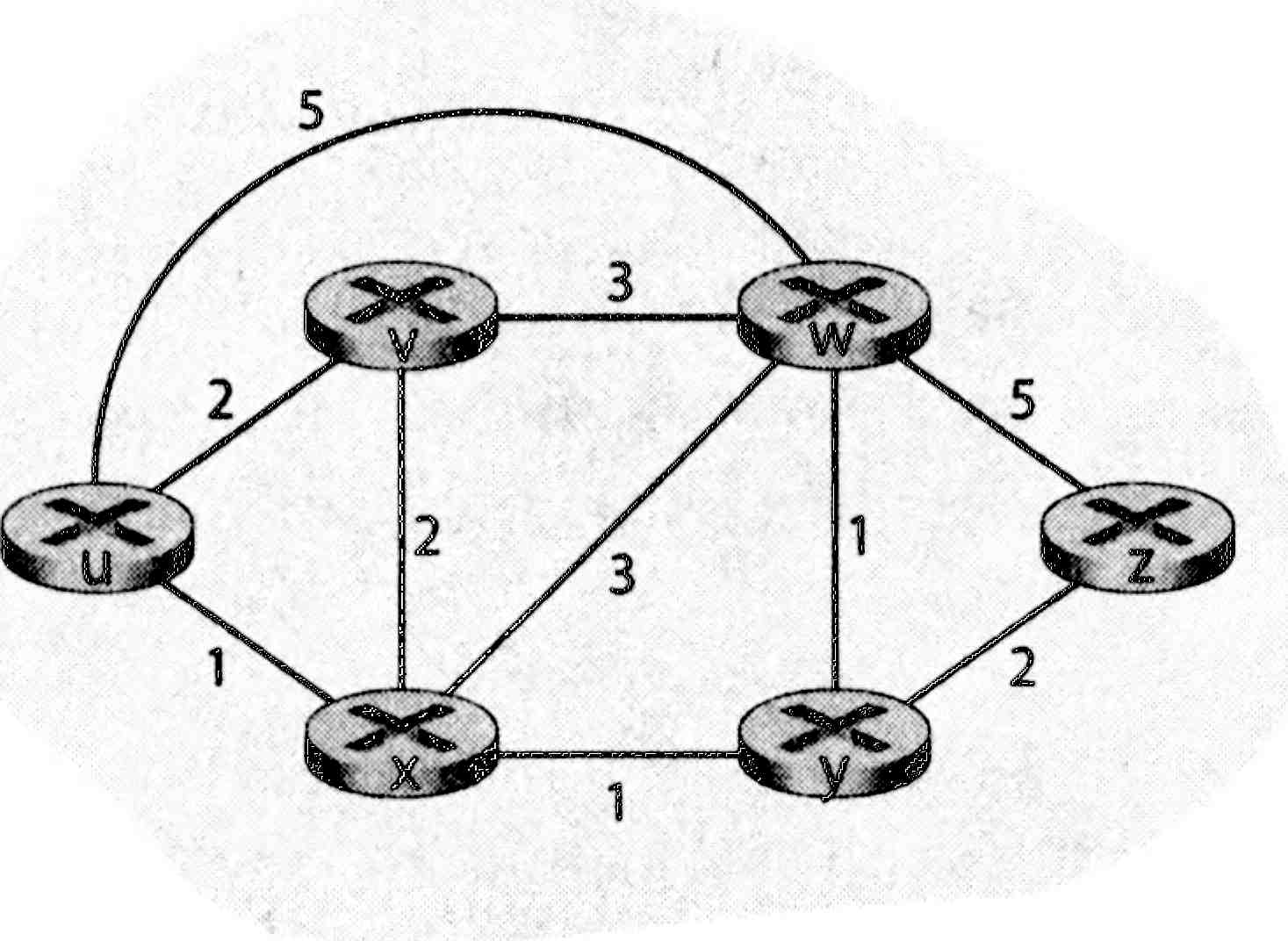
**Алгоритмы маршрутизации**

Ранее мы обсуждали в основном только одну функцию сетевого уровня — перенаправление (переадресацию) данных. Мы узна­ли, что, когда пакет прибывает на маршрутизатор, тот просматривает свою таблицу перенаправления и определяет канальный интерфейс, на который следует послать пакет. Мы также изучили, что алгоритмы маршрутизации, действующие на сетевых маршрутизаторах, обеспе­чивают обмен информацией, нужной для формирования, заполнения и настройки их таблиц перенаправления. Взаимодействие между алго­ритмами маршрутизации и таблицами перенаправления было проде­монстрировано на рис. 4.2. Изучив таблицу маршрутизации достаточ­но подробно, мы перейдем к обсуждению еще одной важной темы, а именно поговорим о функции маршрутизации сетевого уровня, имеющей огромное значение. Независимо от того, реализует ли сетевой уровень дейтаграммный принцип передачи (в таком случае различные пакеты могут следовать по разным маршрутам между конкретной па­рой хостов — исходным и конечным), либо работает с виртуальными каналами (тогда все пакеты между исходным хостом и хостом назначе­ния пойдут по одному и тому же пути), именно сетевой уровень должен определять путь каждого пакета от источника к получателю. Как будет показано ниже, задача маршрутизации — определять хорошие пути (то есть, маршруты) от отправителя к получателю, прокладывая их через сеть маршрутизаторов.

Как правило, хост напрямую подключается к одному маршрутизато­ру, который называется **маршрутизатором по умолчанию** или **первым маршрутизатором на пути доставки.** Маршрутизатор по умолчанию хоста-отправителя мы будем называть **начальным маршрутизатором,** а маршрутизатор по умолчанию хоста-получателя — **конечным марш­рутизатором.** Разумеется, задача маршрутизации пакета от хоста-отправителя к хосту-получателю сводится к пересылке пакета от началь­ного маршрутизатора к конечному, о чем и пойдет речь ниже.

Соответственно, цель алгоритма маршрутизации очень проста: имея набор маршрутизаторов, которые соединены каналами, он находит оптимальный путь от начального маршрутизатора к конечному. В про­стейшем случае оптимальный путь — это наименее затратный. Но на практике приходится учитывать и другие факторы, в частности, детали политики конфиденциальности (например, такое правило: «маршрути­затор x, относящийся к организации Y, не должен пересылать какие-либо пакеты, исходящие из сети организации Z»). Это значительно усложня­ет концептуально простые и красивые алгоритмы, теория которых слу­жит основой практики маршрутизации в современных сетях.

Для формулирования проблем маршрутизации используется граф. Как вы помните, граф*G* = *(N,E) —* это множество из *N* узлов и набор *Е* ребер, где каждое ребро есть пара узлов из *N.* В контексте маршрутиза­ции на сетевом уровне узлы графа соответствуют маршрутизаторам — тем точкам, в которых принимаются решения о перенаправлении паке­та по тому или иному курсу. Ребра между этими узлами представляют физические каналы между данными маршрутизаторами. Подобное аб­страктное представление компьютерной сети в виде графа представлено на рис. 4.27.



**Рис. 4.27. Абстрактная модель компьютерной сети в виде графа**

Как показано на рис. 4.27, ребро также имеет значение, характери­зующее стоимость этого ребра. Как правило, она отражает физическую длину соответствующего канала в сети (например, трансатлантический канал будет иметь более высокую стоимость, чем короткий наземный канал), скорость передачи данных в этом канале или денежную стои­мость передачи информации по нему. В нашем примере мы просто возь­мем готовые значения стоимости ребер, не вдаваясь в подробности их расчета. Для каждого ребра *(х,у)* в *Е* обозначим через *с(х,у)* стоимость ребра между узлами *х* и *у.* Если пара *(х,у)* не относится к £, то *с(х,у)* = ∞. Кроме того, в этом примере мы будем рассматривать лишь неориенти­рованные графы — то есть такие, ребра которых не имеют направлений.

Таким образом, ребро (х,у) равноценно ребру (y,x), a c(y,x) = ∞. Кроме того, узел у называется соседним (смежным) узлу х, если (х,у) принад­лежит к Е.

Учитывая, что при представлении сети в виде графа его ребрам при­сваиваются различные значения стоимости, естественная цель алгорит­ма маршрутизации — найти наименее дорогостоящий путь между исхо­дным и конечным хостом. Чтобы точнее сформулировать эту проблему, оговоримся, что путьв графе *G* = *(N,E) —* это такая последовательность узлов *(xv xv... x* ), где каждая из пар *(xv х2), (х2, х3)>*..., *(х' х* ), является ребром в *Е.* Стоимость пути *(xv* х,,... *х) —* это просто сумма стоимостей всех ребер в его составе, то есть, *c(xv х2)* + *с(хг х3)* +... + *с(х v x* ). Между любыми двумя узлами *х* и ***у*** как правило пролегает много путей, каждый из которых имеет свою стоимость. Один или несколько из них являют­ся **путями наименьшей стоимости** (в сетевой терминологии пути также именуются **трактами).** Соответственно, проблема наименьшей стоимо­сти формулируется очень просто: найти между начальной и конечной точкой такой путь, который обладает самой низкой стоимостью. Так, на рис. 4.27 путь с наименьшей стоимостью между начальным узлом *и* и конечным узлом *w —* это путь *(u,* *х, у, w)* со стоимостью 3. Обратите внимание: если все ребра в графе обладают одинаковой стоимостью, то путь с самой низкой в то же время является и **кратчайшим** (то есть он содержит наименьшее количество звеньев между начальной и конечной точками).

В качестве простого упражнения попробуйте найти на рис. 4.27 путь наименьшей стоимости между узлами *и* и г, а потом задумайтесь, как вы вычислили его. Если вы мыслите как большинство из нас, то поступи­те следующим образом: рассмотрите рис. 4.27, проведете несколько пу­тей между *и* и *z*, а потом логически заключите, что один из них обладает меньшей стоимостью, чем остальные. (Кстати, вы рассмотрели все воз­можные 17 путей между *u* и *z*? Вероятно, нет.) Такое вычисление — при­мер централизованного алгоритма маршрутизации — такого, который вы целиком прогнали у себя в голове, располагая полной информацией о сети. В широком смысле все алгоритмы маршрутизации можно клас­сифицировать по такому признаку: являются ли они глобальными или децентрализованными.

