The Network Layer: Control Plane

We'll complete our journey through the network layer by covering the control-plane. The network-wide logic that controls not only know how a datagram is routed along an end-to-end path from the source host to the destination host, but also how network-layer components and services are configured and managed.

5.1 Introduction

We saw that the forwarding table (in the case of destination-based forwarding) and the flow table (in the case of generalized forwarding) were the principal elements that linked the network layer's data and control planes. We saw that in the case of generalized forwarding, the actions taken
could include not only forwarding a packet to a router's output port, but also dropping a packet, replicating a packet, and/or rewriting layer 2, 3 or 4 packet-header fields.

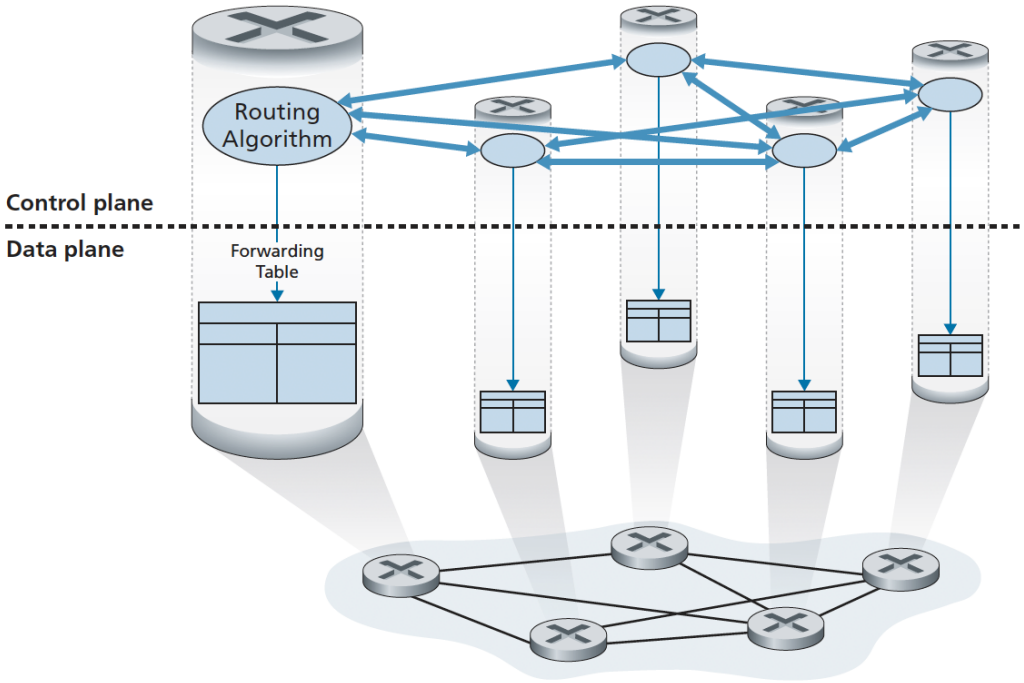

Figure 5.1 ♦ Per-router control: Individual routing algorithm components interact in the control plane

We'll study how those forwarding and flow tables are computed, maintained and installed. In our introduction to the network layer in Section 4.1, we learned that there are two possible approaches for doing so.

- Per-router control. Figure 5.1 illustrates the case where a routing algorithm runs in each and every router; both a forwarding and a routing function are contained within each router. Each router has a routing component that communicates with the routing components in other routers to compute the values for its forwarding table.
- Logically centralized control. Figure 5.2 illustrates the case in which a logically centralized controller computes and distributes the forwarding tables to be used by each and every router.

