Чтобы увидеть, как быстро распространяются хорошие известия, рассмотрим линейную подсеть 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.

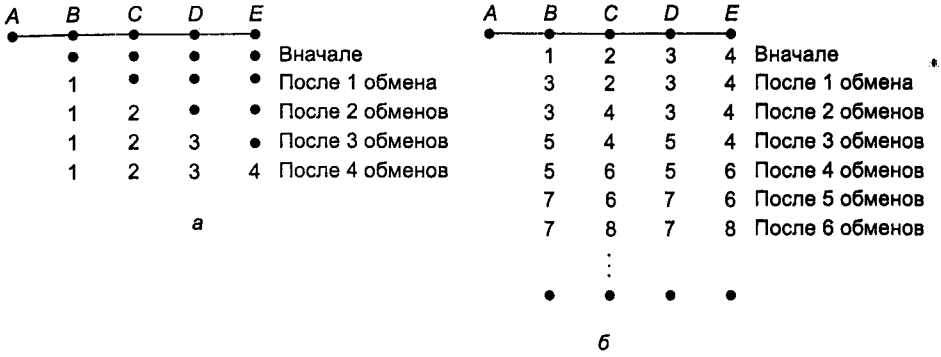


Рис. 5.8. Проблема счета до бесконечности

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 подсети 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 линии и маршрутизаторы изначально находятся во включенном состоянии. Маршрутизаторы *B*, *C*, *D* и *E* находятся от *A* на расстоянии 1, 2, 3 и 4 соответственно. Внезапно *A* отключается или, возможно, 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* не знает, что путь от *C* к *A* проходит через *B*. *B* может только предполагать, что у *C* около 10 выходных линий 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. Если расстояния в подсети измеряются во временных задержках, такую верхнюю границу выбрать сложнее. Следует выбирать достаточно большое ограничивающее значение, чтобы линия с большой задержкой не была сочтена совсем не работающей. Неудивительно, что эта проблема называется **счетом до бесконечности**. Было сделано несколько попыток решить ее (например, алгоритм расколотого горизонта, весьма бесполезный во многих случаях, но внесенный в RFC 1058), но все они оказались безуспешными. 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.

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 по двум причинам. Во-первых, старый алгоритм при выборе пути не учитывал пропускную способность линий. Пока все линии имели пропускную способность 56 Кбит/с, в учете пропускной способности не было необходимости. Однако стали появляться линии со скоростью 230 Кбит/с, а затем и 1,544 Мбит/с, и не принимать во внимание пропускную способность стало невозможно. Конечно, можно было ввести пропускную способность в качестве множителя для единицы измерения, но имелась еще и другая проблема, заключавшаяся в том, что алгоритм слишком долго приходил к устойчивому состоянию (проблема счета до бесконечности). Поэтому он был заменен полностью новым алгоритмом, который сейчас называется **маршрутизацией с учетом состояния линий**. Варианты этого алгоритма широко применяются в наши дни.

В основе алгоритма лежит простая идея, ее можно изложить в *пяти требованиях к маршрутизатору*. Каждый маршрутизатор должен:

1. Обнаруживать своих соседей и узнавать их сетевые адреса.
2. Измерять задержку или стоимость связи с каждым из своих соседей.
3. Создавать пакет, содержащий всю собранную информацию.
4. Посылать этот пакет всем маршрутизаторам.
5. Вычислять кратчайший путь ко всем маршрутизаторам.

В результате каждому маршрутизатору высылаются полная топология и все измеренные значения задержек. После этого для обнаружения кратчайшего пути к каждому маршрутизатору может применяться алгоритм Дейкстры. Далее 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 двухточечным линиям. При этом маршрутизатор на другом конце линии должен послать ответ, сообщая информацию о себе. Имена маршрутизаторов должны быть совершенно уникальными, поскольку, если удаленный маршрутизатор слышит, что три маршрутизатора являются соседями маршрутизатора *F*, не должно возникать разночтений по поводу того, один и тот же маршрутизатор *F* имеется в виду или нет.

Когда два или более маршрутизаторов объединены в локальную сеть, ситуация несколько усложняется. На рис. 5.9, *а* изображена 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.

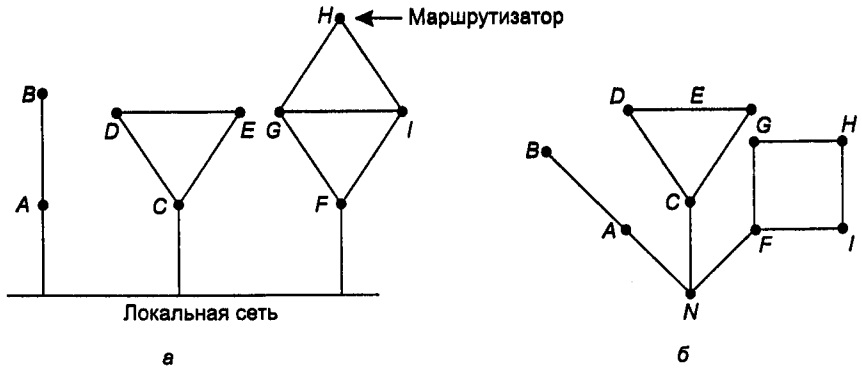


Рис. 5.9. Девять маршрутизаторов и локальная сеть (*а*); графовая модель той же системы (*б*)

Один из способов 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*. Возможность передачи пакетов от *A* к *C* по локальной сети отражается здесь наличием пути *ANC*.

Измерение стоимости линии

Алгоритм маршрутизации с учетом состояния линии требует от каждого маршрутизатора знания или хотя бы обоснованной оценки задержки для всех линий связи со своими соседями. 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. Чтобы получить более точный результат, это действие можно повторить несколько раз, после чего вычислить среднее арифметическое. Конечно, такой метод предполагает, что задержки являются симметричными, что не всегда так.

Возникает интересный вопрос: надо ли учитывать нагрузку на линию во время измерения задержки? Чтобы учесть загруженность линии, таймер должен включаться при отправке пакета ECHO. Чтобы игнорировать загрузку, таймер следует включать, когда пакет ECHO достигает начала очереди.

Оба способа могут быть аргументированы. Учет трафика в линии при измерении задержки означает, что когда у маршрутизатора есть выбор между двумя линиями с одинаковой пропускной способностью, маршрут по менее загруженной линии рассматривается как более короткий. Такой выбор приведет к более сбалансированному использованию линий связи и, следовательно, к более эффективной работе системы.

К сожалению, можно привести аргумент и против учета загруженности линии при расчете задержек. Рассмотрим подсеть, показанную на рис. 5.10. Она разделена на две части — восточную и западную, которые соединены двумя линиями, *CF* и *EI*.

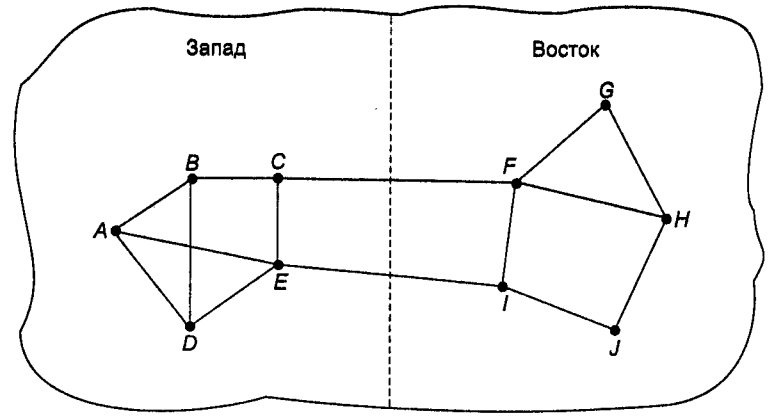


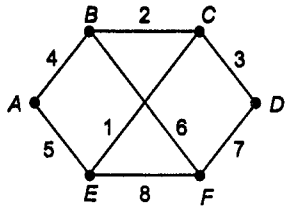
Рис. 5.10. Подсеть, в которой восточная и западная части соединены двумя линиями

Предположим, что основная часть потока данных между востоком и западом использует линию *CF*. В результате эта линия оказывается сильно загруженной и с большими задержками. Учет времени стояния пакета в очередях при подсчете кратчайшего пути сделает линию *EI* более привлекательной. После установки новых таблиц маршрутизации большая часть потока данных между востоком и западом переместится на линию *EI*, и ситуация повторится с точностью до смены одной линии на другую. Аналогично, после еще одного обновления уже линия *CF* окажется более привлекательной. В результате таблицы маршрутизации будут страдать от незатухающих колебаний, что сильно снизит эффективность

работы системы. Если же нагрузку не учитывать, то эта проблема не возникнет. Можно поступать по-другому: распределять нагрузку между двумя линиями. Однако такое решение приведет к неполному использованию наилучшего пути. Тем не менее, во избежание колебаний системы при выборе оптимального пути, по-видимому, лучше всего распределять нагрузку между несколькими линиями, пуская определенные части трафика по каждой из них.

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 далее), а также список соседей. Для каждого соседа указывается соответствующая ему задержка. Пример подсети приведен на рис. 5.11, *а*, на котором показаны задержки для каждой линии. Соответствующие пакеты состояния линий для всех шести маршрутизаторов показаны на рис. 5.11, *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. 5.11. Подсеть (*а*); пакеты состояния линий для этой подсети (*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. По мере распространения и установки пакетов маршрутизаторы, получившие первые пакеты, начинают изменять свои маршруты. Соответственно разные маршрутизаторы будут пользоваться разными версиями топологии, что может привести к противоречиям, появлению в маршрутах петель, недоступных машин, а также к другим проблемам.

Сначала мы опишем основной алгоритм распространения. Затем расскажем о некоторых улучшениях. Основная идея алгоритма распространения пакетов состояния линии 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, обнулится, возникнет путаница. Решение состоит в использовании 32-разрядных 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, его пакеты будут игнорироваться как устаревшие.

В-третьих, может произойти искажение порядкового номера — например, вместо номера 4 будет принято число 65 540 (ошибка в 1-м 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 сохраняет-

ся в течение некоторого периода времени в области промежуточного хранения. Если за это время от того же отправителя успевает прийти еще один пакет, маршрутизатор сравнивает их порядковые номера. Более старый пакет удаляется. Если номера одинаковые, то удаляется дубликат. Для защиты от ошибок на линиях связи между маршрутизаторами получение всех пакетов состояния линий подтверждается. Когда линия освобождается, маршрутизатор сканирует область промежуточного хранения, из которой выбираются для передачи пакеты или подтверждения.

Структура данных, используемая маршрутизатором *B* для работы с подсетью, изображенной на рис. 5.11, *a*, показана на рис. 5.12. Каждый ряд 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 линии. 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. 5.12. Буфер пакетов маршрутизатора *B* с рисунка 5.11

Как видно из рис. 5.12, 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 был получен дважды, по линиям *EAB* и *EFB*. Следовательно, его нужно отослать только *C*, но подтверждения 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 подсети, так как он располагает данными обо всех линиях. На самом деле, каждая линия представлена даже дважды, по одному значению для каждого направления. Эти два значения могут усредняться или использоваться по отдельности.

Теперь для построения кратчайшего пути ко всем возможным адресатам может быть локально применен алгоритм Дейкстры. Результат вычислений может быть установлен в таблицах маршрутов, после чего можно возобновить нормальную работу маршрутизатора.

В подсети, состоящей из n маршрутизаторов, у каждого из которых k соседей, количество памяти, необходимой для хранения входной информации, пропорционально kn. Кроме того, может потребоваться много времени на обработку информации. В больших подсетях это может составлять проблему. Тем не менее, во многих практических ситуациях маршрутизация с учетом состояния линий работает вполне удовлетворительно.

Однако неисправности оборудования или программного обеспечения могут привести к очень серьезным проблемам при использовании данного алгоритма (а также других алгоритмов). Например, если маршрутизатор заявит о существовании линии, которой у него в действительности нет, или наоборот, забудет о существовании имеющейся у него линии, граф подсети 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 неприятности. При увеличении размера подсети 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).

Маршрутизация с учетом состояния линий широко применяется в современных сетях, поэтому следует сказать несколько слов о некоторых примерах протоколов, использующих данный алгоритм. Одним из таких протоколов является протокол OSPF, все чаще применяемый в Интернете, о котором будет рассказано в разделе «Протокол внутреннего шлюза OSPF».

Другим важным протоколом с учетом состояния линий является **IS-IS** (Intermediate System to Intermediate System — связь между промежуточными системами) — протокол, разработанный для сети DECnet и принятый впоследствии Международной организацией по стандартизации ISO для использования вместе с протоколом сетевого уровня CLNP, не требующим соединений. С тех пор он был модифицирован для поддержки также и других протоколов, в частности IP. Протокол IS-IS используется в некоторых магистралях сети Интернет (включая старую магистраль NSFNET) и в некоторых цифровых сотовых системах, например, в CDPD. В сети Novell NetWare применяется разновидность протокола IS-IS (NLSP) для маршрутизации IPX-пакетов.

В основе работы протокола IS-IS лежит распространение картины топологии маршрутизаторов, по которой рассчитываются кратчайшие пути. Каждый маршрутизатор сообщает в информации о состоянии линий доступные ему напрямую адреса сетевого уровня. Эти адреса могут быть адресами IP, IPX, AppleTalk или другими. Протокол IS-IS может даже осуществлять одновременную поддержку нескольких протоколов сетевого уровня.

Многие новшества, разработанные для протокола IS-IS, были приняты несколько лет спустя при разработке протокола OSPF. К ним относятся метод саморегуляции лавинного потока обновлений информации о состоянии линий, концепция выделенного 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 способ кодирования в протоколе IS-IS, в отличие от OSPF, облегчает одновременную поддержку нескольких сетевых протоколов. Это свойство особенно важно в больших многопротокольных средах.

Иерархическая маршрутизация

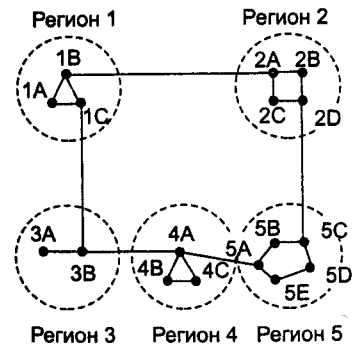
Размер таблиц маршрутов, поддерживаемых маршрутизаторами, увеличивается пропорционально увеличению размеров сети. При этом требуется не только большее количество памяти для хранения этой таблицы, но и большее время центрального процессора для ее обработки. Кроме того, возрастает размер служебных пакетов, 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 записи 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**. 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 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. 5.13 приведен количественный пример маршрутизации в двухуровневой иерархии с пятью регионами. Полная таблица маршрутизатора *1A*, как показано на рис. 5.13, *б*, состоит из 17 записей. При использовании иерархической маршрутизации, как показано на рис. 5.13, *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	3
4	1C	3
5	1C	4

в

Рис. 5.13. Иерархическая маршрутизация

К сожалению, этот выигрыш памяти не достается бесплатно. Платой за уменьшение размеров таблицы маршрутов 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 подсеть с 720 маршрутизаторами. Если иерархии нет, то каждому маршрутизатору необхо-

димо поддерживать таблицу из 720 строк. Если подсеть 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 подсети, 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.

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 распространять соответствующие данные широковещательным способом, предоставляя возможность всем заинтересованным хостам получить их. Итак, **широковещанием** называется рассылка пакетов по всем пунктам назначения. Для ее реализации применяются разнообразные методы.

Один из методов широковещательной маршрутизации не требует никаких особых способностей от подсети и используется просто для того, чтобы рассылать отдельные пакеты по всем направлениям. Он не только отнимает у подсети пропускную способность, но и требует, чтобы у источника пакета был полный список всех хостов. На практике это может быть единственная возможность, но такой метод является наименее желательным.

Еще одним очевидным кандидатом является метод *заливки*. Хотя он плохо подходит для обычных двухточечных соединений, для широковещания это может быть серьезный претендент, особенно если нет возможности применить один из методов, описываемых ниже. Проблема с применением заливки в качестве метода широковещания такая же, как с двухточечным алгоритмом маршрутизации: при заливке генерируется очень много пакетов и отнимается весьма существенная часть пропускной способности.

Третий алгоритм называется **многоадресной маршрутизацией**. При использовании этого метода в каждом пакете содержится либо список адресатов, либо битовая карта, показывающая предпочитаемые хосты назначения. Когда такой пакет прибывает на 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 и будет выглядеть как обычный пакет. Многоадресная маршрутизация подобна индивидуально адресуемым пакетам 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.

Еще один, четвертый алгоритм широковещательной маршрутизации в явном виде использует корневое дерево или любое другое связующее дерево. **Связующее дерево** представляет собой подмножество подсети, 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. 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), но иногда — нет (при маршрутизации по векторам расстояний).

