CHAPTER 20

Unicast Routing

In an internet, the goal of the network layer is to deliver a datagram from its source to its destination or destinations. If a datagram is destined for only one destination (one-to-one delivery), we have *unicast routing*. If the datagram is destined for several destinations (one-to-many delivery), we have *multicast routing*.

In the previous chapters, we have shown that the routing can be possible if a router has a forwarding table to forward a packet to the appropriate next node on its way to the final destination or destinations. To make the forwarding tables of the router, the Internet needs routing protocols that will be active all the time in the background and update the forwarding tables.

In this chapter we discuss only unicast routing; multicast routing will be discussed in the next chapter. This chapter is divided into three sections:

- ❑ The first section introduces the concept of unicast routing and describes the general ideas behind it. The section then describes least-cost routing and least-cost trees.
- ❑ The second section discusses common routing algorithms used in the Internet. The section first describes distance-vector routing. It then describes link-state routing. Finally, it explains path-vector routing.
- ❑ The third section explores unicast-routing protocols corresponding to the unicast-routing algorithms discussed in the second section. This section first defines the structure of the Internet as seen by the unicast-routing protocols. It then describes RIP, a protocol that implements the distance-vector routing algorithm. The section next describes OSPF, a protocol that implements the link-state routing algorithm. Finally, the section describes the BGP, a protocol that implements the path-vector routing algorithm.

20.1 INTRODUCTION

Unicast routing in the Internet, with a large number of routers and a huge number of hosts, can be done only by using hierarchical routing: routing in several steps using different routing algorithms. In this section, we first discuss the general concept of unicast routing in an *internet*: an internetwork made of networks connected by routers. After the routing concepts and algorithms are understood, we show how we can apply them to the Internet using hierarchical routing.

20.1.1 General Idea

In unicast routing, a packet is routed, hop by hop, from its source to its destination by the help of forwarding tables. The source host needs no forwarding table because it delivers its packet to the default router in its local network. The destination host needs no forwarding table either because it receives the packet from its default router in its local network. This means that only the routers that glue together the networks in the internet need forwarding tables. With the above explanation, routing a packet from its source to its destination means routing the packet from a *source router* (the default router of the source host) to a *destination router* (the router connected to the destination network). Although a packet needs to visit the source and the destination routers, the question is what other routers the packet should visit. In other words, there are several routes that a packet can travel from the source to the destination; what must be determined is which route the packet should take.

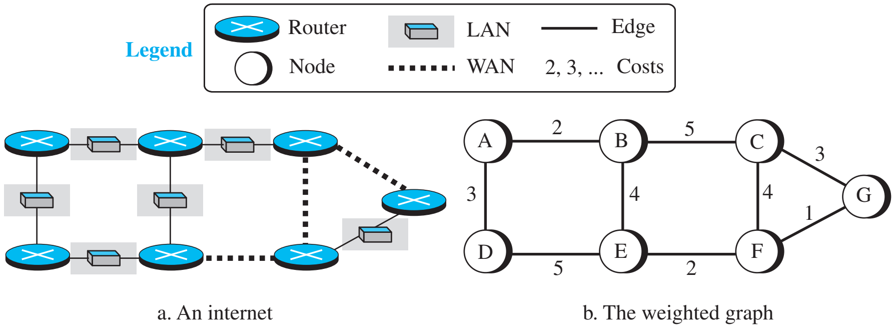
An Internet as a Graph

To find the best route, an internet can be modeled as a *graph*. A graph in computer science is a set of *nodes* and *edges* (lines) that connect the nodes. To model an internet as a graph, we can think of each router as a node and each network between a pair of routers as an edge. An internet is, in fact, modeled as a *weighted graph*, in which each edge is associated with a cost. If a weighted graph is used to represent a geographical area, the nodes can be cities and the edges can be roads connecting the cities; the weights, in this case, are distances between cities. In routing, however, the cost of an edge has a different interpretation in different routing protocols, which we discuss in a later section. For the moment, we assume that there is a cost associated with each edge. If there is no edge between the nodes, the cost is infinity. Figure 20.1 shows how an internet can be modeled as a graph.

20.1.2 Least-Cost Routing

When an internet is modeled as a weighted graph, one of the ways to interpret the *best* route from the source router to the destination router is to find the *least cost* between the two. In other words, the source router chooses a route to the destination router in such a way that the total cost for the route is the least cost among all possible routes. In Figure 20.1, the best route between A and E is A-B-E, with the cost of 6. This means that each router needs to find the least-cost route between itself and all the other routers to be able to route a packet using this criteria.

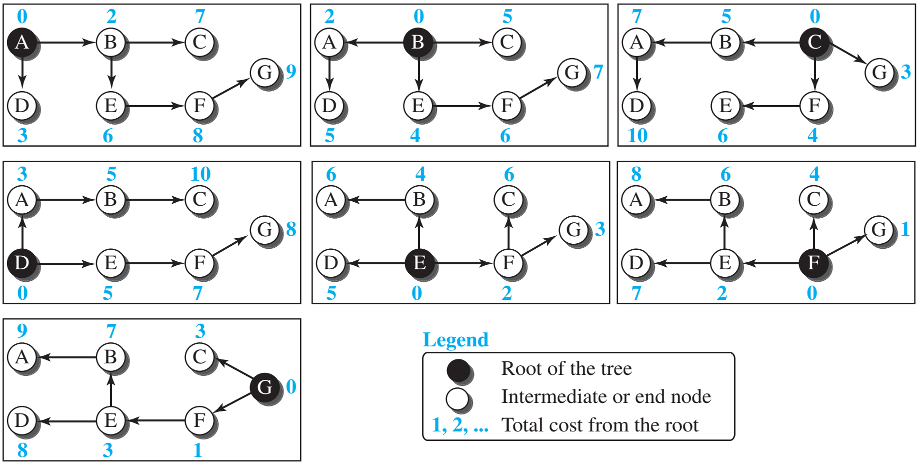
Figure 20.1 *An internet and its graphical representation*



Least-Cost Trees

If there are N routers in an internet, there are $(N - 1)$ least-cost paths from each router to any other router. This means we need $N \times (N - 1)$ least-cost paths for the whole internet. If we have only 10 routers in an internet, we need 90 least-cost paths. A better way to see all of these paths is to combine them in a **least-cost tree**. A least-cost tree is a tree with the source router as the root that spans the whole graph (visits all other nodes) and in which the path between the root and any other node is the shortest. In this way, we can have only one shortest-path tree for each node; we have N least-cost trees for the whole internet. We show how to create a least-cost tree for each node later in this section; for the moment, Figure 20.2 shows the seven least-cost trees for the internet in Figure 20.1.

Figure 20.2 *Least-cost trees for nodes in the internet of Figure 20.1*



The least-cost trees for a weighted graph can have several properties if they are created using consistent criteria.

1. The least-cost route from X to Y in X's tree is the inverse of the least-cost route from Y to X in Y's tree; the cost in both directions is the same. For example, in Figure 20.2, the route from A to F in A's tree is (A → B → E → F), but the route from F to A in F's tree is (F → E → B → A), which is the inverse of the first route. The cost is 8 in each case.
2. Instead of travelling from X to Z using X's tree, we can travel from X to Y using X's tree and continue from Y to Z using Y's tree. For example, in Figure 20.2, we can go from A to G in A's tree using the route (A → B → E → F → G). We can also go from A to E in A's tree (A → B → E) and then continue in E's tree using the route (E → F → G). The combination of the two routes in the second case is the same route as in the first case. The cost in the first case is 9; the cost in the second case is also 9 (6 + 3).

20.2 ROUTING ALGORITHMS

After discussing the general idea behind least-cost trees and the forwarding tables that can be made from them, now we concentrate on the routing algorithms. Several routing algorithms have been designed in the past. The differences between these methods are in the way they interpret the least cost and the way they create the least-cost tree for each node. In this section, we discuss the common algorithms; later we show how a routing protocol in the Internet implements one of these algorithms.

20.2.1 Distance-Vector Routing

The **distance-vector (DV) routing** uses the goal we discussed in the introduction, to find the best route. In distance-vector routing, the first thing each node creates is its own least-cost tree with the rudimentary information it has about its immediate neighbors. The incomplete trees are exchanged between immediate neighbors to make the trees more and more complete and to represent the whole internet. We can say that in distance-vector routing, a router continuously tells all of its neighbors what it knows about the whole internet (although the knowledge can be incomplete).

Before we show how incomplete least-cost trees can be combined to make complete ones, we need to discuss two important topics: the Bellman-Ford equation and the concept of distance vectors, which we cover next.

Bellman-Ford Equation

The heart of distance-vector routing is the famous **Bellman-Ford** equation. This equation is used to find the least cost (shortest distance) between a source node, ***x***, and a destination node, ***y***, through some intermediary nodes (**a, b, c,** . . .) when the costs between the source and the intermediary nodes and the least costs between the intermediary nodes and the destination are given. The following shows the general case in which D_{ij} is the shortest distance and c_{ij} is the cost between nodes ***i*** and ***j***.

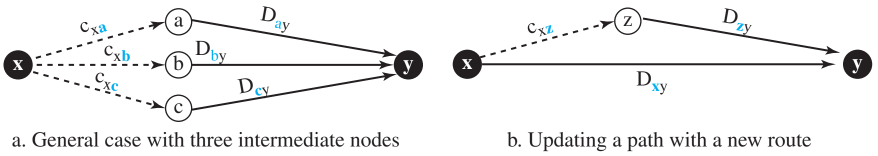
$$D_{xy} = \min\left\{(c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}), \ldots\right\}$$

In distance-vector routing, normally we want to update an existing least cost with a least cost through an intermediary node, such as *z*, if the latter is shorter. In this case, the equation becomes simpler, as shown below:

$$D_{xy} = \min\{D_{xy}, (c_{xz} + D_{zy})\}$$

Figure 20.3 shows the idea graphically for both cases.

Figure 20.3 *Graphical idea behind Bellman-Ford equation*



a. General case with three intermediate nodes

b. Updating a path with a new route

We can say that the Bellman-Ford equation enables us to build a new least-cost path from previously established least-cost paths. In Figure 20.3, we can think of ($a \rightarrow y$), ($b \rightarrow y$), and ($c \rightarrow y$) as previously established least-cost paths and ($x \rightarrow y$) as the new least-cost path. We can even think of this equation as the builder of a new least-cost tree from previously established least-cost trees if we use the equation repeatedly. In other words, the use of this equation in distance-vector routing is a witness that this method also uses least-cost trees, but this use may be in the background.

We will shortly show how we use the Bellman-Ford equation and the concept of distance vectors to build least-cost paths for each node in distance-vector routing, but first we need to discuss the concept of a distance vector.

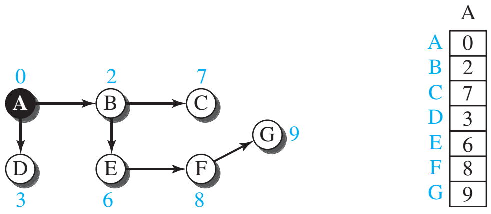
Distance Vectors

The concept of a **distance vector** is the rationale for the name *distance-vector routing*. A least-cost tree is a combination of least-cost paths from the root of the tree to all destinations. These paths are graphically glued together to form the tree. Distance-vector routing unglues these paths and creates a *distance vector*, a one-dimensional array to represent the tree. Figure 20.4 shows the tree for node A in the internet in Figure 20.1 and the corresponding distance vector.

Note that the *name* of the distance vector defines the root, the *indexes* define the destinations, and the *value* of each cell defines the least cost from the root to the destination. A distance vector does not give the path to the destinations as the least-cost tree does; it gives only the least costs to the destinations. Later we show how we can change a distance vector to a forwarding table, but we first need to find all distance vectors for an internet.

We know that a distance vector can represent least-cost paths in a least-cost tree, but the question is how each node in an internet originally creates the corresponding vector. Each node in an internet, when it is booted, creates a very rudimentary distance vector with the minimum information the node can obtain from its neighborhood. The node sends some greeting messages out of its interfaces and discovers the identity of the immediate neighbors and the distance between itself and each neighbor. It then

Figure 20.4 *The distance vector corresponding to a tree*



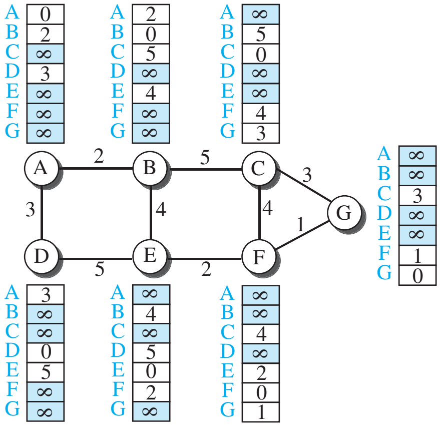
	A
A	0
B	2
C	7
D	3
E	6
F	8
G	9

a. Tree for node A

b. Distance vector for node A

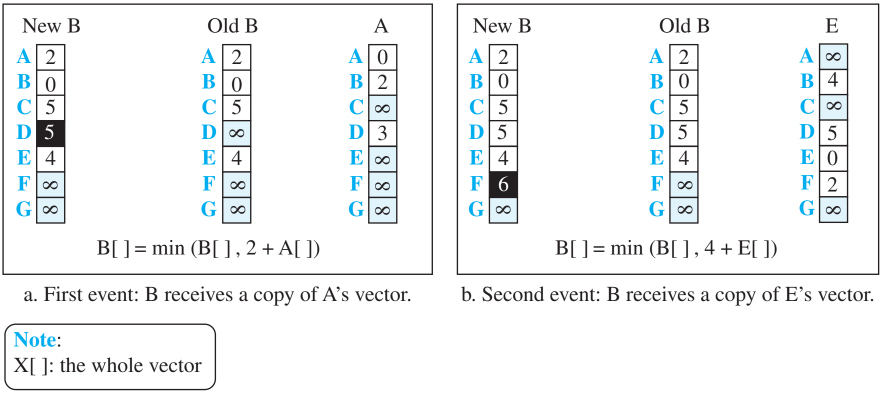
makes a simple distance vector by inserting the discovered distances in the corresponding cells and leaves the value of other cells as infinity. Do these distance vectors represent least-cost paths? They do, considering the limited information a node has. When we know only one distance between two nodes, it is the least cost. Figure 20.5 shows all distance vectors for our internet. However, we need to mention that these vectors are made asynchronously, when the corresponding node has been booted; the existence of all of them in a figure does not mean synchronous creation of them.