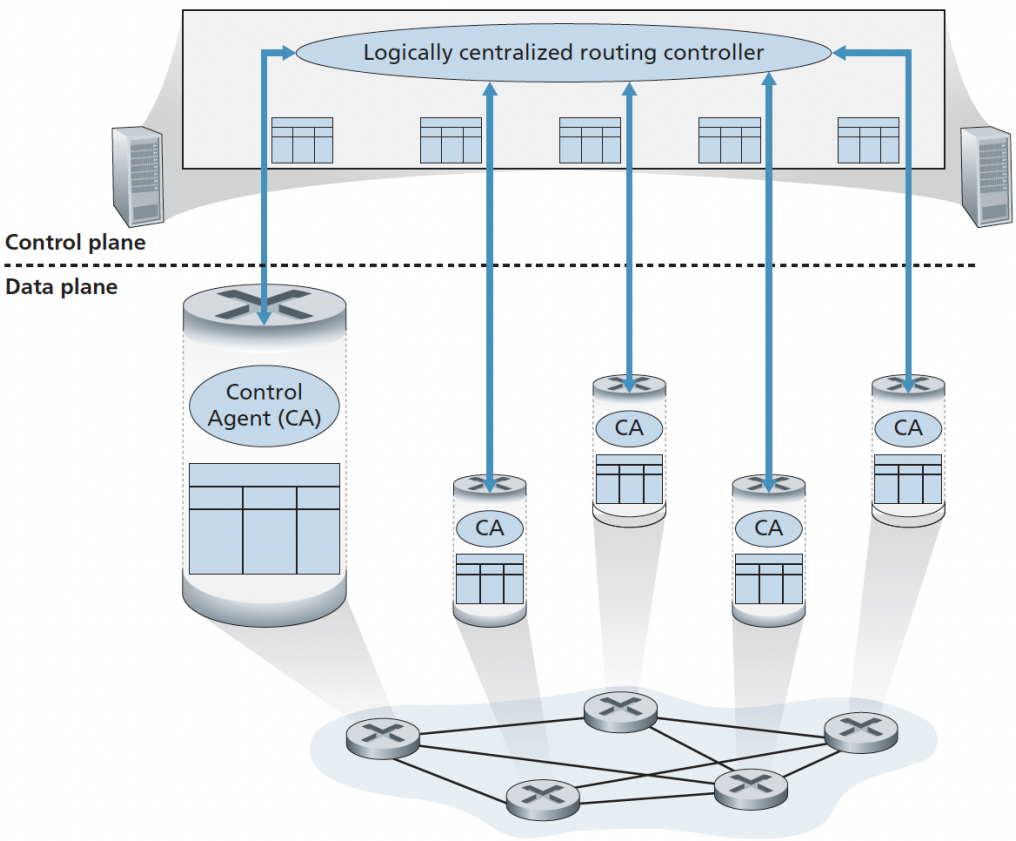

Figure 5.2 ♦ Logically centralized control: A distinct, typically remote, controller interacts with local control agents (CAs)

The controller interacts with a control agent (CA) in each of the routers via a well-defined protocol to configure and manage that router's flow table. Typically, the CA has minimum functionality; its job is to communicate with the controller, and to do as the controller commands. Unlike the routing algorithms in Figure 5.1, the CAs do not directly interact with each other nor do they actively take part in computing the forwarding table. This is a key distinction between per-router control and logically centralized control.

By "logically centralized" control we mean that the routing control service is accessed as if it were a single central service point, even though the service is likely to be implemented via multiple servers for fault-tolerance, and performance scalability reasons. SDN adopts this notion of a logically centralized controller.

5.2 Routing Algorithms

We'll study routing algorithms, whose goal is to determine good paths (equivalently, routes), from senders to receivers, through the network of routers. A "good" path is one that has the least cost. In practice, however, real-world concerns such as policy issues (for example, a rule such as "router x, belonging to organization Y, should not forward any packets originating from the network owned by organization Z") also come into play. We note that whether the network control plane adopts a per-router control approach or a logically centralized approach, there must always be a well-defined sequence of routers that a packet will cross in traveling from sending to receiving host.

A graph is used to formulate routing problems. Recall that a graph G=(N, E) is a set N of nodes and a collection E of edges, where each edge is a pair of nodes from N. The nodes in the graph represent routers and the edges connecting these nodes represent the physical links between these routers.

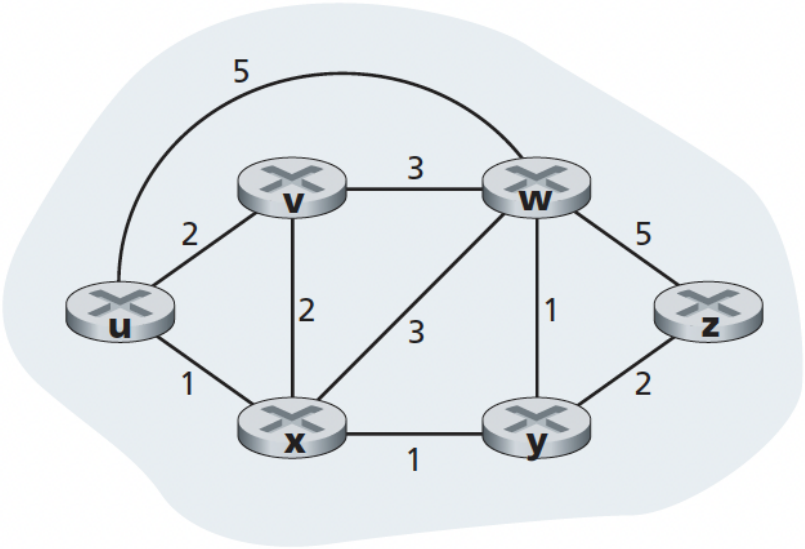

Figure 5.3 ♦ Abstract graph model of a computer network

When we study the BGP inter-domain routing protocol, we'll see that nodes represent networks, and the edge connecting two such nodes represents direction connectivity between two networks.

An edge also has a value representing its cost. Typically, an edge's cost may reflect the physical length of the corresponding link, the link speed, or the monetary cost associated with a link. For any edge (x, y) in E, we denote c(x, y) as the cost of the edge between nodes x and y. If the pair (x, y) does not belong to E, we set c(x, y) = ∞. So that edge (x, y) is the same as edge (y, x) and that c(x, y)=c(y, x). Also, a node y is said to be a neighbor of node x if (x, y) belongs to E.

