# CCIE Professional Development Routing TCP/IP Volume I

## Second Edition

**Jeff Doyle, CCIE No. 1919**
**Jennifer Carroll, CCIE No. 1402**

**Cisco Press**


This chapter covers the following subjects:

- Routing Protocol Basics
- Distance Vector Routing Protocols
- Link State Routing Protocols
- Interior and Exterior Gateway Protocols
- Static or Dynamic Routing?

CHAPTER 4

# Dynamic Routing Protocols

The previous chapter explains what a router needs to recognize to correctly switch packets to their respective destinations, and how that information is put into the route table manually. This chapter shows how routers can discover this information automatically and share that information with other routers via dynamic routing protocols. A *routing protocol* is the language a router speaks with other routers to share information about the reachability and status of networks.

Dynamic routing protocols not only perform these path-determination and route-table-update functions, but also determine the next-best path if the best path to a destination becomes unusable. The capability to compensate for topology changes is the most important advantage dynamic routing offers over static routing.

Obviously, for communication to occur, the communicators must speak the same language. Since the advent of IP routing, there have been eight major IP routing protocols from which to choose;[1] if one router speaks RIP and another speaks OSPF, they cannot share routing information because they are not speaking the same language. Subsequent chapters examine all the IP routing protocols in current use, and even consider how to make a router "bilingual," but first it is necessary to explore some characteristics and issues common to all routing protocols—IP or otherwise.

## Routing Protocol Basics

All dynamic routing protocols are built around an algorithm. Generally, an *algorithm* is a step-by-step procedure for solving a problem. A routing algorithm must, at a minimum, specify the following:

- A procedure for passing reachability information about networks to other routers
- A procedure for receiving reachability information from other routers
- A procedure for determining optimal routes based on the reachability information it has and for recording this information in a route table
- A procedure for reacting to, compensating for, and advertising topology changes in a network

[1] Of these eight protocols, BGP has obsoleted EGP, the Cisco Systems EIGRP obsoleted its IGRP, and RIPv2 is quickly replacing RIPv1.

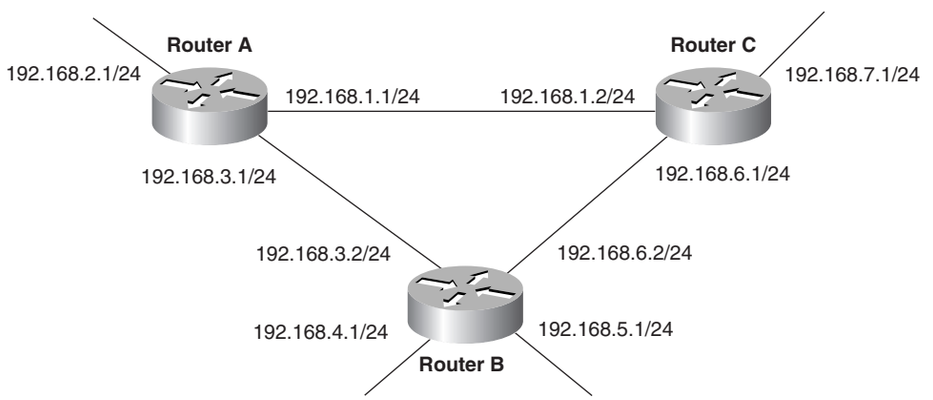
A few issues common to any routing protocol are path determination, metrics, convergence, and load balancing.

## Path Determination

All subnets within a network must be connected to a router, and wherever a router has an interface on a network, that interface must have an address on the network.[2] This address is the originating point for reachability information.

Figure 4-1 shows a simple three-router network. Router A recognizes networks 192.168.1.0, 192.168.2.0, and 192.168.3.0 because it has interfaces on those networks with corresponding addresses and appropriate address masks. Likewise, Router B recognizes 192.168.3.0, 192.168.4.0, 192.168.5.0, and 192.186.6.0; Router C recognizes 192.168.6.0, 192.168.7.0, and 198.168.1.0. Each interface implements the data link and physical protocols of the network to which it is attached, so the router also recognizes the state of the network (up or down).

**Figure 4-1** *Each router knows about its directly connected networks from its assigned addresses and masks.*




At first glance, the information-sharing procedure seems simple. Look at Router A:

1. Router A examines its IP addresses and associated masks and deduces that it is attached to networks 192.168.1.0, 192.186.2.0, and 192.168.3.0.
2. Router A enters these networks into its route table, along with some sort of flag indicating that the networks are directly connected.

[2] Many point-to-point links are configured as "unnumbered" links—that is, there is no address assigned to the connected point-to-point interface—so as to conserve addresses. But unnumbered links do not violate the rule that every interface must have an address; they use another address on the router, usually the loopback address, as a proxy address.

3. Router A places the information into a packet: "My directly connected networks are 192.168.1.0, 192.186.2.0, and 192.168.3.0."
4. Router A transmits copies of these route information packets, or *routing updates*, to Routers B and C.

Routers B and C, having performed the same steps, have sent updates with their directly connected networks to A. Router A enters the received information into its route table, along with the source address of the router that sent the update packet. Router A now recognizes all the networks and the addresses of the routers to which they are attached.

This procedure does seem quite simple. So why are routing protocols so much more complicated than this? Look again at Figure 4-1:

- What should Router A do with the updates from B and C after it has recorded the information in the route table? Should it, for instance, pass B's routing information packet to C and pass C's packet to B?
- If Router A does not forward the updates, information sharing might not be complete. For instance, if the link between B and C does not exist, those two routers would not recognize each other's networks. Router A must forward the update information, but this step opens a new set of problems.
- If Router A hears about network 192.168.4.0 from both Router B and Router C, which router should be used to reach that network? Are they both valid? Which one is the best path?
- What mechanism will be used to ensure that all routers receive all routing information while preventing update packets from circulating endlessly through the network?
- The routers share certain directly connected networks (192.168.1.0, 192.168.3.0, and 192.168.6.0). Should the routers still advertise these networks?

These questions are almost as simplistic as the preceding preliminary explanation of routing protocols, but they should give you an indication for some of the issues that contribute to the complexity of the protocols. Each routing protocol addresses these questions one way or another, which will become clear in following sections and chapters.

## Metrics

When there are multiple routes to the same destination, a router must have a mechanism for calculating the best path. A *metric* is a variable assigned to routes as a means of ranking them from best to worst or from most preferred to least preferred. Consider the following example of why metrics are needed.

Assuming that information sharing has properly occurred in the network of Figure 4-1, Router A might have a route table that looks like Table 4-1.

**Table 4-1** *Rudimentary route table for Router A of Figure 4-1.*

| Network | Next-Hop Router |
|---|---|
| 192.168.1.0 | Directly connected |
| 192.168.2.0 | Directly connected |
| 192.168.3.0 | Directly connected |
| 192.168.4.0 | B, C |
| 192.168.5.0 | B, C |
| 192.168.6.0 | B, C |
| 192.168.7.0 | B, C |

This route table says that the first three networks are directly connected and that no routing is needed from Router A to reach them, which is correct. The last four networks, according to this table, can be reached via Router B or Router C. This information is also correct. But if network 192.168.7.0 can be reached via either Router B or Router C, which path is the preferable path? Metrics are needed to rank the alternatives.

Different routing protocols use different metrics. For example, RIP defines the "best" route as the one with the least number of router hops; EIGRP defines the "best" route based on a combination of the lowest bandwidth along the route and the total delay of the route. The following sections provide basic definitions of these and other commonly used metrics. Further complexities—such as how some routing protocols such as EIGRP use multiple parameters to compute a metric and deal with routes that have identical metric values—are covered later, in the protocol-specific chapters of this book.

## Hop Count

A hop-count metric simply counts router hops. For instance, from Router A, it is one hop to network 192.168.5.0 if packets are sent out interface 192.168.3.1 (through Router B) and two hops if packets are sent out 192.168.1.1 (through Routers C and B). Assuming hop count is the only metric being applied, the best route is the one with the fewest hops, in this case, A-B.

But is the A-B link really the best path? If the A-B link is a DS-0 link and the A-C and C-B links are T-1 links, the two-hop route might actually be best, because bandwidth plays a role in how efficiently traffic travels through the network.

## Bandwidth

A bandwidth metric would choose a higher-bandwidth path over a lower-bandwidth link. However, bandwidth by itself still might not be a good metric. What if one or both of the T1 links are heavily loaded with other traffic and the 56K link is lightly loaded? Or what if the higher-bandwidth link also has a higher delay?

### Load

This metric reflects the amount of traffic utilizing the links along the path. The best path is the one with the lowest load.

Unlike hop count and bandwidth, the load on a route changes, and, therefore, the metric will change. Care must be taken here. If the metric changes too frequently, *route flapping*—the frequent change of preferred routes—might occur. Route flaps can have adverse effects on the router's CPU, the bandwidth of the data links, and the overall stability of the network.

### Delay

*Delay* is a measure of the time a packet takes to traverse a route. A routing protocol using delay as a metric would choose the path with the least delay as the best path. There might be many ways to measure delay. Delay might take into account not only the delay of the links along the route, but also such factors as router latency and queuing delay. On the other hand, the delay of a route might not be measured at all; it might be a sum of static quantities defined for each interface along the path. Each individual delay quantity would be an estimate based on the type of link to which the interface is connected.

### Reliability

*Reliability* measures the likelihood that the link will fail in some way and can be either variable or fixed. Examples of variable-reliability metrics are the number of times a link has failed, or the number of errors it has received within a certain time period. Fixed-reliability metrics are based on known qualities of a link as determined by the network administrator. The path with highest reliability would be selected as best.

### Cost

This metric is configured by a network administrator to reflect more- or less-preferred routes. Cost might be defined by any policy or link characteristic or might reflect the arbitrary judgment of the network administrator. Therefore, "cost" is a term of convenience describing a dimensionless metric.

The term *cost* is often used as a generic term when speaking of route choices. For example, "RIP chooses the lowest-cost path based on hop count." Another generic term is *shortest*, as in "RIP chooses the shortest path based on hop count." When used in this context, either *lowest-cost* (or *highest-cost*) and *shortest* (or *longest*) merely refer to a routing protocol's view of paths based on its specific metrics.

