

4.3	What's Inside a Router?	320
4.3.1	Input Processing	322
4.3.2	Switching	324
4.3.3	Output Processing	326
4.3.4	Where Does Queuing Occur?	327
4.3.5	The Routing Control Plane	331
4.5	Routing Algorithms	363
4.5.1	The Link-State (LS) Routing Algorithm	366
4.5.2	The Distance-Vector (DV) Routing Algorithm	371
4.5.3	Hierarchical Routing	379
4.6	Routing in the Internet	383
4.6.1	Intra-AS Routing in the Internet: RIP	384
4.6.2	Intra-AS Routing in the Internet: OSPF	388
4.6.3	Inter-AS Routing: BGP	390
4.8	Summary	412
	Homework Problems and Questions	413
	Programming Assignments	429
	Wireshark Labs: IP, ICMP	430
	Interview: Vinton G. Cerf	431

4.1 What's Inside a Router?

Now that we've overviewed the network layer's services and functions, let's turn our attention to its **forwarding function**—the actual transfer of packets from a router's incoming links to the appropriate outgoing links at that router. We already took a brief look at a few aspects of forwarding in Section 4.2, namely, addressing and longest prefix matching. We mention here in passing that the terms *forwarding* and *switching* are often used interchangeably by computer-networking researchers and practitioners; we'll use both terms interchangeably in this textbook as well.

A high-level view of a generic router architecture is shown in Figure 4.6. Four router components can be identified:

- *Input ports.* An input port performs several key functions. It performs the physical layer function of terminating an incoming physical link at a router; this is shown in the leftmost box of the input port and the rightmost box of the output port in Figure 4.6. An input port also performs link-layer functions needed to interoperate with the link layer at the other side of the incoming link; this is represented by the middle boxes in the input and output ports. Perhaps most crucially, the lookup function is also performed at the input port; this will occur in the rightmost box of the input port. It is here that the forwarding table is consulted to determine the router output port to which an

arriving packet will be forwarded via the switching fabric. Control packets (for example, packets carrying routing protocol information) are forwarded from an input port to the routing processor. Note that the term *port* here—referring to the physical input and output router interfaces—is distinctly different from the software ports associated with network applications and sockets discussed in Chapters 2 and 3.

- *Switching fabric.* The switching fabric connects the router's input ports to its output ports. This switching fabric is completely contained within the router—a network inside of a network router!
- *Output ports.* An output port stores packets received from the switching fabric and transmits these packets on the outgoing link by performing the necessary link-layer and physical-layer functions. When a link is bidirectional (that is,

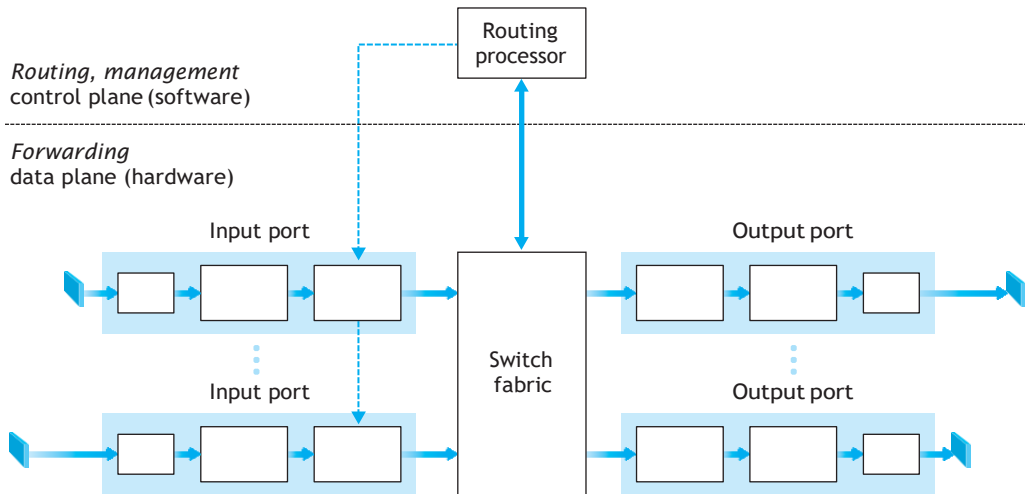


Figure 4.6 ♦ Router architecture

carries traffic in both directions), an output port will typically be paired with the input port for that link on the same line card (a printed circuit board containing one or more input ports, which is connected to the switching fabric).

- *Routing processor.* The routing processor executes the routing protocols (which we'll study in Section 4.6), maintains routing tables and attached link state information, and computes the forwarding table for the router. It also performs the network management functions that we'll study in Chapter 9.

Recall that in Section 4.1.1 we distinguished between a router's forwarding and routing functions. A router's input ports, output ports, and switching fabric together implement the forwarding function and are almost always implemented in hardware, as shown in Figure 4.6. These forwarding functions are sometimes collectively referred to as the **router forwarding plane**. To appreciate why a hardware implementation is needed, consider that with a 10 Gbps input link and a 64-byte IP datagram, the input port has only 51.2 ns to process the datagram before another datagram may arrive. If N ports are combined on a line card (as is often done in practice), the datagram-processing pipeline must operate N times faster—far too fast for software implementation. Forwarding plane hardware can be implemented either using a router vendor's own hardware designs, or constructed using purchased merchant-silicon chips (e.g., as sold by companies such as Intel and Broadcom).

While the forwarding plane operates at the nanosecond time scale, a router's control functions—executing the routing protocols, responding to attached links that

go up or down, and performing management functions such as those we'll study in Chapter 9—operate at the millisecond or second timescale. These **router control plane** functions are usually implemented in software and execute on the routing processor (typically a traditional CPU).


Before delving into the details of a router's control and data plane, let's return to our analogy of Section 4.1.1, where packet forwarding was compared to cars entering and leaving an interchange. Let's suppose that the interchange is a roundabout, and that before a car enters the roundabout, a bit of processing is required—the car stops at an entry station and indicates its final destination (not at the local roundabout, but the ultimate destination of its journey). An attendant at the entry station looks up the final destination, determines the roundabout exit that leads to that final destination, and tells the driver which roundabout exit to take. The car enters the roundabout (which may be filled with other cars entering from other input roads and heading to other roundabout exits) and eventually leaves at the prescribed roundabout exit ramp, where it may encounter other cars leaving the roundabout at that exit.

We can recognize the principal router components in Figure 4.6 in this analogy—the entry road and entry station correspond to the input port (with a lookup function to determine the local outgoing port); the roundabout corresponds to the switch fabric; and the roundabout exit road corresponds to the output port. With this analogy, it's instructive to consider where bottlenecks might occur. What happens if cars arrive blazingly fast (for example, the roundabout is in Germany or Italy!) but the station attendant is slow? How fast must the attendant work to ensure there's no backup on an entry road? Even with a blazingly fast attendant, what happens if cars traverse the roundabout slowly—can backups still occur? And what happens if most of the entering cars all want to leave the roundabout at the same exit ramp—can backups occur at the exit ramp or elsewhere? How should the roundabout operate if we want to assign priorities to different cars, or block certain cars from entering the roundabout in the first place? These are all analogous to critical questions faced by router and switch designers.

In the following subsections, we'll look at router functions in more detail. [Iyer 2008, Chao 2001; Chuang 2005; Turner 1988; McKeown 1997a; Partridge 1998] provide a discussion of specific router architectures. For concreteness, the ensuing discussion assumes a datagram network in which forwarding decisions are based on the packet's destination address (rather than a VC number in a virtual-circuit network). However, the concepts and techniques are quite similar for a virtual-circuit network.

4.1.1 Input Processing

A more detailed view of input processing is given in Figure 4.7. As discussed above, the input port's line termination function and link-layer processing implement the physical and link layers for that individual input link. The lookup performed in the input port is central to the router's operation—it is here that the router uses the forwarding table to look up the output port to which an arriving packet will be



CASE HISTORY

CISCO SYSTEMS: DOMINATING THE NETWORK CORE

As of this writing 2012, Cisco employs more than 65,000 people. How did this gorilla of a networking company come to be? It all started in 1984 in the living room of a Silicon Valley apartment.

Len Bosak and his wife Sandy Lerner were working at Stanford University when they had the idea to build and sell Internet routers to research and academic institutions, the primary adopters of the Internet at that time. Sandy Lerner came up with the name Cisco (an abbreviation for San Francisco), and she also designed the company's bridge logo. Corporate headquarters was their living room, and they financed the project with credit cards and moonlighting consulting jobs. At the end of 1986, Cisco's revenues reached \$250,000 a month. At the end of 1987, Cisco succeeded in attracting venture capital—\$2 million from Sequoia Capital in exchange for one-third of the company. Over the next few years, Cisco continued to grow and grab more and more market share. At the same time, relations between Bosak/Lerner and Cisco management became strained. Cisco went public in 1990; in the same year Lerner and Bosak left the company.