To make this problem more precise, recall that a path g, in a graph G = (N, E) is a sequence of nodes (x_1, x_2, x_p) such that each of the pairs $(x_1, x_2), (x_2, x_3), (x_{p-1}, x_p)$are edges in E. The cost of a path (x1, x2, xp) is simply the sum of all the edge costs along the path. That is $c(x_1, x_2) + c(x_2, x_3) + ... + c(x_{p-1}, x_p)$. Given any two nodes x and y, there are typically many paths between the two nodes, with each path having a cost. One or more of these paths is a least-cost path. The least-cost problem is therefore clear: Find a path between the source and destination that has least cost. Note that if all edges in the graph have the same cost, the least cost path is also the shortest path. Broadly, one way in which we can classify routing algorithms is according to whether they are centralized or decentralized.

- A **centralized routing algorithm** computes the least-cost path between a source and destination using complete, global knowledge about the network. That is, the algorithm takes the connectivity between all nodes and all link costs as inputs. This then requires that the algorithm somehow obtain this information before actually performing the calculation. The calculation itself can be run at one site or could be replicated in the routing component of each and every router. The key distinguishing feature here, however, is that the algorithm has complete information about connectivity and link costs. Algorithms with global state information are often referred to as link-state (LS) algorithms, since the algorithm must be aware of the cost of each link in the network.
- In a **decentralized routing algorithm**, the calculation of the least-cost path is carried out in an iterative, distributed manner by the routers. No node has complete information about the costs of all network links. Instead, each node begins with only the knowledge of the costs of its own directly attached links. Then, through an iterative process of calculation and exchange of information with its neighboring nodes, a node gradually calculates the least-cost path to a destination or set of destinations. The decentralized routing algorithm we'll study is called a distance-

vector (DV) algorithm, because each node maintains a vector of estimates of the costs (distances) to all other nodes in the network.

A second broad way to classify routing algorithms is according to whether they are static or dynamic. In **static routing algorithms**, routes change very slowly over time, often as a result of human intervention. **Dynamic routing algorithms** change the routing paths as the network
traffic loads or topology change. A dynamic algorithm can be run either periodically or in direct response to topology or link cost changes. While dynamic algorithms are more responsive to network changes, they are also more susceptible to problems such as routing loops and route oscillation.

A third way to classify routing algorithms is according to whether they are load sensitive or load-insensitive. In a load-sensitive algorithm, link costs vary dynamically to reflect the current level of congestion in the underlying link. Today's Internet routing algorithms (such as RIP, OSPF, and BGP) are load-insensitive, as a link's cost does not explicitly reflect its current (or recent past) level of congestion.

5.2.1 The Link-State (LS) Routing Algorithm

Recall that in a link-state algorithm, the network topology and all link costs are known. In practice, this is accomplished by having each node broadcast link-state packets to all other nodes in the network, with each link-state packet containing the identities and costs of its attached links. In practice, this is often accomplished by a link-state broadcast algorithm. The result of the nodes' broadcast is that all nodes have an identical and complete view of the network. Each node can then run the LS algorithm and compute the same set of least-cost paths as every other node.

The link-state routing algorithm we present below is known as Dijkstra's algorithm, named after its inventor. Dijkstra's algorithm computes the least-cost path from one node to all other nodes in the network. Dijkstra's algorithm is iterative and has the property that after the kth iteration of the algorithm, the least-cost paths are known to k destination nodes, and among the least-cost paths to all destination nodes, these k paths will have the k smallest costs. Let us define the following notation:

- D(v): cost of the least-cost path from the source node to destination v as of this iteration of the algorithm.

- p(v): previous node (neighbor of v) along the current least-cost path from the source to v.
- N': subset of nodes; v is in N' if the least-cost path from the source to v is definitively known.

Link-State (LS) Algorithm for Source Node u

```
Initialization:
N' = {u}
for all nodes v
  if v is a neighbor of u
   then D(v) = c(u,v)
  else D(v) = ∞
Loop:
find w not in N' such that D(w) is a minimum
add w to N'
update D(v) for each neighbor v of w and not in N':
  D(v) = min(D(v), D(w)+ c(w,v) )
/* new cost to v is either old cost to v or known
  least path cost to w plus cost from w to v *
until N'= N
```

step	*N'*	*D(v), p(v)*	*D(w), p(w)*	*D(x), p(x)*	*D(y), p(y)*	*D(z), p(z)*
0	u	2, u	5, u	1,u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

Table 5.1 ♦ Running the link-state algorithm on the network in Figure 5.3

Before completing our discussion of the LS algorithm, let us consider a pathology that can arise. Network topology where link costs are equal to the load carried on the link, reflecting the delay that would be experienced. In this example, link costs are not

symmetric; that is, c(u,v) equals
c(v,u) only if the load carried on both directions on the link (u,v) is the same.

What can be done to prevent such oscillations (which can occur in any algorithm, not just an LS algorithm, that uses a congestion or delay-based link metric)? One solution would be to mandate that link costs not depend on the amount of traffic carried—an unacceptable solution since one goal of routing is to avoid highly congested (for example, high-delay) links. Another solution is to ensure that not all routers run the LS algorithm at the same time. This seems a more reasonable solution, since we would hope that even if routers ran the LS algorithm with the same periodicity, the execution instance of the algorithm would not be the same at each node. Interestingly, researchers have found that routers in the Internet can self-synchronize among themselves. One way to avoid such self-synchronization is for each router to randomize the time it sends out a link advertisement.

