp. 637–648, Way 1974.

CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W., WALLACH, D., BURROWS, M., CHANDRA, T., FIKES, A., and GRUBER, R.: "Bigtable: A Distributed Storage System for Structured Data," Proc. OSDI 2006 Symp., USENIX, pp. 15–29, 2006.

CHASE, J.S., GALLATIN, A.J., and YOCUM, K.G.: "End System Optimizations for High-Speed TCP," IEEE Commun. Magazine, vol. 39, pp. 68–75, April 2001.

CHEN, S., and NAHRSTEDT, K.: "An Overview of QoS Routing for Next-Generation Networks," IEEE Network Magazine, vol. 12, pp. 64–69, Nov./Dec. 1998.

CHIU, D., and JAIN, R.: "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," Comput. Netw. ISDN Syst., vol. 17, pp. 1–4, June 1989.

CISCO: "Cisco Visual Networking Index: Forecast and Methodology, 2009–2014," Cisco Systems Inc., June 2010.

CLARK, D.D.: "The Design Philosophy of the DARPA Internet Protocols," Proc. SIGCOMM '88 Conf., ACM, pp. 106–114, 1988.

CLARK, D.D.: "Window and Acknowledgement Strategy in TCP," RFC 813, July 1982.

CLARK, D.D., JACOBSON, V., ROMKEY, J., and SAL WEN, H.: "An Analysis of TCP Processing Overhead," IEEE Commun. Magazine, vol. 27, pp. 23–29, June 1989.

CLARK, D.D., SHENKER, S., and ZHANG, L.: "Supporting Real-Time Applications in an Integrated Services Packet Network," Proc. SIGCOMM '92 Conf., ACM, pp. 14–26, 1992.

CLARKE, A.C.: "Extra-Terrestrial Relays," Wireless World, 1945.

CLARKE, I., MILLER, S.G., HONG, T.W., SANDBERG, O., and WILEY, B.: "Protecting Free Expression Online with Freenet," IEEE Internet Computing, vol. 6, pp. 40–49, Jan.-Feb. 2002.

COHEN, B.: "Incentives Build Robustness in BitTorrent," Proc. First Workshop on Economics of Peer-to-Peer Systems, June 2003.

COMER, D.E.: The Internet Book, 4th ed., Englewood Cliffs, NJ: Prentice Hall, 2007.

COMER, D.E.: Internetworking with TCP/IP, vol. 1, 5th ed., Englewood Cliffs, NJ: Prentice Hall, 2005.

CRAVER, S.A., WU, M., LIU, B., STUBBLEFIELD, A., SWARTZLANDER, B., WALLACH, D.W., DEAN, D., and FELTEN, E.W.: "Reading Between the Lines: Lessons from the SDMI Challenge," Proc. 10th USENIX Security Symp., USENIX, 2001.

CROVELLA, M., and KRISHNAMURTHY, B.: Internet Measurement, New York: John Wiley & Sons, 2006.

DAEMEN, J., and RIJMEN, V.: The Design of Rijndael, Berlin: Springer-Verlag, 2002.

DALAL, Y., and METCLFE, R.: "Reverse Path Forwarding of Broadcast Packets," Commun. of the ACM, vol. 21, pp. 1040–1048, Dec. 1978.

DAVIE, B., and FARREL, A.: MPLS: Next Steps, San Francisco: Morgan Kaufmann, 2008.

DAVIE, B., and REKHTER, Y.: MPLS Technology and Applications, San Francisco: Morgan Kaufmann, 2000.

DAVIES, J.: Understanding IPv6, 2nd ed., Redmond, WA: Microsoft Press, 2008.

DAY, J.D.: "The (Un)Revised OSI Reference Model," Computer Commun. Rev., vol. 25, pp. 39-55, Oct. 1995.

DAY, J.D., and ZIMMERMANN, H.: "The OSI Reference Model," Proc. of the IEEE, vol. 71, pp. 1334–1340, Dec. 1983.

DECANDIA, G., HASTORIN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., and VOGELS, W.: "Dynamo: Amazon's Highly Available Key-value Store," Proc. 19th Symp. on Operating Systems Prin., ACM, pp. 205–220, Dec. 2007.

DEERING, S.E.: "SIP: Simple Internet Protocol," IEEE Network Magazine, vol. 7, pp. 16-28, May/June 1993.

DEERING, S., and CHERITON, D.: "Multicast Routing in Datagram Networks and Extended LANs," ACM Trans. on Computer Systems, vol. 8, pp. 85–110, May 1990.

DEMERS, A., KESHAV, S., and SHENKER, S.: "Analysis and Simulation of a Fair Queue-ing Algorithm," Internetwork: Research and Experience, vol. 1, pp. 3–26, Sept. 1990.

DENNING, D.E., and SACCO, G.M.: "Timestamps in Key Distribution Protocols," Commun. of the ACM, vol. 24, pp. 533–536, Aug. 1981.

DEVARAPALLI, V., WAKIKAWA, R., PETRESCU, A., and THUBERT, P.: "Network Mobility (NEMO) Basic Support Protocol," RFC 3963, Jan. 2005.

DIFFIE, W., and HELLMAN, M.E.: "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," IEEE Computer, vol. 10, pp. 74–84, June 1977.

DIFFIE, W., and HELLMAN, M.E.: "New Directions in Cryptography," IEEE Trans, on Information Theory, vol. IT-2, pp. 644–654, Nov. 1976.

DIJKSTRA, E.W.: "A Note on Two Problems in Connexion with Graphs," Numer. Math., vol. 1, pp. 269–271, Oct. 1959.

DILLEY, J., MAGGS, B., PARIKH, J., PROKOP, H., SITARAMAN, R., and WHEIL, B.: "Globally Distributed Content Delivery," IEEE Internet Computing, vol. 6, pp. 50–58, 2002.

DINGLEDINE, R., MATHEWSON, N., SYVERSON, P.: "Tor: The Second-Generation Onion Router," Proc. 13th USENIX Security Symp., USENIX, pp. 303–320, Aug. 2004.

DONAHOO, M., and CALVERT, K.: TCP/IP Sockets in C, 2nd ed., San Francisco: Morgan Kaufmann, 2009.

DONAHOO, M., and CALVERT, K.: TCP/IP Sockets in Java, 2nd ed., San Francisco: Morgan Kaufmann, 2008.

DONALDSON, G., and JONES, D.: "Cable Television Broadband Network Architectures," IEEE Commun. Magazine, vol. 39, pp. 122–126, June 2001.

DORFMAN, R.: "Detection of Defective Members of a Large Population," Annals Math. Statistics, vol. 14, pp. 436–440, 1943.

DUTCHER, B.: The NAT Handbook, New York: John Wiley&Sons, 2001.

DUTTA-ROY, A.: "An Overview of Cable Modem Technology and Market Perspectives," IEEE Commun. Magazine, vol. 39, pp. 81–88, June 2001.

EDELMAN, B., OSTROVSKY, M., and SCHWARZ, M.: "Internet Advertising and the Generalized Second-Price Auction: Selling Billions of Dollars Worth of Keywords," American Economic Review, vol. 97, pp. 242–259, Mar. 2007.

EL GAMAL, T.: "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," IEEE Trans, on Information Theory, vol. IT-1, pp. 469–472, July 1985.

EPCGLOBAL: EPC Radio-Frequency Identity Protocols Class– Generation– UHF RFID Protocol for Communication at 860-MHz to 960-MHz Version 1.2.0, Brussels: EPCglobal Inc., Oct. 2008.