## Convergence

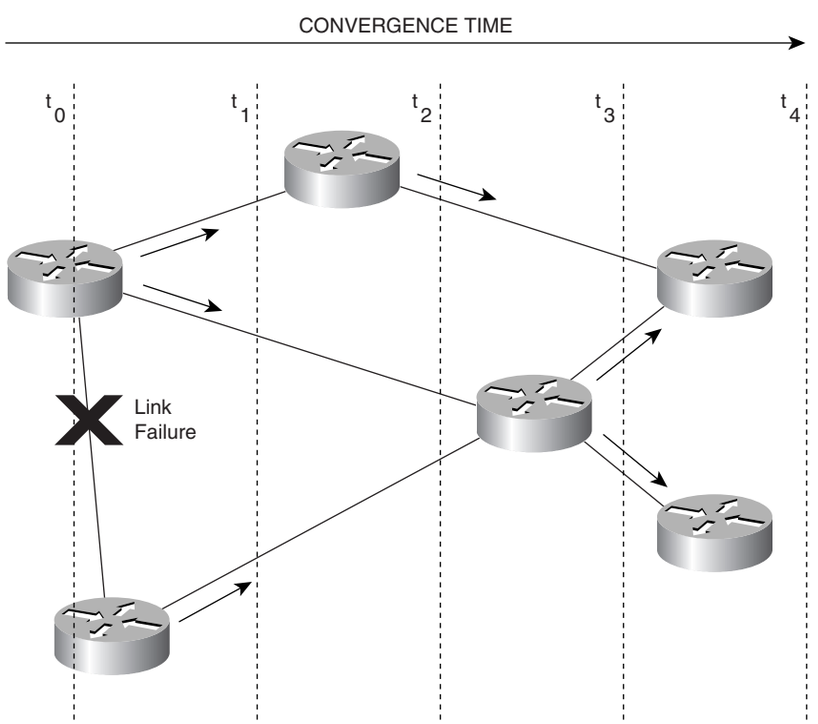
A dynamic routing protocol must include a set of procedures for a router to inform other routers about its directly connected networks, to receive and process the same information from other routers, and to pass along the information it receives from other routers. Further, a routing protocol must define a metric by which best paths might be determined.

A further criterion for routing protocols is that the reachability information in the route tables of all routers in the network must be consistent. If Router A in Figure 4-1 determines that the best path to network 192.168.5.0 is via Router C and if Router C determines that the best path to the same network is through Router A, Router A will send packets destined for 192.168.5.0 to C, C will send them back to A, A will again send them to C, and so on. This continuous circling of traffic between two or more destinations is referred to as a *routing loop*.

The process of bringing all route tables to a state of consistency is called *convergence*. The time it takes to share information across a network and for all routers to calculate best paths is the *convergence time*.

Figure 4-2 shows a network that was converged, but now a topology change has occurred. The link between the two left-most routers has failed; both routers, being directly connected, know about the failure from the data link protocol and proceed to inform their neighbors of the unavailable link. The neighbors update their route tables accordingly and inform their neighbors, and the process continues until all routers know about the change.

**Figure 4-2** *Reconvergence after a topology change takes time. While the network is in an unconverged state, routers are susceptible to bad routing information.*




Notice that at time $t_2$ the three left-most routers recognize the topology change but the three right-most routers have not yet received that information. Those three have old information

and will continue to switch packets accordingly. It is during this intermediate time, when the network is in an unconverged state, that routing errors might occur. Therefore, convergence time is an important factor in any routing protocol. The faster a network can reconverge after a topology change, the better.

### Load Balancing

Recall from Chapter 3, "Static Routing," that load balancing is the practice of distributing traffic among multiple paths to the same destination, so as to use bandwidth efficiently. As an example of the usefulness of load balancing, consider Figure 4-1 again. All the networks in Figure 4-1 are reachable from two paths. If a device on 192.168.2.0 sends a stream of packets to a device on 192.168.6.0, Router A might send them all via Router B or Router C. In both cases, the network is one hop away. However, sending all packets on a single route probably is not the most efficient use of available bandwidth. Instead, load balancing should be implemented to alternate traffic between the two paths. As noted in Chapter 3, load balancing can be equal cost or unequal cost, and per packet or per destination.

## Distance Vector Routing Protocols

Most routing protocols fall into one of two classes: *distance vector* or *link state*. The basics of distance vector routing protocols are examined here; the next section covers link state routing protocols. Most distance vector algorithms are based on the work done of R. E. Bellman,[3] L. R. Ford, and D. R. Fulkerson,[4] and for this reason occasionally are referred to as *Bellman-Ford or Ford-Fulkerson algorithms.* A notable exception is EIGRP, which is based on an algorithm developed by J. J. Garcia Luna Aceves.

The name distance vector is derived from the fact that routes are advertised as vectors of (distance, direction), where distance is defined in terms of a metric and direction is defined in terms of the next-hop router. For example, "Destination A is a distance of five hops away, in the direction of next-hop Router X." As that statement implies, each router learns routes from its neighboring routers' perspectives and then advertises the routes from its own perspective. Because each router depends on its neighbors for information, which the neighbors in turn might have learned from their neighbors, and so on, distance vector routing is sometimes facetiously referred to as "routing by rumor."

Distance vector routing protocols include the following:

- Routing Information Protocol (RIP) for IP
- Xerox Networking System's XNS RIP
- Novell's IPX RIP

[3] R. E. Bellman. *Dynamic Programming*. Princeton, New Jersey: Princeton University Press; 1957.

[4] L. R. Ford Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton, New Jersey: Princeton University Press; 1962.

- The Cisco Systems Internet Gateway Routing Protocol (IGRP) and Enhanced Internet Gateway Routing Protocol (EIGRP)
- DEC's DNA Phase IV
- AppleTalk's Routing Table Maintenance Protocol (RTMP)

## Common Characteristics

A typical distance vector routing protocol uses a routing algorithm in which routers periodically send routing updates to all neighbors by broadcasting their entire route tables.[5]

The preceding statement contains a lot of information. Following sections consider it in more detail.

### Periodic Updates

*Periodic updates* means that at the end of a certain time period, updates will be transmitted. This period typically ranges from 10 seconds for AppleTalk's RTMP to 90 seconds for the Cisco IGRP. At issue here is the fact that if updates are sent too frequently, congestion and router CPU overloading might occur; if updates are sent too infrequently, convergence time might be unacceptably high.

### Neighbors

In the context of routers, *neighbors* means routers sharing a common data link or some higher-level logical adjacency. A distance vector routing protocol sends its updates to neighboring routers[6] and depends on them to pass the update information along to their neighbors. For this reason, distance vector routing is said to use hop-by-hop updates.

### Broadcast Updates

When a router first becomes active on a network, how does it find other routers and how does it announce its own presence? Several methods are available. The simplest is to send the updates to the broadcast address (in the case of IP, 255.255.255.255). Neighboring routers speaking the same routing protocol will hear the broadcasts and take appropriate action. Hosts and other devices uninterested in the routing updates will simply drop the packets.

---

[5] A notable exception to this convention is the Cisco Enhanced IGRP. EIGRP is a distance vector protocol, but its updates are not periodic, are not broadcasted, and do not contain the full route table. EIGRP is covered in Chapter 7, "Enhanced Interior Gateway Routing Protocol (EIGRP)."

[6] This statement is not entirely true. Hosts also can listen to routing updates in some implementations; but all that is important for this discussion is how routers work.

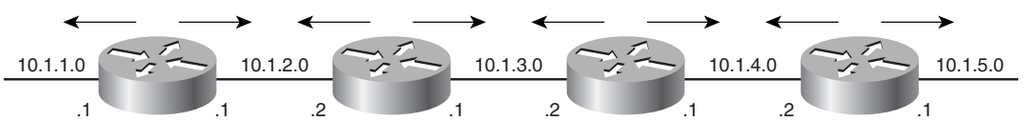
### Full Routing Table Updates

Most distance vector routing protocols take the very simple approach of telling their neighbors everything they know by broadcasting their entire route table, with some exceptions that are covered in following sections. Neighbors receiving these updates glean the information they need and discard everything else.

## Routing by Rumor

Figure 4-3 shows a distance vector algorithm in action. In this example, the metric is hop count. At time $t_0$, Routers A through D have just become active. Looking at the route tables across the top row, at $t_0$ the only information any of the four routers has is its own directly connected networks. The tables identify these networks and indicate that they are directly connected by having no next-hop router and by having a hop count of 0. Each of the four routers will broadcast this information on all links.

**Figure 4-3** *Distance vector protocols converge hop-by-hop.*




| | Router A | | | Router B | | | Router C | | | Router D | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NET | VIA | HOPS | NET | VIA | HOPS | NET | VIA | HOPS | NET | VIA | HOPS |
| $t_0$ | 10.1.1.0 | -- | 0 | 10.1.2.0 | -- | 0 | 10.1.3.0 | -- | 0 | 10.1.4.0 | -- | 0 |
| | 10.1.2.0 | -- | 0 | 10.1.3.0 | -- | 0 | 10.1.4.0 | -- | 0 | 10.1.5.0 | -- | 0 |
| $t_1$ | 10.1.1.0 | -- | 0 | 10.1.2.0 | -- | 0 | 10.1.3.0 | -- | 0 | 10.1.4.0 | -- | 0 |
| | 10.1.2.0 | -- | 0 | 10.1.3.0 | -- | 0 | 10.1.4.0 | -- | 0 | 10.1.5.0 | -- | 0 |
| | 10.1.3.0 | 10.1.2.2 | 1 | 10.1.1.0 | 10.1.2.1 | 1 | 10.1.2.0 | 10.1.3.1 | 1 | 10.1.3.0 | 10.1.4.1 | 1 |
| | | | | 10.1.4.0 | 10.1.3.2 | 1 | 10.1.5.0 | 10.1.4.2 | 1 | | | |
| $t_2$ | 10.1.1.0 | -- | 0 | 10.1.2.0 | -- | 0 | 10.1.3.0 | -- | 0 | 10.1.4.0 | -- | 0 |
| | 10.1.2.0 | -- | 0 | 10.1.3.0 | -- | 0 | 10.1.4.0 | -- | 0 | 10.1.5.0 | -- | 0 |
| | 10.1.3.0 | 10.1.2.2 | 1 | 10.1.1.0 | 10.1.2.1 | 1 | 10.1.2.0 | 10.1.3.1 | 1 | 10.1.3.0 | 10.1.4.1 | 1 |
| | 10.1.4.0 | 10.1.2.2 | 2 | 10.1.4.0 | 10.1.3.2 | 1 | 10.1.5.0 | 10.1.4.2 | 1 | 10.1.2.0 | 10.1.4.1 | 2 |
| | | | | 10.1.5.0 | 10.1.3.2 | 2 | 10.1.1.0 | 10.1.3.1 | 2 | | | |
| $t_3$ | 10.1.1.0 | -- | 0 | 10.1.2.0 | -- | 0 | 10.1.3.0 | -- | 0 | 10.1.4.0 | -- | 0 |
| | 10.1.2.0 | -- | 0 | 10.1.3.0 | -- | 0 | 10.1.4.0 | -- | 0 | 10.1.5.0 | -- | 0 |
| | 10.1.3.0 | 10.1.2.2 | 1 | 10.1.1.0 | 10.1.2.1 | 1 | 10.1.2.0 | 10.1.3.1 | 1 | 10.1.3.0 | 10.1.4.1 | 1 |
| | 10.1.4.0 | 10.1.2.2 | 2 | 10.1.4.0 | 10.1.3.2 | 1 | 10.1.5.0 | 10.1.4.2 | 1 | 10.1.2.0 | 10.1.4.1 | 2 |
| | 10.1.5.0 | 10.1.2.2 | 3 | 10.1.5.0 | 10.1.3.2 | 2 | 10.1.1.0 | 10.1.3.1 | 2 | 10.1.1.0 | 10.1.4.1 | 3 |