Figure 20.5 *The first distance vector for an internet*


A 0, B 2, C ∞, D 3, E ∞, F ∞, G ∞
A 2, B 0, C 5, D ∞, E 4, F ∞, G ∞
A ∞, B 5, C 0, D ∞, E ∞, F 4, G 3
A
B
C
D
E
F
G
2
5
3
4
4
1
5
2
A ∞, B ∞, C 3, D ∞, E ∞, F 1, G 0
A 3, B ∞, C ∞, D 0, E 5, F ∞, G ∞
A ∞, B 4, C ∞, D 5, E 0, F 2, G ∞
A ∞, B ∞, C 4, D ∞, E 2, F 0, G 1

These rudimentary vectors cannot help the internet to effectively forward a packet. For example, node A thinks that it is not connected to node G because the corresponding cell shows the least cost of infinity. To improve these vectors, the nodes in the internet need to help each other by exchanging information. After each node has created its vector, it sends a copy of the vector to all its immediate neighbors. After a node receives a distance vector from a neighbor, it updates its distance vector using the Bellman-Ford equation (second case). However, we need to understand that we need to update, not only one least cost, but *N* of them in which *N* is the number of the nodes in the internet. If we are using a program, we can do this using a loop; if we are showing the concept on paper, we can show the whole vector instead of the *N* separate equations. We show the whole vector instead of seven equations for each update in Figure 20.6. The figure shows two asynchronous events, happening one after another with some time in between. In the first event, node A has sent its vector to node B. Node B updates its vector using the cost $c_{BA} = 2$. In the second event, node E has sent its vector to node B. Node B updates its vector using the cost $c_{EA} = 4$.

Figure 20.6 *Updating distance vectors*



	New B	Old B	A
A	2	2	0
B	0	0	2
C	5	5	∞
D	5	∞	3
E	4	4	∞
F	∞	∞	∞
G	∞	∞	∞

B[ ] = min (B[ ] , 2 + A[ ])

a. First event: B receives a copy of A's vector.

	New B	Old B	E
A	2	2	∞
B	0	0	4
C	5	5	∞
D	5	5	5
E	4	4	0
F	6	∞	2
G	∞	∞	∞

B[ ] = min (B[ ] , 4 + E[ ])

b. Second event: B receives a copy of E's vector.

Note:
X[ ]: the whole vector

After the first event, node B has one improvement in its vector: its least cost to node D has changed from infinity to 5 (via node A). After the second event, node B has one more improvement in its vector; its least cost to node F has changed from infinity to 6 (via node E). We hope that we have convinced the reader that exchanging vectors eventually stabilizes the system and allows all nodes to find the ultimate least cost between themselves and any other node. We need to remember that after updating a node, it immediately sends its updated vector to all neighbors. Even if its neighbors have received the previous vector, the updated one may help more.

Distance-Vector Routing Algorithm

Now we can give a simplified pseudocode for the distance-vector routing algorithm, as shown in Table 20.1. The algorithm is run by its node independently and asynchronously.

Table 20.1 *Distance-Vector Routing Algorithm for a Node*

```
Distance_Vector_Routing ( )
{
    // Initialize (create initial vectors for the node)
    D[myself] = 0
```

Table 20.1 *Distance-Vector Routing Algorithm for a Node (continued)*

```
5       for (y = 1 to N)
6       {
7           if (y is a neighbor)
8               D[y] = c[myself][y]
9           else
10              D[y] = ∞
11      }
12      send vector {D[1], D[2], …, D[N]} to all neighbors
13      // Update (improve the vector with the vector received from a neighbor)
14      repeat (forever)
15      {
16          wait (for a vector $D_w$ from a neighbor $w$ or any change in the link)
17          for (y = 1 to N)
18          {
19              $D[y] = \min [D[y], (c[myself][w] + D_w[y])]$       // Bellman-Ford equation
20          }
21          if (any change in the vector)
22              send vector {D[1], D[2], …, D[N]} to all neighbors
23      }
24    } // End of Distance Vector
```

Lines 4 to 11 initialize the vector for the node. Lines 14 to 23 show how the vector can be updated after receiving a vector from the immediate neighbor. The *for* loop in lines 17 to 20 allows all entries (cells) in the vector to be updated after receiving a new vector. Note that the node sends its vector in line 12, after being initialized, and in line 22, after it is updated.

Count to Infinity

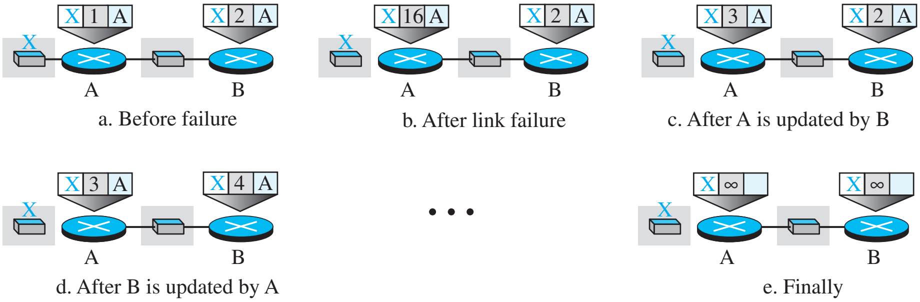
A problem with distance-vector routing is that any decrease in cost (good news) propagates quickly, but any increase in cost (bad news) will propagate slowly. For a routing protocol to work properly, if a link is broken (cost becomes infinity), every other router should be aware of it immediately, but in distance-vector routing, this takes some time. The problem is referred to as *count to infinity*. It sometimes takes several updates before the cost for a broken link is recorded as infinity by all routers.

Two-Node Loop

One example of count to infinity is the two-node loop problem. To understand the problem, let us look at the scenario depicted in Figure 20.7.

The figure shows a system with three nodes. We have shown only the portions of the forwarding table needed for our discussion. At the beginning, both nodes A and B

Figure 20.7 *Two-node instability*



know how to reach node X. But suddenly, the link between A and X fails. Node A changes its table. If A can send its table to B immediately, everything is fine. However, the system becomes unstable if B sends its forwarding table to A before receiving A's forwarding table. Node A receives the update and, assuming that B has found a way to reach X, immediately updates its forwarding table. Now A sends its new update to B. Now B thinks that something has been changed around A and updates its forwarding table. The cost of reaching X increases gradually until it reaches infinity. At this moment, both A and B know that X cannot be reached. However, during this time the system is not stable. Node A thinks that the route to X is via B; node B thinks that the route to X is via A. If A receives a packet destined for X, the packet goes to B and then comes back to A. Similarly, if B receives a packet destined for X, it goes to A and comes back to B. Packets bounce between A and B, creating a two-node loop problem. A few solutions have been proposed for instability of this kind.

Split Horizon

One solution to instability is called ***split horizon***. In this strategy, instead of flooding the table through each interface, each node sends only part of its table through each interface. If, according to its table, node B thinks that the optimum route to reach X is via A, it does not need to advertise this piece of information to A; the information has come from A (A already knows). Taking information from node A, modifying it, and sending it back to node A is what creates the confusion. In our scenario, node B eliminates the last line of its forwarding table before it sends it to A. In this case, node A keeps the value of infinity as the distance to X. Later, when node A sends its forwarding table to B, node B also corrects its forwarding table. The system becomes stable after the first update: both node A and node B know that X is not reachable.

Poison Reverse

Using the split-horizon strategy has one drawback. Normally, the corresponding protocol uses a timer, and if there is no news about a route, the node deletes the route from its table. When node B in the previous scenario eliminates the route to X from its advertisement to A, node A cannot guess whether this is due to the split-horizon strategy (the source of information was A) or because B has not received any news about X recently. In the **poison reverse** strategy B can still advertise the value for X, but if the source of information is A, it can replace the distance with infinity as a warning: “Do not use this value; what I know about this route comes from you.”

Three-Node Instability

The two-node instability can be avoided using split horizon combined with poison reverse. However, if the instability is between three nodes, stability cannot be guaranteed.

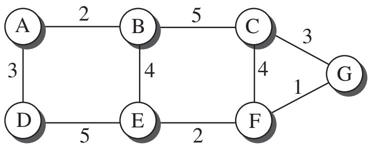
20.2.2 Link-State Routing

A routing algorithm that directly follows our discussion for creating least-cost trees and forwarding tables is **link-state (LS) routing.** This method uses the term *link-state* to define the characteristic of a link (an edge) that represents a network in the internet. In this algorithm the cost associated with an edge defines the state of the link. Links with lower costs are preferred to links with higher costs; if the cost of a link is infinity, it means that the link does not exist or has been broken.

Link-State Database (LSDB)

To create a least-cost tree with this method, each node needs to have a complete *map* of the network, which means it needs to know the state of each link. The collection of states for all links is called the ***link-state database (LSDB).*** There is only one LSDB for the whole internet; each node needs to have a duplicate of it to be able to create the least-cost tree. Figure 20.8 shows an example of an LSDB for the graph in Figure 20.1. The LSDB can be represented as a two-dimensional array(matrix) in which the value of each cell defines the cost of the corresponding link.

Figure 20.8 *Example of a link-state database*



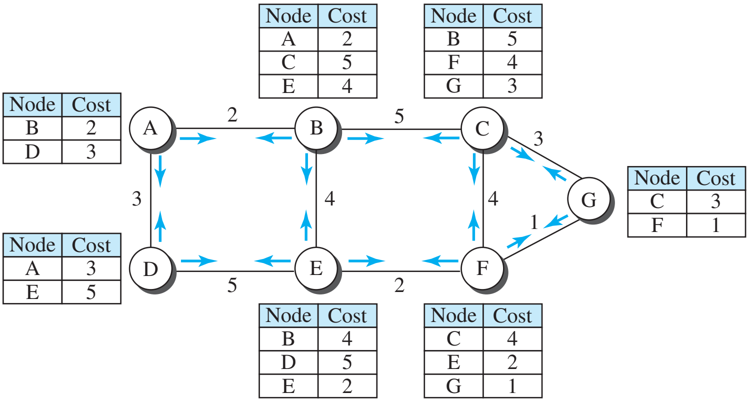
a. The weighted graph

	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

b. Link state database

Now the question is how each node can create this LSDB that contains information about the whole internet. This can be done by a process called **flooding.** Each node can send some greeting messages to all its immediate neighbors (those nodes to which it is connected directly) to collect two pieces of information for each neighboring node: the identity of the node and the cost of the link. The combination of these two pieces of information is called the *LS packet* (LSP); the LSP is sent out of each interface, as shown in Figure 20.9 for our internet in Figure 20.1. When a node receives an LSP from one of its interfaces, it compares the LSP with the copy it may already have. If the newly arrived LSP is older than the one it has (found by checking the sequence number), it discards the LSP. If it is newer or the first one received, the node discards the old LSP (if there is one) and keeps the received one. It then sends a copy of it out of each interface except the one from which the packet arrived. This guarantees that flooding stops somewhere in the network (where a node has only one interface). We need to convince ourselves that, after receiving all new LSPs, each node creates the comprehensive LSDB as shown in Figure 20.9. This LSDB is the same for each node and shows the whole map of the internet. In other words, a node can make the whole map if it needs to, using this LSDB.

Figure 20.9 *LSPs created and sent out by each node to build LSDB*



We can compare the link-state routing algorithm with the distance-vector routing algorithm. In the distance-vector routing algorithm, each router tells its neighbors what it knows about the whole internet; in the link-state routing algorithm, each router tells the whole internet what it knows about its neighbors.

Formation of Least-Cost Trees

To create a least-cost tree for itself, using the shared LSDB, each node needs to run the famous **Dijkstra Algorithm.** This iterative algorithm uses the following steps:

1. The node chooses itself as the root of the tree, creating a tree with a single node, and sets the total cost of each node based on the information in the LSDB.
2. The node selects one node, among all nodes not in the tree, which is closest to the root, and adds this to the tree. After this node is added to the tree, the cost of all other nodes not in the tree needs to be updated because the paths may have been changed.
3. The node repeats step 2 until all nodes are added to the tree.

We need to convince ourselves that the above three steps finally create the least-cost tree. Table 20.2 shows a simplified version of Dijkstra's algorithm.

Table 20.2 *Dijkstra's Algorithm*

```
Dijkstra's Algorithm ( )
{
    // Initialization
    Tree = {root}                    // Tree is made only of the root
```

Table 20.2 *Dijkstra's Algorithm (continued)*

```
5       for (y = 1 to N)                          // N is the number of nodes
6       {
7           if (y is the root)
8               D[y] = 0                          // D[y] is shortest distance from root to node y
9           else if (y is a neighbor)
10              D[y] = c[root][y]                 // c[x][y] is cost between nodes x and y in LSDB
11          else
12              D[y] = ∞
13      }
14      // Calculation
15      repeat
16      {
17          find a node w, with D[w] minimum among all nodes not in the Tree
18          Tree = Tree ∪ {w}                     // Add w to tree
19          // Update distances for all neighbors of w
20          for (every node x, which is a neighbor of w and not in the Tree)
21          {
22              D[x] = min{D[x], (D[w] + c[w][x])}
23          }
24      } until (all nodes included in the Tree)
25  } // End of Dijkstra
```

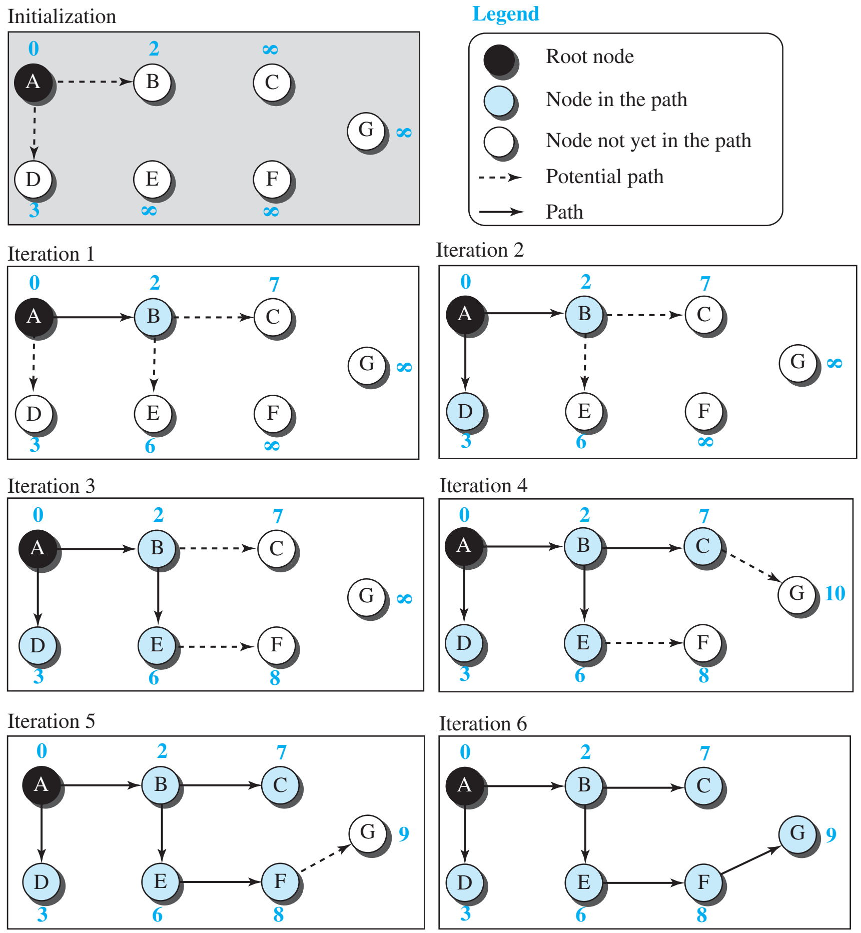
Lines 4 to 13 implement step 1 in the algorithm. Lines 16 to 23 implement step 2 in the algorithm. Step 2 is repeated until all nodes are added to the tree.