Over the years, Cisco has expanded well beyond the router market, selling security, wireless caching, Ethernet switch, datacenter infrastructure, video conferencing, and voice-over IP products and services. However, Cisco is facing increased international competition, including from Huawei, a rapidly growing Chinese network-gear company. Other sources of competition for Cisco in the router and switched Ethernet space include Alcatel-Lucent and Juniper.

forwarded via the switching fabric. The forwarding table is computed and updated by the routing processor, with a shadow copy typically stored at each input port. The forwarding table is copied from the routing processor to the line cards over a separate bus (e.g., a PCI bus) indicated by the dashed line from the routing processor to the input line cards in Figure 4.6. With a shadow copy, forwarding decisions can be made locally, at each input port, without invoking the centralized routing processor on a per-packet basis and thus avoiding a centralized processing bottleneck.

Given the existence of a forwarding table, lookup is conceptually simple—we just search through the forwarding table looking for the longest prefix match, as described

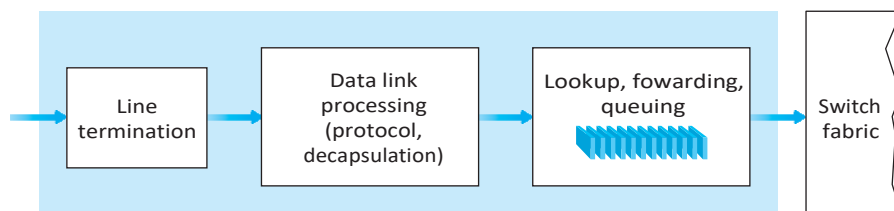


Figure 4.7 ♦ Input port processing

in Section 4.2.2. But at Gigabit transmission rates, this lookup must be performed in nanoseconds (recall our earlier example of a 10 Gbps link and a 64-byte IP datagram). Thus, not only must lookup be performed in hardware, but techniques beyond a simple linear search through a large table are needed; surveys of fast lookup algorithms can be found in [Gupta 2001, Ruiz-Sanchez 2001]. Special attention must also be paid to memory access times, resulting in designs with embedded on-chip DRAM and faster SRAM (used as a DRAM cache) memories. Ternary Content Address Memories (TCAMs) are also often used for lookup. With a TCAM, a 32-bit IP address is presented to the memory, which returns the content of the forwarding table entry for that address in essentially constant time. The Cisco 8500 has a 64K CAM for each input port.

Once a packet's output port has been determined via the lookup, the packet can be sent into the switching fabric. In some designs, a packet may be temporarily blocked from entering the switching fabric if packets from other input ports are currently using the fabric. A blocked packet will be queued at the input port and then scheduled to cross the fabric at a later point in time. We'll take a closer look at the blocking, queuing, and scheduling of packets (at both input ports and output ports) in Section 4.3.4. Although "lookup" is arguably the most important action in input port processing, many other actions must be taken: (1) physical- and link-layer processing must occur, as discussed above; (2) the packet's version number, checksum and time-to-live field—all of which we'll study in Section 4.4.1—must be checked and the latter two fields rewritten; and (3) counters used for network management (such as the number of IP datagrams received) must be updated.

Let's close our discussion of input port processing by noting that the input port steps of looking up an IP address ("match") then sending the packet into the switching fabric ("action") is a specific case of a more general "match plus action" abstraction that is performed in many networked devices, not just routers. In link-layer switches (covered in Chapter 5), link-layer destination addresses are looked up and several actions may be taken in addition to sending the frame into the switching fabric towards the output port. In firewalls (covered in Chapter 8)—devices that filter out selected incoming packets—an incoming packet whose header matches a given criteria (e.g., a combination of source/destination IP addresses and transport-layer port numbers) may be prevented from being forwarded (action). In a network address translator (NAT, covered in Section 4.4), an incoming packet whose transport-layer port number matches a given value will have its port number rewritten before forwarding (action). Thus, the "match plus action" abstraction is both powerful and prevalent in network devices.

4.1.2 Switching

The switching fabric is at the very heart of a router, as it is through this fabric that the packets are actually switched (that is, forwarded) from an input port to an output port. Switching can be accomplished in a number of ways, as shown in Figure 4.8:

- *Switching via memory.* The simplest, earliest routers were traditional computers, with switching between input and output ports being done under direct control of

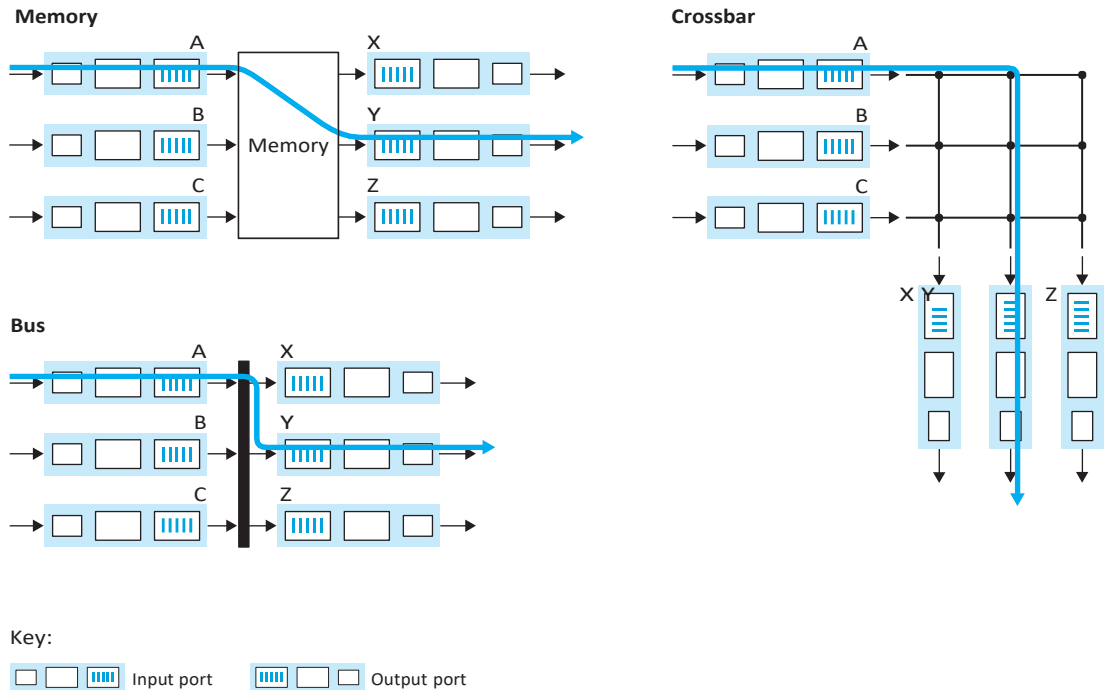


Figure 4.8 ♦ Three switching techniques

the CPU (routing processor). Input and output ports functioned as traditional I/O devices in a traditional operating system. An input port with an arriving packet first signaled the routing processor via an interrupt. The packet was then copied from the input port into processor memory. The routing processor then extracted the destination address from the header, looked up the appropriate output port in the forwarding table, and copied the packet to the output port's buffers. In this scenario, if the memory bandwidth is such that B packets per second can be written into, or read from, memory, then the overall forwarding throughput (the total rate at which packets are transferred from input ports to output ports) must be less than $B/2$. Note also that two packets cannot be forwarded at the same time, even if they have different destination ports, since only one memory read/write over the shared system bus can be done at a time.

Many modern routers switch via memory. A major difference from early routers, however, is that the lookup of the destination address and the storing of the packet into the appropriate memory location are performed by processing on the input line cards. In some ways, routers that switch via memory look very much like shared-memory multiprocessors, with the processing on a line card switching (writing) packets into the memory of the appropriate output port. Cisco's Catalyst 8500 series switches [Cisco 8500 2012] forward packets via a shared memory.