Последний алгоритм широковещания, который мы рассмотрим, представляет собой попытку приблизиться к поведению предыдущего алгоритма, даже когда маршрутизаторы ничего не знают о связующих деревьях. Лежащая в основе данного алгоритма идея, называющаяся **продвижением по встречному пути**, замечательно проста. Когда прибывает широковещательный пакет, маршрутизатор проверяет, используется ли та линия, по которой он прибыл, для нормальной передачи пакетов *источнику широковещания*. В случае положительного ответа 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 линиям, кроме той, по которой он прибыл. Однако если пакет прибывает от того же источника по другой линии, он отвергается как вероятный дубликат.

Пример работы алгоритма продвижения по встречному пути показан на рис. 5.14. Слева изображена подсеть, посередине —входное дерево для маршрутизатора I этой подсети. На первом транзитном участке маршрутизатор I посылает пакеты маршрутизаторам F, H, J и N, являющимся вторым ярусом дерева. Все эти пакеты прибывают к I по предпочитаемым линиям (по пути, совпадающему с входным деревом), что обозначается кружками вокруг символов на рис. 5.14, *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 количест-

вом переданных пакетов, равным 23, тогда как при использовании входного дерева потребовалось бы 4 транзитных участка и 14 пакетов.

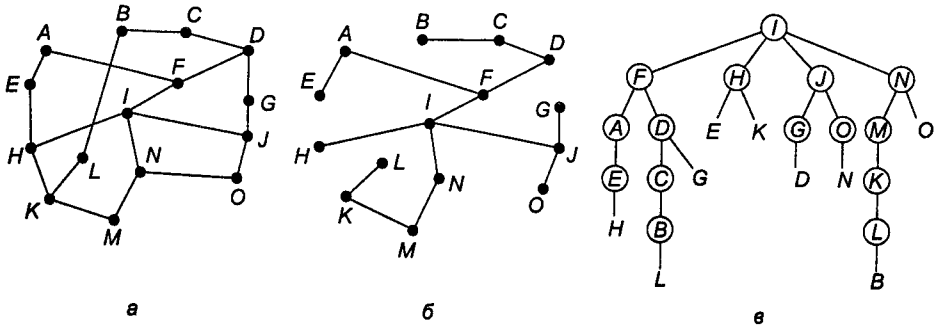


Рис. 5.14. Продвижение по встречному пути: подсеть (*а*); связующее дерево (*б*); дерево, построенное методом продвижения по встречному пути (*в*)

Принципиальное преимущество метода продвижения по встречному пути заключается в его вполне приемлемой эффективности при простоте реализации. Для использования этого метода маршрутизаторам не нужна никакая дополнительная информация о связующих деревьях. Не требуются и дополнительные расходы на список получателей или бит-карту в каждом распространяемом пакете, как в случае многоадресной рассылки. Также не требуется никакого специального механизма для прекращения процесса, как, например, в методе заливки (либо счетчик транзитных участков в каждом пакете и априорные сведения о диаметре сети, либо список уже встречавшихся пакетов от каждого источника).

Многоадресная рассылка

В некоторых приложениях сильно разделенные процессы работают совместными группами. Например, в виде группы процессов может быть реализована распределенная база данных. Часто одному процессу бывает необходимо послать сообщение всем остальным членам группы. Если группа невелика, то можно просто послать каждому члену группы отдельное сообщение. Если же группа достаточно большая, такая стратегия окажется весьма дорогостоящей. Иногда может быть использовано широковещание, но применять его для информирования 1000 машин в сети, состоящей из миллиона узлов, неэффективно, поскольку большинство получателей будут не заинтересованы в данном сообщении (или, что еще хуже, явно заинтересованы, но было бы крайне желательно от них эту информацию скрыть). 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**, а алгоритм маршрутизации этой операции — **многоадресной маршрутизацией**. В этом разделе будет описан один из способов реализации многоадресной маршрутизации. Дополнительные сведения см. в (Chu и др., 2000; Costa и др., 2001; Kasera и др., 2000; Madruga and Garcia-Luna-Aceves, 2001; Zhang and Ryu, 2001).

Многоадресной рассылке требуется управление группами, то есть способ создания и удаления групп, присоединения процесса к группе и ухода процесса из группы. Реализация данных задач, однако, не интересует алгоритм маршрутизации. Зато он заинтересован в том, чтобы процесс информировал свой хост о присоединении к какой-нибудь группе. Важно, чтобы маршрутизаторы знали, какой хост к какой группе принадлежит. Для этого либо хост должен сообщать своим маршрутизаторам об изменении в составе групп, либо маршрутизаторы должны сами периодически опрашивать свои хосты. В любом случае маршрутизаторы узнают, какие из их хостов к каким группам принадлежат. Маршрутизаторы сообщают об этом своим соседям, и таким образом эта информация распространяется по всей подсети.

Для многоадресной рассылки каждый маршрутизатор рассчитывает связующее дерево, покрывающее все остальные маршрутизаторы подсети. Например, на рис. 5.15, *а* мы видим подсеть с двумя группами, 1 и 2. Как показано на рисунке, маршрутизаторы соединены с хостами, принадлежащими к одной или обеим группам. Связующее дерево для самого левого маршрутизатора показано на рис. 5.15, *б*.

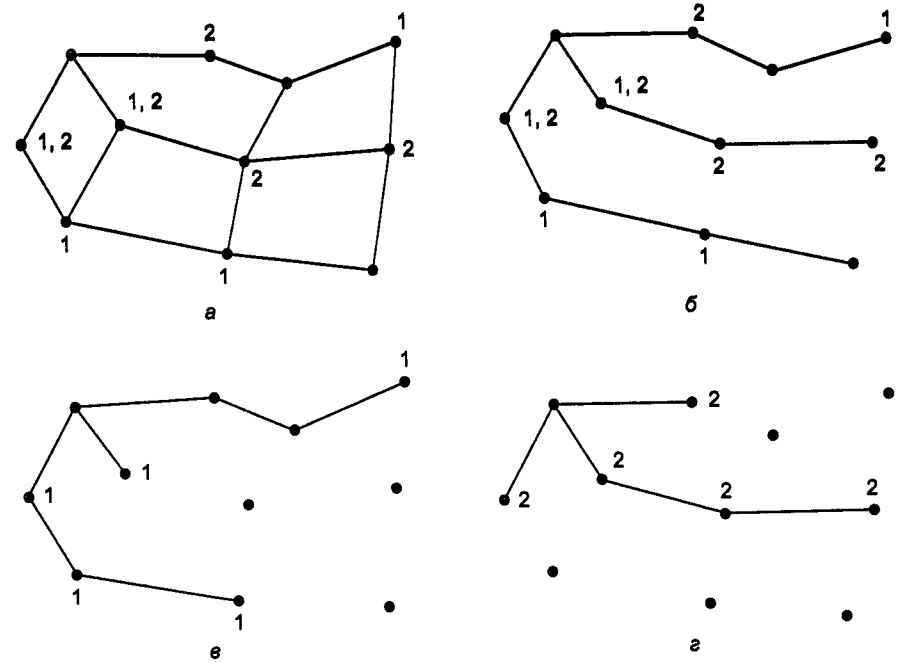


Рис. 5.15. Подсеть (*а*); связующее дерево для самого левого маршрутизатора (*б*); многоадресное дерево для группы 1 (*в*); многоадресное дерево для группы 2 (*г*)

Когда процесс посылает группе многоадресный пакет, первый маршрутизатор изучает свое связующее дерево и отсекает у него линии, не ведущие к хостам, являющимся членами группы. В нашем примере на рис. 5.15, *в* изображено усеченное связующее дерево для группы 1. Аналогично, на рис. 5.15, *г* показано усеченное связующее дерево для группы 2. Многоадресные пакеты рассылаются только вдоль соответствующего их группе усеченного связующего дерева.

Существует несколько способов усечения связующего дерева. Простейший способ может применяться при использовании маршрутизации с учетом состояния линий, когда каждому маршрутизатору известна полная топология подсети, в том числе и состав групп. При этом из связующего дерева могут быть удалены маршрутизаторы, не принадлежащие к данной группе, начиная с конца каждого пути вплоть до корня дерева.

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, он может ответить сообщением PRUNE (отсечь), информируя отправителя, 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 многоадресное сообщение по всем своим линиям. В результате подсеть постепенно рекурсивно усекается.

Недостаток данного алгоритма заключается в его плохой применимости к большим сетям. Предположим, что в сети есть n групп, каждая из которых в среднем состоит из m членов. Для каждой группы должно храниться m усеченных входных деревьев, то есть mn деревьев для всей сети. При большом количестве групп для хранения всех деревьев потребуется много памяти.

Альтернативный метод использует **деревья с основанием в сердцевине** (Ballardie и др., 1993). В этом методе для каждой группы рассчитывается единое связующее дерево с корнем (ядром) около середины группы. Хост посылает многоадресное сообщение ядру группы, откуда оно уже рассылается по всему связующему дереву группы. Хотя это дерево не является оптимальным для всех источников, единое дерево для группы снижает затраты на хранение информации о нем в m раз.

Алгоритмы маршрутизации для мобильных хостов

Сегодня миллионы людей обладают переносными компьютерами, и большинство из них желает читать свою электронную почту и получать доступ к нормальным файловым системам, находясь при этом в любой точке земного шара. Мобильные хосты привносят новое усложнение в и без того непростое дело выбора маршрутов в различных вычислительных сетях — чтобы направить пакет к мобильному хосту, его нужно сначала найти. Вопрос включения мобильных хостов в сети появился не так давно, но в данном разделе мы рассмотрим некоторые проблемы и приведем их возможные решения.

Модель мира, обычно используемая разработчиками сетей, показана на рис. 5.16. Здесь мы видим глобальную сеть, состоящую из маршрутизаторов и хостов. С глобальной сетью соединены локальные и региональные сети и беспроводные соты, которые рассматривались в главе 2.

Хосты, которые никогда не перемещаются, называются стационарными. Они соединены с сетью медными поводами или оптическими кабелями. Мы же будем различать две другие категории хостов. Мигрирующие хосты являются, в основном, стационарными пользователями, но время от времени перемещаются с одного фиксированного места на другое и пользуются сетью только тогда, когда физически соединены с нею. Блуждающие хосты используют переносные компьютеры, и им требуется связь с сетью прямо во время перемещения в пространстве. Для обозначения этих двух категорий, то есть хостов, которые не имеют постоянного местоположения и тем не менее желают быть на связи, мы будем использовать термин **мобильные хосты**.

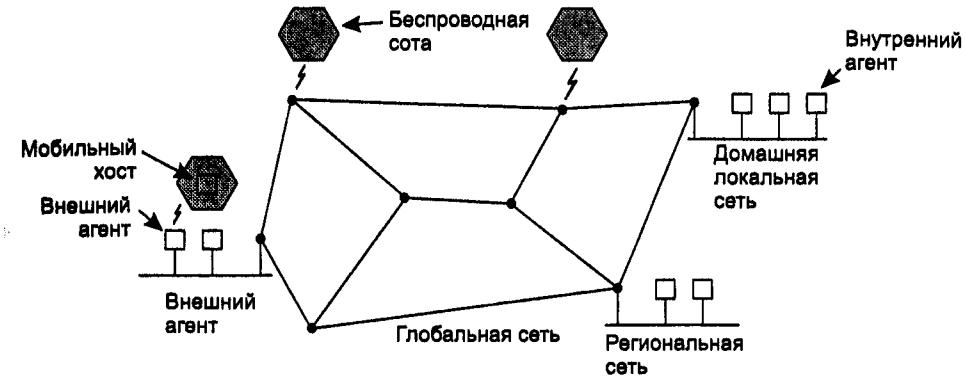


Рис. 5.16. Глобальная сеть, с которой соединены локальные и региональные сети, а также беспроводные соты

Предполагается, что у всех хостов есть постоянное домашнее **местоположение**, которое никогда не меняется. Кроме того, у хостов 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 Манхэттен (212). Целью маршрутизации в системах с мобильными хостами является обеспечение возможности передачи пакетов мобильным пользователям с помощью их домашних адресов. При этом пакеты должны эффективно достигать пользователей, независимо от их расположения. Самое сложное здесь, конечно, — найти пользователя.

В модели, показанной на рис. 5.16, мир разделен на небольшие единицы. Мы будем называть их областями, что обычно будет означать локальную сеть или беспроводную соту. Каждая область может содержать одного или более **внешних агентов**, следящих за всеми мобильными пользователями, посещающими область. Кроме того, в каждой области имеется **внутренний агент**, следящий за временно покинувшими свою область пользователями.

Когда в области появляется новый пользователь — либо подключившийся к ней (соединив свой компьютер с сетью), либо просто переместившийся в соту, — его компьютер должен зарегистрироваться в данной области, связавшись с местным внешним агентом. Процедура регистрации обычно выглядит следующим образом:

1. Периодически каждый внешний агент рассылает пакет, объявляя таким образом о своем существовании и местонахождении. Вновь прибывший мобильный хост может ждать подобного сообщения, но может и сам, не дождавшись его, передать пакет с запросом о наличии внешнего агента в данной области.

2. Мобильный хост регистрируется в данной области, сообщая внешнему агенту свой домашний адрес, текущий адрес уровня передачи данных, а также информацию, подтверждающую его подлинность.
3. Внешний агент связывается с внутренним агентом мобильного пользователя и сообщает ему: «Один из ваших хостов находится в нашей области». Это сообщение содержит адрес сети внешнего агента, а также информацию, подтверждающую подлинность мобильного хоста. Это позволяет убедить внутреннего агента в том, что мобильный хост действительно находится здесь.
4. Внутренний агент проверяет переданный ему идентификатор безопасности мобильного хоста, содержащий временной штамп, доказывающий, что идентификатор был создан буквально несколько секунд назад. Если проверка подлинности хоста проходит успешно, внутренний агент разрешает внешнему агенту продолжать связь.
5. Получив подтверждение от внутреннего агента, внешний агент заносит сведения о мобильном хосте в свою таблицу и сообщает ему, что он зарегистрирован.

В идеальном случае, покидая область, пользователь также должен сообщить об этом внешнему агенту, однако на практике многие пользователи, закончив свои дела, просто выключают свои компьютеры.

Когда пакет посылается мобильному пользователю, он направляется в его домашнюю локальную сеть на домашний адрес пользователя. На рис. 5.17 это действие показано как этап 1. Здесь отправитель из северо-западного города Сиэтла хочет отправить пакет хосту, который обычно находится по другую сторону США, в Нью-Йорке. Пакеты, посланные в домашнюю локальную сеть мобильного хоста (Нью-Йорк), перехватываются внутренним агентом, который узнает новое (временное) расположение мобильной станции (например, Лос-Анджелес) и адрес внешнего агента локальной сети, в которой она в данный момент находится.

Затем внутренний агент выполняет два действия. Во-первых, он помещает пакет, предназначающийся мобильному пользователю, в поле данных внешнего пакета, который посылается внешнему агенту (этап 2 на рис. 5.17). Такой прием называется туннелированием. Позднее мы обсудим ее подробнее. Получив пакет, внешний агент извлекает из поля данных оригинальный пакет, который пересылает мобильному пользователю в виде кадра уровня передачи данных.

Затем внутренний агент сообщает отправителю, что в дальнейшем следует не посылать пакеты мобильному хосту на домашний адрес, а вкладывать их в поле данных пакетов, явно адресованных внешнему агенту (этап 3 на рис. 5.17). Последующие пакеты теперь могут направляться напрямую пользователю через внешнего агента (этап 4), полностью минуя домашний адрес мобильного пользователя.

Различные предложенные схемы маршрутизации отличаются в нескольких аспектах. Во-первых, они отличаются в том, какая часть протокола выполняется маршрутизаторами, а какая — хостами, а также каким уровнем протоколов хостов. Во-вторых, в некоторых схемах маршрутизаторы записывают преобразованные адреса, поэтому они могут перехватывать и переадресовывать пакеты даже еще до того, как они успевают дойти до домашнего адреса мобильного пользователя. В-третьих, в одних схемах каждому посетителю дается уникальный временный адрес, а в других схемах временный адрес ссылается на агента, обрабатывающего трафик для всех посетителей.