• **Глобальный алгоритм маршрутизации** вычисляет наименее затрат­ный путь между исходной и конечной точкой, опираясь на исчер­пывающие сведения обо всей структуре сети. Соответственно, на вход такого алгоритма поступает информация о соединениях между всеми узлами и о стоимости всех каналов в сети. В таком случае не­обходимо, чтобы алгоритм каким-то образом получил всю эту ин­формацию перед тем, как выполнять вычисления. Сами вычисления могут выполняться в одном месте (централизованный глобальный алгоритм маршрутизации), либо воспроизводиться во множестве мест. Основная характерная черта в данном случае такова: глобаль­ный алгоритм располагает полной информацией о соединениях и их стоимости. На практике алгоритмы с такой глобальной информаци­ей зачастую именуются **алгоритмами, учитывающими состояние каналов** (link-state algorithm, LS), так как должны обладать инфор­мацией о состоянии каждого канала в сети.

• **В децентрализованном алгоритме маршрутизации** расчет пути с наименьшей стоимостью выполняется итерационным распределен­ным образом. Ни один узел не обладает полной информацией о стои­мости всех каналов в сети. Напротив, каждому узлу в начале работы известно о стоимости лишь тех каналов, которые связаны непосред­ственно с ним. Затем запускается итерационный процесс вычисле­ния пути и обмена информацией с соседними узлами (то есть теми, что расположены на других концах каналов, связанных с конкрет­ным узлом). Узел постепенно высчитывает путь с наименьшей стои­мостью к одному или нескольким узлам назначения. Децентрализо­ванный алгоритм маршрутизации называется **дистанционно-векторным** (distance-vector, DV). Дело в том, что каждый узел поддержива­ет вектор оценок стоимости (расстояний) до всех остальных узлов в данной сети.

Еще один способ общей классификации алгоритмов маршрутиза­ции — характеристика их по признаку статичности или динамичности. В **статических алгоритмах маршрутизации** маршруты со временем ме­няются очень медленно, зачастую в результате человеческого вмешатель­ства (например, если администратор вручную отредактирует таблицу перенаправления в маршрутизаторе). **Динамические алгоритмы марш­рутизации** изменяют пути в зависимости от активности сетевого трафи­ка или при корректировках топологии сети. Динамический алгоритм мо­жет выполняться либо периодически, либо непосредственно в ответ на изменение топологии сети или стоимости каналов. Хотя динамические алгоритмы лучше адаптируются к изменениям в сети, они также более подвержены проблемам, которые могут возникать в маршрутизаторах — например, к появлению петель или осцилляции маршрутов.

Третий подход к классификации алгоритмов маршрутизации — раз­личение их по признаку чувствительности или нечувствительности к нагрузкам. В **алгоритме, чувствительном к нагрузкам,** стоимость ка­налов динамически варьируется в зависимости от наличия и величины перегрузки в конкретном канале. Если стоимость каналов, которые сей­час перегружены, повышается, то алгоритм маршрутизации будет вы­бирать пути, позволяющие обойти такой канал. Хотя ранние алгоритмы маршрутизации, действовавшие в сети ARPAnet, были чувствительны к нагрузкам, с ними возникал ряд сложностей. Современные алго­ритмы маршрутизации, действующие в Интернете (в частности, RIP, OSPF и BGP), **нечувствительны к нагрузкам.** Это означает, что стои­мость капала не находится в прямой зависимости от того, насколько он загружен.

**Алгоритм маршрутизации, учитывающий состояние каналов**

Как вы помните, при применении алгоритма маршрутизации, учи­тывающего состояние каналов, известны и топология сети, и стоимость каждого отдельного канала. То есть для такого алгоритма все эти све­дения служат входными данными. На практике такой алгоритм реали­зуется на каждом узле путем широковещательной передачи пакетов, учитывающих состояние канала, *всем* другим узлам в сети. В данном случае каждый такой пакет с информацией о состоянии каналов со­держит идентификаторы и значения стоимости тех каналов, которые связаны с узлом — отправителем пакета. На практике (например, при работе с Интернет-протоколом маршрутизации OSPF) эта задача зачастую решается при помощи **широковещательного** алгоритма **с учетом состояния канала** (link-state broadcast, LS). В результате широковещательной работы узлов все они получают идентичное и полное представление о сети. После этого любой узел мо­жет выполнить алгоритм с учетом состояния каналов и рассчитать та­кой же путь с наименьшей стоимостью.

Проиллюстрированный ниже алгоритм маршрутизации с учетом со­стояния каналов также известен под названием *«алгоритм Дейкстры».* Он очень похож на алгоритм Прима; см. работу Кормена, где под­робно обсуждаются алгоритмы графов. Алгоритм Дейкстры вычисляет путь с наименьшей стоимостью от одного узла (исходного, обозначим его *и)* ко всем другим узлам в сети. Алгоритм является итерационным и обладает следующим свойством: после *k*-той итерации пути с наи­меньшей стоимостью становятся известны *k* узлам назначения, а среди всех путей с наименьшей стоимостью, ведущих к узлам назначения, эти *k* путей будут иметь *k* наименьших значений стоимости. Давайте опре­делим следующие условные обозначения:

* *D(v):* цена пути с наименьшей стоимостью от исходного узла к точке назначения v по состоянию на данную итерацию алгоритма.
* *p(v):* предыдущий узел (соседний с *v),* расположенный вдоль теку­щего пути с наименьшей стоимостью от исходного узла к *v.*
* N`: подмножество узлов. Узел *v* находится на расстоянии *N`* если точно известен путь с наименьшей стоимостью от исходного узла до *v.*

Глобальный алгоритм маршрутизации состоит из шага инициализа­ции и следующего за ним цикла. Количество выполнений этого цикла равно количеству узлов в сети. К моменту завершения работы алгоритм рассчитает кратчайшие пути от исходного узла *и* до любого другого узла в сети.

**Алгоритм с учетом состояния каналов для исходного узла** *и*

1. **Инициализация:**
2. N' = {и}
3. для всех узлов v

4 if узел v является соседним с и

5 then D(v) = c(u,v)

6 else D(v)= ∞  
7

1. **Цикл**
2. найти узел w не относящийся к N', чтобы значение D(w) было минимальным
3. добавить w к N'
4. обновить D(v) для каждого v, соседнего для w и не относящегося к N':

12 D(v) = min( D(v), D(w) + c(w,v) )

1. /\* новая стоимость для v есть либо старая стоимость v либо известная
2. стоимость кратчайшего пути к w плюс стоимость пути от w до v\*/
3. **пока N'= N**

Возьмем в качестве примера сеть с рис. 4.27 и определим пути с наи­меньшей стоимостью от узла *и* ко всем возможным узлам назначения. В табл. 4.3 в общем виде представлено вычисление этого алгоритма. В каждой строке таблицы указаны значения переменных алгоритма в кон­це итерации. Давайте подробно рассмотрим несколько первых этапов.

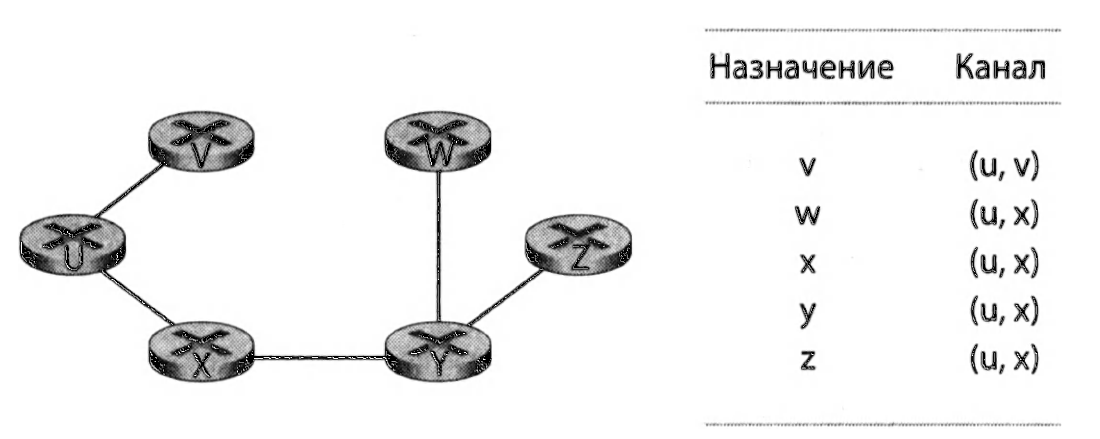
* На этапе инициализации уже известные пути с наименьшей стои­мостью от узла *и* до смежных с ним узлов *v, x и w* принимают, со­ответственно, значения 2, 1 и 5. В частности, следует отметить, что стоимость *w* равна 5 (хотя вскоре мы убедимся, что существует и еще более дешевый путь), поскольку это стоимость пути прямого канала (одного перехода) от *и* до *w.* Стоимости переходов к *у* и *z* равны бес­конечности, поскольку эти узлы не имеют прямой связи с *и.*
* В первой итерации мы просматриваем узлы, которые еще не отно­сятся к множеству N` и находим узел с наименьшей стоимостью по состоянию на конец предыдущей итерации. Это узел *х* со стоимо­стью 1, следовательно, он добавляется ко множеству N`. Затем вы­полняется строка 12 алгоритма LS, которая обновляет значение *D(v)* для всех узлов *v.* Получается результат, показанный во второй стро­ке табл. 4.3 (шаг 1). Стоимость пути к узлу *v* остается неизменной. Стоимость пути к узлу *w* (которая на момент завершения инициа­лизации была равна 5) через узел *х* оказывается равной 4. Соответ­ственно, выбирается путь с меньшей стоимостью, а предшественник *w* на кратчайшем пути от *и* становится равен *х.* Аналогично, пере-считывается стоимость пути до *у* (через *х),* она становится равна 2. Таблица соответствующим образом обновляется.
* На следующем этапе итерации выясняется, что узлы *v* и *у* обладают путями с наименьшей стоимостью (2). Мы произвольно разрываем связь и добавляем *у* ко множеству N`, так, что множество N` теперь содержит узлы w, *x* и *у.* Стоимость пути к оставшимся узлам, еще не принадлежащим ко множеству N` — то есть, к *v, w* и *z*, обновляется после выполнения строки 12 алгоритма LS. Получаем результаты, находящиеся в третьей строке табл. 4.3.
* И т. д.

Когда завершается выполнение алгоритма с учетом состояния кана­лов (LS), мы знаем для каждого узла его узел-предшественник на пути с наименьшей стоимостью от исходного узла. Для каждого предшествен­ника мы также знаем *его* узел-предшественник и т. д.; следовательно, мы можем восстановить весь путь от исходного узла ко всем конечным.