- *Switching via a bus.* In this approach, an input port transfers a packet directly to the output port over a shared bus, without intervention by the routing processor. This is typically done by having the input port pre-pend a switch-internal label (header) to the packet indicating the local output port to which this packet is being transferred and transmitting the packet onto the bus. The packet is received by all output ports, but only the port that matches the label will keep the packet. The label is then removed at the output port, as this label is only used within the switch to cross the bus. If multiple packets arrive to the router at the same time, each at a different input port, all but one must wait since only one packet can cross the bus at a time. Because every packet must cross the single bus, the switching speed of the router is limited to the bus speed; in our roundabout analogy, this is as if the roundabout could only contain one car at a time. Nonetheless, switching via a bus is often sufficient for routers that operate in small local area and enterprise networks. The Cisco 5600 [Cisco Switches 2012] switches packets over a 32 Gbps backplane bus.
- *Switching via an interconnection network.* One way to overcome the bandwidth limitation of a single, shared bus is to use a more sophisticated interconnection network, such as those that have been used in the past to interconnect processors in a multiprocessor computer architecture. A crossbar switch is an interconnection network consisting of $2N$ buses that connect N input ports to N output ports, as shown in Figure 4.8. Each vertical bus intersects each horizontal bus at a crosspoint, which can be opened or closed at any time by the switch fabric controller (whose logic is part of the switching fabric itself). When a packet arrives from port A and needs to be forwarded to port Y, the switch controller closes the crosspoint at the intersection of busses A and Y, and port A then sends the packet onto its bus, which is picked up (only) by bus Y. Note that a packet from port B can be forwarded to port X at the same time, since the A-to-Y and B-to-X packets use different input and output busses. Thus, unlike the previous two switching approaches, crossbar networks are capable of forwarding multiple packets in parallel. However, if two packets from two different input ports are destined to the same output port, then one will have to wait at the input, since only one packet can be sent over any given bus at a time.

More sophisticated interconnection networks use multiple stages of switching elements to allow packets from different input ports to proceed towards the same output port at the same time through the switching fabric. See [Tobagi 1990] for a survey of switch architectures. Cisco 12000 family switches [Cisco 12000 2012] use an interconnection network.

4.1.3 Output Processing

Output port processing, shown in Figure 4.9, takes packets that have been stored in the output port's memory and transmits them over the output link. This includes selecting and de-queueing packets for transmission, and performing the needed link-layer and physical-layer transmission functions.

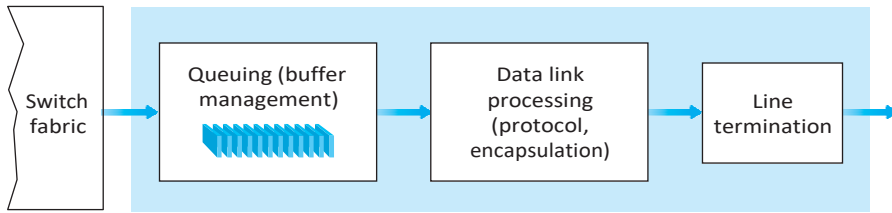


Figure 4.9 ♦ Output port processing

4.1.4 Where Does Queuing Occur?

If we consider input and output port functionality and the configurations shown in Figure 4.8, it's clear that packet queues may form at both the input ports *and* the output ports, just as we identified cases where cars may wait at the inputs and outputs of the traffic intersection in our roundabout analogy. The location and extent of queuing (either at the input port queues or the output port queues) will depend on the traffic load, the relative speed of the switching fabric, and the line speed. Let's now consider these queues in a bit more detail, since as these queues grow large, the router's memory can eventually be exhausted and **packet loss** will occur when no memory is available to store arriving packets. Recall that in our earlier discussions, we said that packets were “lost within the network” or “dropped at a router.” It is here, at these queues within a router, where such packets are actually dropped and lost.

Suppose that the input and output line speeds (transmission rates) all have an identical transmission rate of R_{line} packets per second, and that there are N input ports and N output ports. To further simplify the discussion, let's assume that all packets have the same fixed length, and the packets arrive to input ports in a synchronous manner. That is, the time to send a packet on any link is equal to the time to receive a packet on any link, and during such an interval of time, either zero or one packet can arrive on an input link. Define the switching fabric transfer rate R_{switch} as the rate at which packets can be moved from input port to output port. If R_{switch} is N times faster than R_{line} , then only negligible queuing will occur at the input ports. This is because even in the worst case, where all N input lines are receiving packets, and all packets are to be forwarded to the same output port, each batch of N packets (one packet per input port) can be cleared through the switch fabric before the next batch arrives.

But what can happen at the output ports? Let's suppose that R_{switch} is still N times faster than R_{line} . Once again, packets arriving at each of the N input ports are destined to the same output port. In this case, in the time it takes to send a single packet onto the outgoing link, N new packets will arrive at this output port. Since the output port can transmit only a single packet in a unit of time (the packet transmission time), the N arriving packets will have to queue (wait) for transmission over the outgoing link. Then N more packets can possibly arrive in the time it takes to

transmit just one of the N packets that had just previously been queued. And so on. Eventually, the number of queued packets can grow large enough to exhaust available memory at the output port, in which case packets are dropped.

Output port queuing is illustrated in Figure 4.10. At time t , a packet has arrived at each of the incoming input ports, each destined for the uppermost outgoing port. Assuming identical line speeds and a switch operating at three times the line speed, one time unit later (that is, in the time needed to receive or send a packet), all three original packets have been transferred to the outgoing port and are queued awaiting transmission. In the next time unit, one of these three packets will have been transmitted over the outgoing link. In our example, two *new* packets have arrived at the incoming side of the switch; one of these packets is destined for this uppermost output port.

Given that router buffers are needed to absorb the fluctuations in traffic load, the natural question to ask is how *much* buffering is required. For many years, the rule of thumb [RFC 3439] for buffer sizing was that the amount of buffering (B) should be equal to an average round-trip time (RTT , say 250 msec) times the link capacity (C). This result is based on an analysis of the queuing dynamics of a relatively small number of TCP flows [Villamizar 1994]. Thus, a 10 Gbps link with an RTT of 250 msec would need an amount of buffering equal to $B = RTT \cdot C = 2.5$ Gbits of buffers. Recent

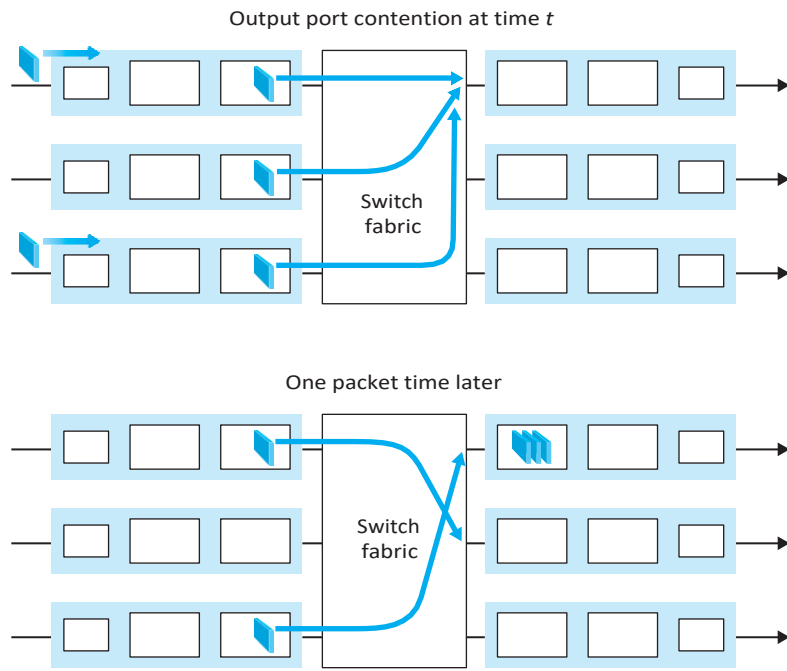


Figure 4.10 ♦ Output port queuing

theoretical and experimental efforts [Appenzeller 2004], however, suggest that when there are a large number of TCP flows (N) passing through a link, the amount of buffering needed is $B = RTT \cdot C\sqrt{N}$. With a large number of flows typically passing through large backbone router links (see, e.g., [Fraleigh 2003]), the value of N can be large, with the decrease in needed buffer size becoming quite significant. [Appenzeller 2004; Wischik 2005; Beheshti 2008] provide very readable discussions of the buffer sizing problem from a theoretical, implementation, and operational standpoint.

A consequence of output port queuing is that a **packet scheduler** at the output port must choose one packet among those queued for transmission. This selection might be done on a simple basis, such as first-come-first-served (FCFS) scheduling, or a more sophisticated scheduling discipline such as weighted fair queuing (WFQ), which shares the outgoing link fairly among the different end-to-end connections that have packets queued for transmission. Packet scheduling plays a crucial role in providing **quality-of-service guarantees**. We'll thus cover packet scheduling extensively in Chapter 7. A discussion of output port packet scheduling disciplines is [Cisco Queue 2012].