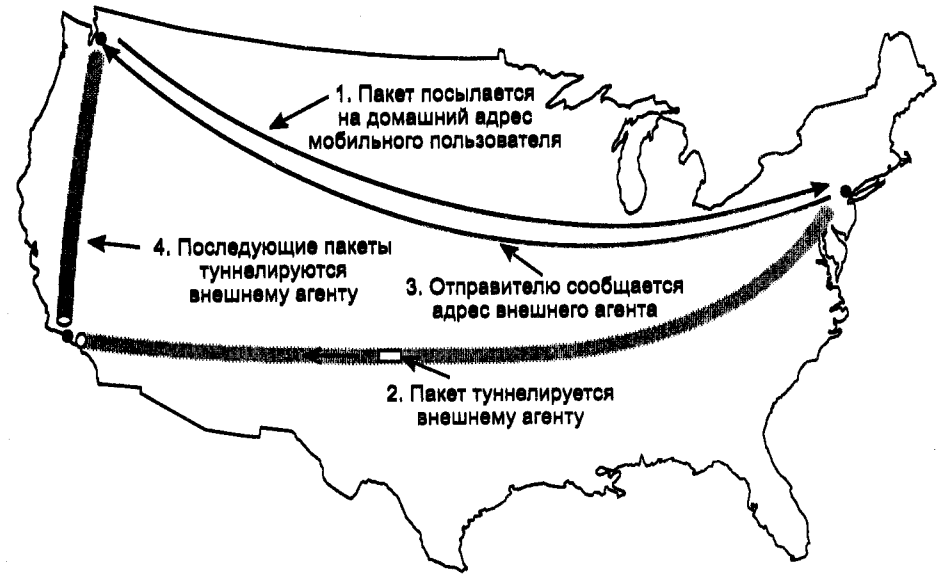


Рис. 5.17. Маршрутизация пакетов мобильным хостам

В-четвертых, схемы различаются способами переадресации пакетов. Один из способов заключается в изменении поля адреса получателя в пакете и передаче измененного пакета. Есть системы, в которых весь пакет, включая домашний адрес, может быть помещен внутрь другого пакета, посылаемого по временному адресу. В любом случае, 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. Несколько протоколов мобильных хостов обсуждаются и сравниваются в (Hac and Guo, 2000; Perkins, 1998a; Snoeren and Balakrishnah, 2000; Solomon, 1998; Wang and Chen, 2001).

Маршрутизация в специализированных 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 возможно, например, в следующих случаях.

- Военная техника на поле боя при отсутствии инфраструктуры.
- Морская флотилия, находящаяся в плавании.
- Работники служб спасения в районах с разрушенной инфраструктурой.
- Собрание людей с портативными компьютерами при отсутствии в помещении сети 802.11.

Во всех подобных случаях каждый узел состоит из маршрутизатора и хоста одновременно, обычно они даже совмещены в пределах одного компьютера. Сети, состоящие из узлов, волею судеб оказавшихся недалеко друг от друга, называются **специализированными сетями**, или **мобильными специализированными сетями** (MANET, Mobile Ad hoc networks). Давайте их вкратце рассмотрим. Более подробную информацию можно найти в книге (Perkins, 2001).

Основное отличие специализированных сетей от обычных проводных сетей состоит в том, что все обычные законы, касающиеся фиксированной топологии, известных соседей, взаимосвязи между IP-адресом и расположением в специализированных сетях, перестают работать. Маршрутизаторы 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 системы. В специализированных 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 сильно отличаться от маршрутизации в стационарных сетях.

Известно множество алгоритмов выбора маршрута для специализированных сетей. Один из наиболее интересных — это алгоритм **AODV** (Ad hoc On-demand Distance Vector — маршрутизация по требованию в специализированных сетях на основе вектора расстояний). Об этом можно прочитать у (Perkins and Royer, 1999). AODV является дальним родственником алгоритма Беллмана—Форда (Bellman—Ford) (метод векторов расстояний), адаптированным для работы в мобильной среде и принимающим в расчет ограниченность пропускной способности и срока службы элементов питания — свойства, характерные для мобильных сетей. Еще одной необычной характеристикой является то, что AODV — это алгоритм «по требованию», то есть он вычисляет маршрут только в тот момент, когда появляется желающий отправить пакет тому или иному адресату. Посмотрим, что это значит.

Построение маршрута

Специализированная сеть в любой момент времени может быть описана с помощью графа узлов (маршрутизаторов и хостов). Два узла считаются соединенными (то есть между ними проведена дуга), если они могут связываться напрямую посредством радио. Поскольку у одного из них может быть более мощный передатчик, чем у другого, то возможна ситуация, когда узел *A* соединен с *B*, но *B* не соединен с *A*. Однако для простоты мы будем считать, что все соединения симметричны. Следует заметить, что нахождение одного из узлов в зоне действия другого еще не означает наличия связи между ними. Их могут разделять холмы, здания и другие местные предметы, блокирующие соединение.

Для описания алгоритма воспользуемся рис. 5.18, на котором изображен процесс, запущенный на узле *A*, которому необходимо отправить пакет на узел *I*. Алгоритм AODV на каждом узле ведет таблицу, 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. Допустим, *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».

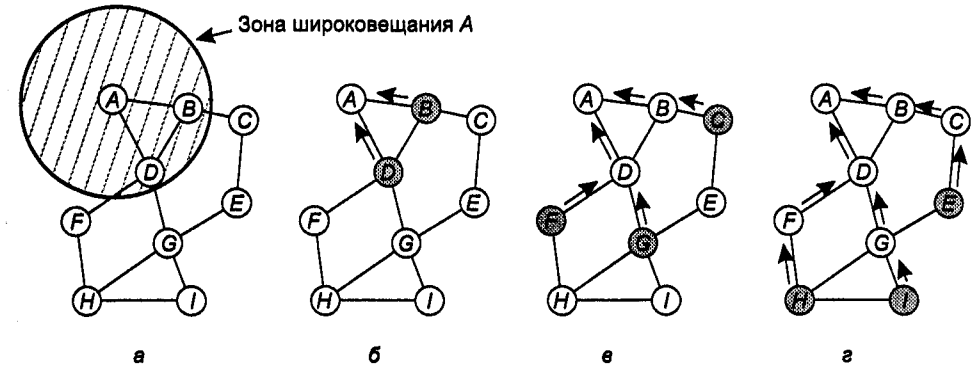


Рис. 5.18. Зона широковещания *A* (*а*); состояние после получения узлами *B* и *D* широковещательного пакета от *A* (*б*); состояние после получения узлами *C*, *F* и *G* широковещательного пакета от *A* (*в*); состояние после получения узлами *E*, *H* и *I* широковещательного пакета от *A* (*г*). Затененными кружочками обозначены новые получатели. Стрелками показаны возможные обратные маршруты

Для поиска *I* узел *A* генерирует специальный пакет запроса маршрута ROUTE REQUEST и распространяет его по сети широковещательным способом. На рис. 5.18, *а* показано, что этот пакет достигает узлов *B* и *D*. На самом деле, причиной установления именно узлами *B* и *D* соединения с *A* является то, что они могут получать пакеты от *A*. Например, *F* не соединен дугой с *A*, потому что он не может принимать радиосигнал от этого узла. То есть *F* не соединен с *A*.

Формат пакета запроса маршрута показан на рис. 5.19. В нем, как видно из этого рисунка, содержатся адреса источника и приемника (обычно IP-адреса), с помощью которых можно понять, кто кого ищет. Также содержится поле *Идентификатор запроса*, которое представляет собой локальный счетчик, обновляемый каждым узлом независимо и инкрементирующийся всякий раз, когда распространяется пакет запроса маршрута. Поля *Адрес источника* и *Идентификатор запроса* вместе единственным образом идентифицируют пакет ROUTE REQUEST, что позволяет узлам обнаруживать и отвергать любые дубликаты.

Адрес отправителя	Идентификатор запроса	Адрес получателя	Порядковый номер отправителя	Порядковый номер получателя	Счетчик переходов

Рис. 5.19. Формат пакета ROUTE REQUEST

В дополнение к счетчику *Идентификатор запроса* каждый узел имеет второй счетчик, который инкрементируется всякий раз при отправке пакета для запроса маршрута или ответе на такой пакет. Его работа напоминает часы, и используется

он для того, чтобы можно было отличить новые маршруты от старых. Четвертое поле, показанное на рис. 5.19, это счетчик узла *A*; пятое — последнее значение порядкового номера пакета, полученного от *I* (оно равно 0, если такого пакета не было). Вскоре мы более подробно раскроем назначение этих полей Наконец, последнее поле — *Счетчик переходов* — запоминает количество пересылок, совершенных пакетом. В начале работы алгоритма оно равно нулю.

Когда пакет запроса маршрута прибывает на узел (например, на узлы *B* и *D*), с ним происходит следующее:

1. Пара значений полей *Адрес источника* и *Идентификатор запроса* ищется в таблице локальной истории. С их помощью можно выяснить, приходил ли уже этот запрос и обрабатывался ли он. Если обнаруживается, что пакет является дубликатом, он отвергается и его обработка прекращается. В противном случае указанная пара значений заносится в таблицу истории, чтобы в будущем можно было обнаружить дубликаты. Обработка запроса продолжается.
2. Приемник ищет адрес назначения в таблице маршрутов. Если известен достаточно свежий маршрут, отправителю посылается пакет наличия маршрута ROUTE REPLY, сообщающий ему о том, как можно достичь получателя (в двух словах: «Используй меня»). Что значит «свежий маршрут»? Имеется в виду, что поле *Порядковый номер получателя* в таблице маршрутизации имеет значение большее или равное *Порядковому номеру получателя* из пакета запроса маршрута. Если оно меньше, значит, хранящийся в таблице маршрут является более старым, нежели предыдущий маршрут, имевшийся у отправителя к тому же пункту назначения. В этом случае выполняется пункт 3.
3. Поскольку у приемника отсутствует свежий маршрут к адресату, он инкрементирует поле *Счетчик переходов* и вновь широковещательным образом распространяет пакет запроса маршрута. Из пакета извлекаются данные и сохраняются в виде новой записи в таблице обратных маршрутов. Эти данные будут использоваться для построения обратного пути, по которому впоследствии необходимо будет послать ответный пакет отправителю. Стрелки на рис. 5.18 как раз показывают процесс построения обратного пути. Для записи о только что созданном обратном пути запускается таймер. При наступлении тайм-аута запись удаляется.

Ни *B*, ни *D* не знают, где находится узел *I*, поэтому каждый из них создает обратный путь к *A*, как показано стрелками на рис. 5.18, и широковещательным способом распространяет пакет со *Счетчиком переходов*, установленным в единицу. Этот пакет от *B* достигает *C* и *D*. Узел *C* делает запись в таблице обратных путей и, в свою очередь, тоже широковещательным способом распространяет пакет далее. Что касается *D*, то он отвергает пакет: для него это дубликат. Разумеется, и *B* отвергает пакеты, полученные от *D*. Тем не менее, *F* и *G* принимают широковещательное сообщение от *D* и сохраняют его, как показано на рис. 5.18, *в*. После того как *E*, *H* и *I* получают широковещательный пакет, запрос маршрута наконец достигает узла назначения (*I*). Этот счастливый миг запечатлен на рис. 5.18, *г*. Обратите внимание: несмотря на то, что мы показали распространение широковещательного пакета в виде трех стадий, на самом деле рассылка этого пакета разными узлами никак не координируется.

В ответ на пришедший запрос узел *I* генерирует пакет наличия маршрута ROUTE REPLY, показанный на рис. 5.20. Поля *Адрес отправителя*, *Адрес получателя* и *Счетчик переходов* копируются из ROUTE REQUEST, а *Порядковый номер получателя* берется из собственного счетчика, хранящегося в памяти. Поле *Счетчик переходов* устанавливается в 0. Поле *Время существования* используется для управления реализуемостью маршрута. Данный пакет распространяется методом одноадресной передачи на тот узел, с которого пришел запрос маршрута. В данном случае он уходит на узел *G*. Затем, в соответствии с установленным обратным путем, он попадет на *D* и наконец на *A*. При проходе каждого узла *Счетчик переходов* инкрементируется, так что узел-отправитель может увидеть, насколько далеко от него находится узел-получатель (*I*).

Адрес отправителя	Адрес получателя	Порядковый номер получателя	Счетчик переходов	Время существования

Рис. 5.20. Формат пакета ROUTE REPLY

Каждый узел, через который проходит пакет на обратном пути (к *A*), проверяет его. При выполнении хотя бы одного из трех условий на его основе строится запись в локальной таблице маршрутов о пути к *I*. Вот эти условия:

1. Не известен ни один маршрут к *I*.
2. Последовательный номер для *I* в пакете ROUTE REPLY больше, чем значение в таблице маршрутизации.
3. Последовательные номера равны, но новый путь короче.

Таким образом, все узлы, стоящие на обратном пути к *A*, совершенно бесплатно получают информацию о маршруте к узлу *I*. Это как бы побочный продукт построения маршрута для *A*. Узлы, получившие исходный пакет запроса маршрута, но не стоящие на обратном пути (узлы *B*, *C*, *E*, *F* и *H* в данном примере) удаляют запись в таблице обратных маршрутов, когда ассоциированный с ней таймер достигает тайм-аута.

В больших сетях алгоритмом генерируется много широковещательных пакетов даже для адресатов, расположенных довольно близко друг к другу. Число этих пакетов может быть уменьшено следующим образом. *Время жизни* IP-пакета устанавливается отправителем в значение, соответствующее ожидаемому диаметру сети, 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 с *Временем жизни*, равным 1. Если в течение разумного времени ответ не приходит, посылается еще один запрос с *Временем жизни*, равным 2, и т. д. Таким образом, поиск, начавшийся в какой-то локальной области, все больше расширяет свой охват.

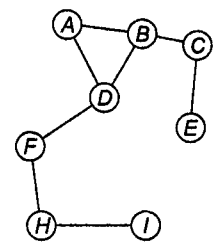
Обслуживание маршрута

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 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 соседи снабжали узел пакетами для данных адресатов в течение последних ΔT секунд. Такие соседи называются **активными соседями** узла *N* для данного адресата. Узел *N* осуществляет сбор подобных сведений с помощью таблицы маршрутизации, которая, как известно, в качестве индекса использует адрес назначения. В этой таблице указан тот узел, на который нужно переслать пакет, чтобы он мог дойти до адресата. Кроме того, в ней имеются сведения об оставшемся числе переходов, последнем порядковом номере получателя, а также об активных соседях данного адресата. Вид возможной таблицы маршрутизации для узла *D* при топологии, рассматриваемой в нашем примере, показан на рис. 5.21, *а*.

Адресат	Следующий переход	Расстояние	Активные соседи	Прочие поля
A	*A*	1	*F, G*	
B	*B*	1	*F, G*	
C	*B*	2	*F*	
E	*G*	2		
F	*F*	1	*A, B*	
G	*G*	1	*A, B*	
H	*F*	2	*A, B*	
I	*G*	2	*A, B*	

а



б

Рис. 5.21. Таблица маршрутизации узла *D* перед выходом из сети узла *G* (*а*); граф-схема сети после выхода из нее *G* (*б*)

Когда какой-либо из соседей узла *N* становится недоступным, проверяется его таблица 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. Активные соседи передают эти новости своим активным соседям, и так далее, пока все пути, зависевшие от ушедшего узла, не будут удалены из всех таблиц.

Рассмотрим наш предыдущий пример, предположив, что *G* внезапно выключился. Образовавшаяся в результате этого события топология показана на рис. 5.21, *б*. Когда *D* обнаруживает, что *G* ушел из сети, он просматривает свою таблицу маршрутизации и видит, что *G* стоял на пути к *E*, *G* и *I*. Объединением активных соседей для данных адресатов является множество {*A*, *B*}. Другими словами, *A* и *B* содержат записи о маршрутах, проходящих через *G*, поэтому их нужно проинформировать о том, что эти маршруты больше не работают. *D* сообщает им об этом, посылая специальные пакеты, заставляющие их обновить свои таблицы соответствующим образом. Сам узел *D* удаляет записи для адресатов *E*, *G* и *I* из таблицы маршрутизации.

Из приведенного описания это, может быть, и не очевидно, но основная разница между AODV и алгоритмом Беллмана—Форда состоит в том, что узлы не занимаются периодической широковещательной рассылкой пакетов, содержащих полные таблицы маршрутизации. Благодаря этому более эффективно используется полоса пропускания и увеличивается время работы элементов питания.

AODV, впрочем, может также заниматься широковещательной и групповой маршрутизацией. Детали см. в (Perkins and Royer, 2001). Маршрутизация в специализированных сетях — чрезвычайно популярная сегодня область исследований. Вопросам, связанным с ней, посвящено большое количество материалов. Например, (Chen и др., 2002; Hu and Johnson, 2001; Li и др., 2001; Raju and Garcia-Luna-Aceves, 2001; Ramanathan and Redi, 2002; Royer and Toh, 1999; Spohn and Garcia-Luna-Aceves, 2001; Tseng и др., 2001; Zadeh и др., 2002).

Поиск узла в равноранговых сетях

Относительно новым явлением являются равноранговые сети, в которых большое количество пользователей, имеющих обычно постоянное кабельное соединение с Интернетом, работает с разделяемыми ресурсами. Первым широко известным применением равноранговых сетей стало создание нелегальной системы Napster, 50 миллионов пользователей которой обменивались звукозаписями без разрешения обладателей авторских прав. Это продолжалось до тех пор, пока сеть Napster после бурной полемики не была закрыта решением суда. Тем не менее, у равноранговых сетей есть множество интересных и в то же время легальных применений. С ней связаны некоторые проблемы, сходные с проблемой маршрутизации, хотя и не такие же, как мы только что изучили. Сейчас мы их вкратце рассмотрим.