FALL, K.: "A Delay-Tolerant Network Architecture for Challenged Internets," Proc. SIGCOMM 2003 Conf., ACM, pp. 27–34, Aug. 2003.

FALOUTSOS, M., FALOUTSOS, P., and FALOUTSOS, C.: "On Power-Law Relationships of the Internet Topology," Proc. SIGCOMM '99 Conf., ACM, pp. 251–262, 1999.

FARRELL, S., and CAHILL, V.: Delay- and Disruption-Tolerant Networking, London: Artech House, 2007.

FELLOWS, D., and JONES, D.: "DOCSIS Cable Modem Technology," IEEE Commun. Magazine, vol. 39, pp. 202–209, Mar. 2001.

FENNER, B., HANDLEY, M., HOLBROOK, H., and KOUVELAS, I.: "Protocol Independent Multicast-Sparse Mode (PIM-SM)," RFC 4601, Aug. 2006.

FERGUSON, N., SCHNEIER, B., and KOHNO, T.: Cryptography Engineering: Design Principles and Practical Applications, New York: John Wiley & Sons, 2010.

FLANAGAN, D.: JavaScript: The Definitive Guide, 6th ed., Sebastopol, CA: O'Reilly, 2010.

FLETCHER, J.: "An Arithmetic Checksum for Serial Transmissions," IEEE Trans. on Commun., vol. COM–0, pp. 247–252, Jan. 1982.

FLOYD, S., HANDLEY, M., PADHYE, J., and WIDMER, J.: "Equation-Based Congestion Control for Unicast Applications," Proc. SIGCOMM 2000 Conf., ACM, pp. 43–56, Aug. 2000.

FLOYD, S., and JACOBSON, V.: "Random Early Detection for Congestion Avoidance," IEEE/ACM Trans, on Networking, vol. 1, pp. 397–413, Aug. 1993.

FLUHRER, S., MANTIN, I., and SHAMIR, A.: "Weakness in the Key Scheduling Algorithm of RC4," Proc. Eighth Ann. Workshop on Selected Areas in Cryptography, Berlin: Springer-Verlag LNCS 2259, pp. 1–24, 2001.

FORD, B.: "Structured Streams: A New Transport Abstraction," Proc. SIGCOMM 2007 Conf., ACM, pp. 361–372, 2007.

FORD, L.R., Jr., and FULKERSON, D.R.: Flows in Networks, Princeton, NJ: Princeton University Press, 1962.

FORD, W., and BAUM, M.S.: Secure Electronic Commerce, Upper Saddle River, NJ: Prentice Hall, 2000.

FOULI, K., and MALER, M.: "The Road to Carrier-Grade Ethernet," IEEE Commun.
Magazine, vol. 47, pp. S30–S38, Mar. 2009.

FOX, A., GRIBBLE, S., BREWER, E., and AMIR, E.: "Adapting to Network and Client
Variability via On-Demand Dynamic Distillation," SIGOPS Oper. Syst. Rev., vol. 30,
pp. 160–170, Dec. 1996.

FRANCIS, P.: "A Near-Term Architecture for Deploying Pip," IEEE Network Magazine, vol. 7, pp. 30–37, May/June 1993.

FRASER, A.G.: "Towards a Universal Data Transport System," IEEE J. on Selected Areas in Commun., vol. 5, pp. 803–816, Nov. 1983.

FRIDRICH, J.: Steganography in Digital Media: Principles, Algorithms, and Applications, Cambridge: Cambridge University Press, 2009.

FULLER, V., and LI, T.: "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan," RFC 4632, Aug. 2006.

GALLAGHER, R.G.: "A Minimum Delay Routing Algorithm Using Distributed Computation," IEEE Trans. on Commun., vol. COM–5, pp. 73–85, Jan. 1977.

GALLAGHER, R.G.: "Low-Density Parity Check Codes," IRE Trans. on Information Theory, vol. 8, pp. 21–28, Jan. 1962.

GARFINKEL, S., with SPAFFORD, G.: Web Security, Privacy, and Commerce, Sebastopol, CA: O'Reilly, 2002.

GAST, M.: 802.11 Wireless Networks: The Definitive Guide, 2nd ed., Sebastopol, CA: O'Reilly, 2005.

GERSHENFELD, N., and KRIKORIAN, R., and COHEN, D.: "The Internet of Things," Scientific American, vol. 291, pp. 76–81, Oct. 2004.

GILDER, G.: "Metcalfe's Law and Legacy," Forbes ASAP, Sepy. 13, 1993.

GOODE, B.: "Voice over Internet Protocol," Proc. of the IEEE, vol. 90, pp. 1495–1517, Sept. 2002.

GORALSKI, W.J.: SONET, 2nd ed., New York: McGraw-Hill, 2002.

GRAYSON, M., SHATZKAMER, K., and WAINNER, S.: IP Design for Mobile Networks, Indianapolis, IN: Cisco Press, 2009.

GROBE, K., and ELBERS, J.: "PON in Adolescence: From TDMA to WDM-PON," IEEE Commun. Magazine, vol. 46, pp. 26–34, Jan. 2008.

GROSS, G., KAYCEE, M., LIN, A., MALIS, A., and STEPHENS, J.: "The PPP Over AAL5," RFC 2364, July 1998.

HA, S., RHEE, I., and LISONG, X.: "CUBIC: A New TCP-Friendly High-Speed TCP Variant," SIGOPS Oper. Syst. Rev., vol. 42, pp. 64–74, June 2008.

HAFNER, K., and LYON, M.: Where Wizards Stay Up Late, New York: Simon & Schuster, 1998.

HALPERIN, D., HEYDT-BENJAMIN, T., RANSFORD, B., CLARK, S., DEFEND, B., MORGAN, W., FU, K., KOHNO, T., and MAISEL, W.: "Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses," IEEE Symp. on Security and Privacy, pp. 129–142, May 2008.

HALPERIN, D., HU, W., SHETH, A., and WETHERALL, D.: "802.11 with Multiple Antennas for Dummies," Computer Commun. Rev., vol. 40, pp. 19–25, Jan. 2010.

HAMMING, R.W.: "Error Detecting and Error Correcting Codes," Bell System Tech. J., vol. 29, pp. 147–160, April 1950.

HARTE, L., KELLOGG, S., DREHER, R., and SCHAFFNIT, T.: The Comprehensive Guide to Wireless Technology, Fuquay-Varina, NC: APDG Publishing, 2000.

HAWLEY, G.T.: "Historical Perspectives on the U.S. Telephone Loop," IEEE Commun. Magazine, vol. 29, pp. 24–28, March 1991.

HECHT, J.: "Understanding Fiber Optics," Upper Saddle River, NJ: Prentice Hall, 2005.

HELD, G.: A Practical Guide to Content Delivery Networks, 2nd ed., Boca Raton, FL: CRC Press, 2010.

HEUSSE, M., ROUSSEAU, F., BERGER-SABBATEL, G., DUDA, A.: "Performance Anomaly of 802.11b," Proc. INFOCOM Conf., IEEE, pp. 836–843, 2003.

HIERTZ, G., DENTENEER, D., STIBOR, L., ZANG, Y., COSTA, X., and WALKE, B.: "The IEEE 802.11 Universe," IEEE Commun. Magazine, vol. 48, pp. 62–70, Jan. 2010.