5.2.2 The Distance-Vector (DV) Routing Algorithm

Whereas the LS algorithm is an algorithm using global information, the distance vector (DV) algorithm is iterative, asynchronous, and distributed. It is distributed in that each node receives some information from one or more of its directly attached neighbors, performs a calculation, and then distributes the results of its calculation back to its neighbors. It is iterative in that this process continues on until no more information is exchanged between neighbors. The algorithm is asynchronous in that it does not require all of the nodes to operate in lockstep with each other.

Before we present the DV algorithm, it will prove beneficial to discuss an important relationship that exists among the costs of the least-cost paths. Let dx(y) be the cost of the least-cost path from node x to node y. Then the least costs are related by the celebrated Bellman-Ford equation, namely,

$$d_x(y) = min_v\{c(x,v) + d_v(y)\}$$

where the min_v in the equation is taken over all of x's neighbors. The Bellman-Ford equation is rather intuitive. Indeed, after traveling from x to v, if we then take the least-cost path from v to y, the path cost will be $c(x,v) + d_v(y)$. Since we must begin

traveling to some neighbor v, the least cost from x to y is the minimum of $c(x,v) + d_v(y)$ taken over all neighbors v.

It actually has significant practical importance: the solution to the Bellman-Ford equation provides the entries in node x's forwarding table.

With the DV algorithm, each node x maintains the following routing information:

- For each neighbor v, the cost c(x,v) from x to directly attached neighbor, v.
- Node x's distance vector, that is, $D_x = [D_x(y) : y \; in \; N]$, containing x's estimate of its cost to all destinations, y, in N.
- The distance vectors of each of its neighbors, that is, $D_x = [D_x(y) : y \; in \; N]$ for each neighbor v of x.

$$D_x(y) = min_v\{c(x,v) + D_v(y)\} \; for \; each \; node \; y \; in \; N$$

Distance-Vector (DV) Algorithm

```
Initialization:
for all destinations y in N:
  Dx(y)= c(x,y)/* if y is not a neighbor then c(x,y)= ∞ */4
for each neighbor w
  Dw(y) = ? for all destinations y in N
for each neighbor w
  send distance vector Dx = [Dx(y): y in N] to w

loop
  wait (until I see a link cost change to some neighbor w or
          until I receive a distance vector from some neighbor w)

  for each y in N:
    Dx(y) = minv{c(x,v) + Dv(y)}

if Dx(y) changed for any destination y
    send distance vector Dx = [Dx(y): y in N] to all neighbors

forever
```

In the DV algorithm, a node x updates its distance-vector estimate when it either sees a cost change in one of its directly attached links or receives a distance-vector update from some neighbor.

Recall that the LS algorithm is a centralized algorithm in the sense that it requires each node to first obtain a complete map of the network before running the Dijkstra algorithm. The DV algorithm is decentralized and does not use such global information. DV-like algorithms are used in many routing protocols in practice, including the Internet's RIP and BGP, ISO IDRP, Novell IPX, and the original ARPAnet.

Distance-Vector Algorithm: Link-Cost Changes and Link Failure

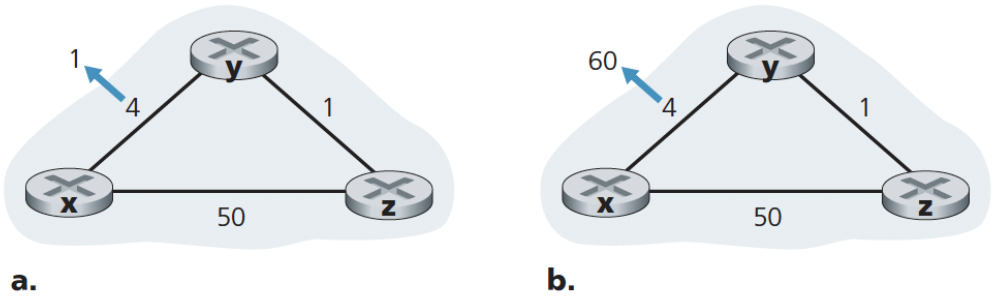

Figure 5.7 ♦ Changes in link cost

When a node running the DV algorithm detects a change in the link cost from itself to a neighbor (Lines 10–11), it updates its distance vector (Lines 13–14) and, if there's a change in the cost of the least-cost path, informs its neighbors (Lines 16–17) of its new distance vector. The DV algorithm causes the following sequence of events to occur:

- At time t_0, y detects the link-cost change (the cost has changed from 4 to 1), updates its distance vector, and informs its neighbors of this change since its distance vector has changed.
- At time t_1, z receives the update from y and updates its table. It computes a new least cost to x (it has decreased from a cost of 5 to a cost of 2) and sends its new distance vector to its neighbors.
- At time t_2, y receives z's update and updates its distance table. y's least costs do not change and hence y does not send any message to z. The algorithm comes to a quiescent state.

Thus, only two iterations are required for the DV algorithm to reach a quiescent state. The good news about the decreased cost between x and y has propagated quickly through the network.

