12

Routing Architecture: Cores, Peers, And Algorithms

12.1 Introduction

Previous chapters concentrate on the communication services TCP/IP offers to applications and the details of the protocols in hosts and routers that provide the services. In the discussion, we assumed that routers always contain correct routes, and saw that a router can use the ICMP redirect mechanism to instruct a directly-connected host to change a route.

This chapter considers two broad questions: what values should each forwarding table contain, and how can those values be obtained? To answer the first question, we will consider the relationship between internet architecture and routing. In particular, we will discuss internets structured around a backbone and those composed of multiple peer networks, and consider the consequences for routing. The former is typical of a corporate intranet; the latter applies to the global Internet. To answer the second question, we will consider the two basic types of route propagation algorithms and see how each supplies routing information automatically.

We begin by discussing forwarding in general. Later sections concentrate on internet architecture and describe the algorithms routers use to exchange routing information. Chapters 13 and 14 continue to expand our discussion of routing. They explore protocols that routers owned by two independent administrative groups use to exchange information, and protocols that a single group uses among all its routers.

12.2 The Origin Of Forwarding Tables

Recall from Chapter 3 that IP routers provide active interconnections among networks. Each router attaches to two or more physical networks and forwards IP datagrams among them, accepting datagrams that arrive over one network interface and sending them out over another interface. Except for destinations on directly attached networks, hosts pass all IP traffic to routers which forward datagrams on toward their final destinations. In the general case, a datagram travels from router to router until it reaches a router that attaches directly to the same network as the final destination. Thus, the router system forms the architectural basis of an internet and handles all traffic except for direct delivery from one host to another.

Chapter 8 describes the algorithm that hosts and routers follow when they forward datagrams, and shows how the algorithm uses a table to make decisions. Each entry in the forwarding table uses an address and a mask to specify the network prefix for a particular destination and gives the address of the next router along a path used to reach that network. In practice, each entry also specifies a local network interface that should be used to reach the next hop.

We have not said how hosts or routers obtain the information for their forwarding tables. The issue has two aspects: *what* values should be placed in the tables, and *how* routers obtain the values. Both choices depend on the architectural complexity and size of the internet as well as administrative policies.

In general, establishing routes involves two steps: initialization and update. A host or router must establish an initial table of routes when it starts, and it must update the table as routes change (e.g., when hardware fails, making a particular network unusable). This chapter will focus on routers; Chapter 22 describes how hosts use DHCP to obtain initial entries for a forwarding table.

Initialization depends on the hardware and operating system. In some systems, the router reads an initial forwarding table from secondary storage at startup, either a disk or flash memory. In others, the router begins with an empty table that is filled in by executing a startup script when the router boots. Among other things, the startup script contains commands that initialize the network hardware and configure an IP address for each network interface. Finally, some systems start by broadcasting (or multicasting) a message that discovers neighbors and requests the neighbors to supply information about network addresses being used.

Once an initial forwarding table has been built, a router must accommodate changes in routes. In small, slowly changing internets, managers can establish and modify routes by hand. In large, rapidly changing environments, however, manual update is impossibly slow and prone to human errors. Automated methods are needed. Before we can understand the automatic protocols used to exchange routing information, we need to review several underlying ideas. The next sections do so, providing the necessary conceptual foundation for routing.

12.3 Forwarding With Partial Information

The principal difference between routers and typical hosts is that hosts usually know little about the structure of the internet to which they connect. Hosts do not have complete knowledge of all possible destination addresses, or even of all possible destination networks. In fact, many hosts have only two entries in their forwarding table: an entry for the local network, and a default entry for a directly-connected router. The host sends all nonlocal datagrams to the local router for delivery. The point is:

> *A host can forward datagrams successfully even if it only has partial forwarding information because it can rely on a router.*

Can routers also forward datagrams with only partial information? Yes, but only under certain circumstances. To understand the criteria, imagine an internet to be a foreign country crisscrossed with dirt roads that have directional signs posted at intersections. Imagine that you have no map, cannot ask directions because you cannot speak the local language, and have no knowledge about visible landmarks, but you need to travel to a village named *Sussex*. You leave on your journey, following the only road out of town, and begin to look for directional signs. The first sign reads:

Norfolk to the left; Hammond to the right; others straight ahead.†

Because the destination you seek is not listed explicitly, you continue straight ahead. In routing jargon, we say you follow a *default route*. After several more signs, you finally find one that reads:

Essex to the left; Sussex to the right; others straight ahead.