Figure 20.10 shows the formation of the least-cost tree for the graph in Figure 20.8 using Dijkstra's algorithm. We need to go through an initialization step and six iterations to find the least-cost tree.

20.2.3 Path-Vector Routing

Both link-state and distance-vector routing are based on the least-cost goal. However, there are instances where this goal is not the priority. For example, assume that there are some routers in the internet that a sender wants to prevent its packets from going through. For example, a router may belong to an organization that does not provide enough security or it may belong to a commercial rival of the sender which might inspect the packets for obtaining information. Least-cost routing does not prevent a packet from passing through an area when that area is in the least-cost path. In other words, the least-cost goal, applied by LS or DV routing, does not allow a sender to apply specific policies to the route a packet may take. Aside from safety and security, there are occasions, as discussed in the next section, in which the goal of routing is merely reachability: to allow the packet to reach its destination more efficiently without assigning costs to the route.

Figure 20.10 *Least-cost tree*



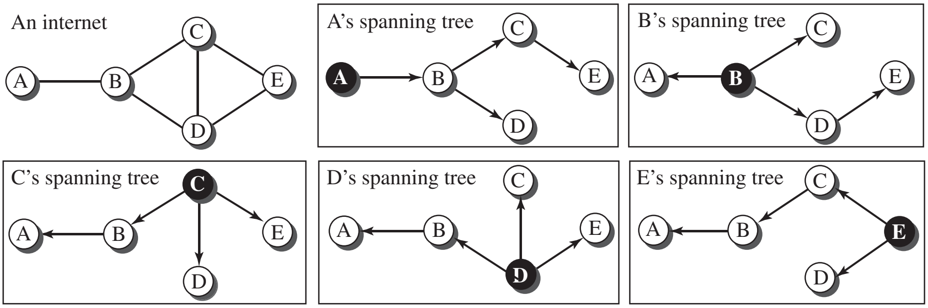
To respond to these demands, a third routing algorithm, called **path-vector (PV) routing** has been devised. Path-vector routing does not have the drawbacks of LS or DV routing as described above because it is not based on least-cost routing. The best route is determined by the source using the policy it imposes on the route. In other words, the source can control the path. Although path-vector routing is not actually used in an internet, and is mostly designed to route a packet between ISPs, we discuss the principle of this method in this section as though applied to an internet. In the next section, we show how it is used in the Internet.

Spanning Trees

In path-vector routing, the path from a source to all destinations is also determined by the *best* spanning tree. The best spanning tree, however, is not the least-cost tree; it is the tree determined by the source when it imposes its own policy. If there is more than one route to a destination, the source can choose the route that meets its policy best. A source may apply several policies at the same time. One of the common policies uses the minimum number of nodes to be visited (something similar to least-cost). Another common policy is to avoid some nodes as the middle node in a route.

Figure 20.11 shows a small internet with only five nodes. Each source has created its own spanning tree that meets its policy. The policy imposed by all sources is to use the minimum number of nodes to reach a destination. The spanning tree selected by A and E is such that the communication does not pass through D as a middle node. Similarly, the spanning tree selected by B is such that the communication does not pass through C as a middle node.

Figure 20.11 *Spanning trees in path-vector routing*



Creation of Spanning Trees

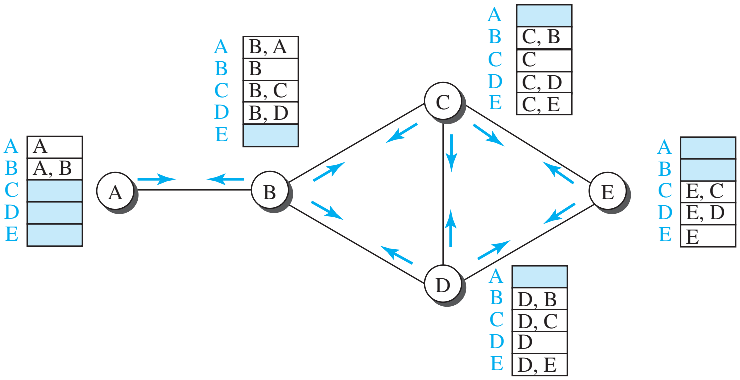
Path-vector routing, like distance-vector routing, is an asynchronous and distributed routing algorithm. The spanning trees are made, gradually and asynchronously, by each node. When a node is booted, it creates a *path vector* based on the information it can obtain about its immediate neighbor. A node sends greeting messages to its immediate neighbors to collect these pieces of information. Figure 20.12 shows all of these path vectors for our internet in Figure 20.11. Note, however, that we do not mean that all of these tables are created simultaneously; they are created when each node is booted. The figure also shows how these path vectors are sent to immediate neighbors after they have been created (arrows).

Each node, after the creation of the initial path vector, sends it to all its immediate neighbors. Each node, when it receives a path vector from a neighbor, updates its path vector using an equation similar to the Bellman-Ford, but applying its own policy instead of looking for the least cost. We can define this equation as

$$\text{Path}(\boldsymbol{x}, \boldsymbol{y}) = \text{best}\ \{\text{Path}(x, y),\ [(\boldsymbol{x} + \text{Path}(\mathbf{v}, \boldsymbol{y})]\} \qquad \text{for all } \mathbf{v}\text{'s in the internet.}$$

In this equation, the operator (+) means to add $\boldsymbol{x}$ to the beginning of the path. We also need to be cautious to avoid adding a node to an empty path because an empty path means one that does not exist.

Figure 20.12 *Path vectors made at booting time*



	A
A	A
B	A, B
C	
D	
E	

	B
A	B, A
B	B
C	B, C
D	B, D
E	

	C
A	
B	C, B
C	C
D	C, D
E	C, E

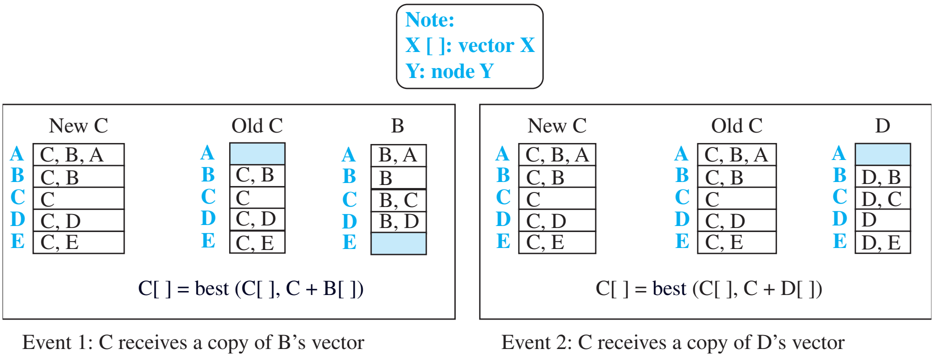
	D
A	
B	D, B
C	D, C
D	D
E	D, E

	E
A	
B	
C	E, C
D	E, D
E	E

The policy is defined by selecting the *best* of multiple paths. Path-vector routing also imposes one more condition on this equation: If Path (v, ***y***) includes ***x***, that path is discarded to avoid a loop in the path. In other words, ***x*** does not want to visit itself when it selects a path to ***y***.

Figure 20.13 shows the path vector of node C after two events. In the first event, node C receives a copy of B's vector, which improves its vector: now it knows how to reach node A. In the second event, node C receives a copy of D's vector, which does not change its vector. As a matter of fact the vector for node C after the first event is stabilized and serves as its forwarding table.

Figure 20.13 *Updating path vectors*



Note:
X [ ]: vector X
Y: node Y

	New C	Old C	B
A	C, B, A		B, A
B	C, B	C, B	B
C	C	C	B, C
D	C, D	C, D	B, D
E	C, E	C, E	

C[ ] = best (C[ ], C + B[ ])

Event 1: C receives a copy of B's vector

	New C	Old C	D
A	C, B, A	C, B, A	
B	C, B	C, B	D, B
C	C	C	D, C
D	C, D	C, D	D
E	C, E	C, E	D, E

C[ ] = best (C[ ], C + D[ ])

Event 2: C receives a copy of D's vector

Path-Vector Algorithm

Based on the initialization process and the equation used in updating each forwarding table after receiving path vectors from neighbors, we can write a simplified version of the path vector algorithm as shown in Table 20.3.

Table 20.3 *Path-vector algorithm for a node*

```
 1  Path_Vector_Routing ( )
 2  {
 3      // Initialization
 4      for ($y$ = 1 to N)
 5      {
 6          if ($y$ is myself)
 7              Path [$y$] = myself
 8          else if (y is a neighbor)
 9              Path [$y$] = myself + neighbor node
10          else
11              Path [$y$] = empty
12      }
13      Send vector {Path[1], Path[2], …, Path[$y$]} to all neighbors
14      // Update
15      repeat (forever)
16      {
17          wait (for a vector $\text{Path}_w$ from a neighbor $w$)
18          for ($y$ = 1 to N)
19          {
20              if ($\text{Path}_w$ includes myself)
21                  discard the path                    // Avoid any loop
22              else
23                  Path[$y$] = best {Path[$y$], (myself + $\text{Path}_w$[$y$])}
24          }
25          If (there is a change in the vector)
26              Send vector {Path[1], Path[2], …, Path[$y$]} to all neighbors
27      }
28  } // End of Path Vector
```

Lines 4 to 12 show the initialization for the node. Lines 17 to 24 show how the node updates its vector after receiving a vector from the neighbor. The update process is repeated forever. We can see the similarities between this algorithm and the DV algorithm.

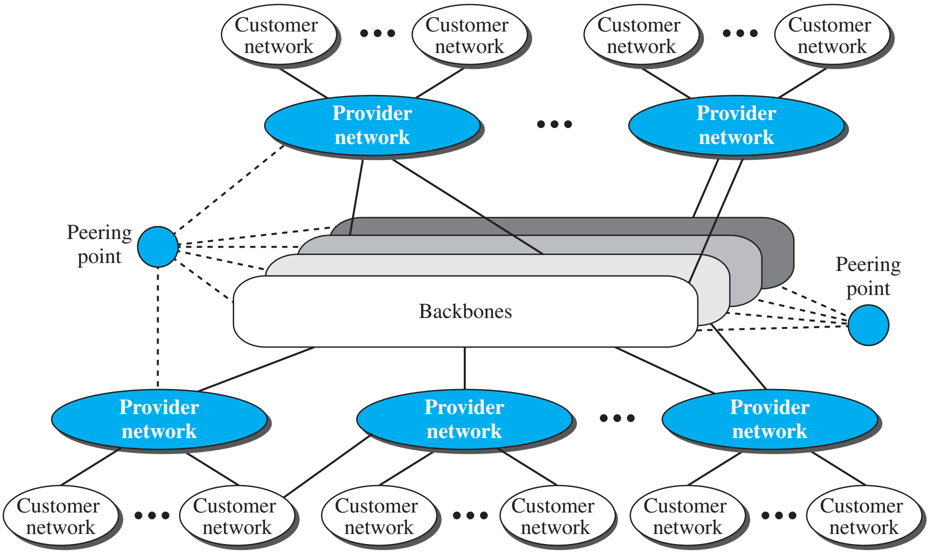
20.3 UNICAST ROUTING PROTOCOLS

In the previous section, we discussed unicast routing algorithms; in this section, we discuss unicast routing protocols used in the Internet. Although three protocols we discuss here are based on the corresponding algorithms we discussed before, a protocol is more than an algorithm. A protocol needs to define its domain of operation, the messages exchanged, communication between routers, and interaction with protocols in other domains. After an introduction, we discuss three common protocols used in the Internet: Routing Information Protocol (RIP), based on the distance-vector algorithm, Open Shortest Path First (OSPF), based on the link-state algorithm, and Border Gateway Protocol (BGP), based on the path-vector algorithm.

20.3.1 Internet Structure

Before discussing unicast routing protocols, we need to understand the structure of today's Internet. The Internet has changed from a tree-like structure, with a single backbone, to a multi-backbone structure run by different private corporations today. Although it is difficult to give a general view of the Internet today, we can say that the Internet has a structure similar to what is shown in Figure 20.14.

Figure 20.14 *Internet structure*



There are several *backbones* run by private communication companies that provide global connectivity. These backbones are connected by some *peering points* that allow connectivity between backbones. At a lower level, there are some *provider networks* that use the backbones for global connectivity but provide services to Internet customers.