Что делает равноранговые системы интересными, так это их полная распределенность. Все узлы симметричны, никакого единого управления или иерархии не существует. В типичной равноранговой сети пользователи обладают какой-то информацией, могущей представлять интерес для других. Это может быть бесплатное программное обеспечение, музыка (являющаяся всеобщим достоянием), фотографии и т. д. Если пользователей много, они друг о друге ничего не знают и, возможно, никогда не узнают. Но совершенно непонятно, как искать что-то нужное в такой сети! Одним из решений является создание централизованной базы данных, но это может быть неосуществимо при некоторых условиях (напри-

мер, в сети нет добровольцев, желающих содержать и обслуживать такую базу). То есть проблема сводится к тому, что пользователю нужен какой-то метод поиска узла, на котором хранится интересующая его информация, в условиях отсутствия централизованной базы данных и даже централизованного индекса.

Предположим, что каждый пользователь владеет одной или более единицей информации, такой как песни, программы, фотографии, файлы и тому подобное — все то, чем другие пользователи, возможно, заинтересуются. У каждого такого объекта имеется строка ASCII, именующая его. Потенциальный пользователь знает только эту строку и стремится найти одного или нескольких пользователей, у которых есть данный объект, и узнать его (их) IP-адрес.

В качестве примера рассмотрим распределенную генеалогическую базу данных. У каждого генеалога есть онлайновый набор записей о его предках и родственниках, возможно, с фотографиями, аудиозаписями или даже видеозаписями. Понятно, что общий прадедушка был у многих, поэтому запись о нем будет иметься на нескольких узлах сети. Запись идентифицируется по имени человека, записанном в какой-нибудь канонической форме. И тут генеалог обнаруживает в архиве завещание прадедушки, в соответствии с которым золотые карманные часы он передает своему племяннику. Теперь необходимо узнать имя этого племянника и того, у кого могут быть какие-либо сведения о нем. Как же это сделать, не имея централизованной базы данных?

Для решения этой проблемы было предложено несколько различных алгоритмов. Мы рассмотрим **метод хорды** (Dabek и др., 2001a; Stoica и др., 2001). Несколько упрощенное объяснение принципа его работы таково. Пусть система состоит из n пользователей. У каждого из них есть какие-то записи, и каждый готов хранить биты и части индекса, которые могут быть использованы другими. У каждого узла есть IP-адрес, который может быть закодирован m-битным номером с помощью хэш-функции. В методе хорды используется алгоритм SHA-1 для вычисления значения хэш-функции. SHA-1 применяется в криптографии, и мы рассмотрим его в главе 8. А сейчас нам просто надо знать, что это некоторая функция с аргументом в виде строки переменной длины и значением — 160-битным числом высокой степени случайности. Таким образом, любой IP-адрес можно закодировать 160-битным числом, называемым **идентификатором узла**.

Концепция состоит в том, что вообще все 2^{160} идентификаторов располагаются в возрастающем порядке, образуя большой круг чисел. Некоторые из идентификаторов соответствуют реально существующим узлам, но это лишь малая часть из них. На рис. 5.22 показан круг идентификаторов для $m = 5$ (на дуги внутри круга пока внимания не обращайте). В этом примере реальным узлам соответствуют идентификаторы 1, 4, 7, 12, 15, 20 и 27. На рисунке кружочки с этими номерами закрашены. Всем остальным идентификаторам нельзя поставить в соответствие какие-либо узлы.

Определим функцию определения последователя *successor*(k) как идентификатор первого реального узла, следующего после k в описанном нами круге (по часовой стрелке). Например, *successor*(6) = 7, *successor*(8) = 12, *successor*(22) = 27.

Названия записей (например, названия песен, имена предков и т. п.) также обрабатываются хэш-функцией *hash* (с помощью алгоритма SHA-1) и превращаются в 160-битные числа, называемые **ключами**. Таким образом, чтобы получить ключ (*key*) из названия записи (*name*), необходимо вычислить *key* = *hash*(*name*). Такое вычисление является просто локальной процедурой, вызовом функции *hash*. Если держатель генеалогической записи хочет сделать ее доступной всем, он должен построить кортеж (набор взаимосвязанных величин), состоящий из полей (*name*, *мой_IP-адрес*), и затем выполнить функцию *successor*(*hash*(*name*)) для сохранения кортежа в общепринятой форме. Если существует несколько записей (на разных узлах) для одного и того же названия, все их кортежи будут храниться на одном и том же узле. То есть индекс распределяется случайным образом между узлами. Во избежание сбоев системы, для хранения кортежей на p узлах используется p различных хэш-функций, но далее мы не будем учитывать этот нюанс.

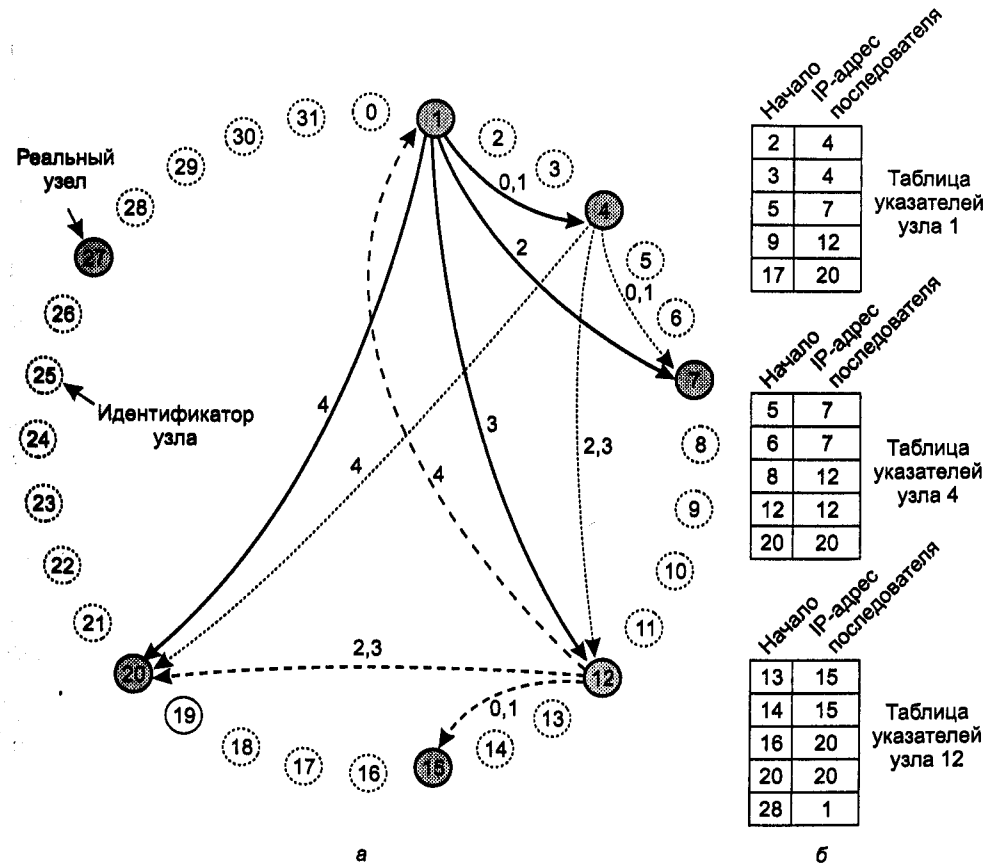


Рис. 5.22. Набор из 32 идентификаторов узлов, выстроенных по кругу (*а*). Закрашенные кружочки соответствуют реальным узлам. Дугами обозначены указатели с узлов 1, 4 и 12. Подписи на дугах соответствуют индексам таблицы. Примеры таблиц указателей (*б*)

Если кто-то из пользователей затем захочет найти название записи (*name*), он вычислит значение хэш-функции, получит ключ (*key*) и затем с помощью функции *successor*(*key*) сможет найти IP-адрес узла, хранящего кортежи индексов. Первый шаг осуществляется довольно просто, не в пример второму. Для того чтобы можно было найти IP-адрес узла, соответствующего какому-либо ключу, каждый узел должен поддерживать определенные служебные структуры данных. Одной из них является IP-адрес последователя. Например, на рис. 5.22 последователь узла 4 — это узел 7, а последователь узла 7 — узел 12.

Теперь можно начинать поиск. Запрашивающий узел посылает последователю пакет, содержащий его IP-адрес и ключ, который он ищет. Этот пакет пересылается по кругу до тех пор, пока не будет найден искомый узел. На нем проверяется соответствие имеющейся информации ключу, и при положительном результате проверки пакет возвращается напрямую запрашивающему узлу, чей IP-адрес содержится в запросе.

В качестве первой оптимизации алгоритма предлагается следующее: каждый узел должен знать IP-адрес не только последователя, но и предшественника, тогда запрос можно будет распространять одновременно в двух направлениях — по часовой стрелке и против часовой стрелки, в зависимости от того, какой из путей кажется более коротким. Например, на рис. 5.22 узел 7 может отправить пакет по часовой стрелке, если ищется узел 10. А если ищется узел 3, то логичнее направить пакет против часовой стрелки.

Даже с этой возможностью выбора направления линейный поиск остается крайне неэффективным методом для больших равноранговых сетей, поскольку среднее число узлов, которое потребуется обойти для поиска записи, равно $n/2$. Значительно ускорить поиск позволяет поддерживаемая каждым узлом специальная таблица, которая в методе хорд носит название **таблицы указателей**. В ней имеется m записей, пронумерованных от 0 до $m - 1$. Каждая запись содержит два поля: *начало* и IP-адрес последователя — *successor*(*start*), как показано на рис. 5.22, *б*. Значения полей записи i на узле k равны:

$$Start = k + 2^i \ (modulo 2^m),$$

IP-адрес *successor* (*start* [i]).

Обратите внимание: на узлах хранится сравнительно небольшое число адресов, и большинство из них расположено довольно близко друг от друга в терминах идентификаторов узлов.

С использованием таблицы указателей процесс поиска ключа на узле k выглядит следующим образом. Если ключ попадает в диапазон между k и *successor*(k), значит, он находится, на самом деле, на узле *successor*(k), и поиск прекращается. В противном случае таблица указателей просматривается на предмет поиска записи, поле *начало* которой является ближайшим предшественником ключа. Запрос посылается напрямую по IP-адресу из таблицы указателей. Узел с этим адресом просят перенять инициативу поиска. Поскольку искомый ключ находится где-то рядом, но идентификатор имеет меньшее значение, велика вероятность того, что конечный ответ будет дан очень скоро, спустя всего несколько дополнительных запросов. Поскольку каждая итерация при поиске вдвое уменьшает

последующую область рассмотрения (то есть оставшееся расстояние до цели), то можно доказать, что среднее число итераций равно $\log_2 n$.

В качестве первого примера рассмотрим поиск ключа *key* = 3 на узле 1. Узел с идентификатором 1 знает, что 3 лежит где-то между ним самим и его последователем — узлом 4. Он делает вывод о том, что ключ находится на узле 4, и поиск прекращается. Результатом является IP-адрес узла 4.

Второй пример. Пусть узлу 1 теперь понадобилось найти ключ *key* = 14. Число 14 не находится между 1 и 4, поэтому необходимо заглянуть в таблицу указателей. Ближайшим предшественником 14 является 9, поэтому запрос направляется на IP-адрес, хранящийся в записи с полем *начало*, равным 9. Конкретно, это узел 12. Последний видит, что 14 находится между ним самим и последователем — узлом 15, поэтому результатом поиска является IP-адрес узла 15.

Третий пример. Допустим, узлу 1 нужно найти ключ *key* = 16. Запрос, как и в предыдущем примере, отправляется на узел 12. Но на этот раз узел не может самостоятельно ответить на него. Он пытается найти тот узел, который находится ближе всего к 16, но имеет меньший идентификатор. Им является узел 14, который ссылается, в свою очередь, на узел 15. Запрос туда и посылается. Узел 15 видит, что 16 находится между ним самим и его последователем (20), поэтому он возвращает IP-адрес 20 просителю. Ответ возвращается сразу на узел 1.

Поскольку узлы могут появляться в сети и исчезать из нее в любое время, в методе хорд необходимо как-то обрабатывать подобные ситуации. Мы предполагаем, что когда система начинала работать, она была достаточно небольшой, и узлы могли обмениваться информацией напрямую, выстраивая первичный круг идентификаторов и таблицы указателей. После этого уже необходима автоматизированная процедура. Когда новый узел r собирается войти в сеть, он должен войти в контакт с какой-либо из уже находящихся в сети станций и попросить ее поискать для него IP-адрес *successor*(r). Затем новый узел выясняет, кто будет его предшественником (*successor*(r) для предшественника). Все это нужно для того, чтобы можно было определить свое место в круге идентификаторов. Например, если узел 24 на рис. 5.22 захочет войти в сеть, он может поинтересоваться у любого узла, чему равно значение *successor*(24). Выясняется, что оно равно 27. Затем у узла 27 надо узнать, кто является его предшественником (20). После того как обоим этим узлам сообщается о существовании нового узла, узел 20 помечает себе, что его последователем становится узел 24, а узел 27 — что этот узел становится его предшественником. Кроме того, узел 27 передает ключи диапазона 21–24 новичку. С этого момента последний считается зарегистрированным в равноранговой сети.

Тем не менее, ситуация изменилась, и многие таблицы указателей теперь содержат ошибки. Чтобы исправить их, все узлы запускают фоновые процессы, которые периодически пересчитывают таблицы указателей, обращаясь к функции *successor*. Когда какой-то из этих запросов наталкивается на новый узел, запись таблицы обновляется.

Если узел уходит вежливо, он передает свои ключи последователю и информирует предшественника о своем скором отбытии. При этом в цепочке идентификаторного круга предшественник теперь оказывается непосредственным соседом последователя. Однако если с узлом происходит какая-то авария, возникают

проблемы у его предшественника, поскольку он не знает, кто является его последователем. Решение данной проблемы состоит в том, что узлы запоминают не только своего прямого последователя, но и еще s последователей. Получается, что без тяжких последствий для окружающих из сети могут выпасть $s - 1$ последовательных узлов, и круг при этом не разорвется.

Метод хорд используется для создания распределенных файловых систем (Dabek и др., 2001b) и других приложений; научные изыскания в этой области идут и сейчас. Одна из равноранговых систем, Pastry, и ее приложения описаны в (Rowston and Druschel, 2001a; Rowston and Druschel, 2001b). Еще одна система, Freenet, обсуждается в (Clarke и др., 2002). Наконец, четвертая система этого типа описана в (Ratnasamy и др., 2001).

Алгоритмы борьбы с перегрузкой

Когда количество пакетов, передаваемых одновременно по подсети (или ее части), превышает некий пороговый уровень, производительность сети начинает снижаться. Такая ситуация называется **перегрузкой**. График на рис. 5.23 изображает симптомы этого явления. Когда число пакетов, посылаемых хостами в сеть, не превышает ее пропускной способности, все они доставляются адресатам (кроме небольшого процента поврежденных ошибками передачи). При этом количество доставленных пакетов пропорционально количеству посланных. Однако по мере роста трафика маршрутизаторы перестают успевать обрабатывать все пакеты и начинают их терять. При дальнейшем увеличении числа отправляемых пакетов ситуация продолжает ухудшаться. Когда число пакетов достигает максимального уровня, производительность сети начинает снижаться. При очень высоком уровне трафика производительность сети падает до совсем низкого уровня и практически никакие пакеты не доставляются.

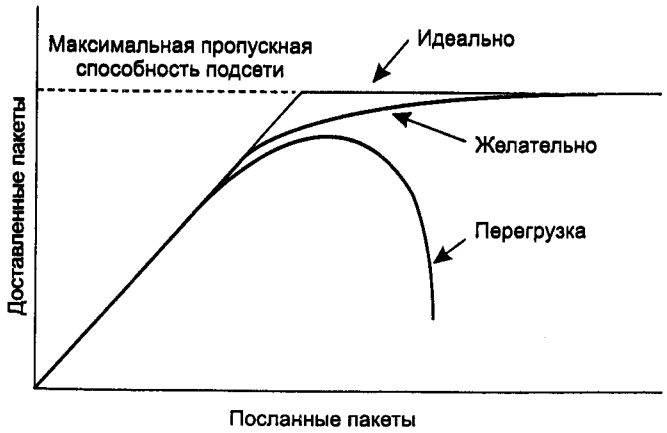


Рис. 5.23. При слишком высоком уровне трафика начинается перегрузка, и производительность сети резко снижается

Перегрузка может быть вызвана несколькими факторами. 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 буферизации всех 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, так как к тому времени, когда пакеты доберутся до начала очереди, они уже запоздают настолько, что источником будут высланы их дубликаты. Все эти пакеты будут посланы следующему маршрутизатору, еще более увеличивая нагрузку на всем протяжении маршрута к получателю.