You turn to the right, follow several more signs, and emerge on a road that leads to Sussex.

Our imagined travel is analogous to a datagram traversing an internet, and the road signs are analogous to forwarding tables in routers along the path. Without a map or other navigational aids, travel is completely dependent on road signs, just as datagram forwarding in an internet depends entirely on forwarding tables. Clearly, it is possible to navigate even though each road sign contains only partial information.

A central question concerns correctness. As a traveler, you might ask: "How can I be sure that following the signs will lead to my destination?" You also might ask: "How can I be sure that following the signs will lead me to my destination along a shortest path?" These questions may seem especially troublesome if you pass many signs without finding your destination listed explicitly. Of course, the answers depend on the topology of the road system and the contents of the signs, but the fundamental idea is that when taken as a whole, the information on the signs should be both consistent and complete. Looking at this another way, we see that it is not necessary for each intersection to have a sign for every destination. The signs can list default paths as long as all

†Fortunately, signs are printed in a language you can read.

explicit signs point along a shortest path, and the turns for shortest paths to all destinations are marked. A few examples will explain some ways that consistency can be achieved.

At one extreme, consider a simple star-shaped topology of roads in which each town has exactly one road leading to it, and all the roads meet at a central point. Figure 12.1 illustrates the topology.

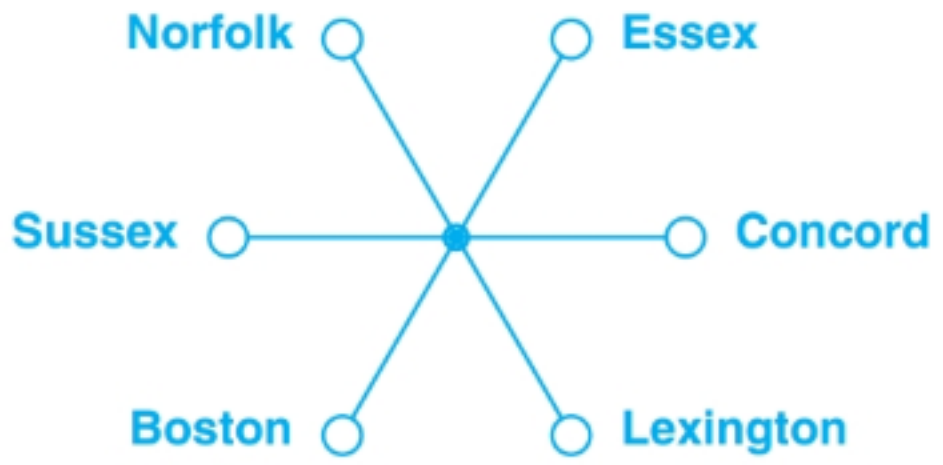

Figure 12.1 An example star-shaped topology of roads connecting towns.

We imagine a sign at the central intersection that lists each possible town and the road to reach that town. In other words, only the central intersection has information about each possible destination; a traveler always proceeds to the central intersection on the way to any destination.

At another extreme, we can imagine an arbitrary set of connected roads and a sign at each intersection listing all possible destinations. To guarantee that the signs lead travelers along the best route, it must be true that at any intersection if the sign for destination D points to road R, no road other than R leads to a shorter path to D.

Neither of the architectural extremes works well for a larger internet routing system. On one hand, the central intersection approach fails because no equipment is fast enough to serve as a central switch through which all traffic passes. On the other hand, having information about all possible destinations in all routers is impractical because it requires propagating large volumes of information whenever a change occurs in the internet. Therefore, we seek a solution that allows groups to manage local routers autonomously, adding new network interconnections and routes without changing the forwarding information in distant routers.

To understand the routing architecture used in the Internet, consider a third topology in which half of the cities lie in the eastern part of the country and half lie in the western part. Suppose a single bridge spans the river that separates east from west. Assume that people living in the eastern part do not like westerners, so they are unwilling to allow any road sign in the east to list destinations in the west. Assume that people living in the west do the opposite. Routing will be consistent if every road sign in the east lists all eastern destinations explicitly and points the default path to the bridge, and every road sign in the west lists all western destinations explicitly and points the default path to the bridge. However, there is a catch: if a tourist arrives who has accidentally

written down the name of a non-existent town, the tourist could cross the bridge one way and then find that the default path points back to the bridge.