Finally, there are some customer *networks* that use the services provided by the provider networks. Any of these three entities (backbone, provider network, or customer network) can be called an Internet Service Provider or ISP. They provide services, but at different levels.

Hierarchical Routing

The Internet today is made of a huge number of networks and routers that connect them. It is obvious that routing in the Internet cannot be done using a single protocol for two reasons: a scalability problem and an administrative issue. *Scalability problem* means that the size of the forwarding tables becomes huge, searching for a destination in a forwarding table becomes time-consuming, and updating creates a huge amount of traffic. The *administrative issue* is related to the Internet structure described in Figure 20.14. As the figure shows, each ISP is run by an administrative authority. The administrator needs to have control in its system. The organization must be able to use as many subnets and routers as it needs, may desire that the routers be from a particular manufacturer, may wish to run a specific routing algorithm to meet the needs of the organization, and may want to impose some policy on the traffic passing through its ISP.

Hierarchical routing means considering each ISP as an **autonomous system (AS).** Each AS can run a routing protocol that meets its needs, but the global Internet runs a global protocol to glue all ASs together. The routing protocol run in each AS is referred to as *intra-AS routing protocol*, *intradomain routing protocol*, or *interior gateway protocol (IGP)*; the global routing protocol is referred to as *inter-AS routing protocol, interdomain routing protocol,* or *exterior gateway protocol (EGP)*. We can have several intradomain routing protocols, and each AS is free to choose one, but it should be clear that we should have only one interdomain protocol that handles routing between these entities. Presently, the two common intradomain routing protocols are RIP and OSPF; the only interdomain routing protocol is BGP. The situation may change when we move to IPv6.

Autonomous Systems

As we said before, each ISP is an autonomous system when it comes to managing networks and routers under its control. Although we may have small, medium-size, and large ASs, each AS is given an autonomous number (ASN) by the ICANN. Each ASN is a 16-bit unsigned integer that uniquely defines an AS. The autonomous systems, however, are not categorized according to their size; they are categorized according to the way they are connected to other ASs. We have stub ASs, multihomed ASs, and transient ASs. The type, as we see will later, affects the operation of the interdomain routing protocol in relation to that AS.

- ❑ ***Stub AS.*** A stub AS has only one connection to another AS. The data traffic can be either initiated or terminated in a stub AS; the data cannot pass through it. A good example of a stub AS is the customer network, which is either the source or the sink of data.
- ❑ ***Multihomed AS.*** A multihomed AS can have more than one connection to other ASs, but it does not allow data traffic to pass through it. A good example of such an AS is some of the customer ASs that may use the services of more than one provider network, but their policy does not allow data to be passed through them.

- ***Transient AS.*** A transient AS is connected to more than one other AS and also allows the traffic to pass through. The provider networks and the backbone are good examples of transient ASs.

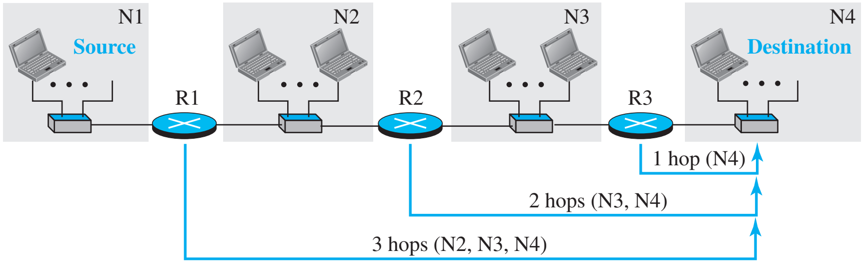
20.3.2 Routing Information Protocol (RIP)

The **Routing Information Protocol** (**RIP**) is one of the most widely used intradomain routing protocols based on the distance-vector routing algorithm we described earlier. RIP was started as part of the Xerox Network System (XNS), but it was the Berkeley Software Distribution (BSD) version of UNIX that helped make the use of RIP widespread.

Hop Count

A router in this protocol basically implements the distance-vector routing algorithm shown in Table 20.1. However, the algorithm has been modified as described below. First, since a router in an AS needs to know how to forward a packet to different networks (subnets) in an AS, RIP routers advertise the cost of reaching different networks instead of reaching other nodes in a theoretical graph. In other words, the cost is defined between a router and the network in which the destination host is located. Second, to make the implementation of the cost simpler (independent from performance factors of the routers and links, such as delay, bandwidth, and so on), the cost is defined as the number of hops, which means the number of networks (subnets) a packet needs to travel through from the source router to the final destination host. Note that the network in which the source host is connected is not counted in this calculation because the source host does not use a forwarding table; the packet is delivered to the default router. Figure 20.15 shows the concept of hop count advertised by three routers from a source host to a destination host. In RIP, the maximum cost of a path can be 15, which means 16 is considered as infinity (no connection). For this reason, RIP can be used only in autonomous systems in which the diameter of the AS is not more than 15 hops.

Figure 20.15 *Hop counts in RIP*



Forwarding Tables

Although the distance-vector algorithm we discussed in the previous section is concerned with exchanging distance vectors between neighboring nodes, the routers in an autonomous system need to keep forwarding tables to forward packets to their destination networks. A forwarding table in RIP is a three-column table in which the first column is the address of the destination network, the second column is the address of the next router to which the packet should be forwarded, and the third column is the cost (the number of hops) to reach the destination network. Figure 20.16 shows the three forwarding tables for the routers in Figure 20.15. Note that the first and the third columns together convey the same information as does a distance vector, but the cost shows the number of hops to the destination networks.

Figure 20.16 *Forwarding tables*

Forwarding table for R1

Destination network	Next router	Cost in hops
N1	——	1
N2	——	1
N3	R2	2
N4	R2	3

Forwarding table for R2

Destination network	Next router	Cost in hops
N1	R1	2
N2	——	1
N3	——	1
N4	R3	2

Forwarding table for R3

Destination network	Next router	Cost in hops
N1	R2	3
N2	R2	2
N3	——	1
N4	——	1

Although a forwarding table in RIP defines only the next router in the second column, it gives the information about the whole least-cost tree based on the second property of these trees, discussed in the previous section. For example, R1 defines that the next router for the path to N4 is R2; R2 defines that the next router to N4 is R3; R3 defines that there is no next router for this path. The tree is then R1 → R2 → R3 → N4.

A question often asked about the forwarding table is what the use of the third column is. The third column is not needed for forwarding the packet, but it is needed for updating the forwarding table when there is a change in the route, as we will see shortly.

RIP Implementation

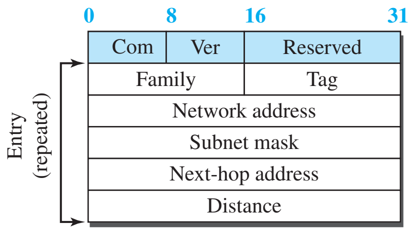
RIP is implemented as a process that uses the service of UDP on the well-known port number 520. In BSD, RIP is a daemon process (a process running in the background), named *routed* (abbreviation for *route daemon* and pronounced *route-dee)*. This means that, although RIP is a routing protocol to help IP route its datagrams through the AS, the RIP messages are encapsulated inside UDP user datagrams, which in turn are encapsulated inside IP datagrams. In other words, RIP runs at the application layer, but creates forwarding tables for IP at the network later.

RIP has gone through two versions: RIP-1 and RIP-2. The second version is backward compatible with the first section; it allows the use of more information in the RIP messages that were set to 0 in the first version. We discuss only RIP-2 in this section.

RIP Messages

Two RIP processes, a client and a server, like any other processes, need to exchange messages. RIP-2 defines the format of the message, as shown in Figure 20.17. Part of the message, which we call *entry*, can be repeated as needed in a message. Each entry carries the information related to one line in the forwarding table of the router that sends the message.

Figure 20.17 *RIP message format*



Fields

Com: Command, request (1), response (2)
Ver: Version, current version is 2
Family: Family of protocol, for TCP/IP value is 2
Tag: Information about autonomous system
Network address: Destination address
Subnet mask: Prefix length
Next-hop address: Address length
Distance: Number of hops to the destination

RIP has two types of messages: request and response. A request message is sent by a router that has just come up or by a router that has some time-out entries. A request message can ask about specific entries or all entries. A response (or update) message can be either solicited or unsolicited. A solicited response message is sent only in answer to a request message. It contains information about the destination specified in the corresponding request message. An unsolicited response message, on the other hand, is sent periodically, every 30 seconds or when there is a change in the forwarding table.

RIP Algorithm

RIP implements the same algorithm as the distance-vector routing algorithm we discussed in the previous section. However, some changes need to be made to the algorithm to enable a router to update its forwarding table:

- Instead of sending only distance vectors, a router needs to send the whole contents of its forwarding table in a response message.
- The receiver adds one hop to each cost and changes the next router field to the address of the sending router. We call each route in the modified forwarding table the *received route* and each route in the old forwarding table the *old route*. The received router selects the old routes as the new ones except in the following three cases:

1. If the received route does not exist in the old forwarding table, it should be added to the route.
2. If the cost of the received route is lower than the cost of the old one, the received route should be selected as the new one.
3. If the cost of the received route is higher than the cost of the old one, but the value of the next router is the same in both routes, the received route should be selected as the new one. This is the case where the route was actually advertised

by the same router in the past, but now the situation has been changed. For example, suppose a neighbor has previously advertised a route to a destination with cost 3, but now there is no path between this neighbor and that destination. The neighbor advertises this destination with cost value infinity (16 in RIP). The receiving router must not ignore this value even though its old route has a lower cost to the same destination.

- The new forwarding table needs to be sorted according to the destination route (mostly using the longest prefix first).

Example 20.1

Figure 20.18 shows a more realistic example of the operation of RIP in an autonomous system. First, the figure shows all forwarding tables after all routers have been booted. Then we show changes in some tables when some update messages have been exchanged. Finally, we show the stabilized forwarding tables when there is no more change.

Timers in RIP

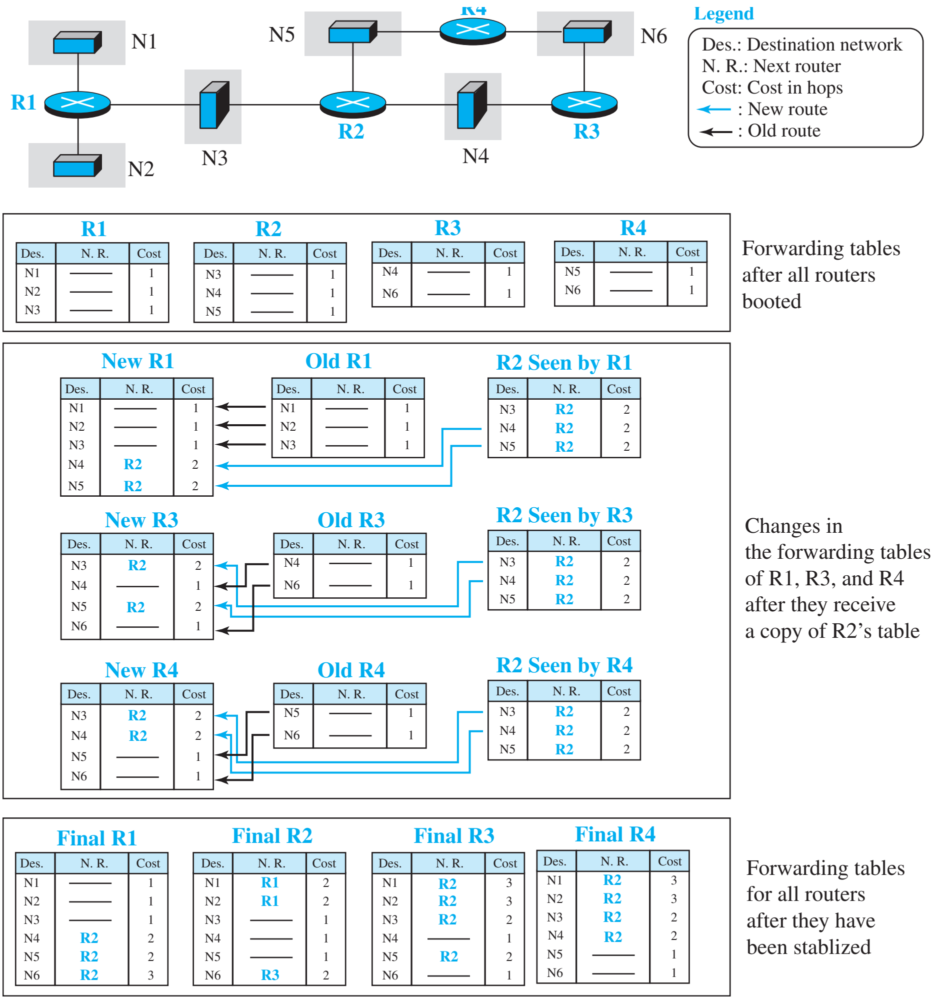
RIP uses three timers to support its operation. The *periodic timer* controls the advertising of regular update messages. Each router has one periodic timer that is randomly set to a number between 25 and 35 seconds (to prevent all routers sending their messages at the same time and creating excess traffic). The timer counts down; when zero is reached, the update message is sent, and the timer is randomly set once again. The *expiration timer* governs the validity of a route. When a router receives update information for a route, the expiration timer is set to 180 seconds for that particular route. Every time a new update for the route is received, the timer is reset. If there is a problem on an internet and no update is received within the allotted 180 seconds, the route is considered expired and the hop count of the route is set to 16, which means the destination is unreachable. Every route has its own expiration timer. The *garbage collection timer* is used to purge a route from the forwarding table. When the information about a route becomes invalid, the router does not immediately purge that route from its table. Instead, it continues to advertise the route with a metric value of 16. At the same time, a garbage collection timer is set to 120 seconds for that route. When the count reaches zero, the route is purged from the table. This timer allows neighbors to become aware of the invalidity of a route prior to purging.

Performance

Before ending this section, let us briefly discuss the performance of RIP:

- ***Update Messages.*** The update messages in RIP have a very simple format and are sent only to neighbors; they are local. They do not normally create traffic because the routers try to avoid sending them at the same time.
- ***Convergence of Forwarding Tables.*** RIP uses the distance-vector algorithm, which can converge slowly if the domain is large, but, since RIP allows only 15 hops in a domain (16 is considered as infinity), there is normally no problem in convergence. The only problems that may slow down convergence are count-to-infinity and loops created in the domain; use of poison-reverse and split-horizon strategies added to the RIP extension may alleviate the situation.