Similarly, if there is not enough memory to buffer an incoming packet, a decision must be made to either drop the arriving packet (a policy known as **drop-tail**) or remove one or more already-queued packets to make room for the newly arrived packet. In some cases, it may be advantageous to drop (or mark the header of) a packet *before* the buffer is full in order to provide a congestion signal to the sender. A number of packet-dropping and -marking policies (which collectively have become known as **active queue management (AQM)** algorithms) have been proposed and analyzed [Labrador 1999, Hollot 2002]. One of the most widely studied and implemented AQM algorithms is the **Random Early Detection (RED)** algorithm. Under RED, a weighted average is maintained for the length of the output queue. If the average queue length is less than a minimum threshold, min_{th} , when a packet arrives, the packet is admitted to the queue. Conversely, if the queue is full or the average queue length is greater than a maximum threshold, max_{th} , when a packet arrives, the packet is marked or dropped. Finally, if the packet arrives to find an average queue length in the interval $[min_{th}, max_{th}]$, the packet is marked or dropped with a probability that is typically some function of the average queue length, min_{th} , and max_{th} . A number of probabilistic marking/dropping functions have been proposed, and various versions of RED have been analytically modeled, simulated, and/or implemented. [Christiansen 2001] and [Floyd 2012] provide overviews and pointers to additional reading.

If the switch fabric is not fast enough (relative to the input line speeds) to transfer *all* arriving packets through the fabric without delay, then packet queuing can also occur at the input ports, as packets must join input port queues to wait their turn to be transferred through the switching fabric to the output port. To illustrate an important consequence of this queuing, consider a crossbar switching fabric and suppose that (1) all link speeds are identical, (2) that one packet can be transferred from any one input port to a given output port in the same amount of time it takes for a packet to be received on an input link, and (3) packets are moved from a given input queue to their

desired output queue in an FCFS manner. Multiple packets can be transferred in parallel, as long as their output ports are different. However, if two packets at the front of two input queues are destined for the same output queue, then one of the packets will be blocked and must wait at the input queue—the switching fabric can transfer only one packet to a given output port at a time.

Figure 4.11 shows an example in which two packets (darkly shaded) at the front of their input queues are destined for the same upper-right output port. Suppose that the switch fabric chooses to transfer the packet from the front of the upper-left queue. In this case, the darkly shaded packet in the lower-left queue must wait. But not only must this darkly shaded packet wait, so too must the lightly shaded packet that is queued behind that packet in the lower-left queue, even though there is *no* contention for the middle-right output port (the destination for the lightly shaded packet). This phenomenon is known as **head-of-the-line (HOL) blocking** in an

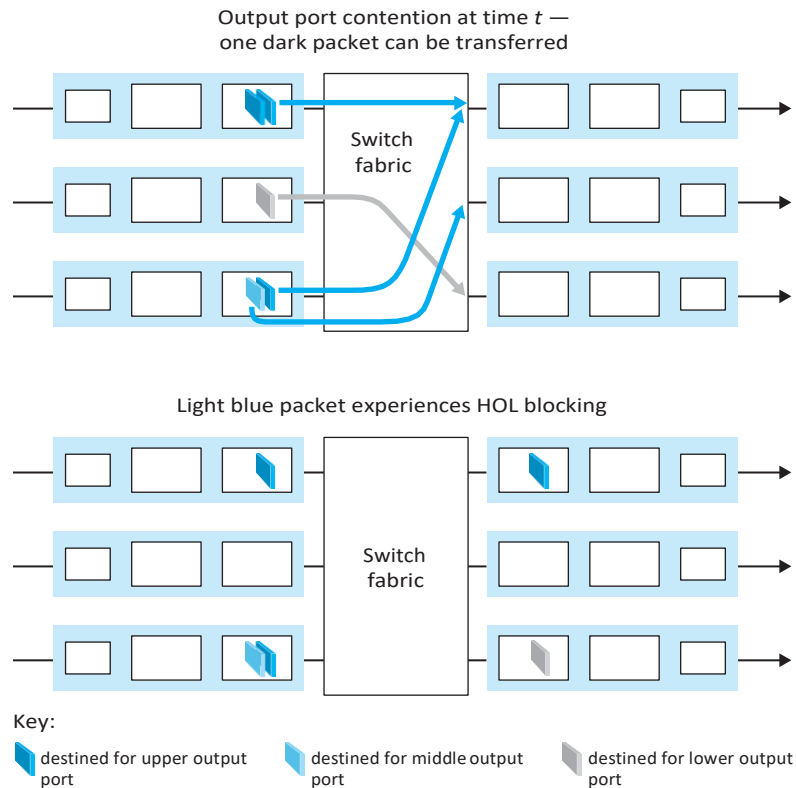


Figure 4.11 ♦ HOL blocking at an input queued switch

input-queued switch—a queued packet in an input queue must wait for transfer through the fabric (even though its output port is free) because it is blocked by another packet at the head of the line. [Karol 1987] shows that due to HOL blocking, the input queue will grow to unbounded length (informally, this is equivalent to saying that significant packet loss will occur) under certain assumptions as soon as the packet arrival rate on the input links reaches only 58 percent of their capacity. A number of solutions to HOL blocking are discussed in [McKeown 1997b].

4.1.5 The Routing Control Plane

In our discussion thus far and in Figure 4.6, we’ve implicitly assumed that the routing control plane fully resides and executes in a routing processor within the router. The network-wide routing control plane is thus decentralized—with different pieces (e.g., of a routing algorithm) executing at different routers and interacting by sending control messages to each other. Indeed, today’s Internet routers and the routing algorithms we’ll study in Section 4.6 operate in exactly this manner. Additionally, router and switch vendors bundle their hardware data plane and software control plane together into closed (but inter-operable) platforms in a vertically integrated product.

Recently, a number of researchers [Caesar 2005a, Casado 2009, McKeown 2008] have begun exploring new router control plane architectures in which part of the control plane is implemented in the routers (e.g., local measurement/reporting of link state, forwarding table installation and maintenance) along with the data plane, and part of the control plane can be implemented externally to the router (e.g., in a centralized server, which could perform route calculation). A well-defined API dictates how these two parts interact and communicate with each other. These researchers argue that separating the software control plane from the hardware data plane (with a minimal router-resident control plane) can simplify routing by replacing distributed routing calculation with centralized routing calculation, and enable network innovation by allowing different customized control planes to operate over fast hardware data planes.

4.2 Routing Algorithms

So far in this chapter, we’ve mostly explored the network layer’s forwarding function. We learned that when a packet arrives to a router, the router indexes a forwarding table and determines the link interface to which the packet is to be directed. We also learned that routing algorithms, operating in network routers, exchange and

compute the information that is used to configure these forwarding tables. The interplay between routing algorithms and forwarding tables was shown in Figure 4.2. Having explored forwarding in some depth we now turn our attention to the other major topic of this chapter, namely, the network layer’s critical routing function. Whether the network layer provides a datagram service (in which case different packets between a given source-destination pair may take different routes) or a VC service (in which case all packets between a given source and destination will take the same path), the network layer must nonetheless determine the path that packets take from senders to receivers. We’ll see that the job of routing is to determine good paths (equivalently, routes), from senders to receivers, through the network of routers.

Typically a host is attached directly to one router, the **default router** for the host (also called the **first-hop router** for the host). Whenever a host sends a packet, the packet is transferred to its default router. We refer to the default router of the source host as the **source router** and the default router of the destination host as the **destination router**. The problem of routing a packet from source host to destination host clearly boils down to the problem of routing the packet from source router to destination router, which is the focus of this section.

The purpose of a routing algorithm is then simple: given a set of routers, with links connecting the routers, a routing algorithm finds a “good” path from source router to destination router. Typically, a good path is one that has the least cost. We’ll see, however, that in practice, 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 to complicate the conceptually simple and elegant algorithms whose theory underlies the practice of routing in today’s networks.

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 . In the context of network-layer routing, the nodes in the graph represent routers—the points at which packet-forwarding decisions are made—and the edges connecting these nodes represent the physical links between these routers. Such a graph abstraction of a computer network is shown in Figure 4.27. To view some graphs representing real network maps, see [Dodge 2012, Cheswick 2000]; for a discussion of how well different graph-based models model the Internet, see [Zegura 1997, Faloutsos 1999, Li 2004].