12.4 Original Internet Architecture And Cores

Much of our knowledge of forwarding and route propagation protocols has been derived from experience with the Internet. When TCP/IP was first developed, participating research sites were connected to the ARPANET, which served as a backbone network connecting all sites on the Internet. During initial experiments, each site managed forwarding tables and installed routes to other destinations by hand. As the fledgling Internet began to grow, it became apparent that manual maintenance of routes was impractical; automated mechanisms were needed. The concept of a backbone network continues to be used: many large enterprises have a backbone that connects sites on the enterprise's intranet.

The Internet designers selected a router architecture that followed the star-shaped topology described above. The original design used a small, central set of routers that kept complete information about all possible destinations, and a larger set of outlying routers that kept partial information. In terms of our analogy, it is like designating a small set of centrally located intersections that have signs listing all destinations, and allowing the outlying intersections to list only local destinations. As long as the default route at each outlying intersection points to one of the central intersections, travelers will eventually reach their destination.

The central set of routers that maintained complete information was known as the *core* of the Internet. Because each core router stores a route for each possible destination, a core router does not need a default route. Therefore, the set of core routers is sometimes referred to as the *default-free zone*. The advantage of partitioning Internet routing into a two-tier system is that it permits local administrators to manage local changes in outlying routers without affecting other parts of the Internet. The disadvantage is that it introduces the potential for inconsistency. In the worst case, an error in an outlying router can make distant routes unreachable.

We can summarize the ideas:

> *The advantage of a core routing architecture lies in autonomy: the manager of a noncore router can make changes locally. The chief disadvantage is inconsistency: an outlying site can introduce errors that make some destinations unreachable.*

Inconsistencies among forwarding tables can arise from errors in the algorithms that compute forwarding tables, incorrect data supplied to those algorithms, or errors that occur while transmitting the results to other routers. Protocol designers look for ways to limit the impact of errors, with the objective being to keep all routes consistent at all times. If routes become inconsistent, the routing protocols should be robust

enough to detect and correct the errors quickly. Most important, the protocols should be designed to constrain the effect of errors.

The early Internet architecture is easy to understand if one remembers that the Internet evolved with a wide-area backbone, the ARPANET, already in place. A major motivation for the core router system came from the desire to connect local networks to the backbone. Figure 12.2 illustrates the idea.