HOE, J.: "Improving the Start-up Behavior of a Congestion Control Scheme for TCP," Proc. SIGCOMM '96 Conf., ACM, pp. 270–280, 1996.

HU, Y., and LI, V.O.K.:"Satellite-Based Internet: A Tutorial," IEEE Commun. Magazine, vol. 30, pp. 154–162, Mar. 2001.

HUITEMA, C.: Routing in the Internet, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1999.

HULL, B., BYCHKOVSKY, V., CHEN, K., GORACZKO, M., MIU, A., SHIH, E., ZHANG, Y., BALAKRISHNAN, H., and MADDEN, S.: "CarTel: A Distributed Mobile Sensor Computing System," Proc. Sensys 2006 Conf., ACM, pp. 125–138, Nov. 2006.

HUNTER, D., RAFTER, J., FAWCETT, J., VAN DER LIST, E., AYERS, D., DUCKETT, J., WATT, A., and MCKINNON, L.: Beginning XML, 4th ed., New Jersey: Wrox, 2007.

IRMER, T.: "Shaping Future Telecommunications: The Challenge of Global Standardization," IEEE Commun. Magazine, vol. 32, pp. 20–28, Jan. 1994.

ITU (INTERNATIONAL TELECOMMUNICATION UNION): ITU Internet Reports 2005: The Internet of Things, Geneva: ITU, Nov. 2005.

ITU (INTERNATIONAL TELECOMMUNICATION UNION): Measuring the Information Society: The ICT Development Index, Geneva: ITU, Mar. 2009.

JACOBSON, V.: "Compressing TCP/IP Headers for Low-Speed Serial Links," RFC 1144, Feb. 1990.

JACOBSON, V.: "Congestion Avoidance and Control," Proc. SIGCOMM '88 Conf., ACM, pp. 314–329, 1988.

JAIN, R., and ROUTHIER, S.: "Packet Trains—Measurements and a New Model for Computer Network Traffic," IEEE J. on Selected Areas in Commun., vol. 6, pp. 986–995, Sept. 1986.

JAKOBSSON, M., and WETZEL, S.: "Security Weaknesses in Bluetooth," Topics in Cryp-tology: CT-RSA 2001, Berlin: Springer-Verlag LNCS 2020, pp. 176–191, 2001.

JOEL, A.: "Telecommunications and the IEEE Communications Society," IEEE Commun. Magazine, 50th Anniversary Issue, pp. 6-14 and 162–167, May 2002.

JOHNSON, D., PERKINS, C., and ARKKO, J.: "Mobility Support in IPv6," RFC 3775, June 2004.

JOHNSON, D.B., MALTZ, D., and BROCH, J.: "DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks," Ad Hoc Networking, Boston: Addison-Wesley, pp. 139–172, 2001.

JUANG, P., OKI, H., WANG, Y., MARTONOSI, M., PEH, L., and RUBENSTEIN, D.: "Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet," SIGOPS Oper. Syst. Rev., vol. 36, pp. 96–107, Oct. 2002.

KAHN, D.: The Codebreakers, 2nd ed., New York: Macmillan, 1995.

KAMOUN, F., and KLEINROCK, L.: "Stochastic Performance Evaluation of Hierarchical Routing for Large Networks," Computer Networks, vol. 3, pp. 337–353, Nov. 1979.

KARN, P.: "MACA—A New Channel Access Protocol for Packet Radio," ARRL/CRRL Amateur Radio Ninth Computer Networking Conf., pp. 134–140, 1990.

KARN, P., and PARTRIDGE, C.: "Improving Round-Trip Estimates in Reliable Transport Protocols," Proc. SIGCOMM '87 Conf., ACM, pp. 2–7, 1987.

KARP, B., and KUNG, H.T.: "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," Proc. MOBICOM 2000 Conf., ACM, pp. 243–254, 2000.

KASIM, A.: Delivering Carrier Ethernet, New York: McGraw-Hill, 2007.

KATABI, D., HANDLEY, M., and ROHRS, C.: "Internet Congestion Control for Future High Bandwidth-Delay Product Environments," Proc. SIGCOMM 2002 Conf., ACM, pp. 89–102, 2002.

KATZ, D., and FORD, P.S.: "TUBA: Replacing IP with CLNP," IEEE Network Magazine, vol. 7, pp. 38–47, May/June 1993.

KAUFMAN, C., PERLMAN, R., and SPECINER, M.: Network Security, 2nd ed., Engle-wood Cliffs, NJ: Prentice Hall, 2002.

KENT, C., and MOGUL, J.: "Fragmentation Considered Harmful," Proc. SIGCOMM '87 Conf., ACM, pp. 390–401, 1987.

KERCKHOFF, A.: "La Cryptographic Militaire," *J. des Sciences Militaires*, vol. 9, pp. 5–38, Jan. 1883 and pp. 161–191, Feb. 1883.

KHANNA, A., and ZINKY, J.: "The Revised ARPANET Routing Metric," Proc. SIGCOMM '89 Conf., ACM, pp. 45–56, 1989.

KIPNIS, J.: "Beating the System: Abuses of the Standards Adoptions Process," IEEE Commun. Magazine, vol. 38, pp. 102–105, July 2000.

KLEINROCK, L.: "Power and Other Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications," Proc. Intl. Conf. on Commun., pp. 43.1.1–43.1.10, June 1979.

KLEINROCK, L., and TOBAGI, F.: "Random Access Techniques for Data Transmission over Packet-Switched Radio Channels," Proc. Nat. Computer Conf., pp. 187-201, 1975.

KOHLER, E., HANDLEY, H., and FLOYD, S.: "Designing DCCP: Congestion Control without Reliability," Proc. SIGCOMM 2006 Conf., ACM, pp. 27–38, 2006.

KOODLI, R., and PERKINS, C.E.: Mobile Inter-networking with IPv6, New York: John Wiley & Sons, 2007.

KOOPMAN, P.: "32-Bit Cyclic Redundancy Codes for Internet Applications," Proc. Intl. Conf. on Dependable Systems and Networks., IEEE, pp. 459–472, 2002.

KRISHNAMURTHY, B., and REXFORD, J.: Web Protocols and Practice, Boston: Addison-Wesley, 2001.

KUMAR, S., PAAR, C., PELZL, J., PFEIFFER, G., and SCHIMMLER, M.: "Breaking Ciphers with COPACOBANA: A Cost-Optimized Parallel Code Breaker," Proc. 8th Cryptographic Hardware and Embedded Systems Wksp., IACR, pp. 101–118, Oct. 2006.

LABOVITZ, C., AHUJA, A., BOSE, A., and JAHANIAN, F.: "Delayed Internet Routing Convergence," IEEE/ACM Trans. on Networking, vol. 9, pp. 293–306, June 2001.

LAM, C.K.M., and TAN, B.C.Y.: "The Internet Is Changing the Music Industry," Commun. of the ACM, vol. 44, pp. 62–66, Aug. 2001.

LAOUTARIS, N., SMARAGDAKIS, G., RODRIGUEZ, P., and SUNDARAM, R.: "Delay Tolerant Bulk Data Transfers on the Internet," Proc. SIGMETRICS 2009 Conf., ACM, pp. 229–238, June 2009.

LARMO, A., LINDSTROM, M., MEYER, M., PELLETIER, G., TORSNER, J., and WIEMANN, H.: "The LTE Link-Layer Design," IEEE Commun. Magazine, vol. 47, pp. 52–59, Apr. 2009.