At time $t_1$, the first updates have been received and processed by the routers. Look at Router A's table at $t_1$. Router B's update to Router A said that Router B can reach networks 10.1.2.0

and 10.1.3.0, both zero hops away. If the networks are zero hops from B, they must be one hop from A. Router A incremented the hop count by one and then examined its route table. It already recognized 10.1.2.0, and the hop count (zero) was less than the hop count B advertised, (one), so A disregarded that information.

Network 10.1.3.0 was new information, however, so A entered this in the route table. The source address of the update packet was Router B's interface (10.1.2.2) so that information is entered along with the calculated hop count.

Notice that the other routers performed similar operations at the same time $t_1$. Router C, for instance, disregarded the information about 10.1.3.0 from B and 10.1.4.0 from C but entered information about 10.1.2.0, reachable via B's interface address 10.1.3.1, and 10.1.5.0, reachable via C's interface 10.1.4.2. Both networks were calculated as one hop away.

At time $t_2$, the update period has again expired and another set of updates has been broadcast. Router B sent its latest table; Router A again incremented B's advertised hop counts by one and compared. The information about 10.1.2.0 is again discarded for the same reason as before. 10.1.3.0 is already known, and the hop count hasn't changed, so that information is also discarded. 10.1.4.0 is new information and is entered into the route table.

The network is converged at time $t_3$. Every router recognizes every network, the address of the next-hop router for every network, and the distance in hops to every network.

It is time for an analogy. You are wandering in the Sangre de Cristo mountains of northern New Mexico—a wonderful place to wander if you aren't lost. But you are lost. You come upon a fork in the trail and a sign pointing west, reading "Taos, 15 miles." You have no choice but to trust the sign. You have no clue what the terrain is like over that 15 miles; you don't know whether there is a better route or even whether the sign is correct. Someone could have turned it around, in which case you will travel deeper into the forest instead of to safety!

Distance vector algorithms provide road signs to networks.[7] They provide the direction and the distance, but no details about what lies along the route. And like the sign at the fork in the trail, they are vulnerable to accidental or intentional misdirection. Following are some of the difficulties and refinements associated with distance vector algorithms.

## Route Invalidation Timers

Now that the network in Figure 4-3 is fully converged, how will it handle reconvergence when some part of the topology changes? If network 10.1.5.0 goes down, the answer is simple enough—Router D, in its next scheduled update, flags the network as unreachable and passes the information along.

But what if, instead of 10.1.5.0 going down, Router D fails? Routers A, B, and C still have entries in their route tables about 10.1.5.0; the information is no longer valid, but

[7] The road sign analogy is commonly used. You can find a good presentation in Radia Perlman's *Interconnections*, pp. 205–210.

there's no router to inform them of this fact. They will unknowingly forward packets to an unreachable destination—a black hole has opened in the network.

This problem is handled by setting a route invalidation timer for each entry in the route table. For example, when Router C first hears about 10.1.5.0 and enters the information into its route table, C sets a timer for that route. At every regularly scheduled update from Router D, C discards the update's already-known information about 10.1.5.0 as described in "Routing by Rumor." But as C does so, it resets the timer on that route.

If Router D goes down, C will no longer hear updates about 10.1.5.0. The timer will expire, C will flag the route as unreachable and will pass the information along in the next update.

Typical periods for route timeouts range from three to six update periods. A router would not want to invalidate a route after a single update has been missed, because this event might be the result of a corrupted or lost packet or some sort of network delay. At the same time, if the period is too long, reconvergence will be excessively slow.

## Split Horizon

According to the distance vector algorithm as it has been described so far, at every update period each router broadcasts its entire route table to every neighbor. But is this really necessary? Every network known by Router A in Figure 4-3, with a hop count higher than zero, has been learned from Router B. Common sense suggests that for Router A to broadcast the networks it has learned from Router B back to Router B is a waste of resources. Obviously, B already "knows" about those networks.

A route pointing back to the router from which packets were received is called a *reverse route*. *Split horizon* is a technique for preventing reverse routes between two routers.

Besides not wasting resources, there is a more important reason for not sending reachability information back to the router from which the information was learned. The most important function of a dynamic routing protocol is to detect and compensate for topology changes—if the best path to a network becomes unreachable, the protocol must look for a next-best path.

Look yet again at the converged network of Figure 4-3 and suppose that network 10.1.5.0 goes down. Router D will detect the failure, flag the network as unreachable, and pass the information along to Router C at the next update interval. However, before D's update timer triggers an update, something unexpected happens. C's update arrives, claiming that it can reach 10.1.5.0, one hop away! Remember the road sign analogy? Router D has no way of recognizing that C is not advertising a legitimate next-best path. It will increment the hop count and make an entry into its route table indicating that 10.1.5.0 is reachable via Router C's interface 10.1.4.1, just two hops away.

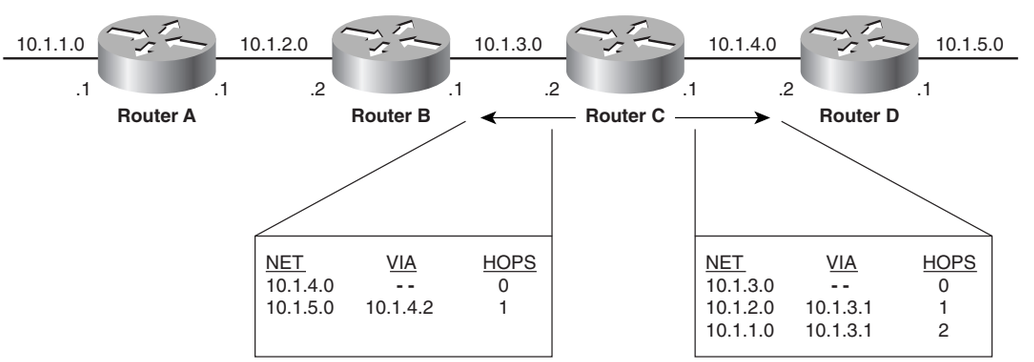
Now a packet with a destination address of 10.1.5.3 arrives at Router C, which consults its route table and forwards the packet to D. Router D consults its route table and forwards the packet to C, C forwards it back to D, *ad infinitum*. A routing loop has occurred.

Implementing split horizon prevents the possibility of such a routing loop. There are two categories of split horizon: simple split horizon and split horizon with poisoned reverse.

The rule for simple split horizon is, when sending updates out a particular interface, do not include networks that were learned from updates received on that interface.

The routers in Figure 4-4 implement simple split horizon. Router C sends an update to Router D for networks 10.1.1.0, 10.1.2.0, and 10.1.3.0. Networks 10.1.4.0 and 10.1.5.0 are not included because they were learned from Router D. Likewise, updates to Router B include 10.1.4.0 and 10.1.5.0 with no mention of 10.1.1.0, 10.1.2.0, and 10.1.3.0.

**Figure 4-4** *Simple split horizon does not advertise routes back to the neighbors from whom the routes were learned.*



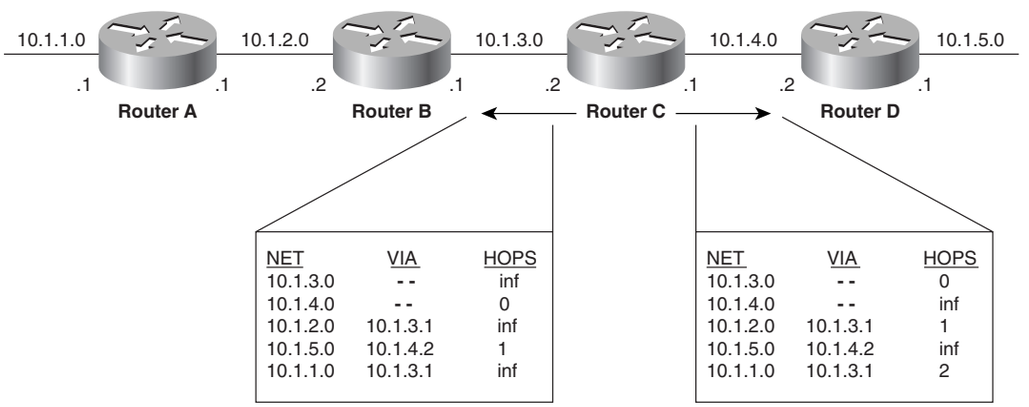

Simple split horizon works by suppressing information. Split horizon with poisoned reverse is a modification that provides more positive information.

The rule for split horizon with poisoned reverse is, when sending updates out a particular interface, designate any networks that were learned from updates received on that interface as unreachable.