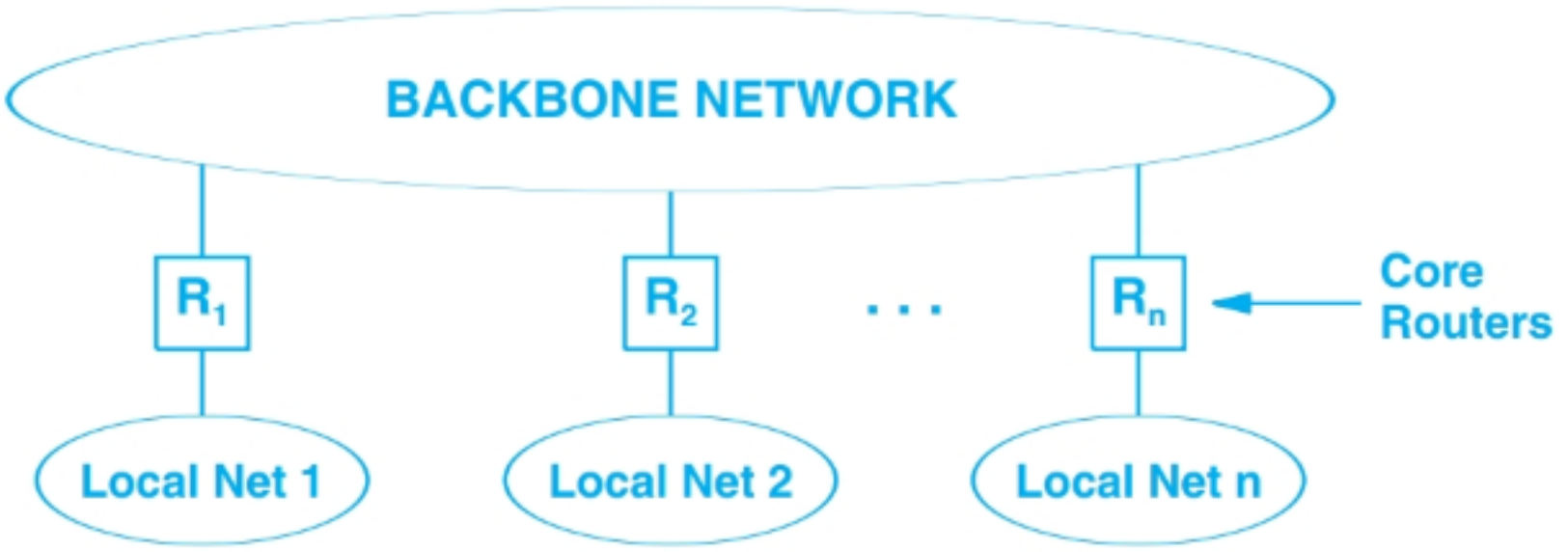

Figure 12.2 The early Internet core router system viewed as a set of routers that connect local area networks to the backbone. The architecture is now used in enterprise networks.

To understand why routers in Figure 12.2 cannot use partial information, consider the path a datagram follows if a set of routers use a default route. At the source site, the local router checks to see if it has an explicit route to the destination, and if not, sends the datagram along the path specified by its default route. All datagrams for which the router has no explicit route follow the same default path regardless of their ultimate destination. The next router along the path diverts datagrams for which it has an explicit route, and sends the rest along its default route. To ensure global consistency, the chain of default routes must reach every router in a giant cycle. Thus, the architecture requires all local sites to coordinate their default routes.

There are two problems with a routing architecture that involves a set of default routes. First, suppose a computer accidentally generates a datagram to a nonexistent destination (i.e., to an IP address that has not been assigned). The host sends the datagram to the local router, the local router follows the default path to the next router, and so on. Unfortunately, because default routes form a cycle, the datagram will go around the cycle until the hop limit expires. Second, if we ignore the problem of nonexistent addresses, forwarding is inefficient. A datagram that follows the default routes may pass through $n - 1$ routers before it reaches a router that connects to the local network of the destination.

To avoid the inefficiencies and potential routing loops that default routes can cause, the early Internet prohibited default routes in core routers. Instead of using default routes, the designers arranged for routers to communicate with one another and exchange routing information so that each router learned how to forward datagrams

directly. Arranging for core routers to exchange routing information is easy — the routers all attach to the backbone network, which means they can communicate directly.

12.5 Beyond The Core Architecture To Peer Backbones

The introduction of the NSFNET backbone into the Internet added new complexity to the routing structure and forced designers to invent a new routing architecture. More important, the change in architecture foreshadowed the current Internet in which a set of *Tier-1 ISPs* each have a wide-area backbone to which customer sites connect. In many ways, the work on NSFNET and the routing architecture that was created to support it was key in moving away from the original Internet architecture to the current Internet architecture.

The primary change that occurred with the NSFNET backbone was the evolution from a single, central backbone to a set of *peer backbone networks*, often called *peers* or *routing peers*. Although the global Internet now has several Tier-1 peers, we can understand the routing situation by considering only two. Figure 12.3 illustrates an Internet topology with a pair of backbone networks.