As shown in Figure 4.27, an edge also has a value representing its cost. Typically, an edge’s cost may reflect the physical length of the corresponding link (for example, a transoceanic link might have a higher cost than a short-haul terrestrial link), the link speed, or the monetary cost associated with a link. For our purposes, we’ll simply take the edge costs as a given and won’t worry about how they are determined. 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 $c(x, y) = \infty$. Also, throughout we consider only undirected graphs (i.e., graphs whose edges do not have a direction), 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 .

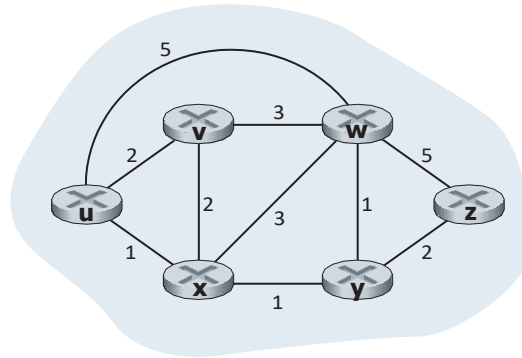


Figure 4.27 ♦ Abstract graph model of a computer network

Given that costs are assigned to the various edges in the graph abstraction, a natural goal of a routing algorithm is to identify the least costly paths between sources and destinations. To make this problem more precise, recall that a **path** in a graph $G = (N, E)$ is a sequence of nodes (x_1, x_2, \dots, x_p) such that each of the pairs (x_1, x_2) , $(x_2, x_3), \dots, (x_{p-1}, x_p)$ are edges in E . The cost of a path (x_1, x_2, \dots, x_p) is simply the sum of all the edge costs along the path, that is, $c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$. 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. In Figure 4.27, for example, the least-cost path between source node u and destination node w is (u, x, y, w) with a path cost of 3. Note that if all edges in the graph have the same cost, the least-cost path is also the **shortest path** (that is, the path with the smallest number of links between the source and the destination).

As a simple exercise, try finding the least-cost path from node u to z in Figure 4.27 and reflect for a moment on how you calculated that path. If you are like most people, you found the path from u to z by examining Figure 4.27, tracing a few routes from u to z , and somehow convincing yourself that the path you had chosen had the least cost among all possible paths. (Did you check all of the 17 possible paths between u and z ? Probably not!) Such a calculation is an example of a centralized routing algorithm—the routing algorithm was run in one location, your brain, with complete information about the network. Broadly, one way in which we can classify routing algorithms is according to whether they are global or decentralized.

- A **global 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

(a centralized global routing algorithm) or replicated at multiple sites. The key distinguishing feature here, however, is that a global algorithm has complete information about connectivity and link costs. In practice, 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. We'll study LS algorithms in Section 4.5.1.

- In a **decentralized routing algorithm**, the calculation of the least-cost path is carried out in an iterative, distributed manner. 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 (that is, nodes that are at the other end of links to which it itself is attached), a node gradually calculates the least-cost path to a destination or set of destinations. The decentralized routing algorithm we'll study below in Section 4.5.2 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 (for example, a human manually editing a router's forwarding table). **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 oscillation in routes.

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. If a high cost is associated with a link that is currently congested, a routing algorithm will tend to choose routes around such a congested link. While early ARPAnet routing algorithms were load-sensitive [McQuillan 1980], a number of difficulties were encountered [Huitema 1998]. 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.

4.2.1 The Link-State (LS) Routing Algorithm

Recall that in a link-state algorithm, the network topology and all link costs are known, that is, available as input to the LS algorithm. 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 (for example, with the Internet's OSPF routing protocol, discussed in Section 4.6.1) this is often accomplished by a **link-state broadcast**

algorithm [Perlman 1999]. We'll cover broadcast algorithms in Section 4.7. 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. A closely related algorithm is Prim's algorithm; see [Cormen 2001] for a general discussion of graph algorithms. Dijkstra's algorithm computes the least-cost path from one node (the source, which we will refer to as u) to all other nodes in the network. Dijkstra's algorithm is iterative and has the property that after the k th 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.

The global routing algorithm consists of an initialization step followed by a loop. The number of times the loop is executed is equal to the number of nodes in the network. Upon termination, the algorithm will have calculated the shortest paths from the source node u to every other node in the network.

Link-State (LS) Algorithm for Source Node u

```

1 Initialization:
2    $N' = \{u\}$ 
3   for all nodes  $v$ 
4     if  $v$  is a neighbor of  $u$ 
5       then  $D(v) = c(u,v)$ 
6       else  $D(v) = \infty$ 
7
8 Loop
9   find  $w$  not in  $N'$  such that  $D(w)$  is a minimum
10  add  $w$  to  $N'$ 
11  update  $D(v)$  for each neighbor  $v$  of  $w$  and not in  $N'$ :
12     $D(v) = \min( D(v), D(w) + c(w,v) )$ 
13  /* new cost to  $v$  is either old cost to  $v$  or known
14   least path cost to  $w$  plus cost from  $w$  to  $v$  */
15 until  $N' = N$ 
```



VideoNote
Dijkstra's algorithm:
discussion and example

As an example, let's consider the network in Figure 4.27 and compute the least-cost paths from u to all possible destinations. A tabular summary of the algorithm's computation is shown in Table 4.3, where each line in the table gives the values of the algorithm's variables at the end of the iteration. Let's consider the few first steps in detail.

- In the initialization step, the currently known least-cost paths from u to its directly attached neighbors, v , x , and w , are initialized to 2, 1, and 5, respectively. Note in particular that the cost to w is set to 5 (even though we will soon see that a lesser-cost path does indeed exist) since this is the cost of the direct (one hop) link from u to w . The costs to y and z are set to infinity because they are not directly connected to u .
- In the first iteration, we look among those nodes not yet added to the set N^t and find that node with the least cost as of the end of the previous iteration. That node is x , with a cost of 1, and thus x is added to the set N^t . Line 12 of the LS algorithm is then performed to update $D(v)$ for all nodes v , yielding the results shown in the second line (Step 1) in Table 4.3. The cost of the path to v is unchanged. The cost of the path to w (which was 5 at the end of the initialization) through node x is found to have a cost of 4. Hence this lower-cost path is selected and w 's predecessor along the shortest path from u is set to x . Similarly, the cost to y (through x) is computed to be 2, and the table is updated accordingly.
- In the second iteration, nodes v and y are found to have the least-cost paths (2), and we break the tie arbitrarily and add y to the set N^t so that N^t now contains u , x , and y . The cost to the remaining nodes not yet in N^t , that is, nodes v , w , and z , are updated via line 12 of the LS algorithm, yielding the results shown in the third row in the Table 4.3.
- And so on. . . .

When the LS algorithm terminates, we have, for each node, its predecessor along the least-cost path from the source node. For each predecessor, we also

step	N^t	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
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	$uxyvww$					4, y
5	$uxyvwwz$					

Table 4.3 ♦ Running the link-state algorithm on the network in Figure 4.27

have its predecessor, and so in this manner we can construct the entire path from the source to all destinations. The forwarding table in a node, say node u , can then be constructed from this information by storing, for each destination, the next-hop node on the least-cost path from u to the destination. Figure 4.28 shows the resulting least-cost paths and forwarding table in u for the network in Figure 4.27.

What is the computational complexity of this algorithm? That is, given n nodes (not counting the source), how much computation must be done in the worst case to find the least-cost paths from the source to all destinations? In the first iteration, we need to search through all n nodes to determine the node, w , not in N' that has the minimum cost. In the second iteration, we need to check $n - 1$ nodes to determine the minimum cost; in the third iteration $n - 2$ nodes, and so on. Overall, the total number of nodes we need to search through over all the iterations is $n(n + 1)/2$, and thus we say that the preceding implementation of the LS algorithm has worst-case complexity of order n squared: $O(n^2)$. (A more sophisticated implementation of this algorithm, using a data structure known as a heap, can find the minimum in line 9 in logarithmic rather than linear time, thus reducing the complexity.)

Before completing our discussion of the LS algorithm, let us consider a pathology that can arise. Figure 4.29 shows a simple network topology where link costs are equal to the load carried on the link, for example, 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. In this example, node z originates a unit of traffic destined for w , node x also originates a unit of traffic destined for w , and node y injects an amount of traffic equal to e , also destined for w . The initial routing is shown in Figure 4.29(a) with the link costs corresponding to the amount of traffic carried.