In the scenario of Figure 4-4, Router C would in fact advertise 10.1.4.0 and 10.1.5.0 to Router D, but the network would be marked as unreachable. Figure 4-5 shows what the route tables from C to B and D would look like. Notice that a route is marked as unreachable by setting the metric to infinity; in other words, the network is an infinite distance away. Coverage of a routing protocol's concept of infinity continues in the next section.

Split horizon with poisoned reverse is considered safer and stronger than simple split horizon—a sort of "bad news is better than no news at all" approach. For example, suppose that Router B in Figure 4-5 receives corrupted information, causing it to proceed as if subnet 10.1.1.0 is reachable via Router C. Simple split horizon would do nothing to correct this misperception, whereas a poisoned reverse update from Router C would immediately stop the potential loop. For this reason, most modern distance vector implementations use split horizon with poisoned reverse. The trade-off is that routing update packets are larger, which might exacerbate any congestion problems on a link.

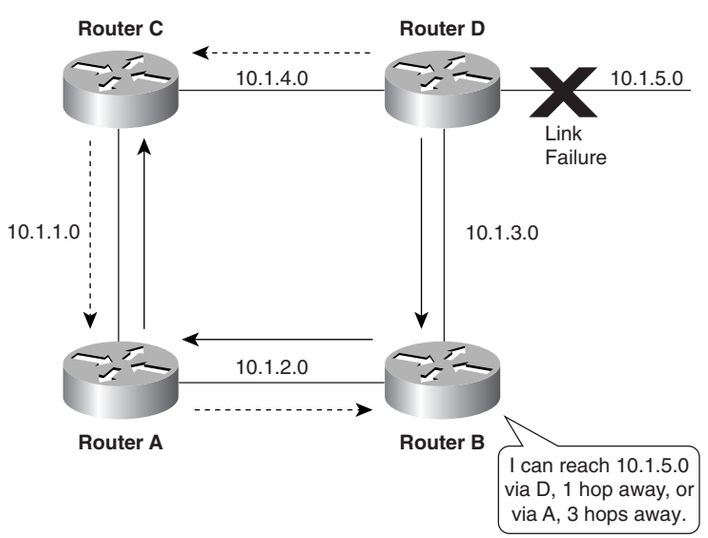
**Figure 4-5** *Split horizon with poisoned reverse advertises reverse routes but with an unreachable (infinite) metric.*




## Counting to Infinity

Split horizon will break loops between neighbors, but it will not stop loops in a network such as the one in Figure 4-6. Again, 10.1.5.0 has failed. Router D sends the appropriate updates to its neighbors, Router C (the dashed arrows), and Router B (the solid arrows). Router B marks the route via D as unreachable, but Router A is advertising a next-best path to 10.1.5.0, which is three hops away. B posts that route in its route table.

**Figure 4-6** *Split horizon will not prevent routing loops here.*



B now informs D that it has an alternative route to 10.1.5.0. D posts this information and updates C, saying that it has a four-hop route to the network. C tells A that 10.1.5.0 is five hops away. A tells B that the network is now six hops away.

"Ah," Router B thinks, "Router A's path to 10.1.5.0 has increased in length. Nonetheless, it's the only route I've got, so I'll use it!"

B changes the hop count to seven, updates D, and around it goes again. This situation is the *counting-to-infinity* problem because the hop count to 10.1.5.0 will continue to increase to infinity. All routers are implementing split horizon, but it doesn't help.

The way to alleviate the effects of counting to infinity is to define infinity. Most distance vector protocols define infinity to be 16 hops. As the updates continue to loop among the routers in Figure 4-6, the hop count to 10.1.5.0 in all routers will eventually increment to 16. At that time, the network will be considered unreachable.

This method is also how routers advertise a network as unreachable. Whether it is a poisoned reverse route, a network that has failed, or a network beyond the maximum network diameter of 15 hops, a router will recognize any 16-hop route as unreachable.

Setting a maximum hop count of 15 helps solve the counting-to-infinity problem, but convergence will still be very slow. Given an update period of 30 seconds, a network could take up to 7.5 minutes to reconverge and is susceptible to routing errors during this time. Triggered updates can be used to reduce this convergence time.

## Triggered Updates

*Triggered updates*, also known as *flash updates*, are very simple: If a metric changes for better or for worse, a router will immediately send out an update without waiting for its update timer to expire. Reconvergence will occur far more quickly than if every router had to wait for regularly scheduled updates, and the problem of counting to infinity is greatly reduced, although not completely eliminated. Regular updates might still occur along with triggered updates. Thus a router might receive bad information about a route from a not-yet-reconverged router after having received correct information from a triggered update. Such a situation shows that confusion and routing errors might still occur while a network is reconverging, but triggered updates will help to iron things out more quickly.

A further refinement is to include in the update only the networks that actually triggered it, rather than the entire route table. This technique reduces the processing time and the impact on network bandwidth.

## Holddown Timers

Triggered updates add responsiveness to a reconverging network. *Holddown timers* introduce a certain amount of skepticism to reduce the acceptance of bad routing information.

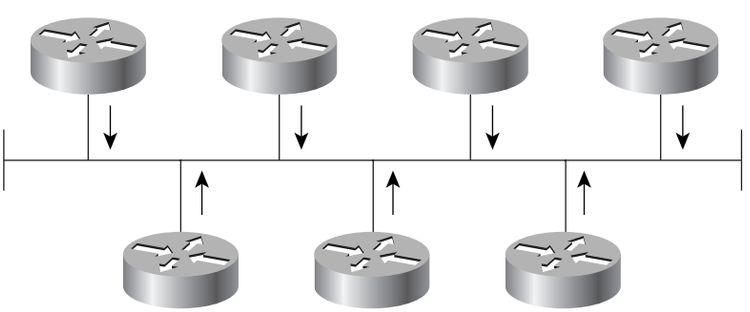
If the distance to a destination increases (for example, the hop count increases from two to four), the router sets a holddown timer for that route. Until the timer expires, the router will not accept any new updates for the route.

Obviously, a trade-off is involved here. The likelihood of bad routing information getting into a table is reduced but at the expense of the reconvergence time. Like other timers, holddown timers must be set with care. If the holddown period is too short, it will be ineffective, and if it is too long, normal routing will be adversely affected.

## Asynchronous Updates

Figure 4-7 shows a group of routers connected to an Ethernet backbone. The routers should not broadcast their updates at the same time; if they do, the update packets will collide. Yet this situation is exactly what can happen when several routers share a broadcast network. System delays related to the processing of updates in the routers tend to cause the update timers to become synchronized. As a few routers become synchronized, collisions will begin to occur, further contributing to system delays, and eventually all routers sharing the broadcast network might become synchronized.

**Figure 4-7** *If update timers become synchronized, collisions might occur.*



Asynchronous updates might be maintained by one of two methods:

- Each router's update timer is independent of the routing process and is, therefore, not affected by processing loads on the router.
- A small random time, or *timing jitter*, is added to each update period as an offset.

If routers implement the method of rigid, system-independent timers, all routers sharing a broadcast network must be brought online in a random fashion. Rebooting the entire group of routers simultaneously—such as might happen during a widespread power outage, for example—could result in all the timers attempting to update at the same time.

Adding randomness to the update period is effective if the variable is large enough in proportion to the number of routers sharing the broadcast network. Sally Floyd and Van

Jacobson[8] have calculated that a too-small randomization will be overcome by a large enough network of routers, and that to be effective the update timer should range up to 50 percent of the median update period.

## Link State Routing Protocols

The information available to a distance vector router has been compared to the information available from a road sign. Link state routing protocols are like a road map. A link state router cannot be fooled as easily into making bad routing decisions, because it has a complete picture of the network. The reason is that unlike the routing-by-rumor approach of distance vector, link state routers have firsthand information from all their peer[9] routers. Each router originates information about itself, its directly connected links, the state of those links (hence the name), and any directly connected neighbors. This information is passed around from router to router, each router making a copy of it, but never changing it. The ultimate objective is that every router has identical information about the network, and each router will independently calculate its own best paths.

Link state protocols, sometimes called *shortest path first* or *distributed database* protocols, are built around a well-known algorithm from graph theory, E. W. Dijkstra's shortest path algorithm. Examples of link state routing protocols are

- Open Shortest Path First (OSPF) for IP
- The ISO's Intermediate System-to-Intermediate System (IS-IS) for CLNS and IP
- DEC's DNA Phase V
- Novell's NetWare Link Services Protocol (NLSP)

Although link-state protocols are rightly considered more complex than distance vector protocols, the basic functionality is not complex at all:

1. Each router establishes a relationship—an adjacency—with each of its neighbors.
2. Each router sends a data unit we will call *link state advertisements* (LSAs) to each neighbor.[10] The LSA lists each of the router's links, and for each link, it identifies the link, the state of the link, the metric cost of the router's interface to the link, and any neighbors that might be connected to the link. Each neighbor receiving an advertisement in turn forwards (*floods*) the advertisement to its own neighbors.
3. Each router stores a copy of all the LSAs it has seen in a database. If all works well, the databases in all routers should be identical.

---

[8] S. Floyd and V. Jacobson. "The Synchronisation of Periodic Routing Messages." ACM Sigcomm '93 Symposium, September 1993.

[9] That is, all routers speaking the same routing protocol.

[10] LSA is an OSPF term. The corresponding data unit created by IS-IS is called link state PDU (LSP). But for this general discussion, "LSA" fits our needs.

4 The completed *topological database*, also called the *link state database*, is used by the Dijkstra algorithm to compute a graph of the network indicating the shortest path to each router. The link state protocol can then consult the link state database to find what subnets are attached to each router, and enter this information into the routing table.

## Neighbors

Neighbor discovery is the first step in getting a link state environment up and running. In keeping with the friendly neighbor terminology, a Hello protocol is used for this step. The protocol will define a Hello packet format and a procedure for exchanging the packets and processing the information the packets contain.

At a minimum, the Hello packet will contain a *router ID* (RID) and the address of the network on which the packet is being sent. The router ID is something by which the router originating the packet can be uniquely distinguished from all other routers; for instance, an IP address from one of the router's interfaces. Other fields of the packet might carry a subnet mask, Hello intervals, a specified maximum period the router will wait to hear a Hello before declaring the neighbor "dead," a descriptor of the circuit type, and flags to help in bringing up adjacencies.