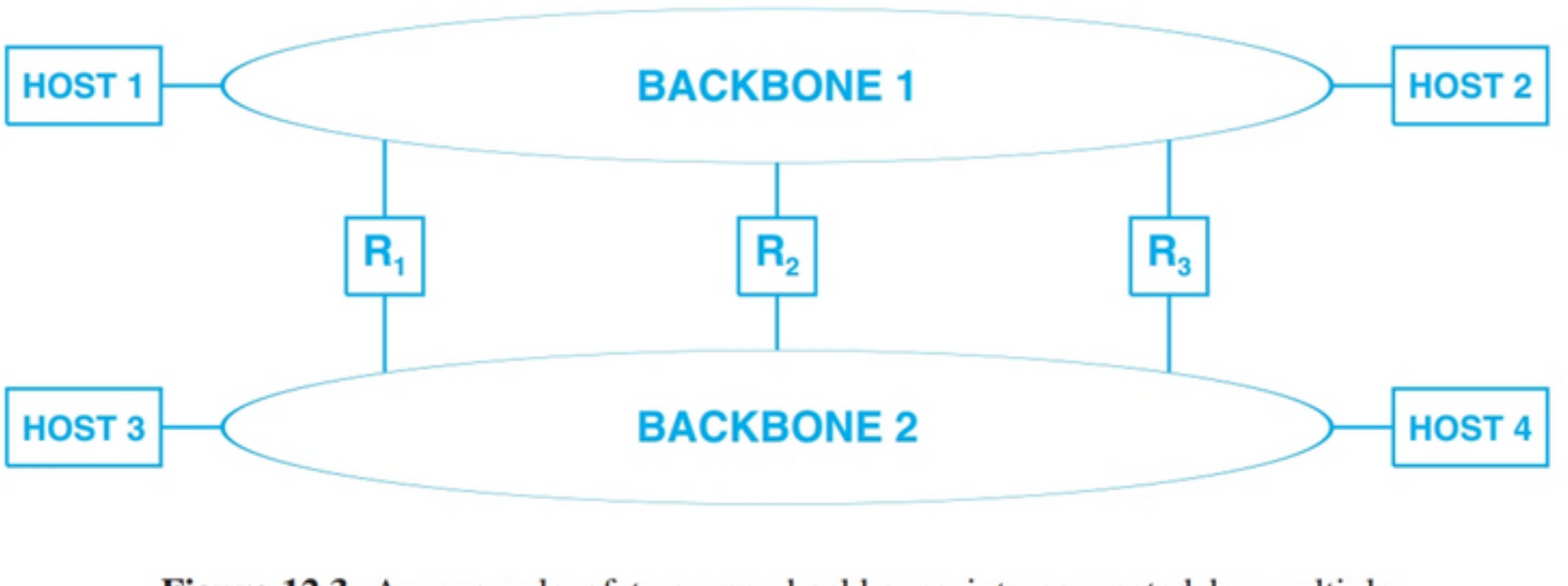

Figure 12.3 An example of two peer backbones interconnected by multiple routers similar to the two peer backbones in the Internet in 1989.

To help us understand the difficulties of IP routing among peer backbones, the figure shows four hosts directly connected to the backbones. Although such direct connection may seem unrealistic, it simplifies the example. Look at the figure and consider routes from host *3* to host *2*. Assume for the moment that the figure shows geographic orientation: host *3* is on the West Coast attached to backbone 2, while host *2* is on the East Coast attached to backbone 1. When establishing routes between hosts *3* and *2*, the managers must decide among three options:

(a) Route the traffic from host *3* through the West Coast router, R_1, and then across backbone 1.

(b) Forward the traffic from host *3* across backbone 2, through the Midwest router, R_2, and then across backbone 1 to host *2*.

(c) Forward the traffic across backbone 2, through the East Coast router, R_3, and then to host *2*.

A more circuitous route is possible as well: traffic could flow from host *3* through the West Coast router, across backbone 1 to the Midwest router, back onto backbone 2 to the East Coast router, and finally across backbone 1 to host *2*. Such a route may or may not be advisable, depending on the policies for network use and the capacity of various routers and backbones. Nevertheless, we will concentrate on routing in which a datagram never traverses a network twice (i.e., never moves to a network, moves off the network, and then moves back to the network again).

Intuitively, we would like all traffic to take a shortest path. That is, we would like traffic between a pair of geographically close hosts to take a short path independent of the routes chosen for long-distance traffic. For example, it is desirable for traffic from host *3* to host *1* to flow through the West Coast router, R_1, because such a path minimizes the total distance that the datagram travels. More important, if a datagram must travel across a backbone, an ISP would like to keep the datagram on its backbone (because doing so is economically less expensive than using a peer).