**Табл. 4.3.** Выполнение алгоритма маршрутизации с учетом состояния каналов для сети, изображенной на рис. 4.27

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***шаг*** | **ЛГ** | ***D(v),p(v)*** | ***D(w),p(w)*** | ***D{x),p(x)*** | ***П(У),Р(У)*** | ***D(z),p(z)*** |
| **0** | **и** | **2,u** | **5,u** | **l,u** | **00** | **00** |
| **1** | **их** | **2,u** | **4,x** |  | **2,x** | **00** |
| **2** | **иху** | **2,u** | **з,у** |  |  | **4.У** |
| **3** | **uxyv** |  | **з,у** |  |  | **4.У** |
| **4** | **uxyvw** |  |  |  |  | **4-У** |
| **5** | **uxyvwz** |  |  |  |  |  |

Далее, опираясь на эту информацию, можно восстановить таблицу перенаправления на заданном узле — скажем, на узле *и.* Для этого нужно найти путь с наименьшей стоимостью от *и* до узла назначения, сохранив для каждого узла его узел следующего перехода. На рис. 4.28 показаны результирующие пути с наименьшей стоимостью, а на рис. 4.27 — табли­ца перенаправления из узла *и.*

****

**Рис. 4.28.** Путь с наименьшей стоимостью и таблица перенаправления для узла *и*

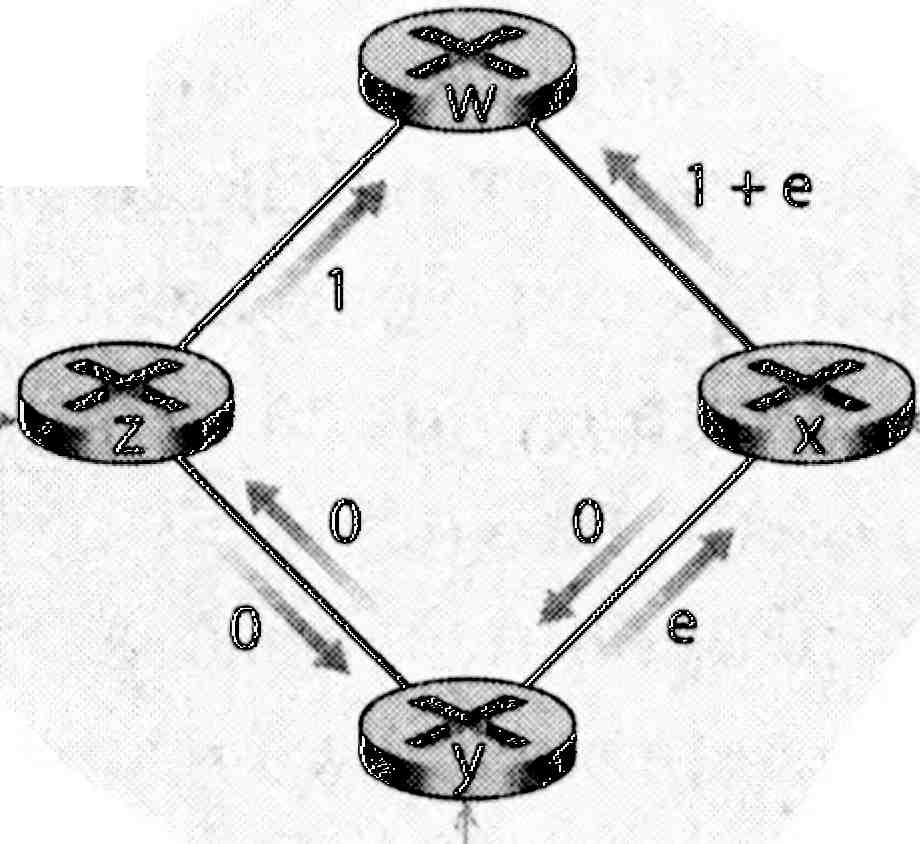
Какова будет трудоемкость этого алгоритма? То есть при наличии *п* узлов (не считая исходного) какой объем вычислений придется выпол­нить, чтобы найти пути с наименьшей стоимостью от источника до всех узлов назначения? В первой итерации потребуется просмотреть все *п* узлов, чтобы определить тот узел *w>* который не относится к множеству *N`* но обеспечивает минимальную стоимость. Во второй итерации для опре­деления минимальной стоимости нам придется проверить *п -* 1 узлов, в третьей итерации — *п - 2* и т. д. Общее количество узлов, которые нам придется перебрать в ходе всех итераций, равно n(n+l)/2. Соответственно, для предыдущей реализации алгоритма с учетом состояния каналов на­блюдается наивысшая сложность порядка *п* в квадрате: *0(п2).* Более слож­ная реализация данного алгоритма, в которой используется особая структура данных под названием «куча», позволяет найти минимум в строке 9 за время, пропорциональное логарифму от числа рассматриваемых узлов. Таким образом, сложность алгоритма значительно снижается.

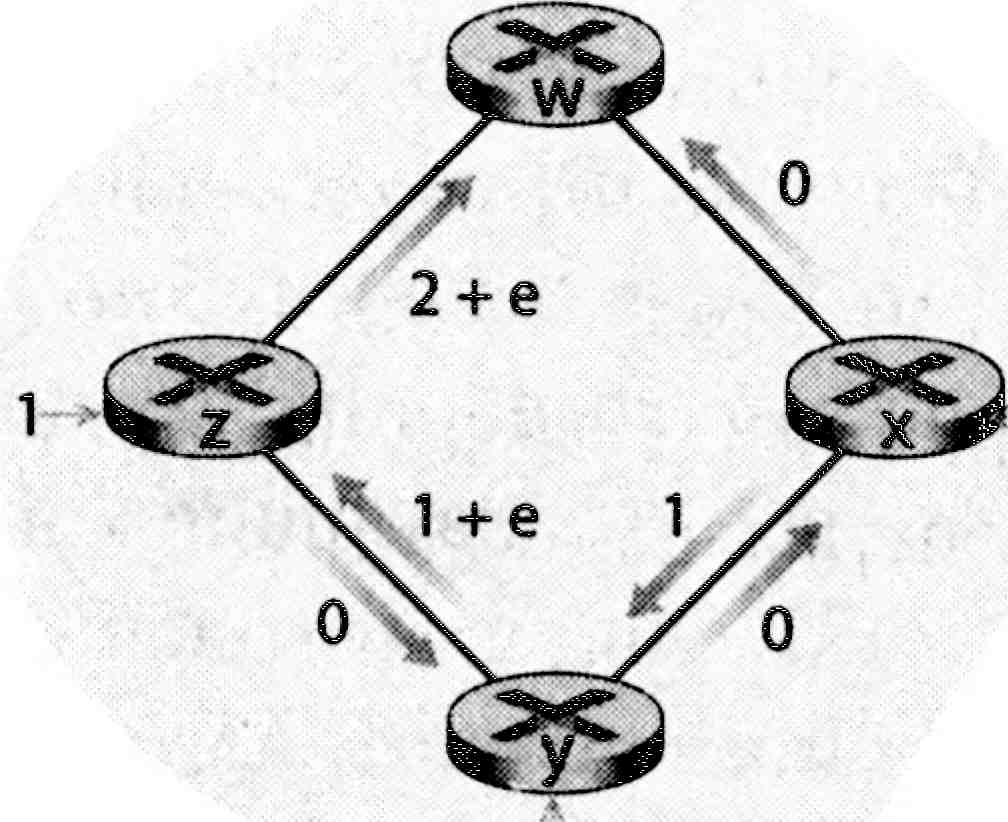
Перед тем как завершить обсуждение алгоритма LS, давайте рас­смотрим один потенциально возможный патологический случай. На рис. 4.29 изображена простая топология сети, в которой стоимость ка­нала равна той нагрузке, что по нему передается. Например, на схеме отражена задержка, которая будет возникать на том или ином участке. В данном случае, стоимости каналов несимметричны. Это значит, что *c(u, v)* равно *c(v, и)* лишь при условии, что нагрузка, передаваемая в обо­их направлениях по каналу между узлами *(и, v*), будет одинаковой. В данном примере узел *z* порождает информацию, предназначенную для *w,* и узел *х* делает то же самое. Наконец, узел *у* внедряет сюда объем ин­формации, равный е, также предназначенный для *т.* Изначальный вари­ант маршрута показан на рис. 4.29(a), стоимость каналов соответствует объему передаваемого по ним трафика.

При следующем прогоне алгоритма LS узел *у* определяет (опираясь на стоимости каналов, показанные на рис. 4.29а), что путь до *w* по ча­совой стрелке имеет стоимость 1, а путь до *w* против часовой стрелки (который и был использован в данном случае) — стоимость 1 *+е.* Соот­ветственно, теперь путь с наименьшей стоимостью от *у* до *w* пролегает по часовой стрелке.

Аналогично *х* определяет, что его новый путь до *w* с наименьшей стоимостью также пролегает против часовой стрелки, результирующие значения стоимости даны на рис. 4.29(6). При следующем прогоне алго­ритма LS все три узла *х, у* и *z —* будут направлять свой трафик против часовой стрелки.

Подобные осцилляции (они могут возникнуть не только в LS, но и в любом другом алгоритме, стоимость каналов в котором зависит от перегрузок и задержек при передаче информации) необходимо предот­вращать. Одно из возможных решений — установить, что стоимость ка­налов не должна зависеть от объема переданного трафика. Правда, такой вариант неприемлем, поскольку сама цель маршрутизации заключается в том, чтобы избегать перегруженных каналов (например, таких, в ко­торых возникают серьезные задержки). Другое решение — обеспечить, чтобы не все маршрутизаторы выполняли алгоритм LS одновременно. Это представляется более разумным, так как мы можем рассчитывать, что даже если все маршрутизаторы будут выполнять алгоритм с одинаковой периодичностью, то длительность выполнения на каждом из них не будет одинаковой. Исследователи обнаружили следующую ин­тересную закономерность: оказывается, маршрутизаторы в Интернете могут синхронизироваться друг с другом. Это означает, что даже если изначально маршрутизаторы выполняют алгоритм с одинаковой перио­дичностью, но в разные моменты времени, то постепенно такие прогоны алгоритма на разных маршрутизаторах могут стать и далее оставаться одновременными. Чтобы избежать такой самопроизвольной синхрони­зации, можно задать на каждом маршрутизаторе рандомизацию момен­тов времени, когда он рассылает сведения о каналах.