When the LS algorithm is next run, node y determines (based on the link costs shown in Figure 4.29(a)) that the clockwise path to w has a cost of 1, while the counterclockwise path to w (which it had been using) has a cost of $1 + e$. Hence y 's

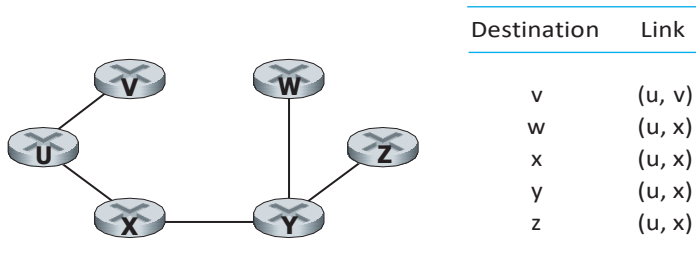


Figure 4.28 ♦ Least cost path and forwarding table for node u

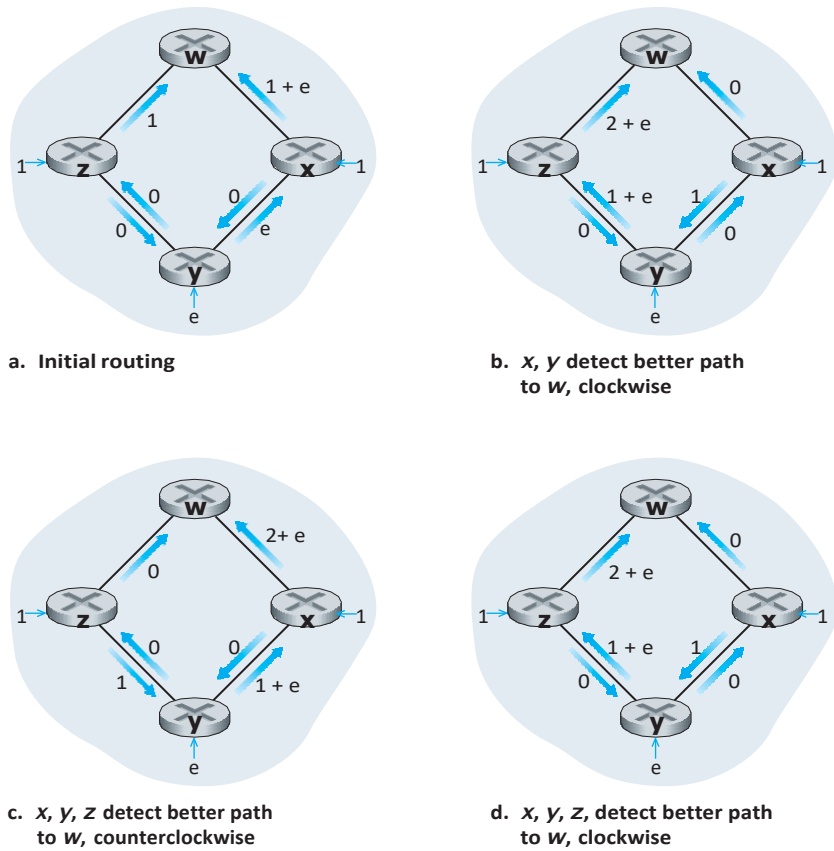


Figure 4.29 ♦ Oscillations with congestion-sensitive routing

least-cost path to w is now clockwise. Similarly, x determines that its new least-cost path to w is also clockwise, resulting in costs shown in Figure 4.29(b). When the LS algorithm is run next, nodes x , y , and z all detect a zero-cost path to w in the counterclockwise direction, and all route their traffic to the counterclockwise routes. The next time the LS algorithm is run, x , y , and z all then route their traffic to the clockwise routes.

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 [Floyd Synchronization 1994]. That is, even though they initially execute the algorithm with the same period but at different instants of time, the algorithm execution instance can eventually become, and remain, synchronized at the routers. One way to avoid such self-synchronization is for each router to randomize the time it sends out a link advertisement.

Having studied the LS algorithm, let's consider the other major routing algorithm that is used in practice today—the distance-vector routing algorithm.

4.2.2 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. (Interestingly, the algorithm is also self-terminating—there is no signal that the computation should stop; it just stops.) The algorithm is *asynchronous* in that it does not require all of the nodes to operate in lockstep with each other. We'll see that an asynchronous, iterative, self-terminating, distributed algorithm is much more interesting and fun than a centralized algorithm!

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 $d_x(y)$ 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,

$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}, \quad (4.1)$$

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 by 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 .

But for those who might be skeptical about the validity of the equation, let's check it for source node u and destination node z in Figure 4.27. The source node u

has three neighbors: nodes v , x , and w . By walking along various paths in the graph, it is easy to see that $d_v(z) = 5$, $d_x(z) = 3$, and $d_w(z) = 3$. Plugging these values into Equation 4.1, along with the costs $c(u,v) = 2$, $c(u,x) = 1$, and $c(u,w) = 5$, gives $d_u(z) = \min\{2 + 5, 5 + 3, 1 + 3\} = 4$, which is obviously true and which is exactly what the Dijkstra algorithm gave us for the same network. This quick verification should help relieve any skepticism you may have.

The Bellman-Ford equation is not just an intellectual curiosity. It actually has significant practical importance. In particular, the solution to the Bellman-Ford equation provides the entries in node x 's forwarding table. To see this, let v^* be any neighboring node that achieves the minimum in Equation 4.1. Then, if node x wants to send a packet to node y along a least-cost path, it should first forward the packet to node v^* . Thus, node x 's forwarding table would specify node v^* as the next-hop router for the ultimate destination y . Another important practical contribution of the Bellman-Ford equation is that it suggests the form of the neighbor-to-neighbor communication that will take place in the DV algorithm.

The basic idea is as follows. Each node x begins with $D_x(y)$, an estimate of the cost of the least-cost path from itself to node y , for all nodes in N . Let $\mathbf{D}_x = [D_x(y): y \text{ in } N]$ be node x 's distance vector, which is the vector of cost estimates from x to all other nodes, y , in N . 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, $\mathbf{D}_x = [D_x(y): y \text{ in } N]$, containing x 's estimate of its cost to all destinations, y , in N
- The distance vectors of each of its neighbors, that is, $\mathbf{D}_v = [D_v(y): y \text{ in } N]$ for each neighbor v of x

In the distributed, asynchronous algorithm, from time to time, each node sends a copy of its distance vector to each of its neighbors. When a node x receives a new distance vector from any of its neighbors v , it saves v 's distance vector, and then uses the Bellman-Ford equation to update its own distance vector as follows:

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

If node x 's distance vector has changed as a result of this update step, node x will then send its updated distance vector to each of its neighbors, which can in turn update their own distance vectors. Miraculously enough, as long as all the nodes continue to exchange their distance vectors in an asynchronous fashion, each cost estimate $D_x(y)$ converges to $d_x(y)$, the actual cost of the least-cost path from node x to node y [Bertsekas 1991]!