Figure 20.18 *Example of an autonomous system using RIP*



R1

Des.	N. R.	Cost
N1	—	1
N2	—	1
N3	—	1

R2

Des.	N. R.	Cost
N3	—	1
N4	—	1
N5	—	1

R3

Des.	N. R.	Cost
N4	—	1
N6	—	1

R4

Des.	N. R.	Cost
N5	—	1
N6	—	1

Forwarding tables after all routers booted

New R1

Des.	N. R.	Cost
N1	—	1
N2	—	1
N3	—	1
N4	R2	2
N5	R2	2

Old R1

Des.	N. R.	Cost
N1	—	1
N2	—	1
N3	—	1

R2 Seen by R1

Des.	N. R.	Cost
N3	R2	2
N4	R2	2
N5	R2	2

New R3

Des.	N. R.	Cost
N3	R2	2
N4	—	1
N5	R2	2
N6	—	1

Old R3

Des.	N. R.	Cost
N4	—	1
N6	—	1

R2 Seen by R3

Des.	N. R.	Cost
N3	R2	2
N4	R2	2
N5	R2	2

New R4

Des.	N. R.	Cost
N3	R2	2
N4	R2	2
N5	—	1
N6	—	1

Old R4

Des.	N. R.	Cost
N5	—	1
N6	—	1

R2 Seen by R4

Des.	N. R.	Cost
N3	R2	2
N4	R2	2
N5	R2	2

Changes in the forwarding tables of R1, R3, and R4 after they receive a copy of R2's table

Final R1

Des.	N. R.	Cost
N1	—	1
N2	—	1
N3	—	1
N4	R2	2
N5	R2	2
N6	R2	3

Final R2

Des.	N. R.	Cost
N1	R1	2
N2	R1	2
N3	—	1
N4	—	1
N5	—	1
N6	R3	2

Final R3

Des.	N. R.	Cost
N1	R2	3
N2	R2	3
N3	R2	2
N4	—	1
N5	R2	2
N6	—	1

Final R4

Des.	N. R.	Cost
N1	R2	3
N2	R2	3
N3	R2	2
N4	R2	2
N5	—	1
N6	—	1

Forwarding tables for all routers after they have been stablized

- ❑ ***Robustness.*** As we said before, distance-vector routing is based on the concept that each router sends what it knows about the whole domain to its neighbors. This means that the calculation of the forwarding table depends on information received from immediate neighbors, which in turn receive their information from their own neighbors. If there is a failure or corruption in one router, the problem will be propagated to all routers and the forwarding in each router will be affected.

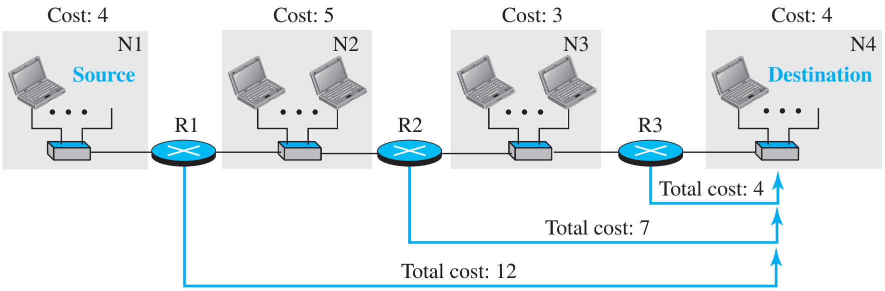
20.3.3 Open Shortest Path First (OSPF)

Open Shortest Path First (**OSPF**) is also an intradomain routing protocol like RIP, but it is based on the link-state routing protocol we described earlier in the chapter. OSPF is an *open* protocol, which means that the specification is a public document.

Metric

In OSPF, like RIP, the cost of reaching a destination from the host is calculated from the source router to the destination network. However, each link (network) can be assigned a weight based on the throughput, round-trip time, reliability, and so on. An administration can also decide to use the hop count as the cost. An interesting point about the cost in OSPF is that different service types (TOSs) can have different weights as the cost. Figure 20.19 shows the idea of the cost from a router to the destination host network. We can compare the figure with Figure 20.15 for the RIP.

Figure 20.19 *Metric in OSPF*



Forwarding Tables

Each OSPF router can create a forwarding table after finding the shortest-path tree between itself and the destination using Dijkstra's algorithm, described earlier in the chapter. Figure 20.20 shows the forwarding tables for the simple AS in Figure 20.19. Comparing the forwarding tables for the OSPF and RIP in the same AS, we find that the only difference is the cost values. In other words, if we use the hop count for OSPF, the tables will be exactly the same. The reason for this consistency is that both protocols use the shortest-path trees to define the best route from a source to a destination.

Areas

Compared with RIP, which is normally used in small ASs, OSPF was designed to be able to handle routing in a small or large autonomous system. However, the formation of shortest-path trees in OSPF requires that all routers flood the whole AS with their LSPs to create the global LSDB. Although this may not create a problem in a small AS, it may have created a huge volume of traffic in a large AS. To prevent this, the AS needs to be divided into small sections called *areas*. Each area acts as a small independent domain for flooding LSPs. In other words, OSPF uses another level of hierarchy in routing: the first level is the autonomous system, the second is the area.

Figure 20.20 *Forwarding tables in OSPF*

Forwarding table for R1

Destination network	Next router	Cost
N1	——	4
N2	——	5
N3	R2	8
N4	R2	12

Forwarding table for R2

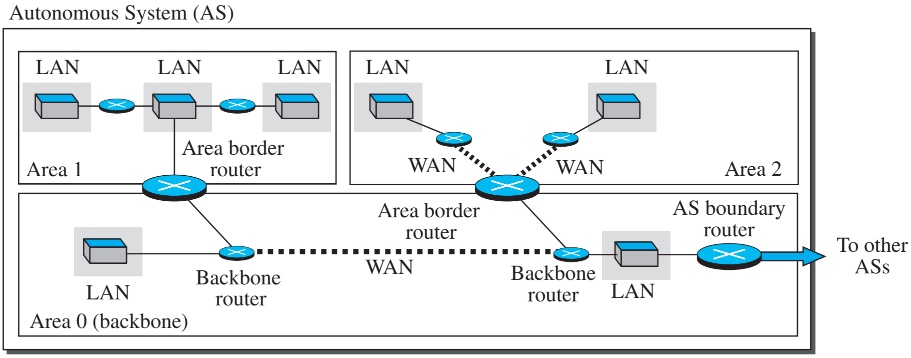
Destination network	Next router	Cost
N1	R1	9
N2	——	5
N3	——	3
N4	R3	7

Forwarding table for R3

Destination network	Next router	Cost
N1	R2	12
N2	R2	8
N3	——	3
N4	——	4

However, each router in an area needs to know the information about the link states not only in its area but also in other areas. For this reason, one of the areas in the AS is designated as the *backbone area,* responsible for gluing the areas together. The routers in the backbone area are responsible for passing the information collected by each area to all other areas. In this way, a router in an area can receive all LSPs generated in other areas. For the purpose of communication, each area has an area identification. The area identification of the backbone is zero. Figure 20.21 shows an autonomous system and its areas.

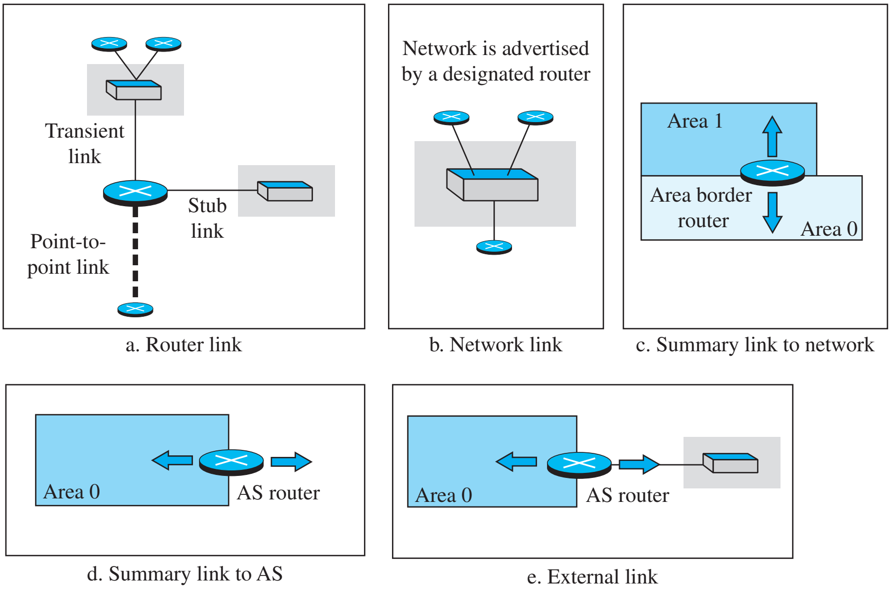
Figure 20.21 *Areas in an autonomous system*



Link-State Advertisement

OSPF is based on the link-state routing algorithm, which requires that a router advertise the state of each link to all neighbors for the formation of the LSDB. When we discussed the link-state algorithm, we used the graph theory and assumed that each router is a node and each network between two routers is an edge. The situation is different in the real world, in which we need to advertise the existence of different entities as nodes, the different types of links that connect each node to its neighbors, and the different types of cost associated with each link. This means we need different types of advertisements, each capable of advertising different situations. We can have five types of link-state advertisements: *router link, network link, summary link to network, summary link to AS border router,* and *external link*. Figure 20.22 shows these five advertisements and their uses.

Figure 20.22 *Five different LSPs*



a. Router link

b. Network link

c. Summary link to network

d. Summary link to AS

e. External link

- **Router link.** A router link advertises the existence of a router as a node. In addition to giving the address of the announcing router, this type of advertisement can define one or more types of links that connect the advertising router to other entities. A *transient link* announces a link to a transient network, a network that is connected to the rest of the networks by one or more routers. This type of advertisement should define the address of the transient network and the cost of the link. A *stub link* advertises a link to a stub network, a network that is not a through network. Again, the advertisement should define the address of the network and the cost. A *point-to-point link* should define the address of the router at the end of the point-to-point line and the cost to get there.

- ***Network link.*** A network link advertises the network as a node. However, since a network cannot do announcements itself (it is a passive entity), one of the routers is assigned as the designated router and does the advertising. In addition to the address of the designated router, this type of LSP announces the IP address of all routers (including the designated router as a router and not as speaker of the network), but no cost is advertised because each router announces the cost to the network when it sends a router link advertisement.

- ***Summary link to network.*** This is done by an area border router; it advertises the summary of links collected by the backbone to an area or the summary of links collected by the area to the backbone. As we discussed earlier, this type of information exchange is needed to glue the areas together.

- ***Summary link to AS.*** This is done by an AS router that advertises the summary links from other ASs to the backbone area of the current AS, information which later can be disseminated to the areas so that they will know about the networks in other ASs. The need for this type of information exchange is better understood when we discuss inter-AS routing (BGP).
- ***External link.*** This is also done by an AS router to announce the existence of a single network outside the AS to the backbone area to be disseminated into the areas.

OSPF Implementation

OSPF is implemented as a program in the network layer, using the service of the IP for propagation. An IP datagram that carries a message from OSPF sets the value of the protocol field to 89. This means that, although OSPF is a routing protocol to help IP to route its datagrams inside an AS, the OSPF messages are encapsulated inside datagrams. OSPF has gone through two versions: version 1 and version 2. Most implementations use version 2.

OSPF Messages

OSPF is a very complex protocol; it uses five different types of messages. In Figure 20.23, we first show the format of the OSPF common header (which is used in all messages) and the link-state general header (which is used in some messages). We then give the outlines of five message types used in OSPF. The *hello* message (type 1) is used by a router to introduce itself to the neighbors and announce all neighbors that it already knows. The *database description* message (type 2) is normally sent in response to the hello message to allow a newly joined router to acquire the full LSDB. The *link-state request* message (type 3) is sent by a router that needs information about a specific LS. The *link-state update* message (type 4) is the main OSPF message used for building the LSDB. This message, in fact, has five different versions (router link, network link, summary link to network, summary link to AS border router, and external link), as we discussed before. The *link-state acknowledgment* message (type 5) is used to create reliability in OSPF; each router that receives a link-state update message needs to acknowledge it.

Authentication

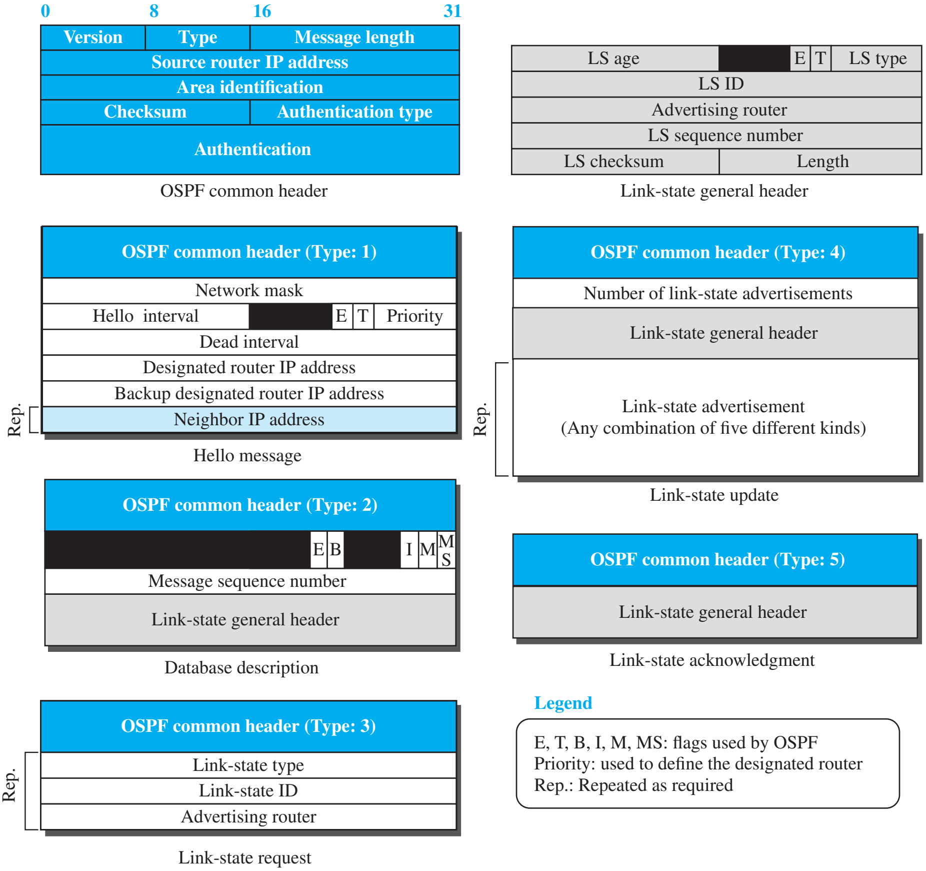
As Figure 20.23 shows, the OSPF common header has the provision for authentication of the message sender. As we will discuss in Chapters 31 and 32, this prevents a malicious entity from sending OSPF messages to a router and causing the router to become part of the routing system to which it actually does not belong.