■;

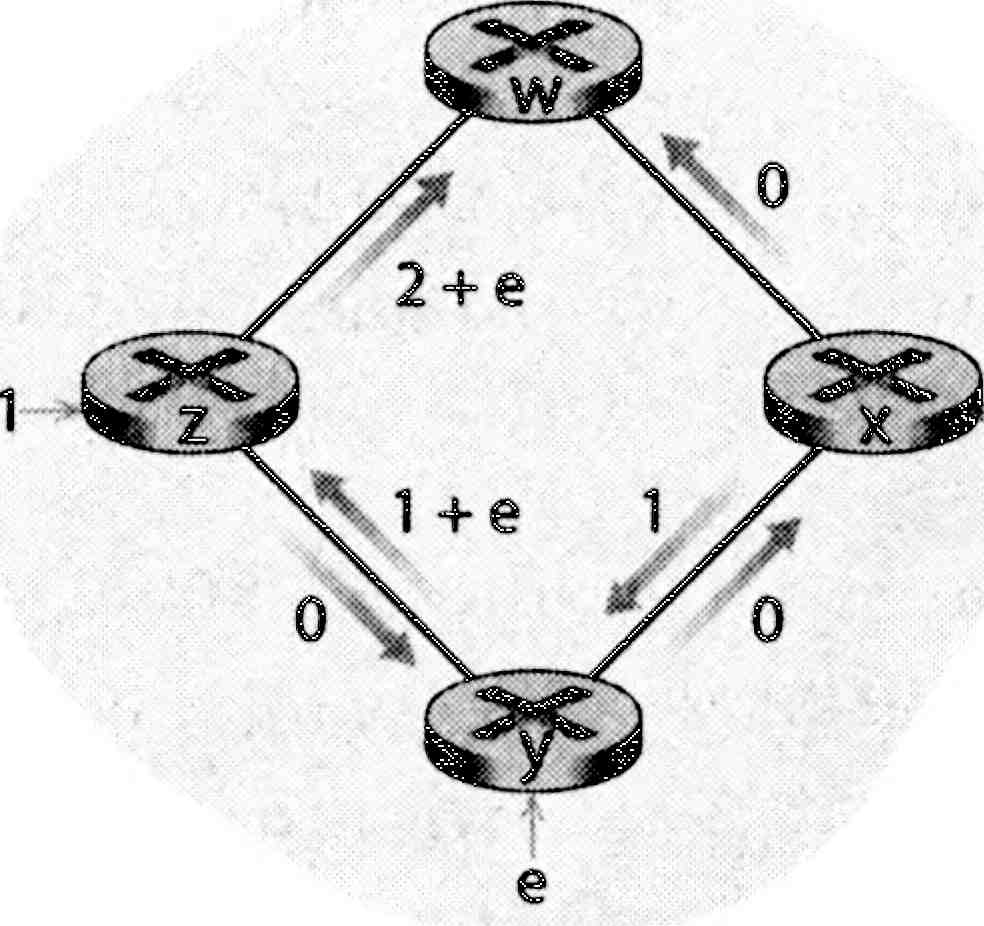
**е**

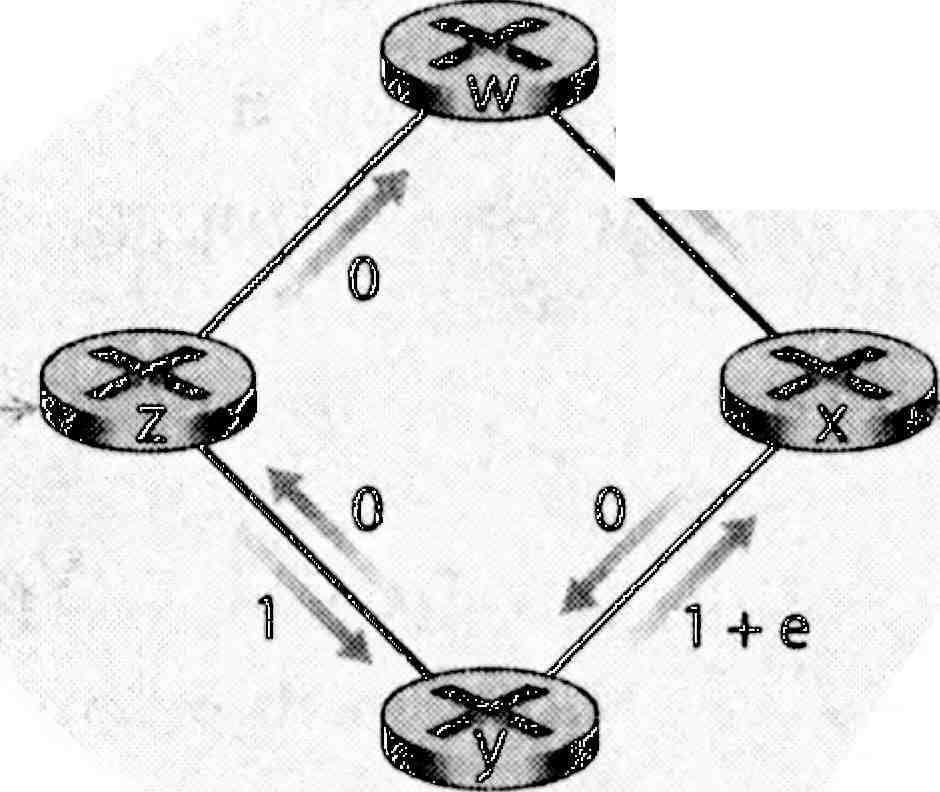
**а. Исходный вариант маршрутизации**

**б. х и у находят более**

**оптимальный путь**

**к w - по часовой стрелке**





5 2+е

**е**

**в. х, у, z находят**

**более оптимальный путь**

**к w - против часовой стрелки**

**г. х, у, z находят более**

**оптимальный путь**

**к w - по часовой стрелке**

**Рис. 4.29. Осцилляции при маршрутизации, чувствительной к нагрузкам**

Изучив алгоритм LS, давайте рассмотрим еще один важный алго­ритм маршрутизации, который сегодня применяется на практике — дистанционно-векторный.

**Дистанционно-векторный алгоритм маршрутизации**

В то время как алгоритм с учетом состояния каналов (LS) использу­ет при работе глобальную информацию, **дистанционно-векторный ал­горитм** (distance-vector, DV) маршрутизации является итерационным, асинхронным и распределенным. Его *распределенность* заключается в том, что, когда узел получает информацию от одного или нескольких *непосредственно связанных* с ним узлов-соседей, этот алгоритм выпол­няет вычисление, а затем вновь распространяет его результат между узлами-соседями. *Итерационностъ* алгоритма означает, что процесс продолжается до тех пор, пока узлы-соседи полностью не обменяются той или иной информацией. Интересно отметить, что этот алгоритм также является самозавершающимся — он не требует никакого сигнала о том, что вычисления нужно прекратить, а просто сам завершает ра­боту. *Асинхронность* дистанционно-векторного алгоритма выражается в том, что все участвующие в нем узлы не обязаны «идти в ногу» друг с другом. Очевидно, что асинхронный итерационный самозавершаю­щийся распределенный алгоритм гораздо интереснее и увлекательнее, чем централизованный!

Прежде чем перейти непосредственно к изучению этого алгоритма, не помешает обсудить одну важную взаимосвязь, которая существует между стоимостями наиболее выгодных (кратчайших) путей. Допу­стим, *dx(y)* — это стоимость кратчайшего пути от узла *х* до узла *у.* В та­ком случае, взаимосвязь между наименьшими значениями стоимости будет описываться знаменитым уравнением Беллмана — Форда:

*dx(y)* = *minv{c(x,v) +dv(y)}* (4.1)

где min^ — значение, взятое с учетом всех узлов, смежных с *х.* Уравне­ние Беллмана — Форда довольно понятное. Действительно, если после перехода от *х* до *v* мы изберем кратчайший путь от *у* до *v,* то стоимость этого пути будет высчитываться по формуле *c(x,v)+dv(y).* Поскольку мы начинаем путь с перехода к какому-либо соседнему узлу *v,* наименьшая стоимость пути от *х* до *у* составит минимальное значение *c(x,v)+dv(y)* с учетом всех узлов-соседей *v.*

Если среди читателей найдутся скептики, сомневающиеся в пра­вильности этого уравнения, предлагаю проверить его для исходного узла *и* и узла назначения *z* на рис. 4.27. Исходный узел *и* имеет трех соседей: это узлы *v, x* и *w.* Проследив различные пути в графе, легко убедиться, что *dh{z)* = 5, *dx(z)* = 3 и *dw(z)* = 3. Подставив эти значения в уравнение 4.1 при стоимостях *c(u,v)* = 2, *c(u,c)* = 1 и *c(u,w) =* 5, имеем *du(z)=* min{2+5, 5+3, 1+3} = 4. Очевидно, это равенство верное, и оно дает ровно такой же результат, как и алгоритм Дейкстры для той же сети. Проверить эти значения можете достаточно быстро, ваш скепти­цизм тут же исчезнет.

Уравнение Беллмана — Форда — не просто любопытное интеллекту­альное достижение. Оно имеет огромное практическое значение. В част­ности, именно по этому уравнению вычисляются все значения, записан­ные в таблице маршрутизации узла *х.* Чтобы убедиться, допустим, что *V\* —* это любой узел-сосед, позволяющий получить минимальный ре­зультат в уравнении 4.1. В таком случае, если узлу *х* требуется отослать пакет узлу *у* по пути с наименьшей стоимостью, то сначала этот пакет должен быть переправлен на узел *V\*.* Соответственно, в таблице марш­рутизации узла *х* узел *V\** будет указан как цель следующего перехода, с учетом того, что конечной целью маршрутизации является узел *V\** Еще один важный практический аспект этого уравнения заключается в том, что в алгоритме DV будет происходить обмен информацией по принципу «от соседа к соседу».

Базовая идея такова. Каждый узел *х* начинает работу, исходя из информации *Dx(y) —* это оценочная стоимость кратчайшего пути от него до узла у, с учетом всех узлов, входящих во множество *N.* Допу­стим, *Dx = [Dx(y):y* принадлежит к *N] —* это вектор расстояния узла *х,* то есть, вектор, оценивающий расстояние от *х* до всех других узлов *уу* от­носящихся к *N.* При применении дистанционно-векторного алгоритма каждый узел х будет содержать следующую информацию о маршрути­зации.