Distance-Vector (DV) Algorithm

At each node, x :

```

1 Initialization:
2   for all destinations  $y$  in  $N$ :
3      $D_x(y) = c(x,y)$  /* if  $y$  is not a neighbor then  $c(x,y) = \infty$  */
4   for each neighbor  $w$ 
5      $D_w(y) = ?$  for all destinations  $y$  in  $N$ 
6   for each neighbor  $w$ 
7     send distance vector  $D_x = [D_x(y): y \text{ in } N]$  to  $w$ 
8
9 loop
10  wait (until I see a link cost change to some neighbor  $w$  or
11       until I receive a distance vector from some neighbor  $w$ )
12
13  for each  $y$  in  $N$ :
14     $D_x(y) = \min_v \{c(x,v) + D_v(y)\}$ 
15
16  if  $D_x(y)$  changed for any destination  $y$ 
17    send distance vector  $D_x = [D_x(y): y \text{ in } N]$  to all neighbors
18
19 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. But to update its own forwarding table for a given destination y , what node x really needs to know is not the shortest-path distance to y but instead the neighboring node $v^*(y)$ that is the next-hop router along the shortest path to y . As you might expect, the next-hop router $v^*(y)$ is the neighbor v that achieves the minimum in Line 14 of the DV algorithm. (If there are multiple neighbors v that achieve the minimum, then $v^*(y)$ can be any of the minimizing neighbors.) Thus, in Lines 13–14, for each destination y , node x also determines $v^*(y)$ and updates its forwarding table for destination y .

Recall that the LS algorithm is a global 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. Indeed, the only information a node will have is the costs of the links to its directly attached neighbors and information it receives from these neighbors. Each node waits for an update from any neighbor (Lines 10–11), calculates its new distance vector when receiving an update (Line 14), and distributes its new distance

vector to its neighbors (Lines 16–17). DV-like algorithms are used in many routing protocols in practice, including the Internet’s RIP and BGP, ISO IDR, Novell IPX, and the original ARPAnet.

Figure 4.30 illustrates the operation of the DV algorithm for the simple three-node network shown at the top of the figure. The operation of the algorithm is illustrated in a synchronous manner, where all nodes simultaneously receive distance vectors from their neighbors, compute their new distance vectors, and inform their neighbors if their distance vectors have changed.

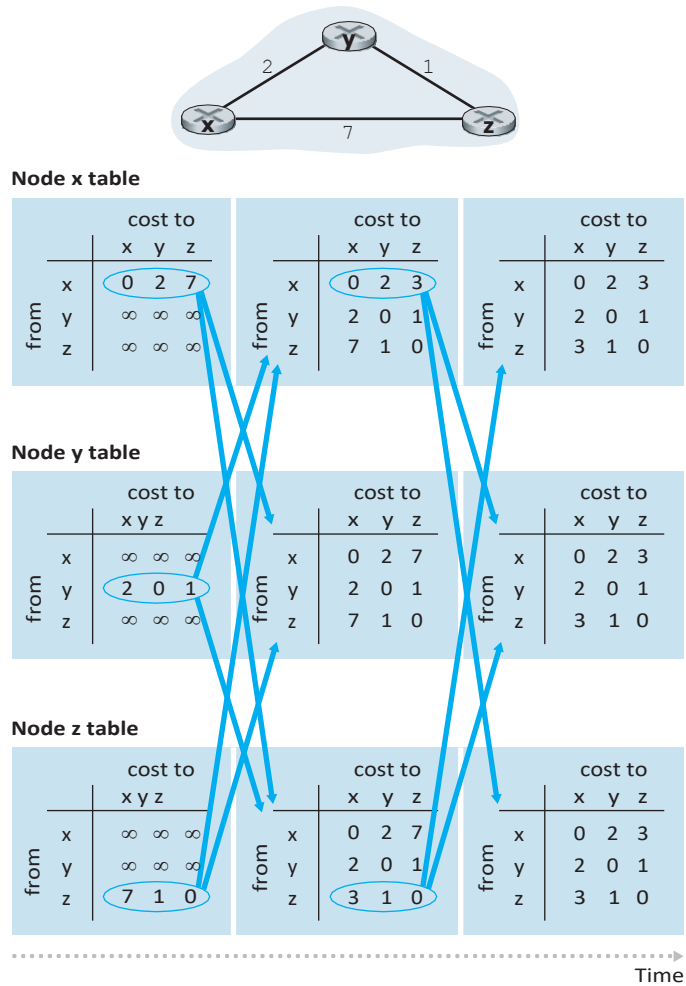


Figure 4.30 ♦ Distance-vector (DV) algorithm

should convince yourself that the algorithm operates correctly in an asynchronous manner as well, with node computations and update generation/reception occurring at any time.

The leftmost column of the figure displays three initial **routing tables** for each of the three nodes. For example, the table in the upper-left corner is node x 's initial routing table. Within a specific routing table, each row is a distance vector—specifically, each node's routing table includes its own distance vector and that of each of its neighbors. Thus, the first row in node x 's initial routing table is $\mathbf{D}_x = [D_x(x), D_x(y), D_x(z)] = [0, 2, 7]$. The second and third rows in this table are the most recently received distance vectors from nodes y and z , respectively. Because at initialization node x has not received anything from node y or z , the entries in the second and third rows are initialized to infinity.

After initialization, each node sends its distance vector to each of its two neighbors. This is illustrated in Figure 4.30 by the arrows from the first column of tables to the second column of tables. For example, node x sends its distance vector $\mathbf{D}_x = [0, 2, 7]$ to both nodes y and z . After receiving the updates, each node recomputes its own distance vector. For example, node x computes

$$D_x(x) = 0$$

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min\{2 + 0, 7 + 1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} = \min\{2 + 1, 7 + 0\} = 3$$

The second column therefore displays, for each node, the node's new distance vector along with distance vectors just received from its neighbors. Note, for example, that node x 's estimate for the least cost to node z , $D_x(z)$, has changed from 7 to 3. Also note that for node x , neighboring node y achieves the minimum in line 14 of the DV algorithm; thus at this stage of the algorithm, we have at node x that $v^*(y) = y$ and $v^*(z) = y$.

After the nodes recompute their distance vectors, they again send their updated distance vectors to their neighbors (if there has been a change). This is illustrated in Figure 4.30 by the arrows from the second column of tables to the third column of tables. Note that only nodes x and z send updates: node y 's distance vector didn't change so node y doesn't send an update. After receiving the updates, the nodes then recompute their distance vectors and update their routing tables, which are shown in the third column.

The process of receiving updated distance vectors from neighbors, recomputing routing table entries, and informing neighbors of changed costs of the least-cost path to a destination continues until no update messages are sent. At this point, since no update messages are sent, no further routing table calculations will occur and the algorithm will enter a quiescent state; that is, all nodes will be performing the wait in Lines 10–11 of the DV algorithm. The algorithm remains in the quiescent state until a link cost changes, as discussed next.

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

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. Figure 4.31(a) illustrates a scenario where the link cost from y to x changes from 4 to 1. We focus here only on y ’ and z ’s distance table entries to destination x . 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 4.31(b).

1. Before the link cost changes, $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$, 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

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

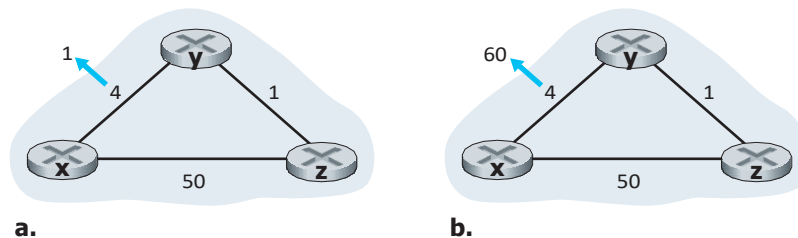


Figure 4.31 ♦ Changes in link cost

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**—in order to get to x , y routes through z , and z routes through y . A routing loop is like a black hole—a packet destined for x arriving at y or z as of t_1 will bounce back and forth between these two nodes forever (or until the forwarding tables are changed).

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. Sometime 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 $D_z(x) = \min\{50 + 0, 1 + 6\} = 7$. Since z 's least cost to x has increased, it then informs y of its new distance vector at t_2 .
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.

How long will the process continue? 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 . The result of the bad news about the increase in link cost has indeed traveled slowly! What would have happened if the link cost $c(y, x)$ had changed from 4 to 10,000 and the cost $c(z, x)$ had been 9,999? 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 $D_z(x) = \infty$ (even though z knows $D_z(x) = 5$ in truth). 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 (and lies about doing so).

Let's now see how poisoned reverse solves the particular looping problem we encountered before in Figure 4.31(b). As a result of the poisoned reverse, y 's distance table indicates $D_z(x) = \infty$. When the cost of the (x, y) link changes from 4 to 60 at time t_0 , y updates its table and continues to route directly to x , albeit at a higher cost of 60, and informs z of its new cost to x , that is, $D_y(x) = 60$. After receiving the

update at t_1 , z immediately shifts its route to x to be via the direct (z, x) link at a cost of 50. Since this is a new least-cost path to x , and since the path no longer passes through y , z now informs y that $D_z(x) = 50$ at t_2 . After receiving the update from z , y updates its distance table with $D_y(x) = 51$. Also, since z is now on y 's least-cost path to x , y poisons the reverse path from z to x by informing z at time t_3 that $D_y(x) = \infty$ (even though y knows that $D_y(x) = 51$ in truth).

Does poisoned reverse solve the general count-to-infinity problem? It does not. 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 towards 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 (that it knows about) in the network. In the LS algorithm, each node talks with *all* other nodes (via broadcast), but it tells them *only* the costs of its directly connected links. 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.* What can happen if a router fails, misbehaves, or is sabotaged? 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. (Indeed, in 1997, a malfunctioning router in a small ISP