Медленные процессоры также могут служить причиной заторов. Если центральные процессоры маршрутизаторов слишком медленно выполняют свои задачи, связанные с учетом, управлением очередями, обновлением таблиц и т. д., то очереди будут появляться даже при достаточно высокой пропускной способности линий. Аналогично, линии с низкой пропускной способностью также могут вызывать заторы в сети. Если заменить линии более совершенными, но оставить старые процессоры, или наоборот, такие действия обычно немного помогают, но часто просто приводят к сдвигу узкого места, вызванного несоответствием производительности разных частей системы. Проблема узкого места сохраняется до тех пор, пока компоненты системы не будут должным образом сбалансированы.

Необходимо пояснить, в чем состоит разница между борьбой с перегрузкой и управлением потоком. Предотвращение перегрузки гарантирует, что подсеть справится с предлагаемым ей трафиком. Это глобальный вопрос, включающий поведение всех хостов и маршрутизаторов, процессов хранения и пересылки на маршрутизаторах, а также множество других факторов, снижающих пропускную способность подсети.

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. Управление потоком обычно реализуется при помощи обратной связи между получателем и отправителем.

Чтобы разница между этими двумя проблемами стала яснее, представьте себе оптоволоконную сеть с пропускной способностью 1000 Гбит/с, по которой суперкомпьютер пытается передать персональному компьютеру файл 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 с промежуточным хранением, состоящую из 1000 больших компьютеров, соединенных линиями с пропускной способностью 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 алгоритмы борьбы с перегрузкой также используют обратную связь в виде специальных сообщений, посылаемых различным отправителям, с просьбой передавать данные помедленнее, когда в сети появляются заторы. 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. В дальнейшем мы еще будем рассматривать этот вопрос.

Мы начнем изучение алгоритмов борьбы с перегрузкой с рассмотрения общей модели. Затем мы познакомимся с общим подходом к предотвращению перегрузки, а также с различными динамическими алгоритмами борьбы с перегрузкой, которую не удалось предотвратить.

Общие принципы борьбы с перегрузкой

Многие проблемы, возникающие в сложных системах, таких как компьютерные сети, следует рассматривать с точки зрения теории управления. При таком подходе все решения делятся на две группы: без обратной связи и с обратной связью. Решения без обратной связи заключаются в попытках решить проблему с помощью улучшения дизайна системы, пытаясь, таким образом, в первую очередь предотвратить возникновение самой ситуации перегрузки. Никаких корректирующих действий во время работы системы не предпринимается.

К методам управления без обратной связи относятся решения о том, когда разрешать новый трафик, когда отвергать пакеты и какие именно, а также составление расписаний для различных участков сети. Общее в этих решениях то, что они не учитывают текущего состояния сети.

Решения с обратной связью, напротив, основываются на учете текущего состояния системы. Этот подход состоит из трех следующих частей:

1. Наблюдения за системой с целью определить, где и когда произойдет перегрузка.
2. Передачи информации о перегрузке в те места, где могут быть предприняты соответствующие действия.
3. Принятия необходимых мер при работе системы для устранения перегрузки.

При наблюдении за состоянием подсети с целью обнаружения перегрузки могут измеряться различные параметры. Среди них следует выделить следующие: процент пакетов, отвергаемых из-за отсутствия свободного места в буфере; средняя длина очереди; процент пакетов, переданных повторно по причине истекшего времени ожидания подтверждения; среднее время задержки пакетов и среднеквадратичное отклонение задержки пакетов. Во всех случаях увеличивающиеся значения параметров являются сигналами о растущей перегрузке.

Второй этап борьбы с перегрузкой состоит в передаче информации о перегрузке от места ее обнаружения туда, где могут быть приняты какие-то меры по

ее устранению. Очевидное решение заключается в том, чтобы маршрутизатор, обнаруживший перегрузку, пересылал источнику или источникам трафика пакет с извещением о наличии проблемы. Такие пакеты, конечно, окажут дополнительную нагрузку на сеть как раз в тот момент, когда нагрузку необходимо снизить.

Существуют, однако, и другие решения. Например, можно зарезервировать в каждом пакете бит или поле, которые будут заполняться маршрутизаторами при достижении перегрузкой порогового уровня. Таким образом, соседи этого маршрутизатора будут предупреждены о том, что на данном участке сети наблюдается перегрузка.

Еще один метод состоит в том, что хосты или маршрутизаторы периодически посылают пробные пакеты, явно спрашивая друг друга о перегрузке. Собранная таким образом информация может затем использоваться для выбора маршрутов в обход участков сети, в которых возникла проблема с перегрузкой. Так, некоторые радиостанции обзавелись вертолетами, летающими над городами и сообщающими слушателям о заторах на дорогах в надежде, что слушающие их водители выберут маршруты для своих пакетов (то есть машин) в обход пробок.

Все системы с обратной связью предполагают, что получившие информацию о перегрузке в сети хосты и маршрутизаторы предпримут какие-нибудь действия для устранения перегрузки. 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 будет спокоен, как слон, и 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. Для правильной работы необходимо некоторое усреднение, однако правильный выбор значения постоянной времени является нетривиальной задачей.

Известны различные алгоритмы борьбы с перегрузкой. Янг (Yang) и Редди (Reddy) (1995) даже разработали специальный метод классификации этих алгоритмов. Они начали с того, что разделили все методы на алгоритмы с обратной связью и без нее, как уже описывалось ранее. Затем они разделили алгоритмы без обратной связи на работающие у отправителя и у получателя. Алгоритмы с обратной связью также были разделены на две подкатегории: с явной и неявной обратной связью. В алгоритмах с явной обратной связью от точки возникновения перегрузки в обратном направлении посылаются пакеты, предупреждающие о заторе. В алгоритмах с неявной обратной связью источник приходит к выводу о наличии перегрузки, основываясь на локальных наблюдениях, — например, по значению интервала времени, требующегося для получения подтверждения.

Наличие перегрузки означает, что нагрузка временно превысила возможности ресурсов данной части системы. Есть два решения данной проблемы: увеличить ресурсы системы или снизить нагрузку. Например, подсеть может использовать телефонные линии с модемами, чтобы увеличить пропускную способность между определенными точками. В спутниковых системах большую пропускную спо-

собность часто дает увеличение мощности передатчика. Распределение трафика по нескольким маршрутам вместо постоянного использования одного и того же, пусть даже оптимального пути также может позволить ликвидировать местную перегрузку. Наконец, для увеличения пропускной способности сети в случае серьезных заторов могут быть задействованы запасные маршрутизаторы, которые обычно применяются для повышения устойчивости системы в случае сбоя.

Однако иногда увеличить пропускную способность бывает невозможно либо она уже увеличена до предела. В таком случае единственный способ борьбы с перегрузкой состоит в уменьшении нагрузки. Для этого существует несколько способов, включая отказ в обслуживании или снижение уровня обслуживания некоторых или всех пользователей, а также составление более четкого расписания потребностей пользователей в обслуживании.

Некоторые из этих методов, которые будут кратко рассмотрены далее, лучше всего применимы к виртуальным каналам. В подсетях, основанных на использовании виртуальных каналов, эти методы могут применяться на сетевом уровне. В дейтаграммных подсетях они иногда также могут применяться в соединениях транспортного уровня. В данной главе основное внимание будет уделено применению методов борьбы с перегрузкой на сетевом уровне. В следующей главе мы обсудим, что можно сделать на транспортном уровне.

Стратегии предотвращения перегрузки

Начнем изучение методов борьбы с перегрузкой с систем без обратной связи. Эти системы разработаны в первую очередь для предотвращения перегрузки, а не для борьбы с уже имеющей место перегрузкой. Они пытаются достичь своей цели, используя соответствующие стратегии на разных уровнях. В табл. 5.2 показаны различные стратегии уровней передачи данных, сетевого и транспортного, способные влиять на перегрузку [162].

Таблица 5.2. Стратегии предотвращения перегрузки

Уровень	Стратегии
Транспортный	Политика повторной передачи
	Политика кэширования пакетов, приходящих в неверном порядке
	Политика подтверждений
	Политика управления потоком
	Определение тайм-аутов
Сетевой	Виртуальные каналы против дейтаграмм в составе подсети
	Политика очередей пакетов и обслуживания
	Политика игнорирования пакетов
	Алгоритм маршрутизации
	Управление временем жизни пакетов
Передачи данных	Политика повторной передачи
	Политика кэширования пакетов, приходящих в неверном порядке
	Политика подтверждений
	Политика управления потоком

Начнем рассмотрение различных стратегий с уровня передачи данных. Стратегия повторной передачи определяет, насколько быстро у отправителя истекает время ожидания подтверждения и что он передает после того как время ожидания истекло. Нетерпеливый отправитель, у которого время ожидания истекает слишком быстро и который повторно посылает все неподтвержденные пакеты с помощью алгоритма возврата на n, окажет более сильную нагрузку на сеть, нежели ленивый отправитель, использующий выборочный повтор. Тесно связана с этим стратегия кэширования. Если получатели просто игнорируют все пакеты, приходящие не в том порядке, то все проигнорированные пакеты придется передавать позднее еще раз, что окажет дополнительную нагрузку на сеть.

Стратегия подтверждений также влияет на перегрузку. Если каждый пакет немедленно подтверждается получателем, то пакеты с подтверждениями образуют дополнительный трафик. Однако если подтверждения добираются обратно «верхом» на попутном потоке кадров, то количество трафика в сети снижается, зато увеличивается среднее время получения подтверждений, что может, в свою очередь, вызвать увеличение повторно переданных пакетов вследствие истечения времени ожидания подтверждений. Более жесткая схема управления потоком (например, с небольшим размером окна) уменьшает скорость передачи данных и помогает бороться с перегрузкой.

Существует также зависимость перегрузки от того, является ли сетевой уровень дейтаграммным или он основан на виртуальных каналах, так как многие алгоритмы борьбы с перегрузкой работают только в подсетях с виртуальным каналами. Политика очередей пакетов и обслуживания определяет количество очередей у каждого маршрутизатора — например, одна общая очередь для всех линий, или по очереди для каждой линии, или какой-нибудь комбинированный вариант. Она также определяет порядок обработки пакетов (например, поочередно или в порядке приоритетов). Политика игнорирования пакетов является правилом, определяющим набор пакетов, подлежащих отвержению, когда не хватает памяти. Хорошо продуманная стратегия может облегчить симптомы перегрузки, тогда как неудачная политика может даже ухудшить ситуацию.

Хороший алгоритм выбора маршрута может помочь избежать локальной перегрузки, перераспределяя трафик по всем линиям, тогда как неудачный алгоритм может направить слишком большое количество пакетов по одной линии и вызвать затор. Наконец, управление временем жизни пакетов определяет, как долго пакет может перемещаться по сети, прежде чем он будет проигнорирован очередным маршрутизатором. Если это время слишком велико, то потерянные пакеты могут засорять собою сеть, однако если время жизни пакета слишком мало, то пакеты не будут успевать достичь адресата, что приведет к необходимости повторных передач.

На транспортном уровне применяются те же стратегии, что и на уровне передачи данных, но к ним добавляется еще и проблема определения времени ожидания подтверждения, что на транспортном уровне осуществить значительно сложнее, поскольку время пересечения всей сети предсказать значительно сложнее, чем время передачи пакета от какого-либо маршрутизатора до его соседа. Если этот интервал слишком короток, будут высылаться излишние повторные пакеты,

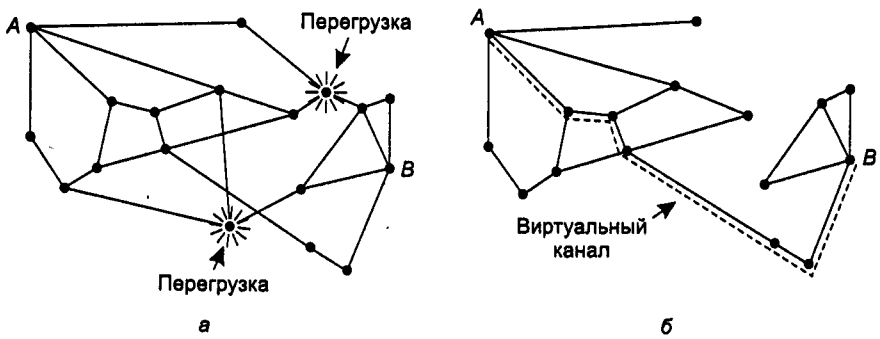
а если слишком велик — перегрузка снизится, но увеличится задержка в случае потери пакета.

Борьба с перегрузкой в подсетях виртуальных каналов

Описанные ранее методы борьбы с перегрузкой в основном являются методами без обратной связи: они в первую очередь пытаются предотвратить перегрузку, а не занимаются устранением уже возникшей перегрузки. В данном разделе мы опишем несколько подходов к динамическому управлению перегрузкой в подсетях виртуальных каналов. В следующих двух разделах будут описаны методы, применимые в любой подсети.

Широко применяемым методом недопущения ухудшения уже начавшейся перегрузки является **управление допуском**. Идея этого метода проста: когда приходит сигнал о перегрузке, никакие новые виртуальные каналы не создаются до тех пор, пока проблема не разрешится. То есть любые попытки установить новые соединения транспортного уровня пресекаются. Понятно, что если пустить в сеть, в которой уже возникла перегрузка, дополнительных пользователей, то ситуация только ухудшится. Хотя такой метод несколько прямолинеен и не эстетичен, зато он дешев, надежен и практичен. В обычных телефонных системах этот метод тоже широко применяется: когда коммутатор оказывается перегруженным, вы поднимаете трубку и не слышите гудка.

Альтернативный подход заключается в том, что создание новых виртуальных каналов разрешается, но эти каналы тщательно прокладываются в обход заторов. Для примера рассмотрим подсеть, показанную на рис. 5.24, в которой два маршрутизатора перегружены.



а *б*

Рис. 5.24. Перегруженная подсеть (*а*); та же подсеть с устраненной перегрузкой (*б*). Показан виртуальный канал между *A* и *B*

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, подсеть усекается, как показано на рис. 5.24, *б*. При этом из нее удаляются перегруженные маршрутизаторы и все их линии связи. Пунктирной линией показан возможный маршрут виртуального канала в обход перегруженных маршрутизаторов.

Другая стратегия, связанная с виртуальными каналами, состоит в достижении соглашения между хостом и подсетью во время установки виртуального канала. Эта договоренность обычно описывает объем и форму трафика, требуемое качество обслуживания и другие параметры. В качестве выполнения своей части соглашения подсеть обычно резервирует ресурсы на пути следования создаваемого канала. К этим ресурсам относятся память для буферов и таблиц маршрутизаторов и пропускная способность линий. При таком подходе возникновение перегрузки в новом виртуальном канале маловероятно, так как все необходимые ресурсы были зарезервированы и их доступность гарантируется.

Подобное резервирование может выполняться как в виде постоянной стандартной процедуры, так и в виде действия, выполняемого только при возникновении перегрузки. Недостаток постоянного резервирования заключается в том, что на эту процедуру расходуются вычислительные ресурсы. Если шесть виртуальных каналов, которым разрешено использовать по 1 Мбит/с, проходят по одной и той же физической линии с пропускной способностью 6 Мбит/с, линия должна быть помечена как полная, хотя маловероятно, что все шесть виртуальных каналов передают данные одновременно, да еще и используют всю доступную пропускную способность. Следовательно, платой за резервирование будет неиспользованная пропускная способность.

Борьба с перегрузкой в дейтаграммных подсетях

Теперь обратимся к методу, применяемому в дейтаграммных подсетях (впрочем, он может использоваться и в подсетях с виртуальным каналом). Каждый маршрутизатор может запросто следить за использованием своих выходных линий и других ресурсов. Например, с каждой линией может быть связана вещественная переменная u, значение которой в пределах от 0,0 до 1,0 отражало бы использование линии за последнее время. Такую усредненную оценку загруженности линии можно получить с помощью несложных вычислений, периодически замеряя мгновенную загруженность линии f (0 либо 1) и рассчитывая новое значение переменной u по формуле

$$u_{\text{нов}} = au_{\text{ст}} + (1 - a)f,$$

где константа a определяет, насколько быстро маршрутизатор забывает свое прошлое.

Когда значение переменной u начинает превышать некий пороговый уровень, это означает, что линия переходит в опасное состояние. Каждый приходящий пакет проходит проверку: если ему предстоит следовать по линии, находящейся в близком к перегрузке состоянии, то выполняется одно из нескольких действий. Далее мы обсудим, как именно маршрутизатор может отреагировать на эту ситуацию.

Биты предупреждения

В старой архитектуре DECNET сигнализация опасного состояния производилась путем установки специального бита в заголовке пакета. То же самое делается в технологии ретрансляции кадров. Пакет доходит до места назначения, и в подтверждение о его доставке, отсылаемое источнику, включается бит предупреждения, увидев который, источник должен снизить трафик.

Продолжая находиться в опасном состоянии, маршрутизатор устанавливает биты предупреждения во все проходящие мимо него пакеты. Это означает, что хосты-источники информируются о проблеме. Со своей стороны источники изучают ту часть подтверждений, в которой установлены биты предупреждения, и соответствующим образом подстраивают свою скорость передачи. Если такие подтверждения продолжают прибывать, источник продолжает снижать скорость. Достигнув такой скорости, когда пакеты еле-еле просачиваются по линии, хост может начать увеличивать скорость. Обратите внимание: бит предупреждения может установить любой маршрутизатор, поэтому трафик может начать повышаться только тогда, когда со всеми маршрутизаторами все в порядке.