* Для каждого узла *v —* стоимость *c(x,v)* от *х* до непосредственной свя­зи с соседом, *v*
* Вектор расстояния узла x, то есть, *Dx = [Dx(y):y* во множестве *N],* век­тор, оценивающий расстояние от *х* до всех других узлов *у,* относя­щихся **к *N.***
* Вектор расстояния для каждого из узлов-соседей, то есть,

*Dv = [Dv(y):y* во множестве *N]* для каждого из соседей *v* узла *х.*

В распределенном асинхронном алгоритме каждый узел время от времени посылает копию своего вектора расстояния каждому из сосе­дей. Когда узел *х* получает новый вектор расстояния от любого из своих соседей *v,* он сохраняет вектор расстояния *v,* а затем использует уравне­ние Беллмана — Форда, чтобы обновить собственный, вот так:

***Dx(y)= minv{c(x,v)+ Dv(y)}*** Для каждого узла *у* во множестве *N*

Если в результате такой операции обновления вектор расстояния узла *х* изменится, то узел *х* пошлет полученное пересчитанное значение каждому из соседей, которые, в свою очередь, также смогут обновить свои векторы расстояний. И как ни удивительно, в ходе такого асин­хронного обмена между узлами, все оценки стоимости *Dx(y)* постепен­но сходятся к *dx(y)y* фактической стоимости кратчайшего пути от узла *х* к узлу г/50!

**Дистанционно-векторный алгоритм**

Для каждого узла *хг.*

1. **Инициализация:**
2. для всех узлов назначения у во множестве N:

3 D (у) = с(х,у) /\* если у не является смежным, то с(х,у) = °° \*/

4 для каждого узла-соседа w

5 Dw(y) = ? для всех узлов назначения у во множестве N

6 для каждого узла-соседа w

7 отправить вектор расстояния Dx = [Dx(y): у *во множестве* N] к w  
8

1. **цикл**
2. **ждем** (пока не увидим изменения стоимости канала к соседу w или

11 пока не получим дистанционный вектор от того или иного соседа w)  
12

13 для каждого узла у во множестве N:

14 Dx(y) = minv{c(x,v) + Dv(y)}  
15

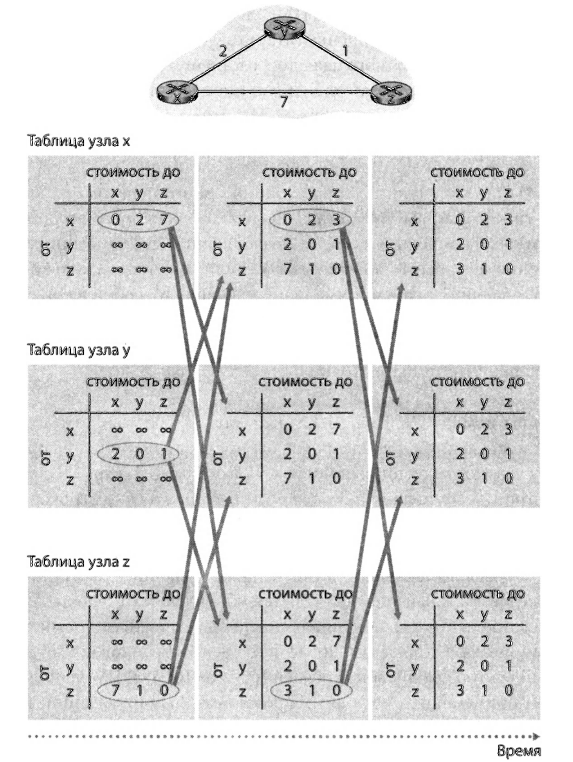
1. **если** Dx(y) изменится для любого узла назначения у
2. посылаем дистанционный вектор Dx = [Dx(y): у во множестве N] всем соседям 18

19 **вечно**

В дистанционно-векторном (DV) алгоритме узел *х* обновляет свою дистанционно-векторную оценку в двух случаях: либо когда фиксирует изменение стоимости одного из непосредственно связанных с ним кана­лов, либо когда получает обновленную информацию о дистанционном векторе от одного из своих соседей. Но чтобы обновить собственную таблицу маршрутизации для заданного узла назначения *у*, узлу *х* на са­мом деле требуется знать не кратчайшее расстояние до *y*, а свой узел-сосед *V\*(y),* находящийся на расстоянии одного перехода на кратчай­шем пути к *у.* Как вы уже догадались, таким маршрутизатором *V\*(y)* в нашем случае является сосед *v,* позволяющий достичь минимально­го значения в строке 14 алгоритма DV. Если имеется несколько узлов-соседей *v,* дающих такое минимальное значение, то любой из них можно подставить на место *V\*(y).* Следовательно, в строках 13-14 для каждого узла назначения *у* узел *х* также определяет *vx(y)* и обновляет свою та­блицу маршрутизации для точки *у.*

Как вы помните, алгоритм LS является глобальным: это означает, что в нем каждый узел должен сначала получить полную информацию о сети, а уже потом будет выполняться алгоритм Дейкстры. В свою оче­редь, алгоритм DV является *децентрализованным* и обходится без такой глобальной информации. Действительно, все сведения, которые будут у узла, сводятся к стоимости каналов, связывающих его с непосред­ственно примыкающими соседями, а также сведениям, полученным от этих соседей. Каждый узел дожидается обновления от любого из своих соседей (строки 10-11), получив обновление, вычисляет новый дистан­ционный вектор (строка 14), после чего распространяет новый дистан­ционный вектор среди своих соседей (строки 16-17). DV-подобные ал­горитмы применяются во многих протоколах маршрутизации: таковы, например, RIP и BGP, действующие в Интернете, ISO IDRP, Novell IPX и оригинальный алгоритм ARPAnet.

На рис. 4.30 проиллюстрирована работа дистанционно-векторного алгоритма для простой сети, состоящей из трех узлов — она показана в верхней части рисунка. Работа алгоритма показана в синхронном ре­жиме, где все узлы одновременно получают дистанционные векторы от своих соседей, вычисляют новые и информируют своих соседей о том, не изменились ли эти векторы. Изучив этот пример, вы окончательно убедитесь, что алгоритм столь же корректно работает и по асинхронно­му принципу, когда вычисления на узлах и генерирование/получение обновлений могут происходить в любой момент.

****

**Рис. 4.30. Дистанционно-векторный алгоритм**

В крайнем левом столбце на рисунке показаны три исходные **табли­цы маршрутизации** для каждого из трех узлов. Например, в верхнем левом углу находится исходная таблица для узла *х.* В рамках каждой отдельно взятой таблицы маршрутизации каждая строка — это дистан­ционный вектор. Точнее говоря, каждая таблица маршрутизации содер­жит собственный дистанционный вектор узла и такие дистанционные векторы его узлов-соседей. Итак, в первой строке исходной таблицы маршрутизации узла х имеем *D = [D* (x), *D (у), D* (z)] - [0,2,7]. Во второй и третьей строках этой таблицы находятся наиболее свежие (недав­но полученные) дистанционные векторы от узлов *у иг* соответственно. Поскольку на момент инициализации узел *х* еще ничего не получил от узлов *у* или 2, то записи во второй и третьей строках после инициализа­ции результируют в бесконечность.

После инициализации каждый узел посылает свой дистанцион­ный вектор каждому из двух узлов-соседей. На рис. 4.30 это показано при помощи стрелок, направленных из таблиц первого столбца в та­блицы второго. Например, узел *х* отсылает дистанционный вектор *Dx* = [0,2,7] одновременно узлам *у и z.* Получив обновления, каждый узел пересчитывает собственный дистанционный вектор. Например, для узла *хг,*

***Dx(x)* = 0**

***Dx(y)* = *тin{с(х,у)* + *Dy(y), c(Xfz)* + *Dy(y)}* = min{2+0,7+l} = 2**

***Dx******(z)* - *min{c(xfy)* + *Dy* (2), *c(xj)* + *Dz* (2)} = min{2+1,7+0} = 3**

Соответственно, в таблицах второго столбца указаны новые дистан­ционные векторы для каждого узла, а вместе с ними — дистанционные векторы, только что полученные от узлов-соседей. В частности, обра­тите внимание на то, что оценка наименьшей стоимости пути от узла *х* до 2, *Dx(z),* изменилась с 7 на 3. Также отметим, что для узла *х* его узел-сосед *у* достигает минимума в строке 14 DV-алгоритма. На узле *х* имеем ***V\*(y) =у*** и ***V\*(z)* = *у.***

После того, как узлы пересчитают свои дистанционные векторы, они опять пошлют обновленные значения своим соседям (если изменения действительно будут). На рис. 4.30 это проиллюстрировано при помощи стрелок, идущих из таблиц второго столбца таблиц в третий. Обрати­те внимание: лишь узлы *хиz*присылают обновления. Дистанционный вектор узла *у* не изменился, поэтому данный узел и не посылает обнов­лений. После получения обновлений узлы пересчитывают свои дистан­ционные векторы и обновляют таблицы маршрутизации, что и показано в третьем столбце.

Процесс получения обновленных дистанционных векторов от узлов-соседей, пересчет записей в таблицах маршрутизации и информирова­ние соседей об изменении стоимости кратчайшего пути к узлу назначе­ния продолжаются до тех пор, пока сообщений об обновлении больше не останется. Когда рассылка сообщений об обновлении прекращает­ся, пересчет таблиц маршрутизации прекратится, и алгоритм перейдет в стационарное состояние. Это значит, что все узлы будут находиться на этапе ожидания, который описан в строках 10-11 алгоритма DV. Ал­горитм останется в стационарном состоянии до тех пор, пока вновь не изменится стоимость путей. Об этом пойдет речь ниже.