OSPF Algorithm

OSPF implements the link-state routing algorithm we discussed in the previous section. However, some changes and augmentations need to be added to the algorithm:

- After each router has created the shortest-path tree, the algorithm needs to use it to create the corresponding routing algorithm.

Figure 20.23 *OSPF message formats*


0
8
16
31
Version
Type
Message length
Source router IP address
Area identification
Checksum
Authentication type
Authentication
OSPF common header
LS age
E
T
LS type
LS ID
Advertising router
LS sequence number
LS checksum
Length
Link-state general header
OSPF common header (Type: 1)
Network mask
Hello interval
E
T
Priority
Dead interval
Designated router IP address
Backup designated router IP address
Neighbor IP address
Rep.
Hello message
OSPF common header (Type: 4)
Number of link-state advertisements
Link-state general header
Link-state advertisement
(Any combination of five different kinds)
Rep.
Link-state update
OSPF common header (Type: 2)
E
B
I
M
MS
Message sequence number
Link-state general header
Database description
OSPF common header (Type: 5)
Link-state general header
Link-state acknowledgment
OSPF common header (Type: 3)
Link-state type
Link-state ID
Advertising router
Rep.
Link-state request
Legend
E, T, B, I, M, MS: flags used by OSPF
Priority: used to define the designated router
Rep.: Repeated as required

- The algorithm needs to be augmented to handle sending and receiving all five types of messages.

Performance

Before ending this section, let us briefly discuss the performance of OSPF:

- ***Update Messages.*** The link-state messages in OSPF have a somewhat complex format. They also are flooded to the whole area. If the area is large, these messages may create heavy traffic and use a lot of bandwidth.
- ***Convergence of Forwarding Tables.*** When the flooding of LSPs is completed, each router can create its own shortest-path tree and forwarding table; convergence is fairly quick. However, each router needs to run Dijkstra's algorithm, which may take some time.

- ***Robustness.*** The OSPF protocol is more robust than RIP because, after receiving the completed LSDB, each router is independent and does not depend on other routers in the area. Corruption or failure in one router does not affect other routers as seriously as in RIP.

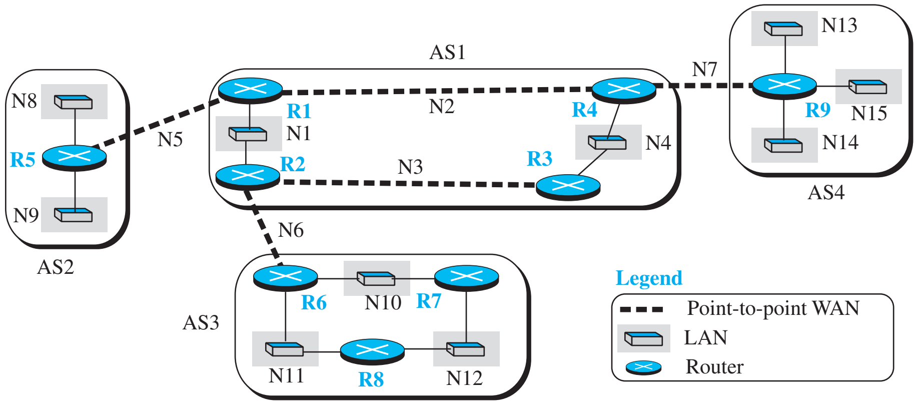
20.3.4 Border Gateway Protocol Version 4 (BGP4)

The **Border Gateway Protocol version 4 (BGP4)** is the only interdomain routing protocol used in the Internet today. BGP4 is based on the path-vector algorithm we described before, but it is tailored to provide information about the reachability of networks in the Internet.

Introduction

BGP, and in particular BGP4, is a complex protocol. In this section, we introduce the basics of BGP and its relationship with intradomain routing protocols (RIP or OSPF). Figure 20.24 shows an example of an internet with four autonomous systems. AS2, AS3, and AS4 are *stub* autonomous systems; AS1 is a *transient* one. In our example, data exchange between AS2, AS3, and AS4 should pass through AS1.

Figure 20.24 *A sample internet with four ASs*



Each autonomous system in this figure uses one of the two common intradomain protocols, RIP or OSPF. Each router in each AS knows how to reach a network that is in its own AS, but it does not know how to reach a network in another AS.

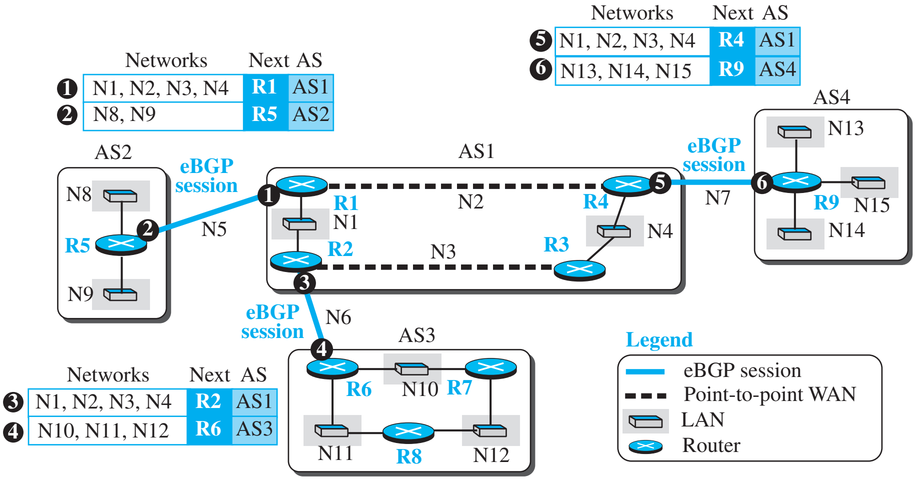
To enable each router to route a packet to any network in the internet, we first install a variation of BGP4, called *external BGP* (*eBGP*), on each *border router* (the one at the edge of each AS which is connected to a router at another AS). We then install the second variation of BGP, called *internal BGP* (*iBGP*), on all routers. This means that the border routers will be running three routing protocols (intradomain, eBGP, and iBGP), but other routers are running two protocols (intradomain and iBGP). We discuss the effect of each BGP variation separately.

Operation of External BGP (eBGP)

We can say that BGP is a kind of point-to-point protocol. When the software is installed on two routers, they try to create a TCP connection using the well-known port 179. In other words, a pair of client and server processes continuously communicate with each other to exchange messages. The two routers that run the BGP processes are called *BGP peers* or *BGP speakers*. We discuss different types of messages exchanged between two peers, but for the moment we are interested in only the update messages (discussed later) that announce reachability of networks in each AS.

The eBGP variation of BGP allows two physically connected border routers in two different ASs to form pairs of eBGP speakers and exchange messages. The routers that are eligible in our example in Figure 20.24 form three pairs: R1-R5, R2-R6, and R4-R9. The connection between these pairs is established over three physical WANs (N5, N6, and N7). However, there is a need for a logical TCP connection to be created over the physical connection to make the exchange of information possible. Each logical connection in BGP parlance is referred to as a *session*. This means that we need three sessions in our example, as shown in Figure 20.25.

Figure 20.25 *eBGP operation*



The figure also shows the simplified update messages sent by routers involved in the eBGP sessions. The circled number defines the sending router in each case. For example, message number 1 is sent by router R1 and tells router R5 that N1, N2, N3, and N4 can be reached through router R1 (R1 gets this information from the corresponding intradomain forwarding table). Router R5 can now add these pieces of information at the end of its forwarding table. When R5 receives any packet destined for these four networks, it can use its forwarding table and find that the next router is R1.

The reader may have noticed that the messages exchanged during three eBGP sessions help some routers know how to route packets to some networks in the internet, but the reachability information is not complete. There are two problems that need to be addressed:

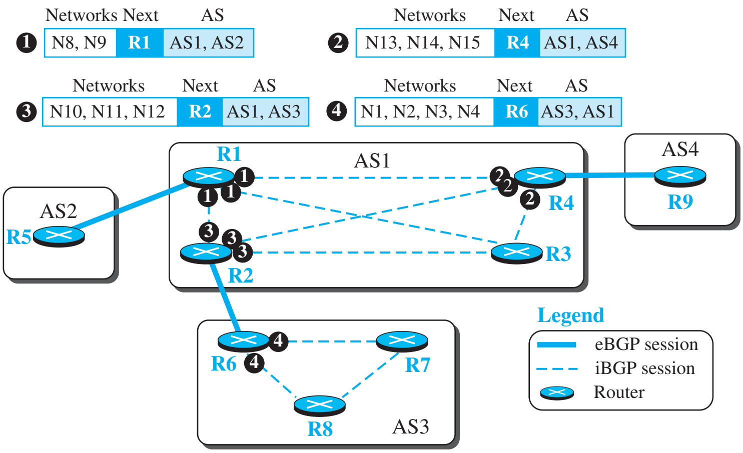
1. Some border routers do not know how to route a packet destined for nonneighbor ASs. For example, R5 does not know how to route packets destined for networks in AS3 and AS4. Routers R6 and R9 are in the same situation as R5: R6 does not know about networks in AS2 and AS4; R9 does not know about networks in AS2 and AS3.
2. None of the nonborder routers know how to route a packet destined for any networks in other ASs.

To address the above two problems, we need to allow all pairs of routers (border or nonborder) to run the second variation of the BGP protocol, iBGP.

Operation of Internal BGP (iBGP)

The iBGP protocol is similar to the eBGP protocol in that it uses the service of TCP on the well-known port 179, but it creates a session between any possible pair of routers inside an autonomous system. However, some points should be made clear. First, if an AS has only one router, there cannot be an iBGP session. For example, we cannot create an iBGP session inside AS2 or AS4 in our internet. Second, if there are n routers in an autonomous system, there should be $[n \times (n - 1) / 2]$ iBGP sessions in that autonomous system (a fully connected mesh) to prevent loops in the system. In other words, each router needs to advertise its own reachability to the peer in the session instead of flooding what it receives from another peer in another session. Figure 20.26 shows the combination of eBGP and iBGP sessions in our internet.

Figure 20.26 *Combination of eBGP and iBGP sessions in our internet*



Note that we have not shown the physical networks inside ASs because a session is made on an overlay network (TCP connection), possibly spanning more than one physical network as determined by the route dictated by intradomain routing protocol. Also note that in this stage only four messages are exchanged. The first message (numbered 1) is sent by R1 announcing that networks N8 and N9 are reachable through the path AS1-AS2, but the next router is R1. This message is sent, through separate sessions, to R2, R3, and R4. Routers R2, R4, and R6 do the same thing but send different messages to different destinations. The interesting point is that, at this stage, R3, R7, and R8 create sessions with their peers, but they actually have no message to send.

The updating process does not stop here. For example, after R1 receives the update message from R2, it combines the reachability information about AS3 with the reachability information it already knows about AS1 and sends a new update message to R5. Now R5 knows how to reach networks in AS1 and AS3. The process continues when R1 receives the update message from R4. The point is that we need to make certain that at a point in time there are no changes in the previous updates and that all information is propagated through all ASs. At this time, each router combines the information received from eBGP and iBGP and creates what we may call a path table after applying the criteria for finding the best path, including routing policies that we discuss later. To demonstrate, we show the path tables in Figure 20.27 for the routers in Figure 20.24. For example, router R1 now knows that any packet destined for networks N8 or N9 should go through AS1 and AS2 and the next router to deliver the packet to is router R5. Similarly, router R4 knows that any packet destined for networks N10, N11, or N12 should go through AS1 and AS3 and the next router to deliver this packet to is router R1, and so on.

Figure 20.27 *Finalized BGP path tables*

Networks	Next	Path
N8, N9	**R5**	AS1, AS2
N10, N11, N12	**R2**	AS1, AS3
N13, N14, N15	**R4**	AS1, AS4

Path table for R1

Networks	Next	Path
N8, N9	**R1**	AS1, AS2
N10, N11, N12	**R6**	AS1, AS3
N13, N14, N15	**R1**	AS1, AS4

Path table for R2

Networks	Next	Path
N8, N9	**R2**	AS1, AS2
N10, N11, N12	**R2**	AS1, AS3
N13, N14, N15	**R4**	AS1, AS4

Path table for R3

Networks	Next	Path
N8, N9	**R1**	AS1, AS2
N10, N11, N12	**R1**	AS1, AS3
N13, N14, N15	**R9**	AS1, AS4

Path table for R4

Networks	Next	Path
N1, N2, N3, N4	**R1**	AS2, AS1
N10, N11, N12	**R1**	AS2, AS1, AS3
N13, N14, N15	**R1**	AS2, AS1, AS4

Path table for R5

Networks	Next	Path
N1, N2, N3, N4	**R2**	AS3, AS1
N8, N9	**R2**	AS3, AS1, AS2
N13, N14, N15	**R2**	AS3, AS1, AS4

Path table for R6

Networks	Next	Path
N1, N2, N3, N4	**R6**	AS3, AS1
N8, N9	**R6**	AS3, AS1, AS2
N13, N14, N15	**R6**	AS3, AS1, AS4

Path table for R7

Networks	Next	Path
N1, N2, N3, N4	**R6**	AS3, AS1
N8, N9	**R6**	AS3, AS1, AS2
N13, N14, N15	**R6**	AS3, AS1, AS4

Path table for R8

Networks	Next	Path
N1, N2, N3, N4	**R4**	AS4, AS1
N8, N9	**R4**	AS4, AS1, AS2
N10, N11, N12	**R4**	AS4, AS1, AS3

Path table for R9

Injection of Information into Intradomain Routing

The role of an interdomain routing protocol such as BGP is to help the routers inside the AS to augment their routing information. In other words, the path tables collected and organized by BPG are not used, per se, for routing packets; they are injected into intradomain forwarding tables (RIP or OSPF) for routing packets. This can be done in several ways depending on the type of AS.