Сдерживающие пакеты

Предыдущий алгоритм борьбы с перегрузкой действует довольно хитро: он использует окольные средства для сообщения источнику о том, что неплохо бы умерить пыл. Но почему бы не сказать ему об этом прямо? Такая идея привела к созданию подхода, при котором маршрутизатор сам отправляет источнику **сдерживающий пакет**. 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.

Когда хост-отправитель получает сдерживающий пакет, он должен уменьшить трафик к указанному получателю на некоторый процент X. Поскольку другие пакеты, направляющиеся к тому же адресату, скорее всего, в этот момент уже находятся в пути, они также породят сдерживающие пакеты. Поэтому в течение фиксированного интервала времени хост должен игнорировать сдерживающие пакеты, относящиеся к тому же получателю. По истечении этого периода времени хост начинает прослушивать другой интервал на предмет сдерживающих пакетов. Если приходит хотя бы один, это означает, что линия все еще перегружена, поэтому хост еще сильнее снижает выходной поток и снова начинает игнорировать последующие сдерживающие пакеты. Если в течение второго интервала времени (периода ожидания сдерживающих пакетов) сдерживающие пакеты не приходят, хост может снова увеличить поток. Обратная связь, присутствующая в данном протоколе, может помочь предотвратить перегрузку, не ограничивая поток до тех пор, пока не возникнет необходимость.

Хосты могут уменьшать трафик, изменяя свои стратегические параметры, например размер окна. Обычно первый сдерживающий пакет уменьшает скорость передачи данных в два раза, следующий — в четыре и т. д. Увеличения скорости производятся меньшими приращениями, чтобы избежать слишком быстрого повторения перегрузки.

Существуют различные варианты этого алгоритма борьбы с перегрузкой. В одном из них маршрутизаторами применяется несколько пороговых уровней загруженности линии. В зависимости от того, какой из порогов пересечен, сдерживающие пакеты могут содержать мягкое или строгое предупреждение либо ультиматум.

В качестве варианта может измеряться длина очереди или объем свободной памяти маршрутизатора. При этом можно использовать ту же экспоненциальную весовую функцию, что и для u.

Сдерживающие пакеты для ретрансляционных участков

При больших скоростях передачи данных и при сильной удаленности хостов с отправкой сдерживающих пакетов возникают проблемы, поскольку реакция на них оказывается крайне запоздалой. Рассмотрим, к примеру, хост в Сан-Франциско (маршрутизатор A на рис. 5.25), посылающий поток данных на хост, расположенный в Нью-Йорке (маршутизатор D на рис. 5.25), со скоростью 155 Мбит/с. Если у нью-йоркского хоста станет кончаться буферная память, сдерживающему пакету потребуется около 30 мс на то, чтобы добраться обратно в Сан-Франциско и сообщить о том, что необходимо снизить объем трафика. Распространение сдерживающего пакета схематично показано на второй, третьей и четвертой диаграммах рис. 5.25, *а*. За те 30 мс, пока этот пакет движется по сети, в сторону нью-йоркского маршрутизатора передается еще 4,6 Мбит данных, с которыми тоже надо как-то совладать. Только к седьмой диаграмме (рис. 5.25, *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. 5.25, *б*. Как только сдерживающий пакет достигает точки F, поток данных от F в сторону D должен быть уменьшен. Таким образом, F резервирует для потока 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.

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 Kanakia, 1992).

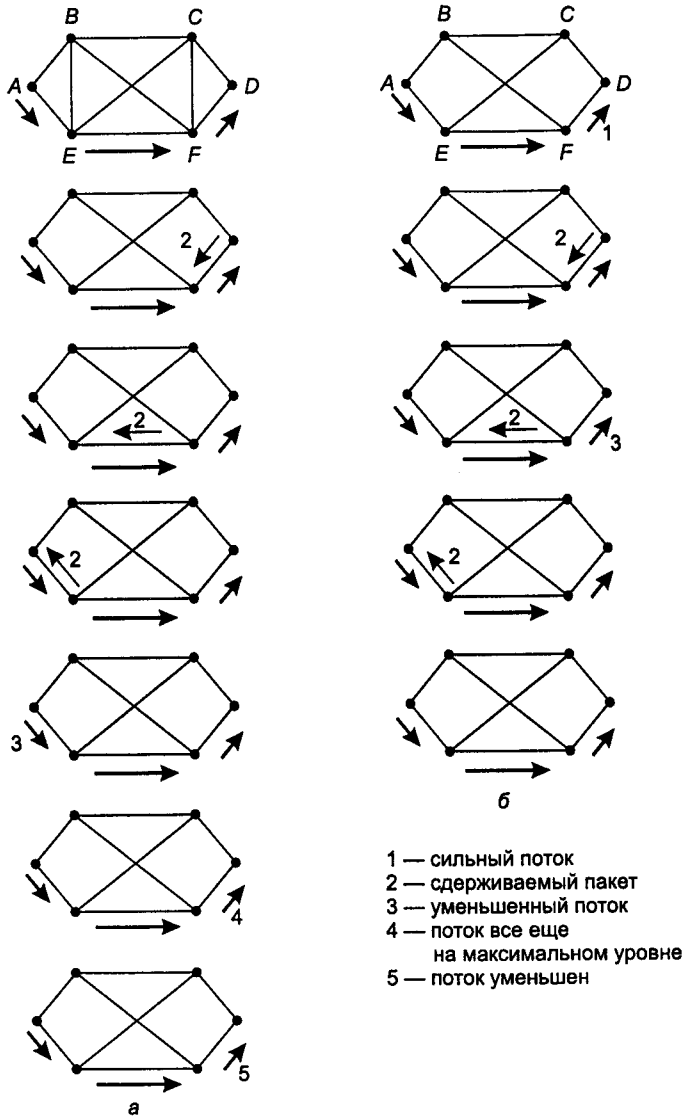


Рис. 5.25. Сдерживающий пакет влияет только на источник (*а*); сдерживающий пакет влияет на все промежуточные участки (*б*)

Сброс нагрузки

Когда ни один из описанных ранее 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 морозные зимние дни, когда потребности в электроэнергии для обогревателей резко возрастают.

Маршрутизатор, заваленный пакетами, может выбирать пакеты просто случайным образом, но обычно имеются более оптимальные варианты. Выбор пакета, который будет отвергнут, может зависеть от приложения, пересылающего этот пакет. Для передачи файла более старый пакет ценится выше нового, так как отвержение пакета номер 6 и сохранение пакетов с номерами с 7-го по 10-й может привести к тому, что получатель запросит еще раз пакеты с 6-го по 10-й (если получатель просто отвергает все пакеты, приходящие не в том порядке). В файле, состоящем из 12 пакетов, выбрасывание 6-го пакета может потребовать повторной передачи пакетов с 7-го по 12-й, тогда как выбрасывание пакета номер 10 может потребовать повторной передачи только пакетов с 10-го по 12-й. Для мультимедийных приложений, напротив, новый пакет важнее старого. Первую стратегию (старое лучше нового) часто называют **винной стратегией**, а вторую (новое лучше старого) — **молочной стратегией**.

Чтобы сделать этот алгоритм еще разумнее, необходимо участие в нем отправителей. Во многих приложениях одни пакеты могут быть значительно важнее других. Например, некоторые алгоритмы сжатия видеосигнала 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. Точно так же при передаче страницы, содержащей текст и рисунок, потеря линии пикселов рисунка может остаться почти незамеченной, тогда как потеря строки текста крайне нежелательна.

Для реализации интеллектуальной стратегии выбрасывания части информации приложения должны помечать свои пакеты классами приоритетов, соответствующими их важности. В этом случае маршрутизаторы смогут сначала выбросить пакеты нижнего 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.

Стимулом может служить стоимость обслуживания, то есть пересылка пакетов низкоприоритетным классом может быть дешевле, чем высокоприоритетным. В качестве альтернативы источникам может быть ультимативно предложено отправлять высокоприоритетные пакеты только в условиях низкого трафика, а с повышением загрузки сети прекращать их отправку.

Еще один вариант состоит в разрешении хостам превышать пределы, указанные в соглашении, заключенном при создании виртуального канала (например, использовать большую пропускную способность, чем договаривались), но при условии, что весь дополнительный трафик 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

Хорошо известно, что при борьбе с перегрузкой гораздо проще вовремя обнаружить затор, чем дать ему развиться до критических размеров, а потом думать, что делать в сложившейся ситуации. Это соображение приводит к идее отвержения некоторых пакетов еще до того, как все буферное пространство будет заполнено скопившимися необработанными данными. Популярный алгоритм, реализующий данную идею, называется **случайным ранним обнаружением** (RED — Random Early Detection, Floyd и Jacobson, 1993). Некоторые транспортные протоколы (включая TCP) на потерю своих пакетов отвечают снижением трафика от источника, чего мы, в сущности, и добиваемся. 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. Этот факт и используется для уменьшения перегрузок.

Если заставить маршрутизаторы сознательно терять пакеты еще до того как ситуация станет безнадежной (именно такой момент зашифрован в слове «раннее» из названия подхода), то останется время на то, чтобы предпринять какие-то действия. Для определения условий, при которых следует начинать терять пакеты, маршрутизаторы постоянно высчитывают скользящее среднее длин своих очередей. Когда средняя длина очереди на какой-либо линии превышает пороговое значение, эта линия объявляется перегруженной и выполняются действия по предотвращению затора.

Маршрутизатор не всегда может определить, кто из отправителей больше всех виноват в заваливании линии данными, поэтому пакет из очереди выбирается случайным образом, и это — самое справедливое, что можно сделать в данной ситуации.

Но как маршрутизатор сообщит источнику о возникшей проблеме? Можно послать ему сдерживающий пакет, как описывалось ранее. Но это приведет к созданию дополнительной нагрузки на и так уже почти перегруженную сеть. Другой подход заключается в том, чтобы просто потерять выбранный пакет и никому не сообщать об этом. Источник в конечном счете среагирует на отсутствие подтверждения. Поскольку он знает, что потеря пакетов обычно связана с перегрузкой сети, он уменьшит скорость выдачи пакетов и не станет пытаться во что бы то ни стало пробиться к загруженному маршрутизатору. Такая неявная форма обратной связи может применяться только тогда, когда источник знает, что на потерю пакетов надо реагировать снижением скорости передачи. В беспроводных сетях, где большинство испорченных и потерянных пакетов обязано своим исчезновением шуму в эфире, такой подход не годится.

Борьба с флуктуациями

Для таких приложений как аудио- и видеопередача, не так уж важно, 20 или 30 мс занимает доставка пакетов, до тех пор, пока время доставки постоянно. Колебание (то есть, среднеквадратичное отклонение) времени доставки пакетов называется **флуктуацией**. Если одни пакеты будут доставляться за 20 мс, а другие — за 30 мс,

изображение или звук начнет дрожать. В этом случае говорят о наличии сильных флуктуаций. На рис. 5.26 изображены примеры флуктуаций. Внутри системы, с другой стороны, может существовать договоренность о том, что 99 % пакетов должны быть доставлены с задержкой в диапазоне от 24,5 до 25,5 мс, и качество при этом будет вполне приемлемым.

Выбранный диапазон должен быть, конечно, выполнимым. При вычислении времени задержки необходимо принимать во внимание время передачи по каналу со скоростью света, минимальную задержку при прохождении маршрутизаторов, а также некоторые другие неизбежные задержки.

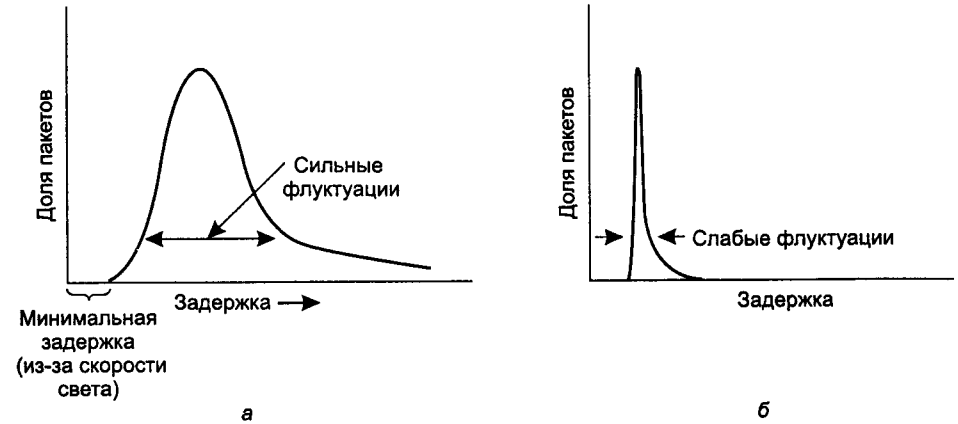


Рис. 5.26. Сильные флуктуации (*а*); слабые флуктуации (*б*)

Для ограничения флуктуаций должно быть вычислено ожидаемое время пересылки по каждому транзитному участку пути. Получив пакет, маршрутизатор проверяет, насколько пакет опаздывает или опережает график. Эта информация хранится в каждом пакете и обновляется каждым маршрутизатором. Если пакет приходит с опережением графика, он удерживается в течение требуемого интервала времени. Если же пакет запаздывает, маршрутизатор пытается отправить его дальше как можно быстрее.

В самом деле, алгоритм, определяющий, какие из пакетов отправить первыми по выходной линии, всегда может выбрать пакет, сильнее всего отстающий от расписания. При этом пакеты, опережающие график, замедляются, а опаздывающие пропускаются в первую очередь, что в обоих случаях уменьшает флуктуации времени доставки пакетов.

В некоторых приложениях, таких как видео по требованию, флуктуации могут быть снижены путем сохранения пакетов в буфере приемника с их последующей выдачей для отображения. При этом сеть не обязана работать в реальном масштабе времени. Тем не менее, в приложениях, которые должны обеспечивать межпользовательское взаимодействие в реальном времени (например, в интернет-телефонии или видеоконференциях), задержка, связанная с буферизацией, совершенно недопустима.

Борьба с перегрузками представляет собой область активных исследований. Текущее положение дел отражено в книге (Gevros и др., 2001).

Качество обслуживания

Методы, рассмотренные нами ранее, направлены на уменьшение перегрузок и повышение производительности в сетях передачи данных. Однако с ростом доли мультимедийной информации таких специализированных параметров оказывается недостаточно. Необходимо предпринимать серьезные попытки обеспечения гарантированного качества обслуживания сетей и улучшения протоколов. В следующих разделах мы продолжим изучение параметров производительности сетей, но больший упор сделаем на методах обеспечения высокого качества обслуживания, соответствующего требованиям конкретных приложений. Для начала следует отметить, что многие идеи пока еще не приобрели законченный вид и со временем могут измениться.

Требования

Последовательность пакетов, передающихся от источника к приемнику, называется **потоком**. При этом в сетях, ориентированных на соединение, все пакеты потока следуют по одному и тому же маршруту, а в сетях без установления соединения они могут идти разными путями. Каждому потоку требуются определенные условия, которые можно охарактеризовать следующими четырьмя основными параметрами: надежность, задержка, флуктуация и пропускная способность. Все вместе они формируют то, что называется **качеством обслуживания** (QoS — Quality of Service), необходимым потоку. Некоторые общие приложения особенно строго подходят к вопросу качества обслуживания, как показано в табл. 5.3.

Таблица 5.3. Строгие требования некоторых приложений к качеству обслуживания

Приложение	Надежность	Задержка	Флуктуации	Пропускная способность
Электронная почта	Высокая	Низкая	Слабые	Низкая
Передача файлов	Высокая	Низкая	Слабые	Средняя
Веб-доступ	Высокая	Средняя	Слабые	Средняя
Удаленный доступ	Высокая	Средняя	Средние	Низкая
Аудио по заказу	Низкая	Низкая	Сильные	Средняя
Видео по заказу	Низкая	Низкая	Сильные	Высокая
Телефония	Низкая	Высокая	Сильные	Низкая
Видеоконференции	Низкая	Высокая	Сильные	Высокая

Первые четыре приложения предъявляют высокие требования к надежности. Некорректная доставка битов должна быть исключена. Обычно это достигается подсчетом контрольной суммы для каждого пакета и ее проверкой у получателя. Если пакет во время передачи был испорчен, подтверждение о его доставке не высылается, и источник вынужден передавать его повторно. Такая стратегия обеспечивает высокую надежность. Четыре последних (аудио/видео) приложения весьма толерантны к ошибкам, поэтому здесь нет никаких вычислений и проверок контрольных сумм.

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 задержкой ровно 2,000 с, пользователи сочтут такую связь неприемлемой. С другой стороны, проигрывание видео- или аудиофайлов, хранящихся на сервере, допускает наличие некоторой задержки.