When two routers have discovered each other as neighbors, they go through a process of synchronizing their databases in which they exchange and verify database information until their databases are identical. The details of database synchronization are described in Chapters 8, "OSPFv2," and 10, "Integrated IS-IS." To perform this database synchronization, the neighbors must be *adjacent*—that is, they must agree on certain protocol-specific parameters such as timers and support of optional capabilities. By using Hello packets to build adjacencies, link state protocols can exchange information in a controlled fashion. Contrast this approach with distance vector, which simply broadcasts updates out any interface configured for that routing protocol.

Beyond building adjacencies, Hello packets serve as keepalives to monitor the adjacency. If Hellos are not heard from an adjacent neighbor within a certain established time, the neighbor is considered unreachable and the adjacency is broken. A typical interval for the exchange of Hello packets is ten seconds, and a typical dead period is four times that.

## Link State Flooding

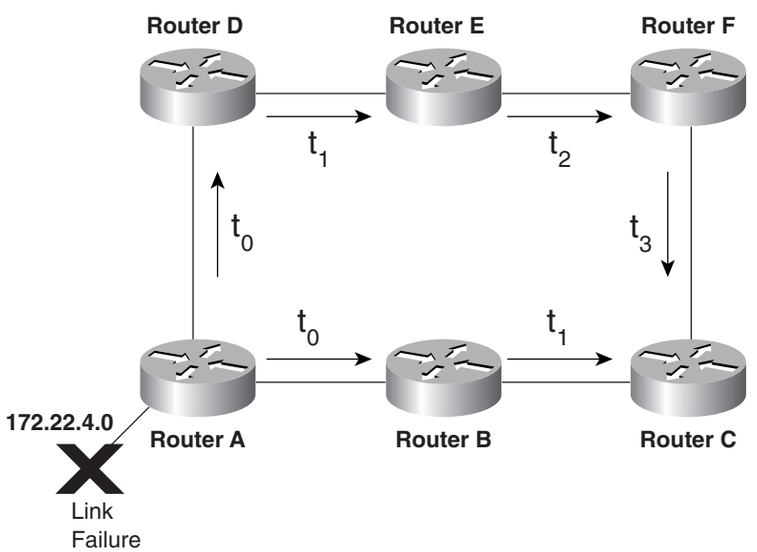
After the adjacencies are established, the routers might begin sending out LSAs. As the term *flooding* implies, the advertisements are sent to every neighbor. In turn, each received LSA is copied and forwarded to every neighbor except the one that sent the LSA. This process is the source of one of link state's advantages over distance vector. LSAs are forwarded almost immediately, whereas distance vector protocols based on the Bellman-Ford algorithm must run its algorithm and update its route table before routing updates, even the triggered ones, can be forwarded. As a result, link state protocols converge much faster than distance vector protocols converge when the topology changes.

The flooding process is the most complex piece of a link state protocol. There are several ways to make flooding more efficient and more reliable, such as using unicast and multicast addresses, checksums, and positive acknowledgments. These topics are examined in the protocol-specific chapters, but two procedures are vitally important to the flooding process: sequencing and aging.

## Sequence Numbers

A difficulty with flooding, as described so far, is that when all routers have received all LSAs, the flooding must stop. A time-to-live value in the packets could simply be relied on to expire, but it is hardly efficient to permit LSAs to wander the network until they expire. Take the network in Figure 4-8. Subnet 172.22.4.0 at Router A has failed, and A has flooded an LSA to its neighbors B and D, advertising the new state of the link. B and D dutifully flood to their neighbors, and so on.

**Figure 4-8** *When a topology change occurs, LSAs advertising the change will be flooded throughout the network.*




Look next at what happens at Router C. An LSA arrives from Router B at time $t_1$, is entered into C's topological database, and is forwarded to Router F. At some later time $t_3$, another copy of the same LSA arrives from the longer A-D-E-F-C route. Router C sees that it already has the LSA in its database; the question is, should C forward this LSA to Router B? The answer is no because B has already received the advertisement. Router C knows this because the sequence number of the LSA it received from Router F is the same as the sequence number of the LSA it received earlier from Router B.

When Router A sent out the LSA, it included an identical sequence number in each copy. This sequence number is recorded in the routers' topological databases along with the rest of the LSA; when a router receives an LSA that is already in the database and its sequence number is the same, the received information is discarded. If the information is the same but the sequence number is greater, the received information and new sequence number are entered into the database and the LSA is flooded. In this way, flooding is abated when all routers have seen a copy of the most recent LSA.

As described so far, it seems that routers could merely verify that their link state databases contain the same LSA as the newly received LSA and make a flood/discard decision based on that information, without needing a sequence number. But imagine that immediately after Figure 4-8's network 172.22.4.0 failed, it came back up. Router A might send out an LSA advertising the network as down, with a sequence number of 166; then it sends out a new LSA announcing the same network as up, with a sequence number of 167. Router C receives the down LSA and then the up LSA from the A-B-C path, but then it receives a delayed down LSA from the A-D-E-F-C path. Without the sequence numbers, C would not know whether or not to believe the delayed down LSA. With sequence numbers, C's database will indicate that the information from Router A has a sequence number of 167; the late LSA has a sequence number of 166 and is therefore recognized as old information and discarded.

Because the sequence numbers are carried in a set field within the LSAs, the numbers must have some upper boundary. What happens when this maximum sequence number is reached?

### Linear Sequence Number Spaces

One approach is to use a linear sequence number space so large that it is unlikely the upper limit will ever be reached. If, for instance, a 32-bit field is used, there are $2^{32} = 4{,}294{,}967{,}296$ available sequence numbers starting with zero. Even if a router was creating a new link state packet every 10 seconds, it would take 1361 years to exhaust the sequence number supply; few routers are expected to last so long.

In this imperfect world, unfortunately, malfunctions occur. If a link state routing process somehow runs out of sequence numbers, it must shut itself down and stay down long enough for its LSAs to age out of all databases before starting over at the lowest sequence number (see the section "Aging," later in this chapter).

A more common difficulty presents itself during router restarts. If Router A restarts, it probably will have no way of remembering the sequence number it last used and must begin again at, say, one. But if its neighbors still have Router A's previous sequence numbers in their databases, the lower sequence numbers will be interpreted as older sequence numbers and will be ignored. Again, the routing process must stay down until all old LSAs are aged out of the network. Given that a maximum age might be an hour or more, this solution is not very attractive.

A better solution is to add a new rule to the flooding behavior described thus far: If a restarted router issues to a neighbor an LSA with a sequence number that appears to be older than the neighbor's stored sequence number, the neighbor will send its own stored LSA and sequence number back to the router. The router will thus learn the sequence number it was using before it restarted and can adjust accordingly.

Care must be taken, however, that the last-used sequence number was not close to the maximum; otherwise, the restarting router will simply have to restart again. A rule must be set limiting the "jump" the router might make in sequence numbers—for instance, a rule might say that the sequence numbers cannot make a single increase more than one-half the total sequence number space. (The actual formulas are more complex than this example, taking into account age constraints.)

IS-IS uses a 32-bit linear sequence number space.

### Circular Sequence Number Spaces

Another approach is to use a circular sequence number space, where the numbers "wrap"—that is, in a 32-bit space, the number following 4,294,967,295 is 0. Malfunctions can cause interesting dilemmas here, too. A restarting router might encounter the same believability problem as discussed for linear sequence numbers.

Circular sequence numbering creates a curious bit of illogic. If $x$ is any number between 1 and 4,294,967,295 inclusive, then $0 < x < 0$. This situation can be managed in well-behaved networks by asserting two rules for determining when a sequence number is greater than or less than another sequence number. Given a sequence number space $n$ and two sequence numbers $a$ and $b$, $a$ is considered more recent (of larger magnitude) in either of the following situations:

$a > b$, and $(a - b) \leq n/2$
$a < b$, and $(b - a) > n/2$

For the sake of simplicity, take a sequence number space of six bits, shown in Figure 4-9:

$n = 2^6 = 64$
so $n/2 = 32$.

Given two sequence numbers 48 and 18, 48 is more recent because by rule (1)

$48 > 18$ and $(48 - 18) = 30$ and $30 < 32$.

Given two sequence numbers 3 and 48, 3 is more recent because by rule (2)

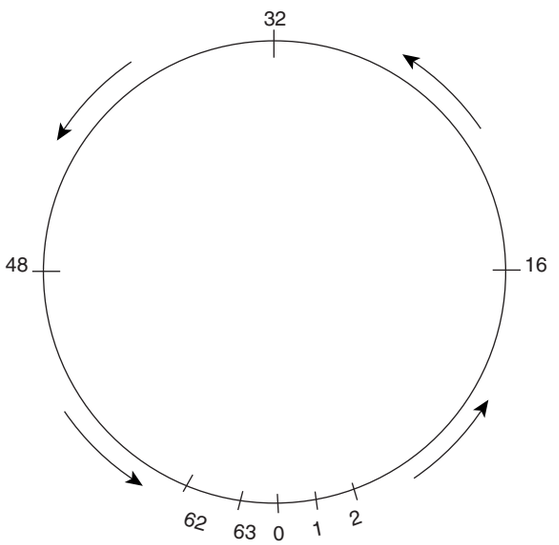
$3 < 48$ and $(48 - 3) = 45$ and $45 > 32$.

Given two sequence numbers 3 and 18, 18 is more recent because by rule (1)

$18 > 3$ and $(18 - 3) = 15$ and $15 < 32$.

So the rules seem to enforce circularity.

**Figure 4-9** *Six-Bit Circular Address Space*




But what about a not-so-well-behaved network? Imagine a network running a six-bit sequence number space. Now imagine that one of the routers on the network malfunctions, but as it does so, it blurts out three identical LSAs with a sequence number of 44 (101100). Unfortunately, a neighboring router is also malfunctioning—it is dropping bits. The neighbor drops a bit in the sequence number field of the second LSA, drops yet another bit in the third LSA, and floods all three. The result is three identical LSAs with three different sequence numbers:

| | |
|---|---|
| 44 | (101100) |
| 40 | (101000) |
| 8 | (001000) |

Applying the circularity rules reveals that 44 is more recent than 40, which is more recent than 8, which is more recent than 44! The result is that every LSA will be continuously flooded, and databases will be continually overwritten with the "latest" LSA, until finally buffers become clogged with LSAs, CPUs become overloaded, and the entire network comes crashing down.