In the case of a stub AS, the only area border router adds a default entry at the end of its forwarding table and defines the next router to be the speaker router at the end of the eBGP connection. In Figure 20.24, R5 in AS2 defines R1 as the default router for all networks other than N8 and N9. The situation is the same for router R9 in AS4 with the default router to be R4. In AS3, R6 set its default router to be R2, but R7 and R8 set their default router to be R6. These settings are in accordance with the path tables we describe in Figure 20.27 for these routers. In other words, the path tables are injected into intradomain forwarding tables by adding only one default entry.

In the case of a transient AS, the situation is more complicated. R1 in AS1 needs to inject the whole contents of the path table for R1 in Figure 20.27 into its intradomain forwarding table. The situation is the same for R2, R3, and R4.

One issue to be resolved is the cost value. We know that RIP and OSPF use different metrics. One solution, which is very common, is to set the cost to the foreign networks at the same cost value as to reach the first AS in the path. For example, the cost for R5 to reach all networks in other ASs is the cost to reach N5. The cost for R1 to reach networks N10 to N12 is the cost to reach N6, and so on. The cost is taken from the intradomain forwarding tables (RIP or OSPF).

Figure 20.28 shows the interdomain forwarding tables. For simplicity, we assume that all ASs are using RIP as the intradomain routing protocol. The shaded areas are the augmentation injected by the BGP protocol; the default destinations are indicated as zero.

Figure 20.28 *Forwarding tables after injection from BGP*

Des.	Next	Cost
N1	—	1
N4	R4	2
N8	R5	1
N9	R5	1
N10	R2	2
N11	R2	2
N12	R2	2
N13	R4	2
N14	R4	2
N15	R4	2

Table for R1

Des.	Next	Cost
N1	—	1
N4	R3	2
N8	R1	2
N9	R1	2
N10	R6	1
N11	R6	1
N12	R6	1
N13	R3	3
N14	R3	3
N15	R3	3

Table for R2

Des.	Next	Cost
N1	R2	2
N4	—	1
N8	R2	3
N9	R2	3
N10	R2	2
N11	R2	2
N12	R2	2
N13	R4	2
N14	R4	2
N15	R4	2

Table for R3

Des.	Next	Cost
N1	R1	2
N4	—	1
N8	R1	2
N9	R1	2
N10	R3	3
N11	R3	3
N12	R3	3
N13	R9	1
N14	R9	1
N15	R9	1

Table for R4

Des.	Next	Cost
N8	—	1
N9	—	1
0	R1	1

Table for R5

Des.	Next	Cost
N10	—	1
N11	—	1
N12	R7	2
0	R2	1

Table for R6

Des.	Next	Cost
N10	—	1
N11	R6	2
N12	—	1
0	R6	2

Table for R7

Des.	Next	Cost
N10	R6	2
N11	—	1
N12	—	1
0	R6	2

Table for R8

Des.	Next	Cost
N13	—	1
N14	—	1
N15	—	1
0	R4	1

Table for R9

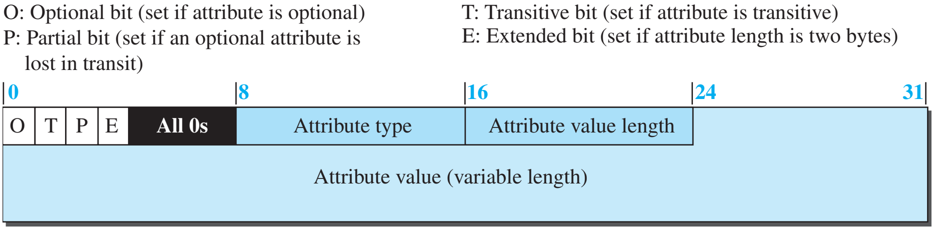
Address Aggregation

The reader may have realized that intradomain forwarding tables obtained with the help of the BGP4 protocols may become huge in the case of the global Internet because many destination networks may be included in a forwarding table. Fortunately, BGP4 uses the prefixes as destination identifiers and allows the aggregation of these prefixes, as we discussed in Chapter 18. For example, prefixes 14.18.20.0/26, 14.18.20.64/26, 14.18.20.128/26, and 14.18.20.192/26, can be combined into 14.18.20.0/24 if all four subnets can be reached through one path. Even if one or two of the aggregated prefixes need a separate path, the longest prefix principle we discussed earlier allows us to do so.

Path Attributes

In both intradomain routing protocols (RIP or OSPF), a destination is normally associated with two pieces of information: next hop and cost. The first one shows the address of the next router to deliver the packet; the second defines the cost to the final destination. Interdomain routing is more involved and naturally needs more information about how to reach the final destination. In BGP these pieces are called *path attributes.* BGP allows a destination to be associated with up to seven path attributes. Path attributes are divided into two broad categories: *well-known* and *optional*. A well-known attribute must be recognized by all routers; an optional attribute need not be. A well-known attribute can be mandatory, which means that it must be present in any BGP update message, or discretionary, which means it does not have to be. An optional attribute can be either transitive, which means it can pass to the next AS, or intransitive, which means it cannot. All attributes are inserted after the corresponding destination prefix in an update message (discussed later). The format for an attribute is shown in Figure 20.29.

Figure 20.29 *Format of path attribute*


O: Optional bit (set if attribute is optional)
P: Partial bit (set if an optional attribute is lost in transit)
T: Transitive bit (set if attribute is transitive)
E: Extended bit (set if attribute length is two bytes)
0
8
16
24
31
O
T
P
E
All 0s
Attribute type
Attribute value length
Attribute value (variable length)

The first byte in each attribute defines the four attribute flags (as shown in the figure). The next byte defines the type of attributes assigned by ICANN (only seven types have been assigned, as explained next). The attribute value length defines the length of the attribute value field (not the length of the whole attributes section). The following gives a brief description of each attribute.

- ❑ ***ORIGIN (type 1).*** This is a well-known mandatory attribute, which defines the source of the routing information. This attribute can be defined by one of the three values: 1, 2, and 3. Value 1 means that the information about the path has been taken from an intradomain protocol (RIP or OSPF). Value 2 means that the information comes from BGP. Value 3 means that it comes from an unknown source.
- ❑ ***AS-PATH (type 2).*** This is a well-known mandatory attribute, which defines the list of autonomous systems through which the destination can be reached. We have used this attribute in our examples. The AS-PATH attribute, as we discussed in path-vector routing in the last section, helps prevent a loop. Whenever an update

message arrives at a router that lists the current AS as the path, the router drops that path. The AS-PATH can also be used in route selection.

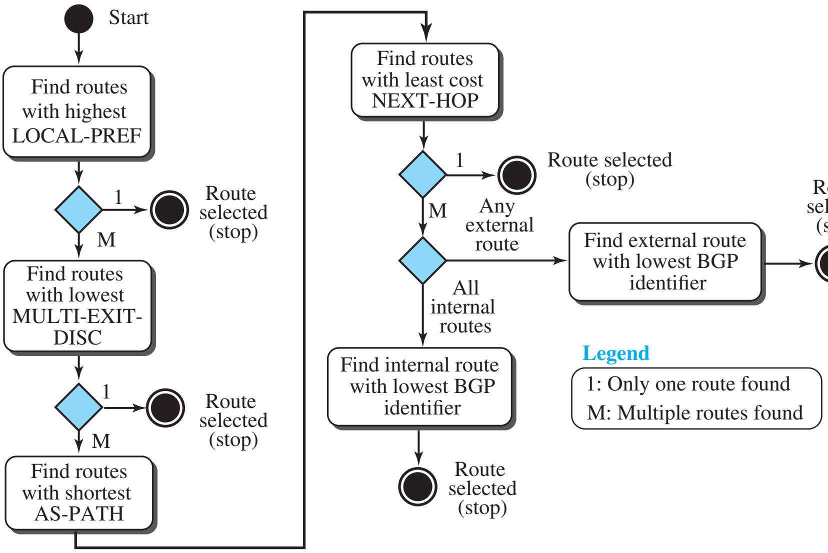
- ❑ ***NEXT-HOP (type 3).*** This is a well-known mandatory attribute, which defines the next router to which the data packet should be forwarded. We have also used this attribute in our examples. As we have seen, this attribute helps to inject path information collected through the operations of eBGP and iBGP into the intradomain routing protocols such as RIP or OSPF.
- ❑ ***MULT-EXIT-DISC (type 4).*** The multiple-exit discriminator is an optional intransitive attribute, which discriminates among multiple exit paths to a destination. The value of this attribute is normally defined by the metric in the corresponding intradomain protocol (an attribute value of 4-byte unsigned integer). For example, if a router has multiple paths to the destination with different values related to these attributes, the one with the lowest value is selected. Note that this attribute is intransitive, which means that it is not propagated from one AS to another.
- ❑ ***LOCAL-PREF (type 5).*** The local preference attribute is a well-known discretionary attribute. It is normally set by the administrator, based on the organization policy. The routes the administrator prefers are given a higher local preference value (an attribute value of 4-byte unsigned integer). For example, in an internet with five ASs, the administrator of AS1 can set the local preference value of 400 to the path AS1 → AS2 → AS5, the value of 300 to AS1 → AS3 → AS5, and the value of 50 to AS1 → AS4 → AS5. This means that the administrator prefers the first path to the second one and prefers the second one to the third one. This may be a case where AS2 is the most secured and AS4 is the least secured AS for the administration of AS1. The last route should be selected if the other two are not available.
- ❑ ***ATOMIC-AGGREGATE (type 6).*** This is a well-known discretionary attribute, which defines the destination prefix as not aggregate; it only defines a single destination network. This attribute has no value field, which means the value of the length field is zero.
- ❑ ***AGGREGATOR (type 7).*** This is an optional transitive attribute, which emphasizes that the destination prefix is an aggregate. The attribute value gives the number of the last AS that did the aggregation followed by the IP address of the router that did so.

Route Selection

So far in this section, we have been silent about how a route is selected by a BGP router mostly because our simple example has one route to a destination. In the case where multiple routes are received to a destination, BGP needs to select one among them. The route selection process in BGP is not as easy as the ones in the intradomain routing protocol that is based on the shortest-path tree. A route in BGP has some attributes attached to it and it may come from an eBGP session or an iBGP session. Figure 20.30 shows the flow diagram as used by common implementations.

The router extracts the routes which meet the criteria in each step. If only one route is extracted, it is selected and the process stops; otherwise, the process continues with the next step. Note that the first choice is related to the LOCAL-PREF attribute, which reflects the policy imposed by the administration on the route.

Figure 20.30 *Flow diagram for route selection*


Start
Find routes with highest LOCAL-PREF
1
Route selected (stop)
M
Find routes with lowest MULTI-EXIT-DISC
1
Route selected (stop)
M
Find routes with shortest AS-PATH
Find routes with least cost NEXT-HOP
1
Route selected (stop)
M
Any external route
Find external route with lowest BGP identifier
Route selected (stop)
All internal routes
Find internal route with lowest BGP identifier
Route selected (stop)
Legend
1: Only one route found
M: Multiple routes found

Messages

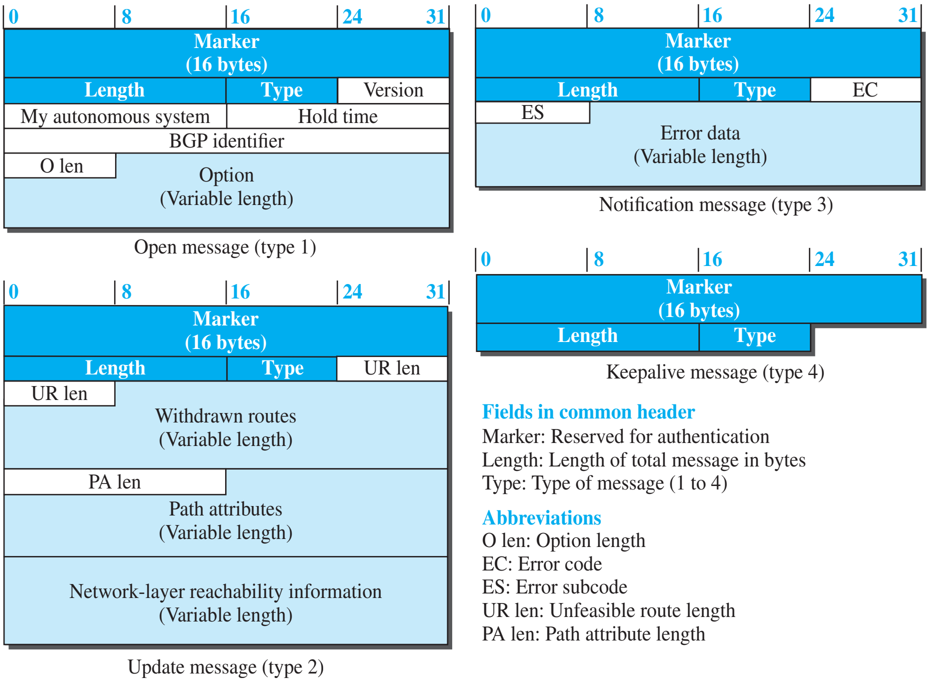
BGP uses four types of messages for communication between the BGP speakers across the ASs and inside an AS: *open, update, keepalive,* and *notification* (see Figure 20.31). All BGP packets share the same common header.

- ***Open Message.*** To create a neighborhood relationship, a router running BGP opens a TCP connection with a neighbor and sends an *open message.*
- ***Update Message.*** The *update message* is the heart of the BGP protocol. It is used by a router to withdraw destinations that have been advertised previously, to announce a route to a new destination, or both. Note that BGP can withdraw several destinations that were advertised before, but it can only advertise one new destination (or multiple destinations with the same path attributes) in a single update message.
- ***Keepalive Message.*** The BGP peers that are running exchange keepalive messages regularly (before their hold time expires) to tell each other that they are alive.
- ***Notification.*** A notification message is sent by a router whenever an error condition is detected or a router wants to close the session.