Первые три приложения спокойно отнесутся к неравномерной задержке доставки пакетов, а при организации удаленного доступа этот фактор имеет более важное значение, поскольку при сильных флуктуациях символы 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, то результат будет просто ужасен. При прослушивании звукозаписей будут заметны флуктуации даже в несколько миллисекунд.

Наконец, приложения могут иметь различные потребности в пропускной способности. Скажем, при передаче электронной почты или при удаленном доступе высокая пропускная способность не требуется, а вот для передачи видеоданных любых типов необходима высокая производительность сети.

В сетях ATM принята следующая классификация потоков по требованиям к качеству обслуживания:

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 через Интернет);
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.

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».

Методы достижения хорошего качества обслуживания

Итак, мы узнали кое-что о требованиях, входящих в понятие «качество обслуживания». Как же заставить систему удовлетворять этим требованиям? Во-первых, необходимо учесть, что не существует никакой волшебной палочки. Ни один метод не обеспечивает безусловно высокое качество обслуживания. Напротив, разработано большое количество методов, практические реализации которых зачастую используют смешанные технологии. Далее мы рассмотрим некоторые методы, применяемые разработчиками систем для повышения качества обслуживания.

Избыточное обеспечение

Проще всего обеспечить такую емкость маршрутизаторов, буферной памяти и такую пропускную способность, при которых пакеты без затруднений пролетали бы по сети. Проблема здесь одна: такое решение обходится очень дорого. Со временем разработчики начинают понимать, какие параметры являются необходимыми и достаточными, и тогда такой подход оправдывает себя. 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.

Буферизация

Потоки можно сохранять в буферной памяти на принимающей стороне перед тем, как доставлять потребителю. Буферизация не сказывается на надежности и пропускной способности, но сказывается на увеличении задержки. Зато с ее помощью можно снизить уровень флуктуаций. При передаче аудио и видео по требованию именно флуктуация представляет собой основную проблему, и буферизация помогает решить ее.

Мы уже видели различия характеристик сети при высокой и низкой флуктуации на рис. 5.26. На рис. 5.27 изображен поток пакетов, доставляемый при значительной флуктуации. Пакет 1 посылается с сервера в момент времени $t = 0$ с и прибывает в момент времени $t = 1$ с. Задержка пакета 2 составляет уже не 1, а 2 с. Прибывающие пакеты буферизируются на клиентской машине.

В момент времени $t = 10$ с начинается воспроизведение, при этом пакеты с 1-го по 6-й уже находятся в буфере, поэтому их можно оттуда извлекать через равные интервалы и воспроизводить. К сожалению, пакету 8 не повезло: он задержался настолько, что его невозможно воспроизвести вовремя. Выдача пакетов приостанавливается до его прибытия. Возникает досадная задержка в проигрывании музыки или фильма. Проблему можно решить увеличением задержки начала выдачи пакетов, но для этого потребуется буфер большей емкости. Все коммерческие веб-сайты, на которых содержится потоковое видео или аудио, используют проигрыватели, которые начинают воспроизведение только после примерно десятисекундной буферизации.

Рис. 5.27. Сглаживание выходного потока путем буферизации пакетов

Формирование трафика

В приведенном ранее примере источник выдает пакеты через фиксированные интервалы времени, однако бывает, что этот процесс не является столь равномерным. Это может приводить к перегрузке сети. Неравномерный выходной поток — это обычное дело для серверов, поддерживающих множество потоков и различные виды действий, таких как быстрая прокрутка вперед и назад, идентификация пользователя и т. д. Подход, описанный ранее (буферизация), не всегда можно применить (например, при видеоконференциях). Тем не менее, если бы удалось заставить серверы (и хосты в целом) передавать данные с предсказуемой скоростью, качество обслуживания было бы лучше. Рассмотрим метод **формирования трафика**, сглаживающий выходной трафик на стороне сервера, а не на стороне клиента.

При формировании трафика происходит регулирование средней и пиковой *скорости* передачи данных. Изучавшиеся нами ранее протоколы скользящего окна ограничивают количество данных, посылаемых сразу, но не скорость, с которой они посылаются. Когда устанавливается виртуальный канал, пользователь и подсеть (то есть клиент и оператор связи) договариваются об определенной схеме (то есть форме) трафика для данного канала. Иногда это действие называется **соглашением об уровне обслуживания**. До тех пор пока клиент выполняет свою часть условий соглашения и посылает пакеты не чаще оговоренного в договоре графика, оператор связи обязуется доставлять их в определенный срок. Формирование трафика снижает перегрузку и, таким образом, помогает оператору связи выполнять свои обязательства. Подобные договоренности не столь важны при передаче файлов, но весьма существенны при передаче данных в режиме реального времени, как, например, для аудио- и видеосвязи, которые плохо переносят перегрузку.

В результате при использовании метода формирования трафика клиент сообщает оператору связи: «Мой график передачи будет выглядеть следующим образом. Сможете ли вы это обеспечить?». Если оператор связи соглашается, то возникает вопрос о том, как оператор связи будет сообщать клиенту, что тот соблюдает соглашение, и что делать, если клиент нарушит договор. Наблюдение за потоком трафика называется **политикой трафика**. Договориться о форме трафика и регулировать его впоследствии легче в подсетях с виртуальными каналами, чем в дейтаграммных подсетях. Тем не менее даже в дейтаграммных подсетях можно применить те же идеи к соединениям транспортного уровня.

Алгоритм дырявого ведра

Представьте себе ведро с маленькой дырочкой в днище, как показано на рис. 5.28, *а*. Независимо от скорости, с которой вода наливается в ведро, выходной поток обладает постоянной скоростью, когда в ведре есть вода, и нулевой скоростью, когда ведро пустое. Кроме того, когда ведро наполняется, вся лишняя вода выливается через край и теряется (то есть не попадает в выходной поток под дырочкой).

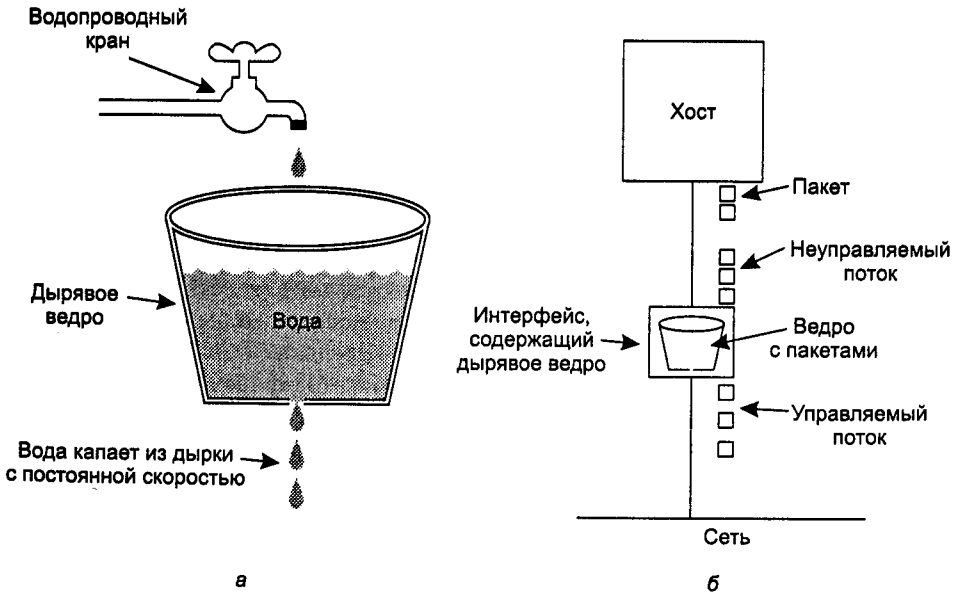


Рис. 5.28. Дырявое ведро с водой (*а*); дырявое ведро с пакетами (*б*)

Та же самая идея применима к пакетам, как показано на рис. 5.28, *б*. Принцип таков: каждый хост соединяется с сетью через интерфейс, содержащий дырявое ведро, то есть конечную внутреннюю очередь. Если пакет появляется в очереди, когда очередь полная, пакет игнорируется. Другими словами, если несколько процессов хоста пытаются послать пакеты, когда в очереди уже стоит максимально допустимое число пакетов, новый пакет игнорируется. Такой интерфейс может быть реализован как аппаратно, так и программно операционной системой хоста. Он был предложен Тернером (Turner, 1986) и называется **алгоритмом дырявого ведра**. По сути это не что иное, как однолинейная система массового обслуживания с постоянным временем обслуживания.

Хосту разрешается посылать в сеть один пакет за один такт. Опять же, это может быть реализовано интерфейсной картой либо операционной системой. Этот механизм преобразует неравномерный поток пакетов от процессов пользователя в равномерный поток пакетов в сети, сглаживая пики и значительно снижая вероятность перегрузки.

Когда размер всех пакетов одинаков (например, в ячейках ATM), этот алгоритм может применяться, как описано ранее. Однако при использовании пакетов переменного размера часто бывает лучше ограничивать количество байтов, переданных в сеть за такт, нежели передавать один пакет за такт. Так, если правилом установлена передача 1024 байт за тактовый интервал, то за этот период могут быть переданы в сеть либо один пакет размером 1024 байта, либо два пакета по 512 байт, либо четыре пакета по 256 байт и т. д. Если оставшееся количество байт меньше размера следующего пакета, следующий пакет должен ждать начала следующего такта.

Реализация исходного алгоритма дырявого ведра проста. Дырявое ведро состоит из конечной очереди. Когда прибывает пакет и в очереди есть место, пакет добавляется к очереди, в противном случае пакет игнорируется. Если очередь не пуста, то в течение каждого тактового интервала в сеть передается по одному пакету.

Алгоритм дырявого ведра со счетчиком байтов реализуется почти также. В каждом тактовом интервале значение счетчика устанавливается равным n. Если размер первого пакета в очереди меньше текущего значения счетчика, он передается, а значение счетчика уменьшается на его размер. Если значение счетчика еще достаточно велико, могут быть посланы и другие пакеты. Когда значение счетчика становится меньше размера следующего пакета в очереди, передача прекращается до следующего такта, после чего все начинается сначала, а остаток счетчика обнуляется.

В качестве примера представьте, что компьютер может производить данные со скоростью 25 млн байт в секунду (200 Мбит/с) и что сеть также работает на этой скорости. Однако маршрутизаторы могут поддерживать эту скорость передачи данных лишь на коротких интервалах (пока не заполнится их буферная память). В течение больших интервалов времени они могут обеспечить не более 2 млн байт в секунду. Теперь предположим, что данные поступают пачками по 1 млн байт, одна пачка продолжительностью 40 мс в каждую секунду. Чтобы уменьшить среднюю скорость до 2 Мбайт/с, можно воспользоваться алгоритмом дырявого ведра с выходной скоростью $\rho = 2$ Мбайт/с и емкостью $C = 1$ Мбайт. Это означает, что пачки до 1 Мбайта могут обрабатываться без потерь и что такие пачки будут передаваться в сеть за 500 мс независимо от того, как быстро они приходят.

На рис. 5.29, *а* показан вход дырявого ведра, на который со скоростью 25 Мбайт/с поступает пачка в течение 40 мс. На рис. 5.29, *б* показан выход, через который данные проходят с постоянной скоростью 2 Мбайт/с в течение 500 мс.

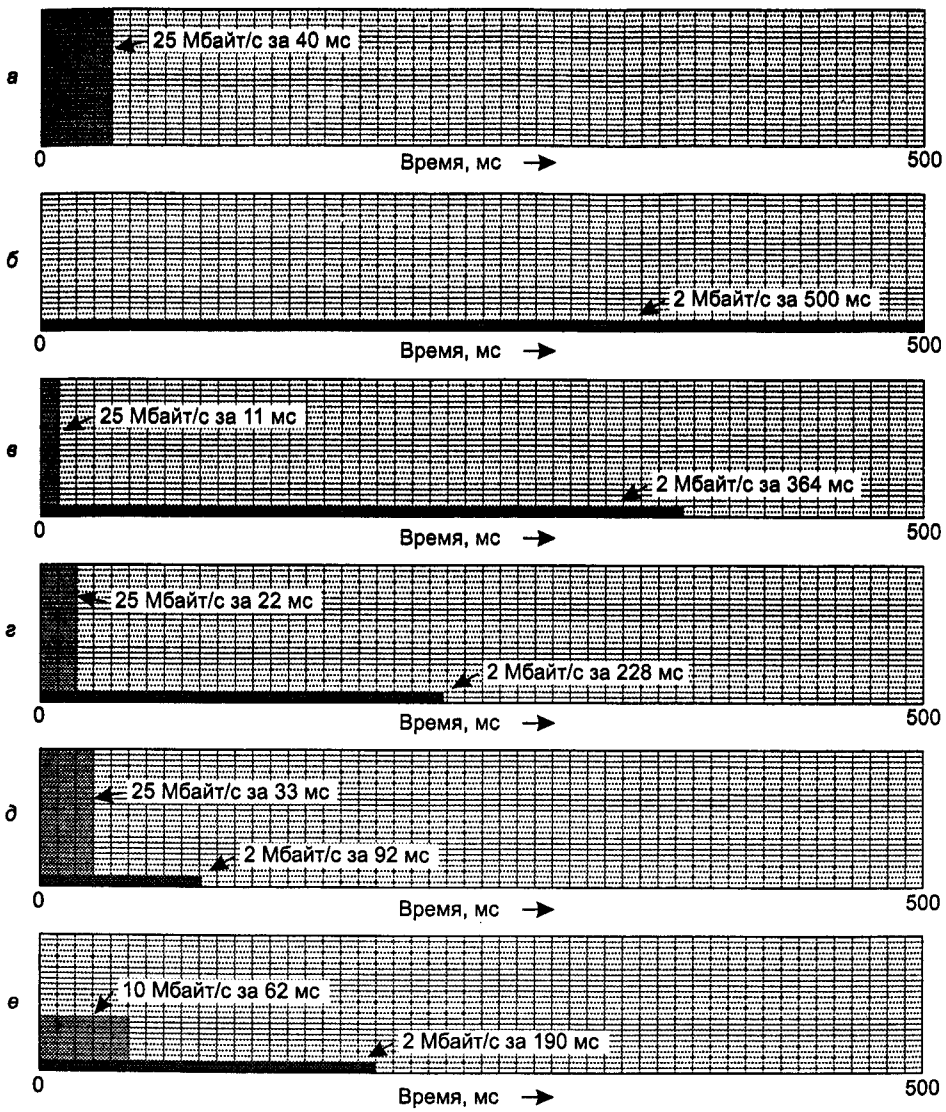


Рис. 5.29. Вход дырявого ведра (*а*); выход дырявого ведра (*б*); выход маркерного ведра емкостью 250 Кбайт (*в*), 500 Кбайт (*г*) и 750 Кбайт (*д*); выход маркерного ведра емкостью 500 Кбайт на входе дырявого ведра со скоростью протекания 10 Мбайт/с (*е*)

Алгоритм маркерного ведра

Алгоритм дырявого ведра формирует строгий выходной поток с постоянной скоростью, не зависящей от неравномерности входного потока. Для многих приложений было бы лучше при поступлении больших пакетов данных немного увеличивать выходную скорость. Таким образом можно было бы попытаться создать более гибкий алгоритм, желательно, не теряющий данные. Одним из таких алгоритмов является **алгоритм маркерного ведра**. В этом алгоритме ведро содержит маркеры, создаваемые через равные интервалы времени ΔT секунд. На рис. 5.30, *а* изображено ведро с тремя маркерами и пятью пакетами, стоящими в очереди. Чтобы передать один пакет, требуется удалить один маркер. На рис. 5.30, *б* мы видим, что три из пяти пакетов прошли в сеть, а оставшиеся два пакета остались ждать двух новых маркеров.

Алгоритм маркерного ведра формирует трафик не так, как алгоритм дырявого ведра. Алгоритм дырявого ведра не позволяет простаивающим хостам запасаться впрок разрешениями на передачу больших пакетов. Алгоритм маркерного ведра разрешает запасаться маркерами до определенного размера ведра n. Это свойство означает, что пачки (пакеты) с величиной до n могут быть переданы в сеть сразу, что создает некоторую неравномерность в выходном потоке, но обеспечивает быструю реакцию на неожиданные входные пачки.

Еще одно различие двух алгоритмов заключается в том, что при переполнении маркерного ведра алгоритм игнорирует маркеры, но никогда не отвергает пакеты. Алгоритм дырявого ведра, напротив, при переполнении выбрасывает сами пакеты.

Возможен вариант алгоритма, при котором маркер может предоставлять право пересылать не один пакет, а k байт. Пакет пересылается только при наличии достаточного числа маркеров, чтобы покрыть его длину. Лишние маркеры сохраняются для будущего использования.

Алгоритмы дырявого и маркерного ведра могут использоваться не только для регулирования выхода хостов, но и для сглаживания трафика между маршрутизаторами. А различаются эти два алгоритма тем, что применение алгоритма маркерного ведра может заставить маршрутизатор остановить передачу пакетов, что может привести к потере данных.