LEE, J.S., and MILLER, L.E.: CDMA Systems Engineering Handbook, London: Artech House, 1998.

LELAND, W., TAQQU, M., WILLINGER, W., and WILSON, D.: "On the Self-Similar Nature of Ethernet Traffic," IEEE/ACM Trans. on Networking, vol. 2, pp. 1–15, Feb. 1994.

LEMON, J.: "Resisting SYN Flood DOS Attacks with a SYN Cache," Proc. BSDCon Conf., USENIX, pp. 88–98, 2002.

LEVY, S.: "Crypto Rebels," Wired, pp. 54–61, May/June 1993.

LEWIS, M.: Comparing, Designing, and Deploying VPNs, Indianapolis, IN: Cisco Press, 2006.

LI, M., AGRAWAL, D., GANESAN, D., and VENKATARAMANI, A.: "Block-Switched Networks: A New Paradigm for Wireless Transport," Proc. NSDI 2009 Conf., USENIX, pp. 423–436, 2009.

LIN, S., and COSTELLO, D.: Error Control Coding, 2nd ed., Upper Saddle River, NJ: Pearson Education, 2004.

LUBACZ, J., MAZURCZYK, W., and SZCZYPIORSKI, K.: "Vice over IP," IEEE Spectrum, pp. 42–47, Feb. 2010.

MACEDONIA, M.R.: "Distributed File Sharing," IEEE Computer, vol. 33, pp. 99–101, 2000.

MADHAVAN, J., KO, D., LOT, L., GANGPATHY, V., RASMUSSEN, A., and HALEVY, A.: "Google's Deep Web Crawl," Proc. VLDB 2008 Conf., VLDB Endowment, pp. 1241–1252, 2008.

MAHAJAN, R., RODRIG, M., WETHERALL, D., and ZAHORJAN, J.: "Analyzing the MAC-Level Behavior of Wireless Networks in the Wild," Proc. SIGCOMM 2006 Conf., ACM, pp. 75–86, 2006.

MALIS, A., and SIMPSON, W.: "PPP over SONET/SDH," RFC 2615, June 1999.

MASSEY, J.L.: "Shift-Register Synthesis and BCH Decoding," IEEE Trans. on Information Theory, vol. IT–5, pp. 122–127, Jan. 1969.

MATSUI, M.: "Linear Cryptanalysis Method for DES Cipher," Advances in Cryptology— Eurocrypt 1993 Proceedings, Berlin: Springer-Verlag LNCS 765, pp. 386–397, 1994.

MAUFER, T.A.: IP Fundamentals, Upper Saddle River, NJ: Prentice Hall, 1999.

MAYMOUNKOV, P., and MAZIERES, D.: "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," Proc. First Intl. Wksp. on Peer-to-Peer Systems, Berlin: Springer-Verlag LNCS 2429, pp. 53–65, 2002.

MAZIERES, D., and KAASHOEK, M.F.: "The Design, Implementation, and Operation of an Email Pseudonym Server," Proc. Fifth Conf. on Computer and Commun. Security, ACM, pp. 27–36, 1998.

MCAFEE LABS: McAfee Threat Reports: First Quarter 2010, McAfee Inc., 2010.
MENEZES, A.J., and VANSTONE, S.A.: "Elliptic Curve Cryptosystems and Their Implementation," Journal of Cryptology, vol. 6, pp. 209–224, 1993.
MERKLE, R.C., and HELLMAN, M.: "Hiding and Signatures in Trapdoor Knapsacks," IEEE Trans. on Information Theory, vol. IT–4, pp. 525–530, Sept. 1978.
METCALFE, R.M.: "Computer/Network Interface Design: Lessons from Arpanet and Ethernet," IEEE J. on Selected Areas in Commun., vol. 11, pp. 173–179, Feb. 1993.
METCALFE, R.M., and BOGGS, D.R.: "Ethernet: Distributed Packet Switching for Local Computer Networks," Commun. of the ACM, vol. 19, pp. 395–404, July 1976.
METZ, C.: "Interconnecting ISP Networks," IEEE Internet Computing, vol. 5, pp. 74–80, Mar.-Apr. 2001.
MISHRA, P.P., KANAKIA, H., and TRIPATHI, S.: "On Hop by Hop Rate-Based Congestion Control," IEEE/ACM Trans. on Networking, vol. 4, pp. 224–239, Apr. 1996.
MOGUL, J.C.: "IP Network Performance," in Internet System Handbook, D.C. Lynch and M.T. Rose (eds.), Boston: Addison-Wesley, pp. 575–675, 1993.
MOGUL, J., and DEERING, S.: "Path MTU Discovery," RFC 1191, Nov. 1990.
MOGUL, J., and MINSHALL, G.: "Rethinking the Nagle Algorithm," Comput. Commun. Rev., vol. 31, pp. 6–20, Jan. 2001.
MOY, J.: "Multicast Routing Extensions for OSPF" Commun. of the ACM, vol. 37, pp. 61–66, AUg. 1994.
MULLINS, J.: "Making Unbreakable Code," IEEE Spectrum, pp. 40-45, May 2002.
NAGLE, J.: "On Packet Switches with Infinite Storage," IEEE Trans, on Commun., vol. COM-5, pp. 435–438, Apr. 1987.
NAGLE, J.: "Congestion Control in TCP/IP Internetworks," Computer Commun. Rev., vol. 14, pp. 11–17, Oct. 1984.
NAUGHTON, J.: "A Brief History of the Future," Woodstock, NY: Overlook Press, 2000.
NEEDHAM, R.M., and SCHROEDER, M.D.: "Using Encryption for Authentication in Large Networks of Computers," Commun. of the ACM, vol. 21, pp. 993–999, Dec. 1978.
NEEDHAM, R.M., and SCHROEDER, M.D.: "Authentication Revisited," Operating Systems Rev., vol. 21, p. 7, Jan. 1987.
NELAKUDITI, S., and ZHANG, Z.-L.: "A Localized Adaptive Proportioning Approach to QoS Routing," IEEE Commun. Magazine vol. 40, pp. 66–71, June 2002.
NEUMAN, C., and TS'O, T.: "Kerberos: An Authentication Service for Computer Networks," IEEE Commun. Mag., vol. 32, pp. 33–38, Sept. 1994.
NICHOLS, R.K., and LEKKAS, P.C.: Wireless Security, New York: McGraw-Hill, 2002.
NIST: "Secure Hash Algorithm," U.S. Government Federal Information Processing Standardise, 1993.
NONNENMACHER, J., BIERSACK, E., and TOWSLEY, D.: "Parity-Based Loss Recovery for Reliable Multicast Transmission," Proc. SIGCOMM '97 Conf., ACM, pp. 289–300, 1997.
NUCCI, A., and PAPAGIANNAKI, D.: Design, Measurement and Management of Large- Scale IP Networks, Cambridge: Cambridge University Press, 2008.
NUGENT, R., MUNAKANA, R., CHIN, A., COELHO, R., and PUIG-SUARI, J.: "The CubeSat: The PicoSatellite Standard for Research and Education," Proc. SPACE 2008 Conf., AIAA, 2008.
ORAN, D.: "OSI IS-IS Intra-domain Routing Protocol," RFC 1142, Feb. 1990.
OTWAY, D., and REES, O.: "Efficient and Timely Mutual Authentication," Operating Systems Rev., pp. 8–10, Jan. 1987.
PADHYE, J., FIROIU, V., TOWSLEY, D., and KUROSE, J.: "Modeling TCP Throughput: A Simple Model and Its Empirical Validation," Proc. SIGCOMM '98 Conf., ACM, pp. 303–314, 1998.
PALAIS, J.C.: Fiber Optic Commun., 5rd ed., Englewood Cliffs, NJ: Prentice Hall, 2004.
PARAMESWARAN, M., SUSARLA, A., and WHINSTON, A.B.: "P2P Networking: An Information-Sharing Alternative," Computer, vol. 34, pp. 31–38, July 2001.