Performance

BGP performance can be compared with RIP. BGP speakers exchange a lot of messages to create forwarding tables, but BGP is free from loops and count-to-infinity. The same weakness we mention for RIP about propagation of failure and corruption also exists in BGP.

Figure 20.31 *BGP messages*



Fields in common header

Marker: Reserved for authentication
Length: Length of total message in bytes
Type: Type of message (1 to 4)

Abbreviations

O len: Option length
EC: Error code
ES: Error subcode
UR len: Unfeasible route length
PA len: Path attribute length

20.4 END-CHAPTER MATERIALS

20.4.1 Recommended Reading

Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], [Koz 05], [Ste 95], [GW 04], [Per 00], [Kes 02], [Moy 98], [WZ 01], and [Los 04].

RFCs

RIP is discussed in RFCs 1058 and 2453. OSPF is discussed in RFCs 1583 and 2328. BGP is discussed in RFCs 1654, 1771, 1773, 1997, 2439, 2918, and 3392.

20.4.2 Key Terms

autonomous system (AS)
Bellman-Ford
Border Gateway Protocol version 4 (BGP4)
Dijkstra's algorithm
distance vector
distance-vector (DV) routing
flooding
least-cost tree

link-state database (LSDB)
link-state (LS) routing
Open Shortest Path First (OSPF)
path-vector (PV) routing
poison reverse
Routing Information Protocol (RIP)
split horizon

20.4.3 Summary

In unicast routing, a packet is routed, hop by hop, from its source to its destination by the help of forwarding tables. Although there are several routes that a packet can travel from the source to the destination, the question is which should be the best. The interpretation of the term *best* depends on the cost and policy imposed on the trip.

Several routing algorithms, and the corresponding protocols, have been devised to find the best route among them; three have survived. In distance-vector routing, the first thing each node creates is its own least-cost tree with the rudimentary information it has about its immediate neighbors. The incomplete trees are exchanged between immediate neighbors to make the trees more and more complete and to represent the whole internet. In other words, in distance-vector routing, a router continuously tells all of its neighbors what it knows about the whole internet. The protocol that implements distance-vector routing is called *Routing Information Protocol (RIP).*

Another routing algorithm that has been used in the Internet is link-state routing. This method uses the term *link-state* to define the characteristic of a link (an edge) that represents a network in the internet. In this algorithm the cost associated with an edge defines the state of the link. In this algorithm, all routers flood the internet, with information related to their link states. When every router has the complete picture of the states, a link-state database can be created. The least-cost tree for each router and the corresponding forwarding table can be made from the link-state database. A protocol that implements link-state routing is called *Open Shortest Path First (OSPF).*

Both link-state and distance-vector routing are based on the least-cost goal. However, there are instances where this goal is not the priority. Path-vector routing algorithms have been designed for this purpose. We can always insert policies in the forwarding table by preventing a packet from visiting a specific router. In path-vector routing, the best route from the source is the best path, the one that complies with the policy imposed. The protocol that implements path-vector routing is the Border Gateway Protocol (BGP).

20.5 PRACTICE SET

20.5.1 Quizzes

A set of interactive quizzes for this chapter can be found on the book website. It is strongly recommended that the student take the quizzes to check his/her understanding of the materials before continuing with the practice set.

20.5.2 Questions

Q20-1. In a graph, if we know that the shortest path from node A to node G is (A → B → E → G), what is the shortest path from node G to node A?

Q20-2. Assume the shortest path in a graph from node A to node H is A → B → H. Also assume that the shortest path from node H to node N is H → G → N. What is the shortest path from node A to node N?

Q20-3. Explain why a router using link-state routing needs to receive the whole LSDB before creating and using its forwarding table. In other words, why can't the router create its forwarding table with a partially received LSDB?

Q20-4. Is the path-vector routing algorithm closer to the distance-vector routing algorithm or to the link-state routing algorithm? Explain.

Q20-5. List three types of autonomous systems (ASs) described in the text, and make a comparison between them.

Q20-6. Explain the concept of hop count in RIP. Can you explain why no hop is counted between N1 and R1 in Figure 20.15?

Q20-7. Assume that we have an isolated AS running RIP. We can say that we have at least two different kinds of datagram traffic in this AS. The first kind carries the messages exchanged between hosts; the second carries messages belonging to RIP. What is the difference between the two kinds of traffic when we think about source and destination IP addresses? Does this show that routers also need IP addresses?

Q20-8. Router A sends two RIP messages to two immediate neighboring routers, B and C. Do the two datagrams carrying the messages have the same source IP addresses? Do the two datagrams have the same destination IP addresses?

Q20-9. At any moment, a RIP message may arrive at a router that runs RIP as the routing protocol. Does it mean that the RIP process should be running all the time?

Q20-10. Why do you think RIP uses UDP instead of TCP?

Q20-11. We say that OSPF is a hierarchical intradomain protocol, but RIP is not. What is the reason behind this statement?

Q20-12. In a very small AS using OSPF, is it more efficient to use only one single area (backbone) or several areas?

Q20-13. Why do you think we need only one RIP update message, but several OSPF update messages?

Q20-14. OSPF messages are exchanged between routers. Does this mean that we need to have OSPF processes run all the time to be able to receive an OSPF message when it arrives?

Q20-15. OSPF messages and ICMP messages are directly encapsulated in an IP datagram. If we intercept an IP datagram, how can we tell whether the payload belongs to OSPF or ICMP?

Q20-16. Explain what type of OSPF link state is advertised in each of the following cases:

a. A router needs to advertise the existence of another router at the end of a point-to-point link.

b. A router needs to advertise the existence of two stub networks and one transient network.

c. A designated router advertises a network as a node.

Q20-17. Can a router combine the advertisement of a link and a network in a single link-state update?

Q20-18. Explain why we can have different intradomain routing protocols in different ASs, but we need only one interdomain routing protocol in the whole Internet.

Q20-19. Can you explain why BGP uses the services of TCP instead of UDP?

Q20-20. Explain why policy routing can be implemented on an interdomain routing, but it cannot be implemented on a intradomain routing.

Q20-21. Explain when each of the following attributes can be used in BGP:

a. LOCAL-PREF **b.** AS-PATH **c.** NEXT-HOP

20.5.3 Problems

P20-1. Assume that the shortest distance between nodes ***a***, ***b***, ***c***, and ***d*** to node ***y*** and the costs from node ***x*** to nodes ***a, b, c,*** and ***d*** are given below:

$\mathbf{D}_{ay} = 5$	$\mathbf{D}_{by} = 6$	$\mathbf{D}_{cy} = 4$	$\mathbf{D}_{dy} = 3$
$\mathbf{c}_{xa} = 2$	$\mathbf{c}_{xb} = 1$	$\mathbf{c}_{xc} = 3$	$\mathbf{c}_{xd} = 1$

What is the shortest distance between node ***x*** and node ***y***, $\mathbf{D}_{xy}$, according to the Bellman-Ford equation?

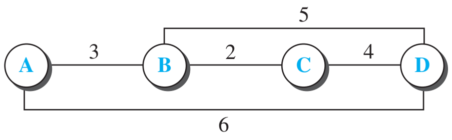
P20-2. Assume a router using RIP has 10 entries in its forwarding table at time t_1. Six of these entries are still valid at time t_2. Four of these entries have been expired 70, 90, 110, and 210 seconds before time t_2. Find the number of periodic timers, expiration timers, and garbage collection timers running at time t_1 and time t_2.

P20-3. When does an OSPF router send each of the following messages?

a. hello **b.** data description **c.** link-state request
d. link-state update **e.** link-state acknowledgment

P20-4. To understand how the distance vector algorithm in Table 20.1 works, let us apply it to a four-node internet as shown in Figure 20.32.

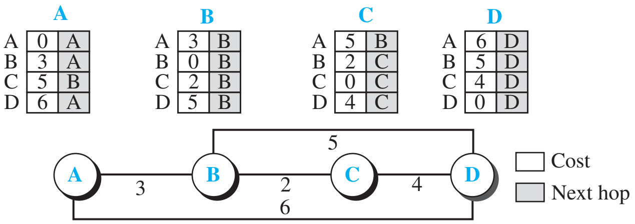
Figure 20.32 *Problem P20-4*



Assume that all nodes are initialized first. Also assume that the algorithm is applied, one at a time, to each node respectively (A, B, C, D). Show that the process converges and all nodes will have their stable distance vectors.

P20-5. In distance-vector routing, good news (decrease in a link metric) will propagate fast. In other words, if a link distance decreases, all nodes quickly learn about it and update their vectors. In Figure 20.33, we assume that a four-node internet is stable, but suddenly the distance between nodes A and D, which is currently 6, is decreased to 1 (probably due to some improvement in the link quality). Show how this good news is propagated, and find the new distance vector for each node after stabilization.

Figure 20.33 *Problem P20-6*



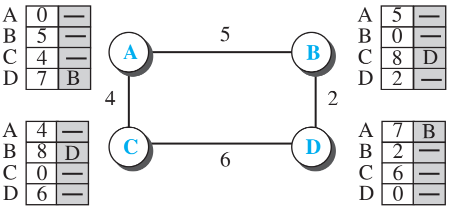
	A		B		C		D	
A	0	A	3	B	5	B	6	D
B	3	A	0	B	2	C	5	D
C	5	B	2	B	0	C	4	D
D	6	A	5	B	4	C	0	D

P20-6. In distance-vector routing, bad news (increase in a link metric) will propagate slowly. In other words, if a link distance increases, sometimes it takes a long time for all nodes to know the bad news. In Figure 20.33 (see the previous problem), we assume that a four-node internet is stable, but suddenly the distance between nodes B and C, which is currently 2, is increased to infinity (link fails). Show how this bad news is propagated, and find the new distance vector for each node after stabilization. Assume that the implementation uses a periodic timer to trigger updates to neighbors (no more updates are triggered when there is change). Also assume that if a node receives a higher cost from the same previous neighbor, it uses the new cost because this means that the old advertisement is not valid anymore. To make the stabilization faster, the implementation also suspends a route when the next hop is not accessible.

P20-7. In computer science, when we encounter an algorithm, we often need to ask about the complexity of that algorithm (how many computations we need to do). To find the complexity of the distance vector's algorithm, find the number of operations a node needs to do when it receives a vector from a neighbor.

P20-8. Assume that the network in Figure 20.34 uses distance-vector routing with the forwarding table as shown for each node.

Figure 20.34 *Problem P20-8*



A	0	—
B	5	—
C	4	—
D	7	B

A	5	—
B	0	—
C	8	D
D	2	—

A	4	—
B	8	D
C	0	—
D	6	—

A	7	B
B	2	—
C	6	—
D	0	—

If each node periodically announces their vectors to the neighbor using the poison-reverse strategy, what is the distance vector advertised in the appropriate period:

a. from A to B? **b.** from C to D? **c.** from D to B? **d.** from C to A?

P20-9. Assume that the network in Figure 20.34 (previous problem) uses distance-vector routing with the forwarding table as shown for each node. If each node periodically announces their vectors to the neighbor using the split-horizon strategy, what is the distance vector advertised in the appropriate period:

a. from A to B? **b.** from C to D? **c.** from D to B? **d.** from C to A?

P20-10. Assume that the network in Figure 20.34 (Problem P20-8) uses distance vector routing with the forwarding table as shown for each node. If node E is added to the network with a link of cost 1 to node D, can you find the new forwarding tables for each node without using the distance-vector algorithm?

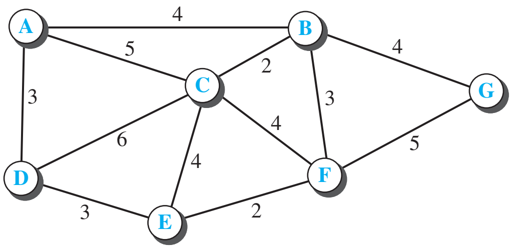
P20-11. Create the forwarding table for node A in Figure 20.10.

P20-12. Create the shortest path tree and the forwarding table for node G in Figure 20.8.

P20-13. Create the shortest path tree and the forwarding table for node B in Figure 20.8.

P20-14. Use Dijkstra's algorithm (Table 20.2) to find the shortest path tree and the forwarding table for node A in the Figure 20.35.

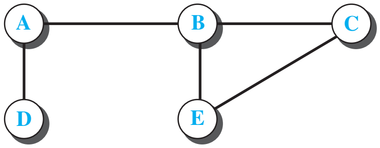
Figure 20.35 *Problem P20-14*



P20-15. In computer science, when we encounter an algorithm, we often need to ask about the complexity of that algorithm (how many computations we need to do). To find the complexity of Dijkstra's algorithm, find the number of searches we have to do to find the shortest path for a single node when the number of nodes is n.

P20-16. Assume that A, B, C, D, and E in Figure 20.36 are autonomous systems (ASs). Find the path vector for each AS using the algorithm in Table 20.3. Assume that the best path in this case is the path which passes through the shorter list of ASs. Also assume that the algorithm first initializes each AS and then is applied, one at a time, to each node respectively (A, B, C, D, E). Show that the process converges and all ASs will have their stable path vectors.

Figure 20.36 *Problem P20-16*



P20-17. In Figure 20.24, assume that the intra-AS routing protocol used by AS1 is OSPF, but the one used by AS2 is RIP. Explain how R5 can find how to route a packet to N4.

P20-18. In Figure 20.24, assume that the intra-AS routing protocol used by AS4 and AS3 is RIP. Explain how R8 can find how to route a packet to N13.

20.6 SIMULATION EXPERIMENTS

20.6.1 Applets

We have created some Java applets to show some of the main concepts discussed in this chapter. It is strongly recommended that the students activate these applets on the book website and carefully examine the protocols in action.

20.7 PROGRAMMING ASSIGNMENT

Write the source code, compile, and test the following program in one of the programming languages of your choice:

Prg20-1. Write a program to simulate the distance-vector algorithm (Table 20.1).

Prg20-2. Write a program to simulate the link-state algorithm (Table 20.2).

Prg20-3. Write a program to simulate the path-vector algorithm (Table 20.3).