Let’s now consider what can happen when a link cost increases. Suppose that the link cost between x and y increases from 4 to 60, as shown in Figure 5.7(b).

1. Before the link cost changes, $D_y(x) = 4,\ D_y(z) = 1,\ D_z(y),\ and\ D_z(x) = 5.$ At time t_0, y detects the link-cost change (the cost has changed from 4 to 60). y computes its new minimum-cost path to x to have a cost of:

$$D_y(x) = min\{c(y,x) + D_x(x), c(y,z) + D_z(x)\} = min\{60 + 0, 1 + 5\} = 6$$

Of course, with our global view of the network, we can see that this new cost via z is wrong. But the only information node y has is that its direct cost to x is 60 and that z has last told y that z could get to x with a cost of 5. So in order to get to x, y would now route through z, fully expecting that z will be able to get to x with a cost of 5. As of t_1 we have a **routing loop**. A packet destined for x arriving at y or z as of t_1 will bounce vack and forth between these two nodes forever.

2. Since node y has computed a new minimum cost to x, it informs z of its new distance vector at time t_1.
3. Sometimes after t_1, z receives y’s new distance vector, which indicates that y’s minimum cost to x is 6. z knows it can get to y with a cost of 1 and hence computes a new least cost to x of $D_z(x) = min\{50 + 0, 1 + 6\} = 7$.
4. In a similar manner, after receiving z’s new distance vector, y determines $D_y(x) = 8$ and sends z its distance vector. z then determines $D_z(x) = 9$ and sends y its distance vector, and so on.

You should convince yourself that the loop will persist for 44 iterations (message exchanges between y and z)—until z eventually computes the cost of its path via y to be greater than 50. At this point, z will (finally!) determine that its least-cost path to x is via its direct connection to x. y will then route to x via z. Because of such scenarios, the problem we have seen is sometimes referred to as the count-to-infinity problem.

Distance-Vector Algorithm: Adding Poisoned reverse

The specific looping scenario just described can be avoided using a technique known as poisoned reverse. The idea is simple—if z routes through y to get to destination x, then z will advertise to y that its distance to x is infinity, that is, z will advertise to y that $D_z(x) = \infty$, z will continue telling this little white lie to y as long as it routes to x via y. Since y believes that z has no path to x, y will never attempt to route to x via z, as long as z continues to route to x via y.

You should convince yourself that loops involving three or more nodes (rather than simply two immediately neighboring nodes) will not be detected by the poisoned reverse technique.

A Comparison of LS and DV Routing Algorithms

The DV and LS algorithms take complementary approaches toward computing routing. In the DV algorithm, each node talks to only its directly connected neighbors, but it provides its neighbors with least-cost estimates from itself to all the nodes in the network. The LS algorithm requires global information. Let's conclude our study of LS and DV algorithms with a quick comparison of some of their attributes. Recall that N is the set of nodes (routers) and E is the set of edges (links).

- *Message complexity.* We have seen that LS requires each node to know the cost of each link in the network. This requires $O(|N||E|)$ messages to be sent. Also, whenever a link cost changes, the new link cost must be sent to all nodes. The DV algorithm requires message exchanges between directly connected neighbors at each iteration. We have seen that the time needed for the algorithm to converge can depend on many factors. When link costs change, the DV algorithm will propagate the results of the changed link cost only if the new link cost results in a changed least-cost path for one of the nodes attached to that link.
- *Speed of convergence.* We have seen that our implementation of LS is an $O(|N|^2)$ algorithm requiring $O(|N||E|)$ messages. The DV algorithm can converge slowly and can have routing loops while the algorithm is converging. DV also suffers from the count-to-infinity problem.
- Robustness. Under LS, a router could broadcast an incorrect cost for one of its attached links
  (but no others). A node could also corrupt or drop any packets it received as part of

an LS broadcast. But an LS node is computing only its own forwarding tables; other nodes are performing similar calculations for themselves. This means route calculations are somewhat separated under LS, providing a degree of robustness. Under DV, a node can advertise incorrect least-cost paths to any or all destinations.

5.3 Intra-AS Routing in the Internet: OSPF

One router was indistinguishable from another in the sense that all routers executed the same routing algorithm to compute routing paths through the entire network. In practice, this model and its view of a homogenous set of routers all executing the same routing algorithm is simplistic for two important reasons:

- *Scale.* As the number of routers becomes large, the overhead involved in communicating, computing, and storing routing information becomes prohibitive. A distance-vector algorithm
  that iterated among such a large number of routers would surely never converge. Clearly, something must be done to reduce the complexity of route computation in a network as large as the Internet.
- Administrative autonomy. An ISP generally desires to operate its network as it pleases or to hide aspects of its network's internal organization from the outside. Ideally, an organization should be able to operate and administer its network as it wishes, while still being able to connect its network to other outside networks.

Both of these problems can be solved by organizing routers into **autonomous systems (ASs)**, with each AS consisting of a group of routers that are under the same administrative control. Often the routers in an ISP, and the links that interconnect them, constitute a single AS.

Routers within the same AS all run the same routing algorithm and have information about each other. The routing algorithm running within an autonomous system is called an intra-autonomous system routing protocol.