PAREKH, A., and GALLAGHER, R.: "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple-Node Case," IEEE/ACM Trans. on Networking, vol. 2, pp. 137–150, Apr. 1994.

PAREKH, A., and GALLAGHER, R.: "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," IEEE/ACM Trans. on Networking, vol. 1, pp. 344–357, June 1993.

PARTRIDGE, C., HUGHES, J., and STONE, J.: "Performance of Checksums and CRCs over Real Data," Proc. SIGCOMM '95 Conf., ACM, pp. 68–76, 1995.

PARTRIDGE, C., MENDEZ, T., and MILLIKEN, W.: "Host Anycasting Service," RFC 1546, Nov. 1993.

PAXSON, V., and FLOYD, S.: "Wide-Area Traffic: The Failure of Poisson Modeling," IEEE/ACM Trans. on Networking, vol. 3, pp. 226–244, June 1995.

PERKINS, C.: "IP Mobility Support for IPv4," RFC 3344, Aug. 2002.

PERKINS, C.E.: RTP: Audio and Video for the Internet, Boston: Addison-Wesley, 2003. *PERKINS, C.E. (ed.)*: Ad Hoc Networking, Boston: Addison-Wesley, 2001.

PERKINS, C.E.: Mobile IP Design Principles and Practices, Upper Saddle River, NJ: Prentice Hall, 1998.

PERKINS, C.E., and ROYER, E.: "The Ad Hoc On-Demand Distance-Vector Protocol," in Ad Hoc Networking, edited by C. Perkins, Boston: Addison-Wesley, 2001.

PERLMAN, R.: Interconnections, 2nd ed., Boston: Addison-Wesley, 2000.

PERLMAN, R.: Network Layer Protocols with Byzantine Robustness, Ph.D. thesis, M.I.T., 1988.

PERLMAN, R.: "An Algorithm for the Distributed Computation of a Spanning Tree in an Extended LAN," Proc. SIGCOMM '85 Conf., ACM, pp. 44–53, 1985.

PERLMAN, R., and KAUFMAN, C.: "Key Exchange in IPsec," IEEE Internet Computing, vol. 4, pp. 50–56, Nov.–Dec. 2000.

PETERSON, W.W., and BROWN, D.T.: "Cyclic Codes for Error Detection," Proc. IRE, vol. 49, pp. 228–235, Jan. 1961.

PIATEK, M., KOHNO, T., and KRISHNAMURTHY, A.: "Challenges and Directions for Monitoring P2P File Sharing Networks—or Why My Printer Received a DMCA Takedown Notice," 3rd Workshop on Hot Topics in Security, USENIX, July 2008.

PIATEK, M., ISDAL, T., ANDERSON, T., KRISHNAMURTHY, A., and VENKATARAMANI, V.: "Do Incentives Build Robustness in BitTorrent?," Proc. NSDI 2007 Conf., USENIX, pp. 1–14, 2007.

PISCITELLO, D.M., and CHAPIN, A.L.: Open Systems Networking: TCP/IP and OSI, Boston: Addison-Wesley, 1993.

PIVA, A., BARTOLINI, F., and BARNI, M.: "Managing Copyrights in Open Networks," IEEE Internet Computing, vol. 6, pp. 18–26, May– 2002.

POSTEL, J.: "Internet Control Message Protocols," RFC 792, Sept. 1981.

RABIN, J., and MCCATHIENEVILE, C.: "Mobile Web Best Practices 1.0," W3C Recommendation, July 2008.

RAMAKRISHNAM, K.K., FLOYD, S., and BLACK, D.: "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sept. 2001.

RAMAKRISHNAN, K.K., and JAIN, R.: "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer," Proc. SIGCOMM '88 Conf., ACM, pp. 303–313, 1988.

RAMASWAMI, R., KUMAR, S., and SASAKI, G.: Optical Networks: A Practical Perspective, 3rd ed., San Francisco: Morgan Kaufmann, 2009.

RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., and SHENKER, S.: "A Scalable Content-Addressable Network," Proc. SIGCOMM 2001 Conf, ACM, pp. 1161–172, 2001.

RIEBACK, M., CRISPO, B., and TANENBAUM, A.: "Is Your Cat Infected with a Computer Virus?," Proc. IEEE Percom, pp. 169–179, Mar. 2006.

RIVEST, R.L.: "The MD5 Message-Digest Algorithm," RFC 1320, Apr. 1992.

RIVEST, R.L., SHAMIR, A., and ADLEMAN, L.: "On a Method for Obtaining Digital Signatures and Public Key Cryptosystems," Commun. of the ACM, vol. 21, pp. 120–126, Feb. 1978.

ROBERTS, L.G.: "Extensions of Packet Communication Technology to a Hand Held Personal Terminal," Proc. Spring Joint Computer Conference, AFIPS, pp. 295–298, 1972.

ROBERTS, L.G.: "Multiple Computer Networks and Intercomputer Communication," Proc. First Symp. on Operating Systems Prin., ACM, pp. 3.1–3.6, 1967.

ROSE, M.T.: The Simple Book, Englewood Cliffs, NJ: Prentice Hall, 1994.

ROSE, M.T.: The Internet Message, Englewood Cliffs, NJ: Prentice Hall, 1993.

ROWSTRON, A., and DRUSCHEL, P.: "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Storage Utility," Proc. 18th Int'l Conf. on Distributed Systems Platforms, London: Springer-Verlag LNCS 2218, pp. 329–350, 2001.

RUIZ-SANCHEZ, M.A., BIERSACK, E.W., and DABBOUS, W.: "Survey and Taxonomy of IP Address Lookup Algorithms," IEEE Network Magazine, vol. 15, pp. 8–23, March-April 2001.

SALTZER, J.H., REED, D.P., and CLARK, D.D.: "End-to-End Arguments in System Design," ACM Trans. on Computer Systems, vol. 2, pp. 277–288, Nov. 1984.

SAMPLE, A., YEAGER, D., POWLEDGE, P., MAMISHEV, A., and SMITH, J.: "Design of an RFID-Based Battery-Free Programmable Sensing Platform," IEEE Trans. on Instrumentation and Measurement, vol. 57, pp. 2608–2615, Nov. 2008.

SAROIU, S., GUMMADI, K., and GRIBBLE, S.: "Measuring and Analyzing the Characteristics of Napster & Gnutella Hosts," Multim. Syst., vol. 9,, pp. 170–184, Aug. *2003.*

SALTZER, J.H., REED, D.P., and CLARK, D.D.: "End-to-End Arguments in System Design," ACM Trans, on Computer Systems, vol. 2, pp. 277-288, Nov. 1984.

SCHALLER, R.: "Moore's Law: Past, Present and Future," IEEE Spectrum, vol. 34, pp. 52–59, June 1997.

SCHNEIER, B.: Secrets and Lies, New York: John Wiley&Sons, 2004.