This chain of events sounds quite far-fetched. It is, however, factual. The ARPANET, the precursor of the modern Internet, ran an early link state protocol with a six-bit circular sequence number space; on October 27, 1980, two routers experiencing the malfunctions just described brought the entire ARPANET to a standstill.[11]

[11] E. C. Rosen. "Vulnerabilities of Network Control Protocols: An Example." Computer Communication Review, July 1981.

### Lollipop-Shaped Sequence Number Spaces

This whimsically-named construct was proposed by Dr. Radia Perlman.[12] Lollipop-shaped sequence number spaces are a hybrid of linear and circular sequence number spaces; if you think about it, a lollipop has a linear component and a circular component. The problem with circular spaces is that there is no number less than all other numbers. The problem with linear spaces is that they are—well—not circular. That is, their set of sequence numbers is finite.

When Router A restarts, it would be nice to begin with a number *a* that is less than all other numbers. Neighbors will recognize this number for what it is, and if they have a pre-restart number *b* in their databases from Router A, they can send that number to Router A and Router A will jump to that sequence number. Router A might be able to send more than one LSA before hearing about the sequence number it had been using before restarting. Therefore, it is important to have enough restart numbers so that A cannot use them all before neighbors either inform it of a previously used number, or the previously used number ages out of all databases.

These linear restart numbers form the stick of the lollipop. When they have been used up, or after a neighbor has provided a sequence number to which A can jump, A enters a circular number space, the candy part of the lollipop.

One way of designing a lollipop address space is to use signed sequence numbers, where $-k < 0 < k$. The negative numbers counting up from $-k$ to 1 form the stick, and the positive numbers from 0 to $k$ are the circular space. Perlman's rules for the sequence numbers are as follows. Given two numbers $a$ and $b$ and a sequence number space $n$, $b$ is more recent than $a$ if and only if

$a < 0$ and $a < b$, or

$a > 0, a < b$, and $(b - a) < n/2$, or

$a > 0, b > 0, a > b$, and $(a - b) > n/2$

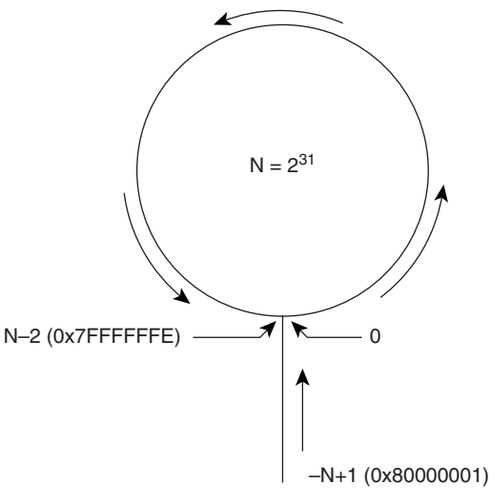
Figure 4-10 shows an implementation of the lollipop-shaped sequence number space. A 32-bit signed number space N is used, yielding $2^{31}$ positive numbers and $2^{31}$ negative numbers. $-N$ ($-2^{31}$, or 0x80000000) and $N - 1$ ($2^{31} - 1$, or 0x7FFFFFFF) are not used. A router coming online will begin its sequence numbers at $-N + 1$ (0x80000001) and increment up to zero, at which time it has entered the circular number space. When the sequence reaches $N - 2$ (0x7FFFFFFE), the sequence wraps back to zero (again, $N - 1$ is unused).

Next, suppose the router restarts. The sequence number of the last LSA sent before the restart is 0x00005de3 (part of the circular sequence space). As it synchronizes its database with its neighbor after the restart, the router sends an LSA with a sequence number of 0x80000001 ($-N + 1$). The neighbor looks into its own database and finds the pre-restart LSA with a sequence number of 0x00005de3. The neighbor sends this LSA to the restarted router, essentially saying, "This is where you left off." The restarted router then records

[12] R. Perlman. "Fault-Tolerant Broadcasting of Routing Information." Computer Networks, Vol. 7, December 1983, pp. 395–405.

the LSA with the positive sequence number. If it needs to send a new copy of the LSA at some future time, the new sequence number will be 0x00005de6.

**Figure 4-10** *Lollipop-shaped sequence number space.*




Lollipop sequence spaces were used with the original version of OSPF, OSPFv1 (RFC 1131). Although the use of signed numbers was an improvement over the linear number space, the circular part was found to be vulnerable to the same ambiguities as a purely circular space. The deployment of OSPFv1 never progressed beyond the experimental stage. The current version of OSPF, OSPFv2 (originally specified in RFC 1247), adopts the best features of linear and lollipop sequence number spaces. It uses a signed number space like lollipop sequence numbers, beginning with 0x80000001. However, when the sequence number goes positive, the sequence space continues to be linear until it reaches the maximum of 0x7FFFFFFF. At that point, the OSPF process must flush the LSA from all link state databases before restarting.

## Aging

The LSA format should include a field for the age of the advertisement. When an LSA is created, the router sets this field to one. As the packet is flooded, each router increments the age of the advertisement.[13]

This process of aging adds another layer of reliability to the flooding process. The protocol defines a maximum age difference (MaxAgeDiff) value for the network. A router might receive multiple copies of the same LSA with identical sequence numbers but different

[13] Of course, another option would be to start with some maximum age and decrement. OSPF increments; IS-IS decrements.

ages. If the difference in the ages is lower than the MaxAgeDiff, it is assumed that the age difference was the result of normal network latencies; the original LSA in the database is retained, and the newer LSA (with the greater age) is not flooded. If the difference is greater than the MaxAgeDiff value, it is assumed that an anomaly has occurred in the network in which a new LSA was sent without incrementing the sequence number. In this case, the newer LSA will be recorded, and the packet will be flooded. A typical MaxAgeDiff value is 15 minutes (used by OSPF).

The age of an LSA continues to be incremented as it resides in a link state database. If the age for a link state record is incremented up to some maximum age (MaxAge)—again defined by the specific routing protocol—the LSA, with age field set to the MaxAge value, is flooded to all neighbors and the record is deleted from the databases.

If the LSA is to be flushed from all databases when MaxAge is reached, there must be a mechanism to periodically validate the LSA and reset its timer before MaxAge is reached. A link state refresh time (LSRefreshTime)[14] is established; when this time expires, a router floods a new LSA to all its neighbors, who will reset the age of the sending router's records to the new received age. OSPF defines a MaxAge of 1 hour and an LSRefreshTime of 30 minutes.

## Link State Database

In addition to flooding LSAs and discovering neighbors, a third major task of the link state routing protocol is establishing the link state database. The link state or topological database stores the LSAs as a series of records. Although a sequence number and age and possibly other information are included in the LSA, these variables exist mainly to manage the flooding process. The important information for the shortest path determination process is the advertising router's ID, its attached networks and neighboring routers, and the cost associated with those networks or neighbors. As the previous sentence implies, LSAs might include two types of generic information:[15]

- *Router link information* advertises a router's adjacent neighbors with a triple of (Router ID, Neighbor ID, Cost), where cost is the cost of the link to the neighbor.
- *Stub network information* advertises a router's directly connected stub subnets (subnets with no neighbors) with a triple of (Router ID, Network ID, Cost).

The shortest path first (SPF) algorithm is run once for the router link information to establish shortest paths to each router, and then stub network information is used to add these networks to the routers. Figure 4-11 shows a network of routers and the links between them; stub networks are not shown for the sake of simplicity. Notice that several links have different costs associated with them at each end. A cost is associated with the outgoing

[14] LSRefreshTime, MaxAge, and MaxAgeDiff are OSPF architectural constants.

[15] Actually, there can be more than two types of information and multiple types of link state packets. They are covered in the chapters on specific link state protocols.

direction of an interface. For instance, the link from RB to RC has a cost of 1, but the same link has a cost of 5 in the RC to RB direction.

**Figure 4-11** *Link costs are calculated for the outgoing direction from an interface and do not necessarily have to be the same at all interfaces on a link.*

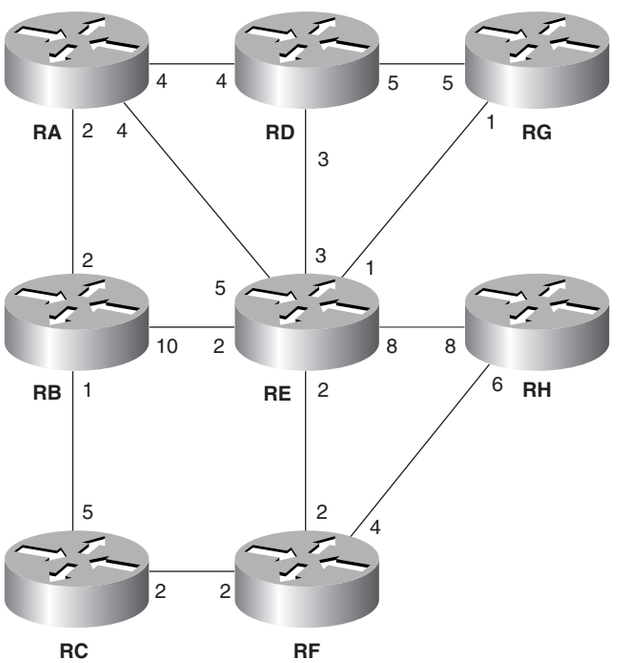



Table 4-2 shows a generic link state database for the network of Figure 4-11, a copy of which is stored in every router. As you read through this database, you will see that it completely describes the network. Now it is possible to compute a tree that describes the shortest path to each router by running the SPF algorithm.