Open Shortest Path First (OSPF)

OSPF routing and its closely related cousin, IS-IS, are widely used for intra-AS routing in the Internet. The Open in OSPF indicates that the routing protocol specification is publicly available.

OSPF is a link-state protocol that uses flooding of link-state information and a Dijkstra's least-cost path algorithm. With OSPF, each router constructs a complete topological map (that is, a graph) of the entire autonomous system. Each router then locally runs Dijkstra's shortest-path algorithm to determine a shortest-path tree to all subnets, with itself as the root node. Individual link costs are configured by the network administrator. The administrator might choose to set all link costs to 1, thus achieving minimum-hop routing, or might choose to set the link weights to
be inversely proportional to link capacity in order to discourage traffic from using low-bandwidth links. OSPF does not mandate a policy for how link weights are set but instead provides the mechanisms (protocol) for determining least-cost path routing for the given set of link weights.

With OSPF, a router broadcasts routing information to all other routers in the autonomous system, not just to its neighboring router. A router broadcasts link-state information whenever there is a change in a link's state. It also broadcasts a link's state periodically even if the link's state has not changed. OSPF advertisements are contained in OSPF messages that are carried directly by IP, with an upper-layer protocol of 89 for OSPF. Thus, the OSPF protocol must itself implement functionality such as reliable message transfer and link-state broadcast. The OSPF protocol also checks that links are operational (via a HELLO message that is sent to an attached neighbor) and allows an OSPF router to obtain a neighboring router's database of network-wide link state.

Some of the advances embodied in OSPF include the following:

- *Security.* Exchanges between OSPF routers can be authenticated. With authentication, only trusted routers can participate in the OSPF protocol within an AS, thus preventing malicious intruders from injecting incorrect information into router tables. By default, OSPF packets between routers are not authenticated and could be forged. Two types of authentication can be configured—simple and MD5. With simple authentication, the same password is con
  figured on each router. When a router sends an OSPF packet, it includes the password in plaintext. Clearly, simple authentication is not very secure. MD5 authentication is based on shared secret keys that are configured in all the routers.

For each OSPF packet that it sends, the router computes the MD5 hash of the content of the OSPF packet appended with the secret key.

- *Multiple same-cost paths.* When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used.
- *Integrated support for unicast and multicast routing.* Multicast OSPF provides simple extensions to OSPF to provide for multicast routing. MOSPF uses the existing OSPF link database and adds a new type of link-state advertisement to the existing OSPF link-state broadcast mechanism.
- *Support for hierarchy within a single AS.* An OSPF autonomous system can be configured hierarchically into areas. Each area runs its own OSPF link-state routing algorithm, with each router in an area broadcasting its link state to all other routers in that area. Within each area, one or more area border routers are responsible for routing packets outside the area. Lastly, exactly one OSPF area in the AS is configured to be the backbone area. The primary role of the backbone area is to route traffic between the other areas in the AS. The backbone always contains all area border routers in the AS and may contain non-border routers as well. Inter-area routing within the AS requires that the packet be first routed to an area border router (intra-area routing), then routed through the backbone to the area border router that is in the destination area, and then routed to the final destination.

5.4 Routing Among the ISPs: BGP

We need an **inter-autonomous system routing protocol.** Since an inter-AS routing protocol involves coordination among multiple ASs, communicating ASs must run the same inter-AS routing protocol. In fact, in the Internet, all ASs run the same inter-AS routing protocol, called the Border Gateway Protocol, more commonly known as **BGP.**

5.4.1 The Role of BGP

In BGP, packets are not routed to a specific destination address, but instead to CIDRized prefixes, with each prefix representing a subnet or a collection of subnets.

As an inter-AS routing protocol, BGP provides each router a means to:

1. Obtain prefix reachability information from neighboring ASs. In particular, BGP allows each subnet to advertise its existence to the rest of the Internet. A subnet screams, “I exist and I am here,” and BGP makes sure that all the routers in the Internet know about this subnet.
2. Determine the “best” routes to the prefixes. A router may learn about two or more different routes to a specific prefix. To determine the best route, the router will locally run a BGP route-selection procedure. The best route will be determined based on policy as well as the reachability information.

5.4.2 Advertising BGP Route Information

For each AS, each router is either a gateway router or an internal router. A gateway router is a router on the edge of an AS that directly connects to one or more routers in other ASs. An internal router connects only to hosts and routers within its own AS.

Although the discussion in the above paragraph about advertising BGP reachability information should get the general idea across, it is not precise in the sense that autonomous systems do not actually send messages to each other, but instead routers do. In BGP, pairs of routers exchange routing information over semi-permanent TCP connections using port 179. Each such TCP connection, along with all the BGP messages sent over the connection, is called a **BGP connection.** Furthermore, a BGP connection that spans two ASs is called an **external BGP (eBGP)** connection, and a BGP session between routers in the same AS is called an **internal BGP (iBGP) connection.** There is typically one eBGP connection for each link that directly connects gateway routers in different ASs.

There are also iBGP connections between routers within each of the ASs.