The goals above seem straightforward and sensible. However, they cannot be translated into a reasonable routing scheme for two reasons. First, although the standard IP forwarding algorithm uses the network portion of an IP address to choose a route, optimal forwarding in a peer backbone architecture requires individual routes for individual hosts. For example, consider the forwarding table in host *3*. The optimal next hop for host *1* is the west-coast router, R_1, and the optimal next hop for host *2* is the east-coast router, R_3. However, hosts *1* and *2* both connect to backbone 1, which means they have the same network prefix. Therefore, instead of using network prefixes, the host forwarding table must contain host-specific routes. Second, managers of the two backbones must agree to keep routes consistent among all routers or a *forwarding loop* (*routing loop*) can develop in which a set of the routers forward to each other in a cycle.

12.6 Automatic Route Propagation And A FIB

We said that the original Internet core system avoided default routes because it propagated complete information about all possible destinations to every core router. Many corporate intranets now use a similar approach — they propagate information about each destination in the corporation to all routers in their intranet. The next sections discuss two basic types of algorithms that compute and propagate routing information; later chapters discuss protocols that use the algorithms.

Routing protocols serve two important functions. First, they compute a set of shortest paths. Second, they respond to network failures or topology changes by continually updating the routing information. Thus, when we think about route propagation, it is important to consider the dynamic behavior of protocols and algorithms.

Conceptually, routing protocols operate independently from the forwarding mechanism. That is, routing protocol software runs as a separate process that uses IP to exchange messages with routing protocol software on other routers. Routing protocols learn about destinations, compute a shortest path to each destination, and pass information to the routing protocol software on other routers.

Although a routing protocol computes shortest paths, the routing protocol software does not store information directly in the router's forwarding table. Instead, routing software creates a *Forwarding Information Base* (*FIB*). A FIB may contain extra information not found in a forwarding table, such as the source of the routing information, how old the information is (i.e., the last time a routing protocol on another router sent a message about the route), and whether a manager has temporarily overridden a specific route.

When the FIB changes, routing software recomputes a forwarding table for the router and installs the new forwarding table. A crucial step occurs between items being placed in a FIB and the items being propagated to the forwarding table: policy rules are applied. Policies allow a manager to control which items are automatically installed in the forwarding table. Therefore, even if routing software finds a shorter path to a particular destination and places the information in the FIB, policies may prevent the path from being injected into the forwarding table.

12.7 Distance-Vector (Bellman-Ford) Routing

The term *distance-vector*‡ refers to a class of algorithms used to propagate routing information. The idea behind distance-vector algorithms is quite simple. Each router keeps a list of all known destinations in its FIB. When it boots, a router initializes its FIB to contain an entry for each directly connected network. Each entry in the FIB identifies a destination network, a next-hop router used to reach the destination, and the "distance" to the network (according to some measure of distance). For example, some distance-vector protocols use the number of network hops as a measure of distance. A directly-connected network is zero hops away; if a datagram must travel through N routers to reach a destination, the destination is N hops away. Figure 12.4 illustrates the initial contents of a FIB on a router that attaches to two networks. In the figure, each entry corresponds to a directly-connected network (zero hops away).

‡The terms *vector-distance*, *Ford-Fulkerson*, *Bellman-Ford*, and *Bellman* are synonymous with *distance-vector*; the last three are taken from the names of researchers who published the idea.

Destination	Distance	Route
Net 1	0	direct
Net 2	0	direct

Figure 12.4 An initial FIB used with a distance-vector algorithm. Each entry contains the IP address of a directly connected network and an integer distance to the network.

When using distance-vector, routing software on each router sends a copy of its FIB to any other router it can reach directly. When a report arrives at router *K* from router *J*, *K* examines the set of destinations reported and the distance to each and applies three rules:

- If *J* lists a destination that *K* does not have in its FIB, *K* adds a new entry to its FIB with the next hop of *J*.
- If *J* knows a shorter way to reach a destination *D*, *K* replaces the next hop in its FIB entry for *D* with *J*.
- If *K*'s FIB entry for destination *D* already lists *J* as the next hop and *J*'s distance to the destination has changed, *K* replaces the distance in its FIB entry.

The first rule can be interpreted, "If my neighbor knows a way to reach a destination that I don't know, I can use the neighbor as a next hop." The second rule can be interpreted, "If my neighbor has a shorter route to a destination, I can use the neighbor as a next hop." The third rule can be interpreted, "If I am using my neighbor as the next hop for a destination and the neighbor's cost to reach the destination changes, my cost must change."