SCHNEIER, B.: E-Mail Security, New York: John Wiley&Sons, 1995.

SCHNORR, C.P.: "Efficient Signature Generation for Smart Cards," Journal of Cryptol-ogy, vol. 4, pp. 161–174, 1991.

SCHOLTZ, R.A.: "The Origins of Spread-Spectrum Communications," IEEE Trans, on Commun., vol. COM–0, pp. 822–854, May 1982.

SCHWARTZ, M., and ABRAMSON, N.: "The AlohaNet: Surfing for Wireless Data," IEEE Commun. Magazine, vol. 47, pp. 21–25, Dec. 2009.

SEIFERT, R., and EDWARDS, J.: The All-New Switch Book, NY: John Wiley, 2008.

SENN, J.A.: "The Emergence of M-Commerce," IEEE Computer, vol. 33, pp. 148–150, Dec. 2000.

SERJANTOV, A.: "Anonymizing Censorship Resistant Systems," Proc. First Int'l Workshop on Peer-to-Peer Systems, London: Springer-Verlag LNCS 2429, pp. 111–120, 2002.

SHACHAM, N., and MCKENNY, P.: "Packet Recovery in High-Speed Networks Using Coding and Buffer Management," Proc. INFOCOM Conf., IEEE, pp. 124–131, June 1990.

SHAIKH, A., REXFORD, J., and SHIN, K.: "Load-Sensitive Routing of Long-Lived IP Flows," Proc. SIGCOMM '99 Conf., ACM, pp. 215–226, Sept. 1999.

SHALUNOV, S., and CARLSON, R.: "Detecting Duplex Mismatch on Ethernet," Passive and Active Network Measurement, Berlin: Springer-Verlag LNCS 3431, pp. 3135–3148, 2005.

SHANNON, C.: "A Mathematical Theory of Communication," Bell System Tech. J., vol. 27, pp. 379–423, July 1948; and pp. 623–656, Oct. 1948.

SHEPARD, S.: SONET/SDH Demystified, New York: McGraw-Hill, 2001.

SHREEDHAR, M., and VARGHESE, G.: "Efficient Fair Queueing Using Deficit Round Robin," Proc. SIGCOMM '95 Conf., ACM, pp. 231–243, 1995.

SIMPSON, W.: Video Over IP, 2nd ed., Burlington, MA: Focal Press, 2008.

SIMPSON, W.: "PPP in HDLC-like Framing," RFC 1662, July 1994b.

SIMPSON, W.: "The Point-to-Point Protocol (PPP)," RFC 1661, July 1994a.

SIU, K., and JAIN, R.: "A Brief Overview of ATM: Protocol Layers, LAN Emulation, and Traffic," ACM Computer Communications Review, vol. 25, pp. 6–20, Apr. 1995.
SKOUDIS, E. and LISTON, T.: Counter Hack Reloaded, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2006.
SMITH, O.K., and ALEXANDER, R.C.: Fumbling the Future, New York: William Morrow, 1988.
SNOEREN, A.C., and BALAKRISHNAN, H.: "An End-to-End Approach to Host Mobility," Intel Conf. on Mobile Computing and Networking, ACM, pp. 155–166, 2000.
SOBEL, D.L.: "Will Carnivore Devour Online Privacy," Computer, vol. 34, pp. 87–88, May 2001.
SOTIROV, A., STEVENS, M., APPELBAUM, J., LENSTRA, A., MOLNAR, D., OSVIK, D., and DE WEGER, B.: "MD5 Considered Harmful Today," Proc. 25th Chaos Communication Congress, Verlag Art d'Ameublement, 2008.
SOUTHEY, R.: The Doctors, London: Longman, Brown, Green and Longmans, 1848.
SPURGEON, C.E.: Ethernet: The Definitive Guide, Sebastopol, CA: O'Reilly, 2000.
STALLINGS, W.: Data and Computer Communications, 9th ed., Upper Saddle River, NJ: Pearson Education, 2010.
STEVENS, W.R.: TCP/IP Illustrated, TheProtocols, Boston: Addison-Wesley, 1994.
STINSON, D.R.: Cryptography Theory and Practice, 2nd ed., Boca Raton, FL: CRC Press, 2002.
STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M.F., and BALAKRISHNAN, H.: "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," Proc. SIGCOMM 2001 Conf., ACM, pp. 149–160, 2001.
STUBBLEFIELD, A., IOANNIDIS, J., and RUBIN, A.D.: "Using the Fluhrer, Mantin, and Shamir Attack to Break WEP," Proc Network and Distributed Systems Security Symp., ISOC, pp. 1–11,2002.
STUTTARD, D., and PINTO, M.: The Web Application Hacker's Handbook, New York John Wiley & Sons, 2007.
SU, S.: The UMTS Air Interface in RF Engineering, New York: McGraw-Hill, 2007.
SULLIVAN, G., and WIEGAND, T.: "Tree Algorithms for Packet Broadcast Channels,' Proc. of the IEEE, vol. 93, pp. 18–31, Jan. 2005.
SUNSHINE, C.A., and DALAL, Y.K.: "Connection Management in Transport Protocols," Computer Networks, vol. 2, pp. 454–473, 1978.
TAN, K., SONG, J., ZHANG, Q., and SRIDHARN, M.: "A Compound TCP Approach for High-Speed and Long Distance Networks," Proc. INFOCOM Conf., IEEE, pp. 1–12, 2006.
ТАНЕНБАУМ, Э. Современные операционные системы. 3-е изд. — СПб.: Питер, 2010.
TANENBAUM, A.S., and VAN STEEN, M.: Distributed Systems: Principles and Paradigms, Upper Saddle River, NJ: Prentice Hall, 2007.
TOMLINSON, R.S.: "Selecting Sequence Numbers," Proc. SIGCOMM/SIGOPS Interprocess Commun. Workshop, ACM, pp. 11–23, 1975.
TUCHMAN, W.: "Hellman Presents No Shortcut Solutions to DES," IEEE Spectrum, vol. 16, pp. 40–41, July 1979.
TURNER, J.S.: "New Directions in Communications (or Which Way to the Information Age)," IEEE Commun. Magazine, vol. 24, pp 8–15, Oct. 1986.
UNGERBOECK, G.: "Trellis-Coded Modulation with Redundant Signal Sets Part I: Introduction," IEEE Commun. Magazine, vol. 25, pp. 5–11, Feb. 1987.
VALADE, J.,: PHP & MySQL for Dummies, 5th ed., New York: John Wiley & Sons, 2009.
VARGHESE, G.: Network Algorithmics, San Francisco: Morgan Kaufmann, 2004.
VARGHESE, G., and LAUCK, T.: "Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility," Proc. llth Symp. on Operating Systems Prin., ACM, pp. 25–38, 1987.
VERIZON BUSINESS: 2009 Data Breach Investigations Report, Verizon, 2009.
VITERBI, A.: CDMA: Principles of Spread Spectrum Communication, Englewood Cliffs, NJ: Prentice Hall, 1995.

VON AHN, L., BLUM, B., and LANGFORD, J.: "Telling Humans and Computers Apart Automatically," Commun. of the ACM, vol. 47, pp. 56–60, Feb. 2004.

WAITZMAN, D., PARTRIDGE, C., and DEERING, S.: "Distance Vector Multicast Routing Protocol," RFC 1075, Nov. 1988.