5.4.3 Determining the Best Routes

There may be many paths from a given router to a destination subnet. How does a router choose among these paths (and then configure its forwarding table accordingly)?

Before addressing this critical question, we need to introduce a little more BGP terminology. When a router advertises a prefix across a BGP connection, it includes with

the prefix several **BGP attributes.** In BGP jargon, a prefix along with its attributes is called a **route.** Two of the more important attributes are AS-PATH and NEXT-HOP. The AS-PATH attribute contains the list of ASs through which the advertisement has passed, as we've seen in our examples above. To generate the AS-PATH value, when a prefix is passed to an AS, the AS adds its ASN to the existing list in the AS-PATH. BGP routers also use the AS-PATH attribute to detect and prevent looping advertisements; specifically, if a router sees that its own AS is contained in the path list, it will reject the advertisement.

Providing the critical link between the inter-AS and intra-AS routing protocols, the NEXT-HOP attribute has a subtle but important use. The NEXT-HOP is the IP address of the router interface that begins the AS-PATH.

Here, each BGP route is written as a list with three components: NEXT-HOP; ASPATH; destination prefix. Note that the NEXT-HOP attribute is an IP address of a router that does not belong to AS1; however, the subnet that contains this IP address directly attaches to AS1.

Hot Potato Routing

We are now finally in position to talk about BGP routing algorithms in a precise manner. We will begin with one of the simplest routing algorithms, namely, **hot potato routing.**

As just described, this router will learn about two possible BGP routes to prefix x. In hot potato routing, the route chosen (from among all possible routes) is that route with the least cost to the NEXT-HOP router beginning that route.

The steps for adding an outside-AS prefix in a router's forwarding table for hot potato routing are summarized in Figure 5.11. It is important to note that when adding an outside-AS prefix into a forwarding table, both the inter-AS routing protocol (BGP) and the intra-AS routing protocol (e.g., OSPF) are used.

The idea behind hot-potato routing is for router 1b to get packets out of its AS as quickly as possible (more specifically, with the least cost possible) without worrying about the cost of the remaining portions of the path outside of its AS to the destination. It tries to reduce the cost in its own AS while ignoring the other components of the end-to-end costs outside its AS.

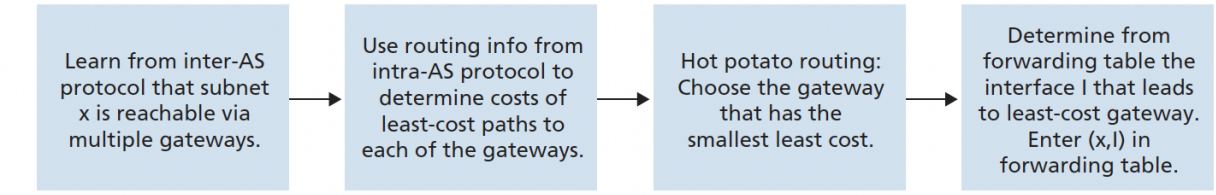

Figure 5.11 ♦ Steps in adding outside-AS destination in a router's forwarding table

Route-Selection Algorithm

In practice, BGP uses an algorithm that is more complicated than hot potato routing, but nevertheless incorporates hot potato routing. For any given destination prefix, the input into BGP's route-selection algorithm is the set of all routes to that prefix that have been learned and accepted by the router. If there is only one such route, then BGP obviously selects that route. If there are two or more routes to the same prefix, then BGP sequentially invokes the following elimination rules until one route remains:

1. A route is assigned a **local preference** value as one of its attributes. The local preference of a route could have been set by the router or could have been learned from another router in the same AS. The value of the local preference attribute is a policy decision that is left entirely up to the AS's network administrator. The routes with the highest local preference values are selected.
2. From the remaining routes (all with the same highest local preference value), the route with the shortest AS-PATH is selected. If this rule were the only rule for route selection, then BGP would be using a DV algorithm for path determination, where the distance metric uses the number of AS hops rather than the number of router hops.
3. From the remaining routes (all with the same highest local preference value and the same AS-PATH length), hot potato routing is used, that is, the route with the closest NEXT-HOP router is selected.
4. If more than one route still remains, the router uses BGP identifiers to select the route.

As noted above, BGP is the de facto standard for inter-AS routing for the Internet.

5.4.4 IP-Anycast

BGP is often used to implement the IP-anycast service, which is commonly used in DNS. To motivate IP-anycast, consider that in many applications, we are interested in (1) replicating the same content on different servers in many different dispersed geographical locations, and (2) having each user access the content from the server that is closest. For example, a CDN may replicate videos and other objects on servers in different countries. Similarly, the DNS system can replicate DNS records on DNS servers throughout the world. When a user wants to access this replicated content, it is desirable to point the user to the "nearest" server with the replicated content. BGP's route-selection algorithm provides an easy and natural mechanism for doing so.