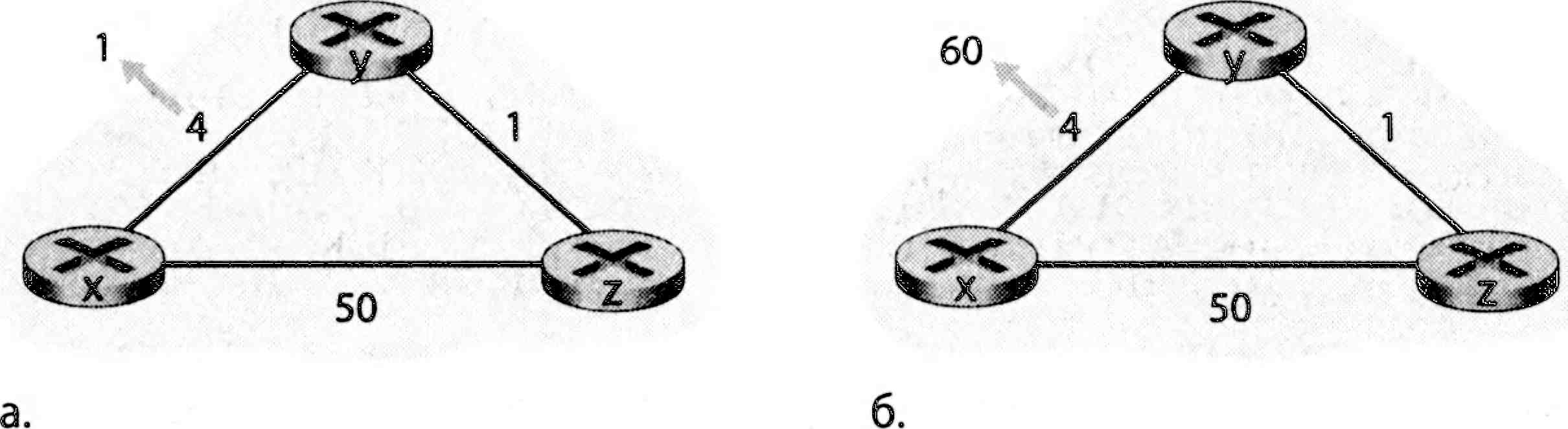
**Дистанционно-векторный алгоритм при изменении стоимостей и неисправностях каналов**

Когда узел, на котором работает дистанционно-векторный алго­ритм, обнаруживает изменение стоимости канала, направляющегося от него к соседу (строки 10-11), он обновляет свой дистанционный век­тор (строки 13-14) и, если наименьшая стоимость пути изменилась, он извещает об этом своих соседей (строки 16-17). Эту ситуацию ил­люстрирует рис. 4.31(a). В данном примере стоимость канала от узла *х* до узла *у* изменяется с 4 до 1. Здесь мы рассматриваем только записи таблиц расстояний узлов *у* и z, содержащие стоимости путей до узла *х.* Дистанционно-векторный алгоритм приводит к следующей последова­тельности событий.

1. В момент времени *t0* узел *у* замечает изменение стоимости канала (с 4 до 1) и информирует об этом своих соседей, так как благодаря обновлению стоимости линии меняются минимальные стоимости путей.
2. В момент времени *tx* узел *z* получает сообщение от узла *у* и обновля­ет свою таблицу. Затем он вычисляет новое значение минимальной стоимости маршрута до узла *х* (она уменьшилась с 5 до 2) и извеща­ет об этом своих соседей.
3. В момент времени *t2* узел *у* получает сообщение от узла *z* и обновля­ет свою таблицу. Его значения минимальных стоимостей остались прежними (хотя стоимость пути до узла *х* через узел *z* изменилась), поэтому узел *у* не посылает сообщений своим соседям. Алгоритм переходит в стационарное состояние.

Соответственно, дистанционно-векторному алгоритму требуется только две итерации, чтобы перейти в стационарное состояние. «Хо­рошие новости» об уменьшившийся стоимости канала между узлами *х* и *у* быстро распространяются по сети.

Рассмотрим теперь, что произойдет в случае *увеличения* стоимости канала. Предположим, что стоимость канала между узлами *х* и *у* возрос­ла с 4 до 60, как показано на рис. 4.31(6).



**Рис. 4.31. Изменение стоимости канала**

1. До изменения стоимости канала *D (х)* = 4, *D* (z) = 1, *Dz(y)* = 1  
и *Dz(x)* = 5. В момент времени *t0* узел *у* замечает изменение стои­  
мости канала (с 4 до 60). Узел *у* вычисляет, что новая наименьшая  
стоимость пути до узла *х* равна

***Dy(x) = min{c(x,y)* + *Dx(x), c(y,z)* + *Dz(x)}* = min{60+0,l+5} = 6.**

Поскольку мы можем видеть всю сеть сразу (глобально), нам понят­но, что это новое значение неверно. Но узлу *у* известно лишь то, что стоимость его прямого соединения с узлом *х* равна 60 и что узел *z* в последний раз сообщил узлу *у* о наличии у узла *z* пути к узлу *х* стоимостью 5. Узел *у* решает, что теперь ему дешевле посылать па­кеты узлу *х* через узел *z.* Таким образом, в момент времени *t{* у нас образовалась петля — узел *у* направляет пакеты для узла *х* через узел 2, а узел *z* направляет пакеты для узла *х* через узел *у.* Петля подобна черной дыре — пакет, предназначенный узлу *х,* будет «отфутболи­ваться» узлами *yuz* друг другу до тех пор, пока таблицы маршрути­зации на этих узлах не изменятся.

1. Так как узел *у* вычислил новый путь наименьшей стоимости до узла *х,* он информирует об этом узел *z* в момент времени *tv*
2. Спустя некоторое время после *tx* узел *z* получает новый дистанцион­ный вектор с измененной наименьшей стоимостью пути от узла *у* до узла *х* (узел *у* сообщил узлу 2, что это значение теперь равно 6). Узел 2 знает, что он может добраться до узла *у* по каналу стоимости 1, и вы­числяет новое значение стоимости маршрута до узлах (все также через узел *у),* равное 7. Поскольку наименьшая стоимость пути от узла *z* до узла *х* увеличилась, узел *z* извещает об этом узел *у* в момент времени *tT*
3. Аналогичным образом узел *у* обновляет свою таблицу и информиру­ет узел 2 о том, что значение минимальной стоимости теперь равно 8. Затем узел *z* обновляет свою таблицу и информирует узел *у* о том, что значение минимальной стоимости стало равно 9, и т. д.

Как долго может продолжаться этот процесс? Вы можете убедиться в том, что петля будет сохраняться в течение 44 итераций (необходимых узлам *у* и *z* для обмена сообщениями) — пока наконец узел *z* не опреде­лит, что стоимость пути до узла *х* через узел *у* больше 50. В этот момент узел *z* (наконец-то!) выяснит, что путь наименьшей стоимости к узлу *х* пролегает по прямой линии с узлом *х.* Таким образом, «дурным вестям» об увеличении стоимости канала между узлами *х* и *у* для распростра­нения по сети потребовалось очень много времени! Что бы произошло, если бы стоимость канала *с(у,х)* увеличилась с 4 до 10 000, а стоимость канала *c(z,x)* была бы равна 9999? Подобная проблема иногда называет­ся проблемой «счета до бесконечности».

**Дистанционно-векторный алгоритм и обратная коррекция**

Обсуждавшегося выше сценария со счетом до бесконечности можно избежать, если использовать метод, называемый обратной коррекцией, или *«отравлением» обратного пути.* Идея этого метода проста — если узел *z* направляет пакеты узлу *х* через узел *у,* тогда узел *z* объявит узлу *у у* что его (узла *z)* расстояние до узла *х* равно бесконечности (хотя на самом деле узлу *z* известно, что *Dz(x)* = 5). Узел *z* будет продолжать го­ворить узлу *у* эту «маленькую ложь» до тех пор, пока узел *z* направляет пакеты узлу *х* через узел *у.* Поскольку узел *у* полагает, что у узла *z* нет пути к узлу *х,* узел *у* никогда не станет пытаться посылать пакеты узлу *х* через узел *z,* пока узел *z* продолжает посылать пакеты узлу *х* через узел *у* (и лгать о том, что у него нет пути к узлу *х).*

Давайте рассмотрим, как метод обратной коррекции решает пробле­му петель, с которой мы столкнулись выше на рис. 4.31 (б). В результате обратной коррекции в таблице расстояний узла *у* в записи стоимости пути до узла *х* через узел *z* указана бесконечность (так как узел *z* со­общил узлу Y, что расстояние от узла *z* до узла *х* равно бесконечности). Когда стоимость линии *(х,у)* увеличивается с 4 до 60, узел *у* обновляет свою таблицу, но продолжает направлять пакеты узлу *х* напрямую, не­смотря на увеличившуюся стоимость маршрута, а также информирует узел *z* об изменении стоимости, сообщая, что *D (х)* = 60. Получив это извещение в момент времени *tv* узел *z* немедленно изменяет свой марш­рут к узлу *х* на прямую линию *(z,x)* стоимости 50. Поскольку маршрут к узлу *х* изменился и более не проходит через узел *у,* узел *z* сообщает узлу *у* в момент времени *tv* что *Dz(x)* = 50. Получив эту информацию, узел *у* обновляет свою таблицу расстояний, так что *D (х)* = 51. Кроме того, поскольку через узел *z* теперь проходит путь наименьшей стоимости от узла ***у*** к узлу *х,* узел ***у*** «отравляет» обратный путь от узла *z* до узла *ху* сообщая в момент времени *t3* узлу z, что *D (х)* = оо (хотя на самом деле узлу ***у*** известно, что *D (х)* = 51).

Позволяет ли метод «отравления» обратного пути решить пробле­му счета до бесконечности в общем случае? Нет. Вы можете убедиться сами, что петли, состоящие из трех и более узлов (а не просто из двух смежных), метод обратной коррекции распознать не сможет.

**Сравнение алгоритмов маршрутизации LS и DV**

Дистанционно-векторный алгоритм (DV) и алгоритм с учетом со­стояния каналов (LS) взаимно дополняют друг друга при планировании маршрутизации. В алгоритме DV каждый узел обменивается информа­цией *только* со своими непосредственными соседями, но предоставля­ет этим смежным узлам оценку наименьшей стоимости от себя до *всех остальных* узлов в сети (насколько ему это известно). В алгоритме LS каждый узел обменивается информацией со *всеми* остальными узлами в сети (широковещательным образом), но сообщает им стоимости *лишь* тех каналов, которые непосредственно с ним связаны. Давайте завер­шим наше изучение алгоритмов LS и DV, вкратце сравнив некоторые их свойства. Как вы помните, *N* — это множество узлов (маршрутизато­ров), а *Е* — множество ребер (каналов).