Figure 12.5 shows an existing FIB in a router, *K*, and a distance-vector update message from another router, *J*. Three items in the message cause changes in the FIB. Note that if *J* reports a distance *N* hops to a given destination and *K* uses *J* as a next hop, the distance stored in *K*'s FIB will have distance *N* + *1* (i.e., the distance from *J* to the destination plus the distance to reach *J*). The third column in our example FIB is labeled *Route*. In practice, the column contains the IP address of a next-hop router. To make it easy to understand, the figure simply lists a symbolic name (e.g., *Router J*).

The term *distance-vector* comes from the information sent in the periodic messages. A message contains a list of pairs (*D, V*), where *D* is a distance to a destination and *V* identifies the destination (called the *vector*). Note that distance-vector algorithms report routes in the first person (i.e., we think of a router advertising, "I can reach destination *V* at distance *D*"). In such a design, all routers must participate in the distance-vector exchange for the routes to be efficient and consistent.

Destination	Distance	Route
Net 1	0	direct
Net 2	0	direct
Net 4	8	Router L
Net 17	5	Router M
Net 24	6	Router J
Net 30	2	Router Q
Net 42	2	Router J

(a)

Destination	Distance
Net 1	2
→ Net 4	3
Net 17	6
→ Net 21	4
Net 24	5
Net 30	10
→ Net 42	3

(b)

Figure 12.5 (a) An existing FIB in router *K*, and (b) an incoming routing update message from router *J* that will cause changes.

Although they are easy to implement, distance-vector algorithms have several disadvantages. In a completely static environment, distance-vector algorithms do indeed compute shortest paths and correctly propagate routes to all destinations. When routes change rapidly, however, the computations may not stabilize. When a route changes (i.e, a new connection appears or an old one fails), the information propagates slowly from one router to another. Meanwhile, some routers may have incorrect routing information.

12.8 Reliability And Routing Protocols

Most routing protocols use connectionless transport — early protocols encapsulated messages directly in IP; modern routing protocols usually encapsulate in UDP†. Unfortunately, UDP offers the same semantics as IP: messages can be lost, delayed, duplicated, corrupted, or delivered out of order. Thus, a routing protocol that uses them must compensate for failures.

Routing protocols use several techniques to handle reliability. First, checksums are used to handle corruption. Loss is either handled by *soft state*‡ or through acknowledgements and retransmission. *Sequence numbers* are used to handle two problems. First, sequence numbers allow a receiver to handle out-of-order delivery by placing incoming messages back in the correct order. Second, sequence numbers can be used to handle *replay*, a condition that can occur if a duplicate of a message is delayed and arrives long after newer updates have been processed. Chapter 14 illustrates how distance-vector protocols can exhibit slow convergence, and discusses additional techniques that distance-vector protocols use to avoid problems. In particular, the chapter covers split horizon and poison reverse techniques.

†The next chapter discusses an exception — a routing protocol that uses TCP.

‡Recall that soft state relies on timeouts to remove old information.

12.9 Link-State (SPF) Routing

The main disadvantage of the distance-vector algorithm is that it does not scale well. Besides the problem of slow response to change mentioned earlier, the algorithm requires the exchange of large messages — because each routing update contains an entry for every possible network, message size is proportional to the total number of networks in an internet. Furthermore, because a distance-vector protocol requires every router to participate, the volume of information exchanged can be enormous.

The primary alternative to distance-vector algorithms is a class of algorithms known as *link state*, *link status*, or *Shortest Path First*† (*SPF*). The SPF algorithm requires each participating router to compute topology information. The easiest way to think of topology information is to imagine that every router has a map that shows all other routers and the networks to which they connect. In abstract terms, the routers correspond to nodes in a graph, and networks that connect routers correspond to edges. There is an edge (link) between two nodes in the topology graph if and only if the corresponding routers can communicate directly.

Instead of sending messages that contain a list of destinations that a router can reach, each router participating in an SPF algorithm performs two tasks:

- Actively test the status of each neighboring router. Two routers are considered neighbors if they attach to a common network.
- Periodically broadcast link-state messages of the form, "The link between me and router X is up" or "The link between me and router X is down."