**Table 4-2** *Topological database for the network in Figure 4-11.*

| Router ID | Neighbor | Cost |
|---|---|---|
| RA | RB | 2 |
| RA | RD | 4 |
| RA | RE | 4 |
| RB | RA | 2 |
| RB | RC | 1 |
| RB | RE | 10 |

*continues*

**Table 4-2** *Topological database for the network in Figure 4-11. (Continued)*

| Router ID | Neighbor | Cost |
|---|---|---|
| RC | RB | 5 |
| RC | RF | 2 |
| RD | RA | 4 |
| RD | RE | 3 |
| RD | RG | 5 |
| RE | RA | 5 |
| RE | RB | 2 |
| RE | RD | 3 |
| RE | RF | 2 |
| RE | RG | 1 |
| RE | RH | 8 |
| RF | RC | 2 |
| RF | RE | 2 |
| RF | RH | 4 |
| RG | RD | 5 |
| RG | RE | 1 |
| RH | RE | 8 |
| RH | RF | 6 |

## SPF Algorithm

It is unfortunate that Dijkstra's algorithm is so commonly referred to in the routing world as the shortest path first algorithm. After all, the objective of every routing protocol is to calculate shortest paths. It is also unfortunate that Dijkstra's algorithm is often made to appear more esoteric than it really is; many writers just can't resist putting it in set theory notation. The clearest description of the algorithm comes from E. W. Dijkstra's original paper. Here it is in his own words, followed by a "translation" for the link state routing protocol:

> Construct [a] tree of minimum total length between the *n* nodes. (The tree is a graph with one and only one path between every two nodes.)
>
> In the course of the construction that we present here, the branches are divided into three sets:
>
> I. the branches definitely assigned to the tree under construction (they will be in a subtree);
> II. the branches from which the next branch to be added to set I, will be selected;
> III. the remaining branches (rejected or not considered).
>
> The nodes are divided into two sets:
>
> A. the nodes connected by the branches of set I,
> B. the remaining nodes (one and only one branch of set II will lead to each of these nodes).

We start the construction by choosing an arbitrary node as the only member of set A, and by placing all branches that end in this node in set II. To start with, set I is empty. From then onwards we perform the following two steps repeatedly.

*Step 1:* The shortest branch of set II is removed from this set and added to set I. As a result, one node is transferred from set B to set A.

*Step 2:* Consider the branches leading from the node, which has just been transferred to set A, to the nodes that are still in set B. If the branch under construction is longer than the corresponding branch in set II, it is rejected; if it is shorter, it replaces the corresponding branch in set II, and the latter is rejected.

We then return to step 1 and repeat the process until sets II and B are empty. The branches in set I form the tree required.[16]

Adapting the algorithm for routers, first note that Dijkstra describes three sets of branches: I, II, and III. In the router, three databases represent the three sets:

- *The Tree Database*—This database represents set I. Links (branches) are added to the shortest path tree by adding them here. When the algorithm is finished, this database will describe the shortest path tree.
- *The Candidate Database*—This database corresponds to set II. Links are copied from the link state database to this list in a prescribed order, where they become candidates to be added to the tree.
- *The Link State Database*—The repository of all links, as has been previously described. This topological database corresponds to set III.

Dijkstra also specifies two sets of nodes, set A and set B. Here the nodes are routers. Specifically, they are the routers represented by Neighbor ID in the Router Links triples (Router ID, Neighbor ID, Cost). Set A comprises the routers connected by the links in the Tree database. Set B is all other routers. Since the whole point is to find a shortest path to every router, set B should be empty when the algorithm is finished.

Here's a version of Dijkstra's algorithm adapted for routers:

1. A router initializes the Tree database by adding itself as the root. This entry shows the router as its own neighbor, with a cost of 0.
2. All triples in the link state database describing links to the root router's neighbors are added to the Candidate database.
3. The cost from the root to each link in the Candidate database is calculated. The link in the Candidate database with the lowest cost is moved to the Tree database. If two or more links are an equally low cost from the root, choose one.
4. The Neighbor ID of the link just added to the Tree database is examined. With the exception of any triple whose Neighbor ID is already in the Tree database, triples in the link state database describing that router's neighbors are added to the Candidate database.

[16] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs." Numerische Mathematik, Vol. 1, 1959, pp. 269–271.

5 If entries remain in the Candidate database, return to step 3. If the Candidate database is empty, terminate the algorithm. At termination, a single Neighbor ID entry in the Tree database should represent every router, and the shortest path tree is complete.

Table 4-3 summarizes the process and results of applying Dijkstra's algorithm to build a shortest path tree for the network in Figure 4-11. Router RA from Figure 4-11 is running the algorithm, using the link state database of Table 4-2. Figure 4-12 shows the shortest path tree constructed for Router RA by this algorithm. After each router calculates its own tree, it can examine the other routers' network link information and add the stub networks to the tree fairly easily. From this information, entries might be made into the routing table.

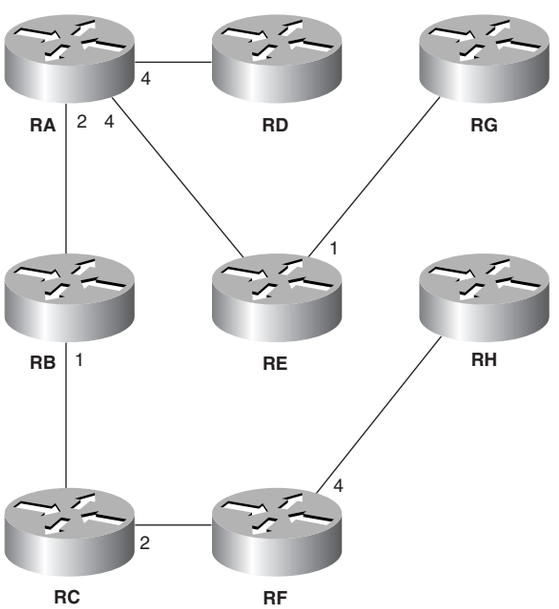
**Table 4-3** *Dijkstra's Algorithm Applied to the Database of Table 4-2*

| Candidate | Cost to Root | Tree | Description |
|---|---|---|---|
| | | RA,RA,0 | Router A adds itself to the tree as root. |
| RA,RB,2<br>RA,RD,4<br>RA,RE,4 | 2<br>4<br>4 | RA,RA,0 | The links to all of RA's neighbors are added to the candidate list. |
| RA,RD,4<br>RA,RE,4<br>RB,RC,1<br>~~RB,RE,10~~ | 4<br>4<br>3<br>12 | RA,RA,0<br>RA,RB,2 | (RA,RB,2) is the lowest-cost link on the candidate list, so it is added to the tree. All of RB's neighbors except those already in the tree are added to the candidate list. (RA,RE,4) is a lower-cost link to RE than (RB,RE,10), so the latter is dropped from the candidate list. |
| RA,RD,4<br>RA,RE,4<br>RC,RF,2 | 4<br>4<br>5 | RA,RA,0<br>RA,RB,2<br>RB,RC,1 | (RB,RC,1) is the lowest-cost link on the candidate list, so it is added to the tree. All of RC's neighbors except those already on the tree become candidates. |
| RA,RE,4<br>RC,RF,2<br>RD,RE,3<br>RD,RG,5 | 4<br>5<br>7<br>9 | RA,RA,0<br>RA,RB,2<br>RB,RC,1<br>RA,RD,4 | (RA,RD,4) and (RA,RE,4) are both a cost of 4 from RA; (RC,RF,2) is a cost of 5. (RA,RD,4) is added to the tree and its neighbors become candidates. Two paths to RE are on the candidate list; (RD,RE,3)is a higher cost from RA and is dropped. |
| RC,RF,2<br>~~RD,RG,5~~<br>RE,RF,2<br>RE,RG,1<br>RE,RH,8 | 5<br>9<br>6<br>5<br>12 | RA,RA,0<br>RA,RB,2<br>RB,RC,1<br>RA,RD,4<br>RA,RE,4 | (RF,RE,1) is added to the tree. All of RE's neighbors not already on the tree are added to the candidate list. The higher-cost link to RG is dropped. |
| RE,RF,2<br>RE,RG,1<br>~~RE,RH,8~~<br>RF,RH,4 | 6<br>5<br>12<br>9 | RA,RA,0<br>RA,RB,2<br>RB,RC,1<br>RA,RD,4<br>RA,RE,4<br>RC,RF,2 | (RC,RF,2) is added to the tree, and its neighbors are added to the candidate list. (RE,RG,1) could have been selected instead because it has the same cost (5) from RA. The higher-cost path to RH is dropped. |

**Table 4-3** *Dijkstra's Algorithm Applied to the Database of Table 4-2 (Continued)*

| Candidate | Cost to Root | Tree | Description |
|---|---|---|---|
| RF,RH,4 | 9 | RA,RA,0<br>RA,RB,2<br>RB,RC,1<br>RA,RD,4<br>RA,RE,4<br>RC,RF,2<br>RE,RG,1 | (RE,RG,1) is added to the tree. RG has no neighbors that are not already on the tree, so nothing is added to the candidate list. |
| | | RA,RA,0<br>RA,RB,2<br>RB,RC,1<br>RA,RD,4<br>RA,RE,4<br>RC,RF,2<br>RE,RG,1<br>RF,RH,4 | (RF,RH,4) is the lowest-cost link on the candidate list, so it is added to the tree. No candidates remain on the list, so the algorithm is terminated. The shortest path tree is complete. |

**Figure 4-12** *Shortest path tree derived by the algorithm in Table 4-3.*



## Areas

An *area* is a subset of the routers that make up a network. Dividing a network into areas is a response to three concerns commonly expressed about link state protocols:

- The necessary databases require more memory than a distance vector protocol requires.
- The complex algorithm requires more CPU time than a distance vector protocol requires.
- The flooding of link state packets adversely affects available bandwidth, particularly in unstable networks.

Modern link state protocols and the routers that run them are designed to reduce these effects, but cannot eliminate them. The last section examined what the link state database might look like, and how an SPF algorithm might work, for a small eight-router network. Remember that the stub networks that would be connected to those eight routers and that would form the leaves of the SPF tree were not taken into consideration. Now, imagine an 8000-router network, and you can understand the concern about the impact on memory, CPU, and bandwidth.

This impact can be greatly reduced by the use of areas, as in Figure 4-13. When a network is subdivided into areas, the routers within an area need to flood LSAs only within that area and therefore need to maintain a link state database only for that area. The smaller database means less required memory in each router and fewer CPU cycles to run the SPF algorithm on that database. If frequent topology changes occur, the resulting flooding will be confined to the area of the instability.