WALDMAN, M., RUBIN, A.D., and CRANOR, L.F.: "Publius: A Robust, Tamper-Evident, Censorship-Resistant, Web Publishing System," Proc. Ninth USEN1X Security Symp., USENIX, pp. 59–72, 2000.

WANG, Z., and CROWCROFT, J.: "SEAL Detects Cell Misordering," IEEE Network Magazine, vol. 6, pp. 8–9, July 1992.

WANT, R.: RFID Explained, San Rafael, CA: Morgan Claypool, 2006.

WARNEKE, B., LAST, M., LIEBOWITZ, B., and PISTER, K.S.J.: "Smart Dust: Communicating with a Cubic Millimeter Computer," Computer, vol. 34, pp. 44–51, Jan. 2001.

WAYNER, P.: Disappearing Cryptography: Information Hiding, Steganography, and Watermarking, 3nded., San Francisco: Morgan Kaufmann, 2008.

WEI, D., CHENG, J., LOW, S., and HEGDE, S.: "FAST TCP: Motivation, Architecture, Algorithms, Performance," IEEE/ACM Trans. on Networking, vol. 14, pp. 1246–1259, Dec. 2006.

WEISER, M.: "The Computer for the Twenty-First Century," Scientific American, vol. 265, pp. 94–104, Sept. 1991.

WELBOURNE, E., BATTLE, L., COLE, G., GOULD, K., RECTOR, K., RAYMER, S., BALAZINSKA, M., and BORRIELLO, G.: "Building the Internet of Things Using RFID," IEEE Internet Computing, vol. 13, pp. 48–55, May 2009.

WITTENBURG, N.: Understanding Voice Over IP Technology, Clifton Park, NY: Delmar Cengage Learning, 2009.

WOLMAN, A., VOELKER, G., SHARMA, N., CARDWELL, N., KARLIN, A., and LEVY, H.: "On the Scale and Performance of Cooperative Web Proxy Caching," Proc. 17th Symp. on Operating Systems Prin., ACM, pp. 16–31, 1999.

WOOD, L., IVANCIC, W., EDDY, W., STEWART, D., NORTHAM, J., JACKSON, C., and DA SILVA CURIEL, A.: "Use of the Delay-Tolerant Networking Bundle Protocol from Space," Proc. 59th Int'l Astronautical Congress, Int'l Astronautical Federation, pp. 3123–3133, 2008.

WU, T.: "Network Neutrality, Broadband Discrimination," Journal on Telecom. and High-Tech. Law, vol. 2, pp. 141–179, 2003.

WYLIE, J., BIGRIGG, M.W., STRUNK, J.D., GANGER, G.R., KILICCOTE, H., and KHOSLA, P.K.: "Survivable Information Storage Systems," Computer, vol. 33, pp. 61–68, Aug. 2000.

YU, T., HARTMAN, S., and RAEBURN, K.: "The Perils of Unauthenticated Encryption: Kerberos Version 4," Proc. NDSS Symposium, Internet Society, Feb. 2004.

YUVAL, G.: "How to Swindle Rabin," Cryptologia, vol. 3, pp. 187–190, July 1979.

ZACKS, M.: "Antiterrorist Legislation Expands Electronic Snooping," IEEE Internet Computing, vol. 5, pp. 8–9, Nov.-Dec. 2001.

ZHANG, Y., BRESLAU, L., PAXSON, V., and SHENKER, S.: "On the Characteristics and
Origins of Internet Flow Rates," Proc. SIGCOMM 2002 Conf., ACM, pp. 309–322, 2002.

ZHAO, B., LING, H., STRIBLING, J., RHEA, S., JOSEPH, A., and KUBIATOWICZ, J.: "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," IEEE J. on Selected Areas in Commun., vol. 22, pp. 41–53, Jan. 2004.

ZIMMERMANN, P.R.: The Official PGP User's Guide, Cambridge, MA: M.I.T. Press, 1995a.

ZIMMERMANN, P.R.: PGP: Source Code and Internals, Cambridge, MA: M.I.T. Press, 1995b.

ZIPF, G.K.: Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology, Boston: Addison-Wesley, 1949.

ZIV, J., and LEMPEL, Z.: "A Universal Algorithm for Sequential Data Compression," IEEE Trans, on Information Theory, vol. IT-3, pp. 337–343, May 1977.

Алфавитный указатель

В

Г

Д

З

И

К

Л

М

Н

О

П

P

C

T

У

Ф

Х

Ц

Ч

Ш

Э

Э. Таненбаум, Д. Уэзеролл

Компьютерные сети. 5-е изд.

Перевел с английского А. Гребеньков

Заведующий редакцией	*А. Кривцов*
Руководитель проекта	*А. Юрченко*
Ведущий редактор	*Ю. Сергиенко*
Научный редактор	*А. Гущин*
Художественный редактор	*К. Радзевич*
Корректор	*В. Листова*
Верстка	*Е. Егорова*

ООО «Мир книг», 198206, Санкт-Петербург, Петергофское шоссе, 73, лит. А29.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.
Подписано в печать 27.09.11. Формат 70x100/16. Усл. п. л. 77,400. Тираж 2000. Заказ 0000.
Отпечатано по технологии CtP в ОАО «Первая Образцовая типография», обособленное подразделение «Печатный двор».
197110, Санкт-Петербург, Чкаловский пр., 15.

ВАМ НРАВЯТСЯ НАШИ КНИГИ? ЗАРАБАТЫВАЙТЕ ВМЕСТЕ С НАМИ!

У Вас есть свой сайт?
Вы ведете блог?
Регулярно общаетесь на форумах? Интересуетесь литературой, любите рекомендовать хорошие книги и хотели бы стать нашим партнером?
ЭТО ВПОЛНЕ РЕАЛЬНО!

СТАНЬТЕ УЧАСТНИКОМ ПАРТНЕРСКОЙ ПРОГРАММЫ ИЗДАТЕЛЬСТВА «ПИТЕР»!

- *Зарегистрируйтесь на нашем сайте в качестве партнера по адресу* ***www.piter.com/ePartners***
- *Получите свой персональный уникальный номер партнера*
- *Выбирайте книги на сайте* ***www.piter.com,*** *размещайте информацию о них на своем сайте, в блоге или на форуме и добавляйте в текст ссылки на эти книги (на сайт* ***www.piter.com****)*

ВНИМАНИЕ! *В каждую ссылку необходимо добавить свой персональный уникальный номер партнера.*

С этого момента получайте 10% от стоимости каждой покупки, которую совершит клиент, придя в интернет-магазин «Питер» по ссылке с Вашим партнерским номером. А если покупатель приобрел не только эту книгу, но и другие издания, Вы получаете дополнительно по 5% от стоимости каждой книги.

Деньги с виртуального счета Вы можете потратить на покупку книг в интернет-магазине издательства «Питер», а также, если сумма будет больше 500 рублей, перевести их на кошелек в системе Яндекс.Деньги или Web.Money.