To test the status of a directly connected neighbor, the two neighbors exchange short messages that verify that the neighbor is alive and reachable. If the neighbor replies, the link between them is said to be *up*. Otherwise, the link is said to be *down*‡. To inform all other routers, each router periodically broadcasts a message that lists the status (state) of each of its links. A status message does not specify routes — it simply reports whether communication is possible between pairs of routers. When using a link-state algorithm, the protocol software must find a way to deliver each link-state message to all routers, even if the underlying network does not support broadcast. Thus, individual copies of each message may be forwarded point-to-point.

Whenever a link-state message arrives, software running on the router uses the information to update its map of the internet. First, it extracts the pair of routers mentioned in the message and makes sure that the local graph contains an edge between the two. Second, it uses the status reported in the message to mark the link as up or down. Whenever an incoming message causes a change in the local topology graph, the link-state algorithm recomputes routes by applying Dijkstra's algorithm. The algorithm computes the shortest path from the local router to each destination. The resulting information is placed in the FIB, and if policies permit, used to change the forwarding table.

---

†Dijkstra, the inventor of the algorithm, coined the name "shortest path first," but it is misleading because all routing algorithms compute shortest paths.

‡In practice, to prevent oscillations between the up and down states, many protocols use a *k-out-of-n rule* to test liveness, meaning that the link remains up until a significant percentage of requests have no reply, and then it remains down until a significant percentage of messages receive a reply.

One of the chief advantages of SPF algorithms is that each router computes routes independently using the same original status data; they do not depend on the computation of intermediate routers. Compare the approach to a distance-vector algorithm in which each router updates its FIB and then sends the updated information to neighbors — if the software in any router along the path is incorrect, all successive routers will receive incorrect information. With a link-state algorithm, routers do not depend on intermediate computation — the link-status messages propagate throughout the network unchanged, making it is easier to debug problems. Because each router performs the shortest path computation locally, the computation is guaranteed to converge. Finally, because each link-status message only carries information about the direct connection between a pair of routers, the size does not depend on the number of networks in the underlying internet. Therefore, SPF algorithms scale better than distance-vector algorithms.

12.10 Summary

To ensure that all networks remain reachable with high reliability, an internet must provide globally consistent forwarding. Hosts and most routers contain only partial routing information; they depend on default routes to send datagrams to distant destinations. Originally, the global Internet solved the routing problem by using a core router architecture in which a set of core routers each contained complete information about all networks.

When additional backbone networks were added to the Internet, a new routing architecture arose to match the extended topology. Currently, a set of separately managed peer backbone networks exist that interconnect at multiple places.

When they exchange routing information, routers usually use one of two basic algorithms; distance-vector or link-state (also called SPF). The chief disadvantage of distance-vector algorithms is that they perform a distributed shortest path computation that may not converge if the status of network connections changes continually. Thus, for large internets or internets where the underlying topology changes quickly, SPF algorithms are superior.

EXERCISES

12.1 Suppose a router discovers it is about to forward an IP datagram back over the same network interface on which the datagram arrived. What should it do? Why?

12.2 After reading RFC 823 and RFC 1812, explain what an Internet core router (i.e., one with complete routing information) should do in the situation described in the previous question.

12.3 How can routers in a core system use default routes to send all illegal datagrams to a specific machine?

12.4 Imagine that a manager accidentally misconfigures a router to advertise that it has direct connections to six specific networks when it does not. How can other routers that receive the advertisement protect themselves from invalid advertisements while still accepting other updates from "untrusted" routers?

12.5 Which ICMP messages does a router generate?

12.6 Assume a router is using unreliable transport for delivery. How can the router determine whether a designated neighbor's status is *up* or *down*? (Hint: consult RFC 823 to find out how the original core system solved the problem.)

12.7 Suppose two routers each advertise the same cost, k, to reach a given network, N. Describe the circumstances under which forwarding through one of them may take fewer total hops than forwarding through the other one.

12.8 How does a router know whether an incoming datagram carries a routing update message?

12.9 Consider the distance-vector update shown in Figure 12.5 carefully. For each item updated in the table, give the reason why the router will perform the update.

12.10 Consider the use of sequence numbers to ensure that two routers do not become confused when datagrams are duplicated, delayed, or delivered out of order. How should initial sequence numbers be selected? Why?