Реализация исходного алгоритма маркерного ведра подразумевает наличие переменной, считающей маркеры. Счетчик увеличивается на единицу через равные интервалы времени ΔT и уменьшается при посылке пакета. Когда счетчик уменьшается до нуля, передача пакетов останавливается. В варианте с учетом количества переданных байт счетчик увеличивается на k байт каждые ΔT секунд и уменьшается на размер переданного пакета.

Суть алгоритма маркерного ведра состоит в том, что он допускает передачу данных пачками, но ограничивает длительность пачки. Для примера рассмотрим рис. 5.29, *в*, на котором изображено маркерное ведро емкостью 250 Кбайт. Маркеры появляются с частотой, соответствующей выходной скорости 2 Мбайт/с. Предположим, что маркерное ведро заполнено, когда прибывает пакет данных размером 1 Мбайт. Ведро может быть освобождено с максимальной скоростью 25 Мбайт/с примерно за 11 мс. Затем оно должно уменьшить скорость передачи до 2 Мбайт/с, пока не будет передан весь входной пакет данных.

При расчете длительности выходной пачки (на максимальной скорости) нужно учитывать, что пока ведро опорожняется, появляются новые маркеры. При длительности пачки S секунд, емкости маркерного ведра C байт, скорости появления маркеров ρ байт/с, и максимальной выходной скорости M байт/с очевидно, что максимальное количество переданных байтов в пачке будет равно $C + \rho S$ байт.

Мы также знаем, что количество байтов, переданных в пачке с максимальной скоростью, равно MS. Таким образом,

$$C + \rho S = MS.$$

Решив это уравнение, получим: $S = C/(M - \rho)$. При наших параметрах $C = 250$ Кбайт, $M = 25$ Мбайт/с и $\rho = 2$ Мбайт/с получаем длительность пачки около 11 мс. На рис. 5.29, *г* и *д*, показаны маркерные ведра емкостью 500 и 750 Кбайт соответственно.

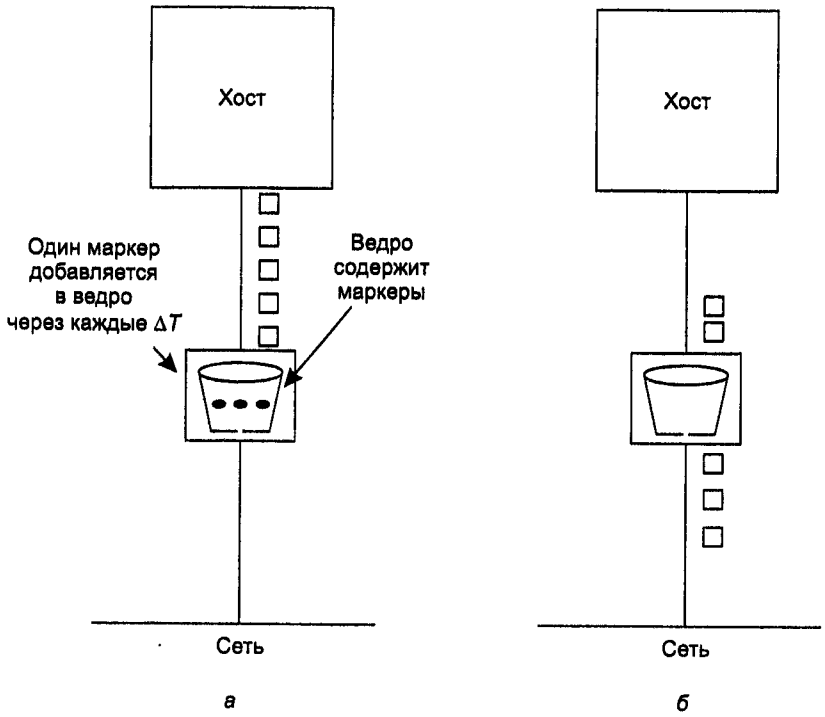


Рис. 5.30. Алгоритм маркерного ведра: до (*а*) и после (*б*)

Недостатком алгоритма маркерного ведра является слишком большая скорость передачи данных при опустошении ведра, несмотря на то что длительность пачки можно регулировать тщательным подбором ρ и M. Часто бывает желательно уменьшить пиковую скорость, не возвращаясь при этом к скорости алгоритма дырявого ведра.

Один из способов получения более гладкого трафика состоит в помещении дырявого ведра после маркерного ведра. Скорость дырявого ведра должна быть выше минимальной скорости маркерного ведра ρ, но ниже максимальной скорости сети. На рис. 5.29, *е* показан выходной поток маркерного ведра емкостью 500 Кбайт, за которым установлено дырявое ведро со скоростью протекания, равной 10 Мбайт/с.

Управление такими схемами может оказаться непростым. По сути, сеть должна имитировать алгоритм и гарантировать, 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 к качеству обслуживания.

Резервирование ресурсов

Возможность управления трафиком — это неплохой начальный шаг в деле обеспечения гарантированного качества обслуживания. Однако на самом деле использование этих методов неявно означает, что все пакеты в потоке должны следовать 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.

Раз у нас есть особый путь, по которому направляется поток, становится возможным резервирование ресурсов вдоль этого пути, что позволяет гарантировать доступность необходимой емкости. Резервироваться могут три типа ресурсов:

1. Пропускная способность.
2. Буферное пространство.
3. Время центрального процессора.

Наиболее очевидно резервирование пропускной способности. Если потоку необходима скорость 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 обычно оседает на сетевой интерфейсной карте (это действие управляется аппаратно). Затем программному обеспечению маршрутизатора необходимо скопировать пакет в буфер оперативной памяти и поставить содержимое этого буфера в очередь на отправку по выбранной исходящей линии. 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 — это еще один очень ценный ресурс. На что расходуется время работы процессора в маршрутизаторе? На обработку пакетов. Поэтому существует предельная скорость, с которой маршрутизатор может обрабатывать пакеты. Необходимо быть уверенным в том, что процессор не перегружен, — это залог своевременной обработки каждого пакета.

На первый взгляд кажется, что если на обработку пакета уходит, скажем, 1 мкс, то маршрутизатор способен управиться с миллионом пакетов за секунду. Однако это предположение ошибочно, так как при доставке потока всегда есть промежутки времени, в течение которых ничего не передается. Если центральному процессору для совершения своей работы важен каждый отдельный такт, то пропуск нескольких тактов из-за молчания на линии приведет к накоплению невыполненных заказов, от которых невозможно избавиться.

Однако даже если нагрузка несколько меньше теоретической емкости, все равно могут образовываться очереди и возникать задержки. Рассмотрим ситуацию, когда пакеты прибывают нерегулярно со средней скоростью прибытия λ пакетов в секунду. Время, необходимое процессору на обработку каждого пакета, также меняется, но в среднем составляет μ пакетов в секунду. Предположим, что как скорость прибытия, так и скорость обслуживания имеют пуассоновское распределение. Тогда, используя теорию массового обслуживания, можно доказать, что средняя задержка T, присущая пакету, составляет

$$T = \frac{1}{\mu} \times \frac{1}{1-\lambda/\mu} = \frac{1}{\mu} \times \frac{1}{1-\rho}$$

где $\rho = \lambda/\mu$ — коэффициент использования центрального процессора. Первый сомножитель $1/\mu$ — это задержка при отсутствии конкуренции. Второй сомножитель представляет собой дополнительную задержку, возникающую в результате конкурентной борьбы с другими потоками. Например, если $\lambda = 950\,000$ пакетов/с, а $\mu = 1\,000\,000$ пакетов/с, тогда $\rho = 0{,}95$, 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.

Управление доступом

Итак, в результате проведенной работы мы получили входящий трафик в виде хорошо сформированного и, возможно, следующего по единому маршруту потока. На пути потока можно заранее резервировать ресурсы. Когда маршрутизатору предлагается обработать такой поток, он может принять или отвергнуть его, обосновывая свое решение доступной емкостью и количеством уже находящихся в обработке потоков.

Процесс принятия решения об обработке или игнорировании потока сложнее, нежели простое сравнение запрашиваемых потоком параметров (пропускной способности, буферной памяти, времени центрального процессора) с имеющимися. Во-первых, хотя многие приложения и знают свои требования к 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. Далее, приложения весьма различаются по толерантности в отношении пропущенного предельного срока обработки. 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**. 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 снижение трафика — никто не станет сознательно брать на себя больше работы, чем требует заказчик (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, базирующийся на RFC 2210 и RFC 2211 (табл. 5.4). В спецификации содержится пять параметров, первый из которых, *Скорость маркерного ведра*, хранит число байтов, поступающих в «ведро» за секунду. Это максимум, который отправитель может поддерживать в течение длительного времени, усредненный по большому временному отрезку.

Таблица 5.4. 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

Второй параметр — размер маркерного ведра в байтах. Если, к примеру, *Скорость маркерного ведра* составляет 1 Мбит/с, а размер ведра равен 500 Кбайт, то его можно будет наполнять данными в течение 4 с. Все, что будет посылаться после этого, будет теряться.

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 важен, 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 по 50 байт в секунду 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? Это отображение является специфическим и не стандартизованным действием. Допустим, маршрутизатор может обрабатывать 100 000 пакетов/с. Если ему предлагается пропустить через себя поток со скоростью 1 Мбайт/с с максимальным размером пакета, составляющим 512 байт, он может легко посчитать, что такой поток дает 2048 пакетов/с, значит, под него необходимо отвести 2 % времени работы процессора, а лучше немного больше, чтобы избежать больших задержек обслуживания. Если политика маршрутизатора не позволяет ему резервировать более 50 % процессорного времени (что подразумевает половинную задержку) и если 49 % уже зарезервировано, то поток будет отвергнут. Подобные вычисления необходимо производить для всех резервируемых ресурсов.

Чем строже спецификация потока, тем лучше для маршрутизаторов. Если же в спецификации говорится, что *Скорость маркерного ведра* составляет 5 Мбайт/с, однако пакеты могут быть размером от 50 до 1500 байт, значит, скорость передачи пакетов может колебаться от 3500 до 105 000 пакетов/с. Маршрутизатор, ужаснувшись при виде последнего числа, может отвергнуть такой поток. При минимальном размере пакета, равном 1000 байт, 5-мегабайтный поток тем же самым маршрутизатором мог бы быть принят.

Пропорциональная маршрутизация

Большинство алгоритмов маршрутизации пытаются искать наилучшие пути для каждого адресата и направлять весь трафик по оптимальному пути. Альтернативный подход, позволяющий повысить качество обслуживания, состоит в разделении трафика для одного и того же адресата между несколькими маршрутами. Поскольку маршрутизаторы обычно не следят за нагрузкой на всю сеть в целом, остается лишь один способ разделения трафика — на основе доступной локальной информации. Одним из простых методов является маршрутизация, пропорциональная или эквивалентная емкостям исходящих связей. Однако существуют и более сложные алгоритмы (Nelakuditi и Zhang, 2002).

Диспетчеризация пакетов

Если маршрутизатор имеет поддержку нескольких потоков, существует опасность того, что один из них захватит слишком большую часть пропускной способности и не даст жить всем остальным потокам. Обработка пакетов в порядке поступления может привести к тому, что агрессивный источник загрузит все производственные мощности маршрутизаторов, через которые проходит его поток, и тем самым снизит качество обслуживания других источников. Для пресечения

подобных попыток были разработаны алгоритмы диспетчеризации пакетов (Bhatti и Crowcroft, 2000).

Одним из первых был алгоритм **справедливого обслуживания** (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, 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. Агрессивному хосту не поможет то, что в его очереди стоит больше пакетов, чем у остальных.

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. То есть очереди сканируются побайтно до того момента, пока не будет выхвачен последний байт последнего пакета. После этого пакеты отправляются в том порядке, в котором они заканчивались при опросе очередей. Алгоритм проиллюстрирован на рис. 5.31.

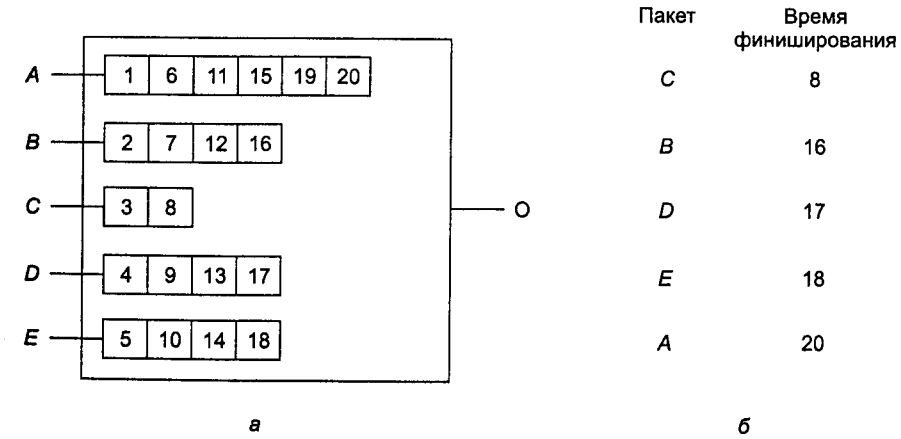


Рис. 5.31. Маршрутизатор с пятью очередями пакетов для линии O (*а*); время окончания сканирования для пяти пакетов (*б*)

На рис. 5.31, *а* мы видим пакеты длиной от 2 до 6 байт. Во время (виртуального) первого такта извлекается первый байт пакета с линии *A*. Затем следует первый байт пакета с линии *B*, и т. д. Первым, через 8 тактов, закончится пакет *C*. Порядок сортировки пакетов приведен на рисунке справа (рис. 5.31, *б*). При отсутствии новых поступлений пакеты будут отсылаться в указанном порядке, начиная с *C* и заканчивая *A*.

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 такт опроса. Такая модификация алгоритма называется **взвешенным справедливым обслуживанием**. Иногда весовой коэффициент эквивалентен числу потоков, генерируемых машиной, таким образом все процессы получают равные доли пропускной способности. Одна из эффективных реализаций данного алгоритма описывается в (Shreedhar и Varghese, 1995). Все чаще и чаще встречаются аппаратные реализации передачи пакетов через маршрутизаторы или коммутаторы (Elhanany и др., 2001).

Интегральное обслуживание

В 1995–1997 годы 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 2205–2210. Общее название этих трудов — **потоковые алгоритмы** или **интегральное обслуживание**. 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. Далее мы более подробно рассмотрим многоадресную рассылку, поскольку однонадресная 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

Главный протокол архитектуры интегрального обслуживания, разработанный IETF, называется **протоколом резервирования ресурсов** (RSVP — Resource reSerVation Protocol). Он описывается в документе RFC 2205 и других документах-протоколах. 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. 5.32, *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. 5.32, *б* и *в* соответственно.

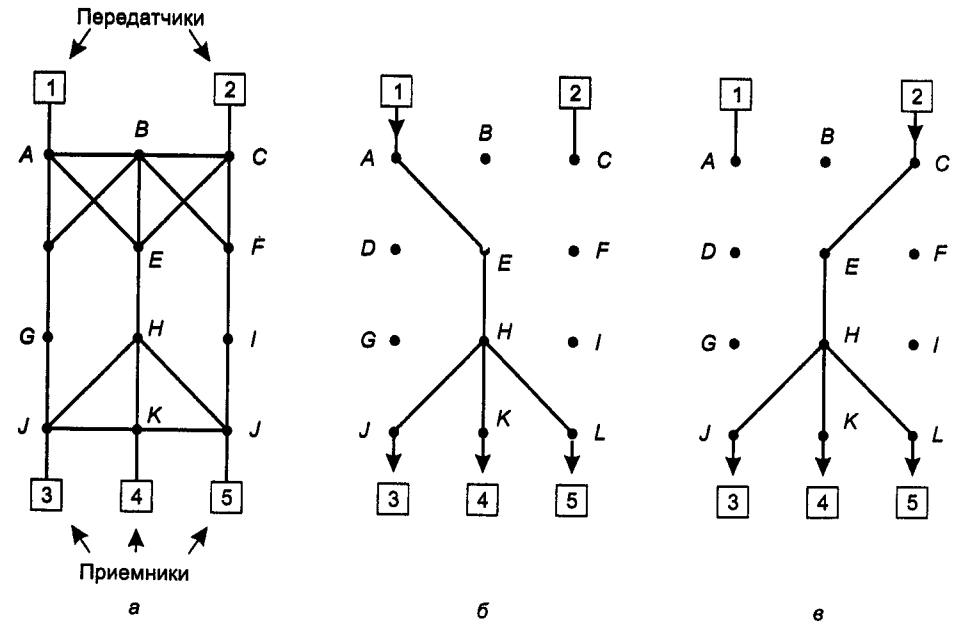


Рис. 5.32. Сеть (*а*); связующее дерево многоадресной рассылки для хоста 1 (*б*); связующее дерево многоадресной рассылки для хоста 2 (*в*)

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. 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.