Пример партнерской ссылки:
http://www.piter.com/book.phtml?978538800282 – обычная ссылка
http://www.piter.com/book.phtml?97853880028**2&refer=0000** – партнерская ссылка, где 0000 – это ваш уникальный партнерский номер

Подробно о Партнерской программе ИД «Питер» читайте на сайте WWW.PITER.COM

КНИГА-ПОЧТОЙ

ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР» МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ:

- на нашем сайте: **www.piter.com**
- по электронной почте: **postbook@piter.com**
- по телефону: **(812) 703-73-74**
- по почте: **197198, Санкт-Петербург, а/я 127, ООО «Питер Мейл»**
- по ICQ: **413763617**

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ:

- Наложенным платежом с оплатой при получении в ближайшем почтовом отделении.
- С помощью банковской карты. Во время заказа Вы будете перенаправлены на защищенный сервер нашего оператора, где сможете ввести свои данные для оплаты.
- Электронными деньгами. Мы принимаем к оплате все виды электронных денег: от традиционных Яндекс.Деньги и Web-money до USD E-Gold, MoneyMail, INOCard, RBK Money (RuPay), USD Bets, Mobile Wallet и др.
- В любом банке, распечатав квитанцию, которая формируется автоматически после совершения Вами заказа.

Все посылки отправляются через «Почту России». Отработанная система позволяет нам организовывать доставку Ваших покупок максимально быстро. Дату отправления Вашей покупки и предполагаемую дату доставки Вам сообщат по e-mail.

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ:

- фамилию, имя, отчество, телефон, факс, e-mail;
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру;
- название книги, автора, количество заказываемых экземпляров.

ИЗДАТЕЛЬСКИЙ ДОМ
ПИТЕР®
WWW.PITER.COM

ПРЕДСТАВИТЕЛЬСТВА ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
предлагают эксклюзивный ассортимент компьютерной, медицинской, психологической, экономической и популярной литературы

РОССИЯ

Санкт-Петербург м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-73, 703-73-72; e-mail: sales@piter.com

Москва м. «Электрозаводская», Семеновская наб., д. 2/1, корп. 1, 6-й этаж
тел./факс: (495) 234-38-15, 974-34-50; e-mail: sales@msk.piter.com

Воронеж Ленинский пр., д. 169; тел./факс: (4732) 39-61-70
e-mail: piterctr@comch.ru

Екатеринбург ул. Бебеля, д. 11а; тел./факс: (343) 378-98-41, 378-98-42
e-mail: office@ekat.piter.com

Нижний Новгород ул. Совхозная, д. 13; тел.: (8312) 41-27-31
e-mail: office@nnov.piter.com

Новосибирск ул. Станционная, д. 36; тел.: (383) 363-01-14
факс: (383) 350-19-79; e-mail: sib@nsk.piter.com

Ростов-на-Дону ул. Ульяновская, д. 26; тел.: (863) 269-91-22, 269-91-30
e-mail: piter-ug@rostov.piter.com

Самара ул. Молодогвардейская, д. 33а; офис 223; тел.: (846) 277-89-79
e-mail: pitvolga@samtel.ru

УКРАИНА

Харьков ул. Суздальские ряды, д. 12, офис 10; тел.: (1038057) 751-10-02
758-41-45; факс: (1038057) 712-27-05; e-mail: piter@kharkov.piter.com

Киев Московский пр., д. 6, корп. 1, офис 33; тел.: (1038044) 490-35-69
факс: (1038044) 490-35-68; e-mail: office@kiev.piter.com

БЕЛАРУСЬ

Минск ул. Притыцкого, д. 34, офис 2; тел./факс: (1037517) 201-48-79, 201-48-81
e-mail: gv@minsk.piter.com

Ищем зарубежных партнеров или посредников, имеющих выход на зарубежный рынок.
Телефон для связи: **(812) 703-73-73**. **E-mail:** fuganov@piter.com

---

Издательский дом «Питер» приглашает к сотрудничеству авторов. Обращайтесь по телефонам: **Санкт-Петербург** – **(812) 703-73-72**, **Москва** – **(495) 974-34-50**

Заказ книг для вузов и библиотек по тел.: (812) 703-73-73.
Специальное предложение – e-mail: kozin@piter.com

Заказ книг по почте: на сайте **www.piter.com**; по тел.: (812) 703-73-74
по ICQ 413763617

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР®
WWW.PITER.COM

УВАЖАЕМЫЕ ГОСПОДА!
КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
ВЫ МОЖЕТЕ ПРИОБРЕСТИ
ОПТОМ И В РОЗНИЦУ У НАШИХ
РЕГИОНАЛЬНЫХ ПАРТНЕРОВ.

ДАЛЬНИЙ ВОСТОК

Владивосток
«Приморский торговый дом книги»
тел./факс: (4232) 23-82-12
e-mail: bookbase@mail.primorye.ru

Хабаровск, «Деловая книга», ул. Путевая, д. 1а
тел.: (4212) 36-06-65, 33-95-31
e-mail: dkniga@mail.kht.ru

Хабаровск, «Книжный мир»
тел.: (4212) 32-85-51, факс: (4212) 32-82-50
e-mail: postmaster@worldbooks.kht.ru

Хабаровск, «Мирс»
тел.: (4212) 39-49-60
e-mail: zakaz@booksmirs.ru

ЕВРОПЕЙСКИЕ РЕГИОНЫ РОССИИ

Архангельск, «Дом книги», пл. Ленина, д. 3
тел.: (8182) 65-41-34, 65-38-79
e-mail: marketing@avfkniga.ru

Воронеж, «Амиталь», пл. Ленина, д. 4
тел.: (4732) 26-77-77
http://www.amital.ru

Калининград, «Вестер»,
сеть магазинов «Книги и книжечки»
тел./факс: (4012) 21-56-28, 6 5-65-68
e-mail: nshibkova@vester.ru
http://www.vester.ru

Самара, «Чакона», ТЦ «Фрегат»
Московское шоссе, д.15
тел.: (846) 331-22-33
e-mail: chaconne@chaccone.ru

Саратов, «Читающий Саратов»
пр. Революции, д. 58
тел.: (4732) 51-28-93, 47-00-81
e-mail: manager@kmsvrn.ru

СЕВЕРНЫЙ КАВКАЗ

Ессентуки, «Россы», ул. Октябрьская, 424
тел./факс: (87934) 6-93-09
e-mail: rossy@kmw.ru

СИБИРЬ

Иркутск, «ПродаЛитЪ»
тел.: (3952) 20-09-17, 24-17-77
e-mail: prodalit@irk.ru
http://www.prodalit.irk.ru

Иркутск, «Светлана»
тел./факс: (3952) 25-25-90
e-mail: kkcbooks@bk.ru
http://www.kkcbooks.ru

Красноярск, «Книжный мир»
пр. Мира, д. 86
тел./факс: (3912) 27-39-71
e-mail: book-world@public.krasnet.ru

Новосибирск, «Топ-книга»
тел.: (383) 336-10-26
факс: (383) 336-10-27
e-mail: office@top-kniga.ru
http://www.top-kniga.ru

ТАТАРСТАН

Казань, «Таис»,
сеть магазинов «Дом книги»
тел.: (843) 272-34-55
e-mail: tais@bancorp.ru

УРАЛ

Екатеринбург, ООО «Дом книги»
ул. Антона Валека, д. 12
тел./факс: (343) 358-18-98, 358-14-84
e-mail: domknigi@k66.ru

Екатеринбург, ТЦ «Люмна»
ул. Студенческая, д. 1в
тел./факс: (343) 228-10-70
e-mail: igm@lumna.ru
http://www.lumna.ru

Челябинск, ООО «ИнтерСервис ЛТД»
ул. Артиллерийская, д. 124
тел.: (351) 247-74-03, 247-74-09,
247-74-16
e-mail: zakup@intser.ru
http://www.fkniga.ru, www.intser.ru