* *Сложность сообщений.* Как мы видели, алгоритм, основанный на состоянии линий, требует от каждого узла знать стоимость каждой линии сети. Для этого необходимо отправить 0(|N| |E|) сообщений. Кроме того, каждый раз, когда стоимость линии изменяется, об этом следует известить все узлы. Дистанционно-векторный алгоритм тре­бует обмена сообщениями только между напрямую соединенными узлами на каждой итерации. Как было показано, время, необходимое для схождения алгоритма, может зависеть от многих факторов. Ког­да изменяется стоимость линии, дистанционно-векторный алгоритм распространяет результаты только в том случае, если обновление приводит к изменению пути с наименьшей стоимостью для одного из узлов, присоединенного к этому каналу.
* *Скорость схождения.* Как мы видели, количество вычислений в на­шей реализации алгоритма, основанного на состоянии каналов, рас­тет пропорционально квадрату узлов сети, требуя передачи 0(|N| |E|) сообщений. Дистанционно-векторный алгоритм может сходиться медленно (в зависимости от относительной стоимости путей, как было показано в примере на рис. 4.10), и во время схождения могут образовываться петли. Кроме того, дистанционно-векторный алго­ритм страдает от «приступов» счета до бесконечности.

• *Живучесть.* Что может случиться, если маршрутизатор выйдет из строя, «сойдет с ума» или объявит забастовку? В алгоритме марш­рутизации, основанном на состоянии каналов, маршрутизатор мо­жет передать всем остальным маршрутизаторам неверные сведения о стоимости одного из присоединенных к нему каналов. Узел может также повредить или потерять один из широковещательных пакетов LS-алгоритма, который он получил. Но узел рассчитывает только собственную таблицу перенаправления. Остальные узлы заполняют свои таблицы сами. Это означает, что в алгоритме, основанном на со­стоянии каналов, расчеты маршрутов выполняются в значительной степени раздельно, что обеспечивает определенную степень живуче­сти. В случае дистанционно-векторного алгоритма узел может пере­дать другим узлам неверно рассчитанные им значения минимальной стоимости путей. (Такие ситуации встречаются на практике. В 1997 году неисправный маршрутизатор, принадлежащий небольшой ком­пании, занимающейся предоставлением доступа в Интернет, снабжал маршрутизаторы национальной магистрали ошибочной информа­цией о маршрутах. Это привело к тому, что другие маршрутизаторы завалили трафиком неисправный, в результате большие фрагменты Интернета в течение нескольких часов были отрезаны.) Можно ска­зать, что в дистанционно-векторном алгоритме на каждой итерации результаты вычислений узла непосредственно передаются соседнему узлу, а затем на следующей итерации они попадают к соседу соседа и т. д. Таким образом, в дистанционно-векторном алгоритме некор­ректно вычисленные данные могут распространиться по всей сети.

В заключение скажем, что ни один алгоритм нельзя считать побе­дителем. Как мы увидим в разделе «Маршрутизация в Интернете», на практике применяются оба эти алгоритма.

**Другие алгоритмы маршрутизации**

Рассмотренные нами дистанционно-векторный алгоритм и алго­ритм, основанный на состоянии каналов, представляют собой не про­сто популярные алгоритмы маршрутизации. По сути *только* эти два ал­горитма и применяются на практике. Тем не менее за последние 30 лет исследователями было предложено множество других алгоритмов маршрутизации, варьирующихся от крайне простых до очень сложных.

В основе широкого класса алгоритмов маршрутизации лежит точка зре­ния на пакетный трафик как на потоки данных между отправителями и получателями. При таком подходе проблема выбора маршрута может быть сформулирована математически как задача оптимизации при огра­ничениях, известная как задача сетевых потоков. Еще одно семейство алгоритмов маршрутизации, о которых следует сказать здесь несколько слов, обязаны своим происхождением телефонным сетям. Эти **алгорит­мы маршрутизации коммутируемых каналов** представляют интерес для сетей с коммутацией пакетов, когда для каждого соединения резер­вируются ресурсы линии, такие как пропускная способность части ли­нии связи или объем буферов. И хотя формулировка задачи маршрути­зации значительно отличается от метода путей наименьшей стоимости, у подобных алгоритмов довольно много общего, по крайне мере в том, что касается алгоритмов определения маршрутов.

**Иерархическая маршрутизация**

Ранее мы рассматривали сеть просто как множе­ство соединенных друг с другом маршрутизаторов. Один маршрути­затор ничем не отличался от других в том смысле, что на всех работал один и тот же алгоритм для расчета маршрутов через всю сеть. Однако данная модель с однородным набором маршрутизаторов является упро­щением и не применяется на практике по двум причинам.

• *Масштабирование.* Когда количество маршрутизаторов оказыва­ется очень большим, накладные расходы на вычисление, хранение данных и обмен данными о маршрутах (такими как обновление состояния каналов или изменения путей наименьшей стоимости) между маршрутизаторами становятся неприемлемыми. Сегодняш­ний Интернет состоит из сотен миллионов хостов. Хранение инфор­мации о маршрутах на каждом из этих хостов потребовало бы памя­ти огромных размеров. Накладные расходы на широковещательную рассылку пакетов с информацией о состоянии линий между всеми маршрутизаторами Интернета просто не оставили бы пропускной способности для пакетов с данными! А дистанционно-векторный алгоритм при таком огромном количестве узлов сети никогда бы не достиг схождения! Очевидно, необходимо было что-то предпринять, чтобы упростить задачу вычисления маршрутов в такой огромной сети, как Интернет.

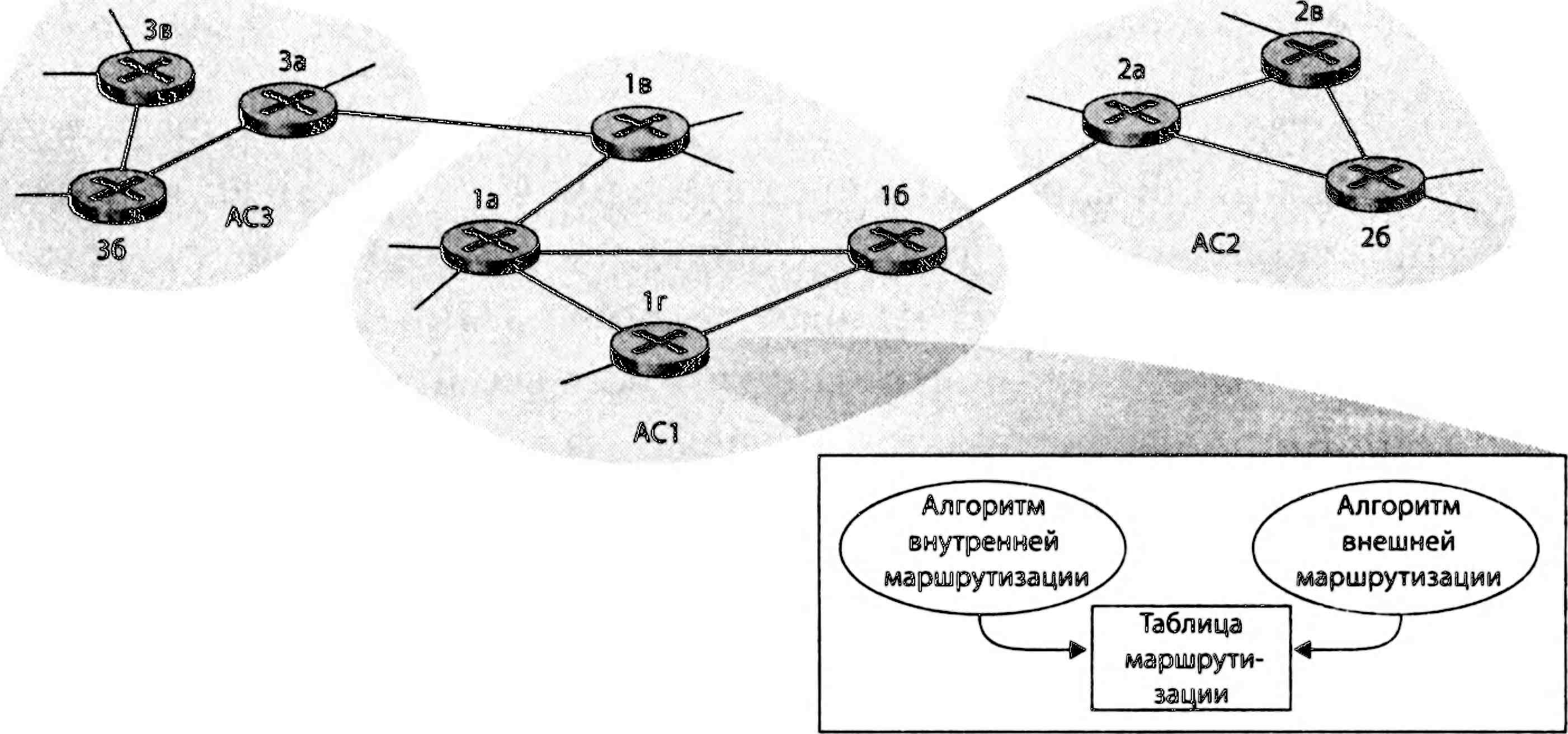
• *Административная автономия.* Хотя инженеры часто игнорируют такие вопросы, как пожелания компаний управлять маршрутизато­рами, как им хочется (например, использовать алгоритм маршру­тизации по своему выбору), или скрыть внутреннюю организацию сети от внешних наблюдателей, важность подобных вопросов может быть высокой. В идеальном случае организация должна иметь воз­можность управлять своей сетью так, как ей заблагорассудится, не теряя при этом возможности ее соединения с «внешним» миром.

Обе эти задачи могут быть решены путем объединения маршрути­заторов в отдельные области, называемые **автономными системами** (Autonomous System, AC). Маршрутизаторы в пределах одной автоном­ной системы используют один и тот же алгоритм маршрутизации (на­пример, дистанционно-векторный алгоритм или алгоритм, основанный на состоянии каналов) и обладают информацией обо всех маршрутиза­торах своей автономной системы, как это было в рассмотренном ранее идеальном случае. Алгоритм маршрутизации, используемый внутри автономной системы, называется **протоколом внутренней маршрути­зации.** Разумеется, необходимо соединить автономные системы друг с другом, и поэтому один или несколько маршрутизаторов в автономной системе будут отвечать за пересылку пакетов за пределы автономной системы. Эти маршрутизаторы называются **шлюзовыми (граничными) маршрутизаторами,** или просто **шлюзами.**

Данный сценарий проиллюстрирован на рис. 4.32. Здесь изображе­ны три автономные системы, АС1, АС2 и АС3. Жирные линии пред­ставляют прямые соединения между парами маршрутизаторов. Тонкие линии, проведенные от маршрутизаторов, представляют подсети, непо­средственно подключенные к маршрутизаторам. В автономной системе АС1 имеются четыре маршрутизатора, 1а, 1б, 1в и 1г, на которых рабо­тает протокол внутренней маршрутизации, используемый в пределах автономной системы АС1. Соответственно, каждый из этих четырех маршрутизаторов знает, как пересылать пакеты по оптимальным путям на любой целевой узел в системе АС1. Аналогично в автономных систе­мах АС2 и АС3 есть по три маршрутизатора. Обратите внимание, что протоколы внутренней маршрутизации, применяемые в автономных системах АС1, АС2 и АС3, могут быть разными. Шлюзовыми маршру­тизаторами являются 1б,1в, 2а и За.