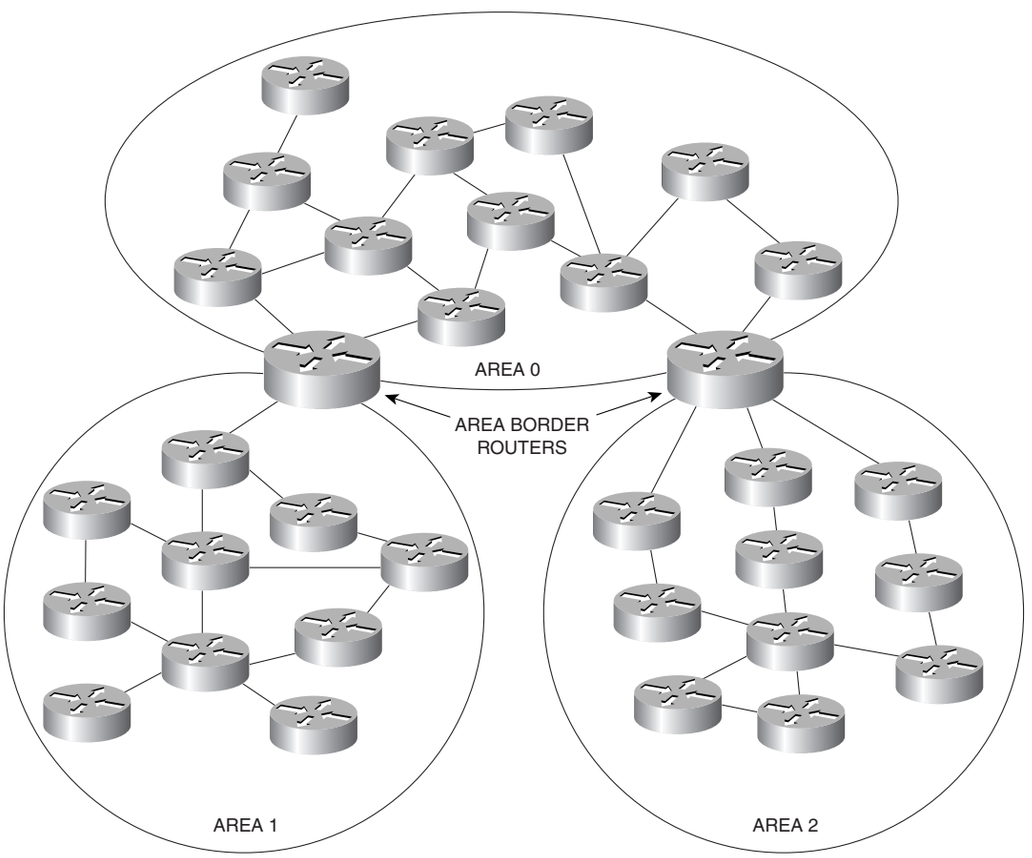
The routers connecting two areas (*Area Border Routers*, in OSPF terminology) belong to both areas and must maintain separate topological databases for each. Just as a host on one network that wants to send a packet to another network only needs to know how to find its local router, a router in one area that wants to send a packet to another area only needs to know how to find its local Area Border Router. In other words, the intra-area router/inter-area router relationship is the same as the host/router relationship but at a higher hierarchical level.

Distance vector protocols, such as RIP and IGRP, do not use areas. Given that these protocols have no recourse but to see a large network as a single entity, must calculate a route to every network, and must broadcast the resulting huge route table every 30 or 90 seconds, it becomes clear that link state protocols utilizing areas can conserve system resources.

# Interior and Exterior Gateway Protocols

Areas introduce a hierarchy to the network architecture. Another layer is added to this hierarchical structure by grouping areas into larger areas. These higher-level areas are called *autonomous systems* in the IP world and *routing domains* in the ISO world.

**Figure 4-13** *Use of areas reduces link state's demand for system resources.*




An autonomous system was once defined as a group of routers under a common administrative domain running a common routing protocol. Given the fluidity of modern networking life, the latter part of the definition is no longer very accurate. Departments, divisions, and even entire companies frequently merge, and networks that were designed with different routing protocols merge along with them. The result is that many networks nowadays combine multiple routing protocols with multiple degrees of inelegance, all under common administrations. So a contemporary definition of an autonomous system is a network under a common administration.

The routing protocols that run within an autonomous system are referred to as *Interior Gateway Protocols* (IGP). All the protocols given in this chapter as examples of distance vector or link state protocols are IGPs.

Routing protocols that route between autonomous systems or routing domains are referred to as *Exterior Gateway Protocols* (EGP). Whereas IGPs discover paths between networks, EGPs discover paths between autonomous systems. Examples of EGPs include the following:

- Border Gateway Protocol (BGP) for IP
- Exterior Gateway Protocol (EGP) for IP (yes, an EGP named EGP)
- The ISO's InterDomain Routing Protocol (IDRP)

Novell also incorporates an EGP functionality, called Level 3 Routing, into NLSP.

Having given these definitions, it must be said that the common usage of the term *autonomous system* is not so absolute. Various standards documents, literature, and people tend to give various meanings to the term. As a result, it is important to understand the context in which one is reading or hearing the term.

This book uses *autonomous system* in one of two contexts:

- *Autonomous system* might refer to a routing domain, as defined at the beginning of this section. In this context, an autonomous system is a system of one or more IGPs that is autonomous from other systems of IGPs. An EGP is used to route between these autonomous systems.
- *Autonomous system* might also refer to a *process domain*, or a single IGP process that is autonomous from other IGP processes. For example, a system of OSPF-speaking routers might be referred to as an OSPF autonomous system. EIGRP also uses autonomous system in this context. Redistribution is used to route between these autonomous systems.

The context will indicate which form of autonomous system is under discussion at different points throughout this book.

## Static or Dynamic Routing?

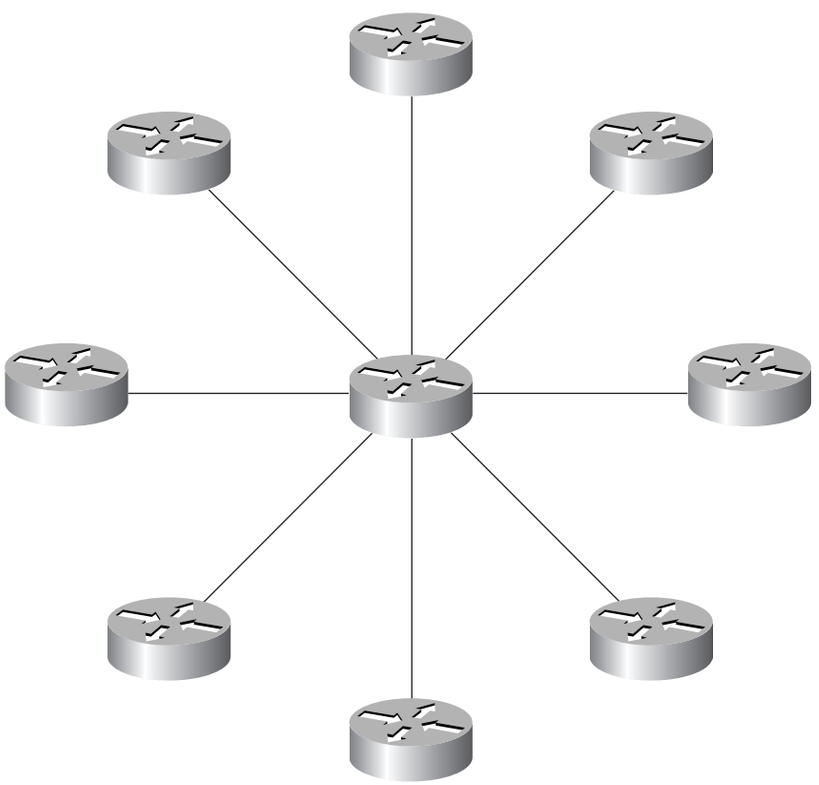
When reading (or being lectured about) all the glorious details of dynamic routing protocols, it's hard not to come away with the impression that dynamic routing is always better than static routing. It's important to keep in mind that the primary duty of a dynamic routing protocol is to automatically detect and adapt to topological changes in the network. The price of this "automation" is paid in bandwidth and maybe queue space, in memory, and in processing time.

Another price of dynamic routing is a reduced control of routing choices. The routing protocol decides what the best path is to a given destination, you don't. If precise control of path selection is important, particularly when the path you want is different from the path a routing protocol would choose, static routing is a better choice.

A frequent objection to static routing is that it is hard to administer. This criticism might be true of medium to large topologies with many alternative routes, but it is certainly not true of small networks with few or no alternative routes.

The network in Figure 4-14 has a hub-and-spoke topology popular in smaller networks. If a spoke to any router breaks, is there another route for a dynamic routing protocol to choose? This network is an ideal candidate for static routing. Configure one static route in the hub router for each spoke router and a single default route in each spoke router pointing to the hub, and the network is ready to go. (Default routes are covered in Chapter 12, "Default Routes and On-Demand Routing.")

**Figure 4-14** *This hub-and-spoke network is ideal for static routing.*



When designing a network, the simplest solution is almost always the best solution. It is good practice to choose a dynamic routing protocol only after determining that static routing is not a practical solution for the design at hand.

## Looking Ahead

Now that the basics of dynamic routing protocols have been examined, it is time to examine specific routing protocols. The following chapter looks at RIP, the oldest and simplest of the dynamic routing protocols.

## Recommended Reading

Perlman, R. *Interconnections: Bridges and Routers.* Reading, Massachusetts: Addison-Wesley; 1992.

## Review Questions

1. What is a routing protocol?
2. What basic procedures should a routing algorithm perform?
3. Why do routing protocols use metrics?
4. What is convergence time?
5. What is load balancing? Name four different types of load balancing.
6. What is a distance vector routing protocol?
7. Name several problems associated with distance vector protocols.
8. What are neighbors?
9. What is the purpose of route invalidation timers?
10. Explain the difference between simple split horizon and split horizon with poisoned reverse.
11. What is the counting-to-infinity problem, and how can it be controlled?
12. What are holddown timers, and how do they work?
13. What are the differences between distance vector and link state routing protocols?
14. What is the purpose of a topological database?
15. Explain the basic steps involved in converging a link state network.
16. Why are sequence numbers important in link state protocols?

**17** What purpose does aging serve in a link state protocol?

**18** Explain how an SPF algorithm works.

**19** How do areas benefit a link state network?

**20** What is an autonomous system?

**21** What is the difference between an IGP and an EGP?