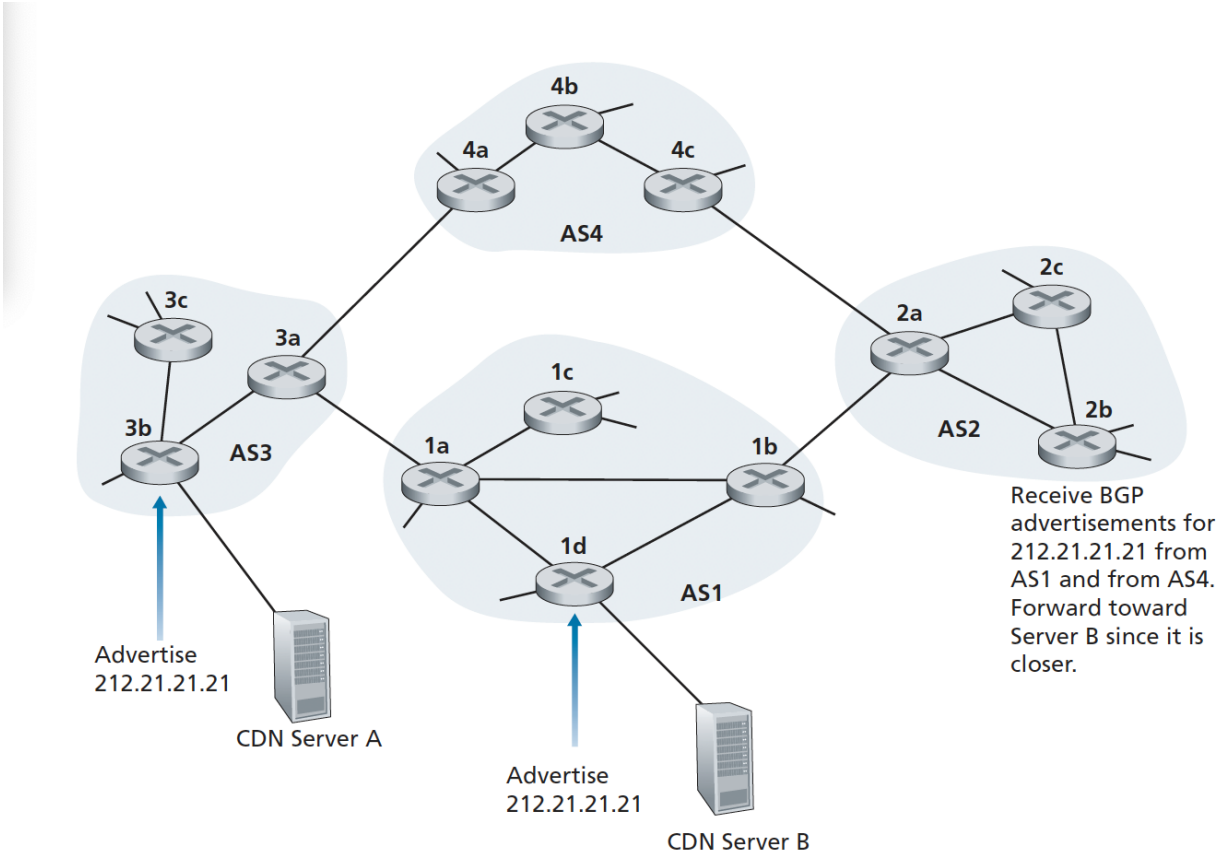

Figure 5.12 ♦ Using IP-anycast to bring users to the closest CDN server

To make our discussion concrete, let’s describe how a CDN might use IP-anycast. As shown in Figure 5.12, during the IP-anycast configuration stage, the CDN company assigns the same IP address to each of its servers, and uses standard BGP to advertise this IP address from each of the servers. When a BGP router receives multiple route advertisements for this IP address, it treats these advertisements as providing different paths to the same physical location (when, in fact, the advertisements are for different paths to different physical locations). When configuring its routing table, each router will locally use the BGP route-selection algorithm to pick the “best” (for example, closest, as determined by AS-hop counts) route to that IP address. For example, if one BGP route (corresponding to one location) is only one AS hop away from the router, and all other BGP routes (corresponding to other locations) are two or more AS hops away, then the BGP
router would choose to route packets to the location that is one hop away. After this initial BGP address-advertisement phase, the CDN can do its main job of distributing content. When a client requests the video, the CDN returns to the client the common IP address used by the geographically dispersed servers, no matter where the client is located. When the client sends a request to that IP address, Internet routers then forward the request packet to the “closest” server, as defined by the BGP route-selection algorithm.

Although the above CDN example nicely illustrates how IP-anycast can be used, in practice, CDNs generally choose not to use IP-anycast because BGP routing changes can result in different packets of the same TCP connection arriving at different instances of the Web server. But IP-anycast is extensively used by the DNS system to direct DNS queries to the closest root DNS server.

5.4.5 Routing Policy

When a router selects a route to a destination, the AS routing policy can trump all other considerations, such as shortest AS path or hot potato routing. Indeed, in the route-selection algorithm, routes are first selected according to the local-preference attribute, whose value is fixed by the policy of the local AS.

There are currently no official standards that govern how backbone ISPs route among themselves. However, a rule of thumb followed by commercial ISPs is that any traffic flowing across an ISP’s backbone network must have either a source or a destination

(or both) in a network that is a customer of that ISP; otherwise the traffic would be getting a free ride on
the ISP’s network.