Итак, должно быть ясно, как маршрутизаторы в автономной системе определяют пути доставки для таких пар «исходный узел — конечный узел», которые являются внутренними для автономной системы. Однако в нашей задаче о маршрутизации из конца в конец по-прежнему нет отве­та на один большой вопрос: как маршрутизатор, расположенный внутри автономной системы, сумеет направить пакет на узел, находящийся вне этой автономной системы? Ответить на этот вопрос не составляет труда, если в АС имеется лишь один шлюз, соединяющий ее ровно с одной дру­гой автономной системой. В таком случае, поскольку внутренний алго­ритм маршрутизации, действующий в АС, определяет путь с наименьшей стоимостью от любого внутреннего маршрутизатора до шлюза, каждому внутреннему маршрутизатору известно, как пересылать пакеты.

Как только шлюз получает пакет, он пересылает его по единствен­ному каналу, ведущему из шлюза за пределы автономной системы. Вторая автономная система, расположенная на другом конце канала, будет завершать доставку пакета к месту назначения. Рассмотрим при­мер: маршрутизатор 2б на рис. 4.32 получает пакет, чей узел назначения находится за пределами АС2. В таком случае маршрутизатор 2б пере­шлет пакет на 2а или 2в — в зависимости от того, что указано в таблице перенаправления 2б, которая была сконфигурирована в соответствии со внутренним протоколом маршрутизации автономной системы АС2. Рано или поздно пакет поступит на шлюз 2а, откуда будет переправлен на 1б. Как только пакет покинет 2а, работа системы АС2 с этим пакетом будет завершена.

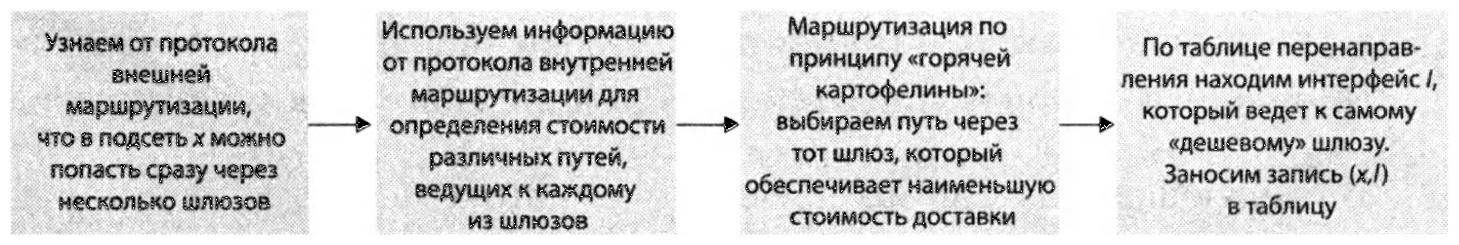


**Рис. 4.32. Пример взаимосвязанных автономных систем**

Итак, проблема тривиальна, если в автономной системе есть всего один канал, ведущий за ее пределы. Но что если таких каналов будет два и более (соответственно, также будет два и более шлюзов)? В таком случае определить направление, по которому требуется пустить пакет, становит­ся существенно сложнее. Возьмем для примера маршрутизатор в системе АС1 и предположим, что его нужно отправить на узел, расположенный вне этой АС. Маршрутизатор должен передать пакет на один из шлюзов, 1б или 1в, но на какой именно? Для решения этой проблемы АС1 должна узнать (1) какие места назначения достижимы через АС2, а какие — через АСЗ и (2) распространить эту информацию о доступности всем марш­рутизаторам в пределах АС1. После этого каждый из маршрутизаторов данной автономной системы сможет соответствующим образом сконфи­гурировать свою таблицу перенаправления, чтобы иметь возможность работать с такими внешними узлами назначения. Две эти задачи — по­лучение информации о достижимости от соседних АС и распространение этой информации среди всех внутренних маршрутизаторов конкретной автономной системы — решаются при помощи **протокола внешней марш­рутизации автономной системы.** Поскольку такой протокол внешней маршрутизации при работе задействует соединения между двумя авто­номными системами, эти две системы должны использовать один и тот же протокол внешней маршрутизации. Так и есть: все автономные системы, работающие в Интернете, применяют для внешней маршрутизации один и тот же протокол BGP4, о котором мы поговорим позднее. Как показано на рис. 4.32, каждый маршрутизатор получает информацию от протокола внутренней маршрутизации автономной системы и от про­токола внешней маршрутизации, а таблица перенаправления заполняет­ся с учетом информации обоих этих протоколов.

В качестве примера давайте рассмотрим подсеть *х* (обозначаемую собственным CIDR-адресом) и предположим, что АС1 узнает по внеш­нему протоколу маршрутизации следующую информацию: подсеть *х* достижима из системы АС3, *но* недостижима из системы АС2. Система АС1 распространяет эту информацию между всеми своими маршрути­заторами. Когда маршрутизатор 1г узнает, что подсеть *х* достижима из АС3, то есть в нее можно попасть через шлюз 1в, то уже по информации из внутреннего протокола маршрутизации 1г определяет интерфейс маршрутизации, который расположен на пути с наименьшей стоимо­стью от 1г к шлюзу 1в. Обозначим этот интерфейс I. В таком случае маршрутизатор 1г может внести в свою таблицу перенаправления за­пись (х,l). Этот пример в целом верно иллюстрируют механизмы, действующие в Интернете, но упрощает их. Ниже будет дано более детальное, хотя и более сложное описание этих механизмов — оно понадобится нам при изучении протокола BGP.

Исходя из предыдущего примера, далее предположим, что АС2 и АСЗ подключены к другим автономным системам, которые не пока­заны на схеме. Также допустим, что по внешнему протоколу маршрути­зации АС1 узнает, что подсеть *х* достижима как из АС2 через шлюз 1б, так и из АС3 через шлюз 1в. Тогда система АС1 распространит эту ин­формацию между всеми своими маршрутизаторами, включая 1г. Марш­рутизатор 1г, чтобы сконфигурировать свою таблицу перенаправления, должен определить, на какой из шлюзов он будет отправлять пакеты, адресованные в подсеть х — на 1б или на 1в. В таких случаях на практике часто применяется маршрутизация по принципу **«горячей картофели­ны»** (hot-potato routing). Другими словами, автономная система должна избавиться от пакета («горячей картофелины») как можно быстрее (точ­нее — с наименьшими затратами). Для этого система обязывает обыч­ный маршрутизатор послать пакет на тот шлюзовой маршрутизатор, до которого пролегает путь с наименьшей стоимостью, если выбирать из альтернативных шлюзов, расположенных между данным маршрутизато­ром и хостом назначения. В контексте рассматриваемого нами примера при использовании принципа горячей картофелины на маршрутизаторе 1г мы будем задействовать информацию внутреннего протокола марш­рутизации данной автономной системы, чтобы определить стоимости путей до 1в и 1б, а затем выберем путь с наименьшей стоимостью. Когда путь выбран, 1б добавляет в свою таблицу перенаправления запись, соот­ветствующую подсети *х.* На рис. 4.33 обобщенно представлены все дей­ствия, которые выполняются на маршрутизаторе 1г для добавления в его таблицу перенаправления новой записи относительно подсети *х.*

****

**Рис. 4.33. Этапы, входе которых хост назначения, расположенный во внешней автономной системе, добавляется в таблицу перенаправления маршрутизатора**

Когда АС узнает о месте назначения из соседней АС, первая АС объ­являет эту информацию о маршрутизации некоторым из соседствующих с ней АС. Например, АС1 узнает от АС2, что подсеть *х* достижима через АС2. После этого АС1 может сообщить АСЗ, что подсеть *х* достижима че­рез АС1. Таким образом, если АСЗ требуется переслать пакет, предназна­ченный для доставки в подсеть *ху* то эта автономная система переправит пакет в АС1, а АС1 в свою очередь перешлет этот пакет в АС2. Как будет показано в разделе о BGP, автономная система может достаточно гибко выбирать те места, о которых она будет объявлять своим соседним авто­номным системам. Это уже вопрос *политик* администрирования, кото­рые чаще обусловлены экономическими, а не техническими факторами.

Как вы помните из раздела 1.5, Интернет представляет собой ие­рархическую структуру взаимосвязанных провайдеров. Итак, как же соотносятся Интернет-провайдер и автономная система? Вы могли бы предположить, что все маршрутизаторы одного провайдера образуют одну автономную систему. На практике зачастую это не так, многие про­вайдеры сегментируют свои сети на множество автономных систем. На­пример, некоторые провайдеры первого эшелона используют каждый по одной автономной системе для всей своей сети; другие подразделяют свои сети на десятки взаимосвязанных автономных систем.

Итак, проблемы масштабирования и административной автономии решаются путем использования автономных систем. В рамках автоном­ной системы все маршрутизаторы используют для внутренней маршру­тизации один и тот же протокол. Автономные системы обмениваются информацией между собой, опять же с применением унифицированно­го внешнего протокола маршрутизации. Проблема масштабирования решается так: маршрутизатору, расположенному внутри автономной системы, требуется информация лишь о других маршрутизаторах, рабо­тающих в этой автономной системе. Проблема административной авто­номии также легко решается, поскольку организация в своих пределах может использовать для внутренней маршрутизации любой протокол на свой выбор. Однако любые две взаимосвязанные автономные системы должны применять один и тот же протокол внешней маршрутизации, по которому они будут обмениваться информацией о достижимости сетей.

В следующем разделе мы изучим два протокола внутренней марш­рутизации в автономных системах (это будут протоколы RIP и OSPF) и протокол внешней маршрутизации BGP, используемые в современном Интернете. Эти практические примеры красиво завершат наш разговор об иерархической маршрутизации.