provided national backbone routers with erroneous routing information. This caused other routers to flood the malfunctioning router with traffic and caused large portions of the Internet to become disconnected for up to several hours [Neumann 1997].) More generally, we note that, at each iteration, a node's calculation in DV is passed on to its neighbor and then indirectly to its neighbor's neighbor on the next iteration. In this sense, an incorrect node calculation can be diffused through the entire network under DV.

In the end, neither algorithm is an obvious winner over the other; indeed, both algorithms are used in the Internet.

Other Routing Algorithms

The LS and DV algorithms we have studied are not only widely used in practice, they are essentially the *only* routing algorithms used in practice today in the Internet. Nonetheless, many routing algorithms have been proposed by researchers over the past 30 years, ranging from the extremely simple to the very sophisticated and complex. A broad class of routing algorithms is based on viewing packet traffic as flows between sources and destinations in a network. In this approach, the routing problem can be formulated mathematically as a constrained optimization problem known as a network flow problem [Bertsekas 1991]. Yet another set of routing algorithms we mention here are those derived from the telephony world. These **circuit-switched routing algorithms** are of interest to packet-switched data networking in cases where per-link resources (for example, buffers, or a fraction of the link bandwidth) are to be reserved for each connection that is routed over the link. While the formulation of the routing problem might appear quite different from the least-cost routing formulation we have seen in this chapter, there are a number of similarities, at least as far as the path-finding algorithm (routing algorithm) is concerned. See [Ash 1998; Ross 1995; Girard 1990] for a detailed discussion of this research area.

4.2.3 Hierarchical Routing

In our study of LS and DV algorithms, we've viewed the network simply as a collection of interconnected routers. 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 homogeneous set of routers all executing the same routing algorithm is a bit simplistic for at least two important reasons:

- *Scale.* As the number of routers becomes large, the overhead involved in computing, storing, and communicating routing information (for example,

LS updates or least-cost path changes) becomes prohibitive. Today's public Internet consists of hundreds of millions of hosts. Storing routing information at each of these hosts would clearly require enormous amounts of memory. The overhead required to broadcast LS updates among all of the routers in the public Internet would leave no bandwidth left for sending data packets! 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 networks as large as the public Internet.

- *Administrative autonomy.* Although researchers tend to ignore issues such as a company's desire to run its routers as it pleases (for example, to run whatever routing algorithm it chooses) or to hide aspects of its network's internal organization from the outside, these are important considerations. Ideally, an organization should be able to run 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 typically under the same administrative control (e.g., operated by the same ISP or belonging to the same company network). Routers within the same AS all run the same routing algorithm (for example, an LS or DV algorithm) and have information about each other—exactly as was the case in our idealized model in the preceding section. The routing algorithm running within an autonomous system is called an **intra-autonomous system routing protocol**. It will be necessary, of course, to connect ASs to each other, and thus one or more of the routers in an AS will have the added task of being responsible for forwarding packets to destinations outside the AS; these routers are called **gateway routers**.

Figure 4.32 provides a simple example with three ASs: AS1, AS2, and AS3. In this figure, the heavy lines represent direct link connections between pairs of routers. The thinner lines hanging from the routers represent subnets that are directly connected to the routers. AS1 has four routers—1a, 1b, 1c, and 1d—which run the intra-AS routing protocol used within AS1. Thus, each of these four routers knows how to forward packets along the optimal path to any destination within AS1. Similarly, autonomous systems AS2 and AS3 each have three routers. Note that the intra-AS routing protocols running in AS1, AS2, and AS3 need not be the same. Also note that the routers 1b, 1c, 2a, and 3a are all gateway routers.

It should now be clear how the routers in an AS determine routing paths for source-destination pairs that are internal to the AS. But there is still a big missing piece to the end-to-end routing puzzle. How does a router, within some AS, know how to route a packet to a destination that is outside the AS? It's easy to answer this question if the AS has only one gateway router that connects to only one other AS. In this case, because the AS's intra-AS routing algorithm has determined the least-cost path from each internal router to the gateway router, each

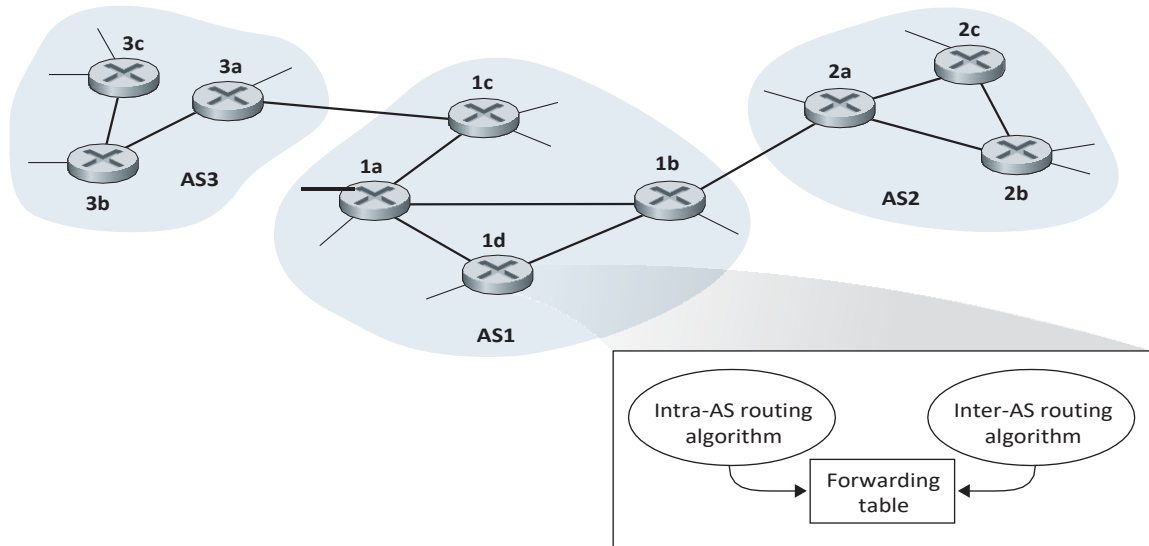


Figure 4.32 ♦ An example of interconnected autonomous systems

internal router knows how it should forward the packet. The gateway router, upon receiving the packet, forwards the packet on the one link that leads outside the AS. The AS on the other side of the link then takes over the responsibility of routing the packet to its ultimate destination. As an example, suppose router 2b in Figure 4.32 receives a packet whose destination is outside of AS2. Router 2b will then forward the packet to either router 2a or 2c, as specified by router 2b's forwarding table, which was configured by AS2's intra-AS routing protocol. The packet will eventually arrive to the gateway router 2a, which will forward the packet to 1b. Once the packet has left 2a, AS2's job is done with this one packet.

So the problem is easy when the source AS has only one link that leads outside the AS. But what if the source AS has two or more links (through two or more gateway routers) that lead outside the AS? Then the problem of knowing where to forward the packet becomes significantly more challenging. For example, consider a router in AS1 and suppose it receives a packet whose destination is outside the AS. The router should clearly forward the packet to one of its two gateway routers, 1b or 1c, but which one? To solve this problem, AS1 needs (1) to learn which destinations are reachable via AS2 and which destinations are reachable via AS3, and (2) to propagate this reachability information to all the routers within AS1, so that each router can configure its forwarding table to handle external-AS destinations. These

two tasks—obtaining reachability information from neighboring ASs and propagating the reachability information to all routers internal to the AS—are handled by the **inter-AS routing protocol**. Since the inter-AS routing protocol involves communication between two ASs, the two 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 BGP4, which is discussed in the next section. As shown in Figure 4.32, each router receives information from an intra-AS routing protocol and an inter-AS routing protocol, and uses the information from both protocols to configure its forwarding table.

As an example, consider a subnet x (identified by its CIDRized address), and suppose that AS1 learns from the inter-AS routing protocol that subnet x is reachable from AS3 but is *not* reachable from AS2. AS1 then propagates this information to all of its routers. When router 1d learns that subnet x is reachable from AS3, and hence from gateway 1c, it then determines, from the information provided by the intra-AS routing protocol, the router interface that is on the least-cost path from router 1d to gateway router 1c. Say this is interface I . The router 1d can then put the entry (x, I) into its forwarding table. (This example, and others presented in this section, gets the general ideas across but is a simplification of what really happens in the Internet. In the next section we'll provide a more detailed description, albeit more complicated, when we discuss BGP.)

Following up on the previous example, now suppose that AS2 and AS3 connect to other ASs, which are not shown in the diagram. Also suppose that AS1 learns from the inter-AS routing protocol that subnet x is reachable both from AS2, via gateway 1b, and from AS3, via gateway 1c. AS1 would then propagate this information to all its routers, including router 1d. In order to configure its forwarding table, router 1d would have to determine to which gateway router, 1b or 1c, it should direct packets that are destined for subnet x . One approach, which is often employed in practice, is to use **hot-potato routing**. In hot-potato routing, the AS gets rid of the packet (the hot potato) as quickly as possible (more precisely, as inexpensively as possible). This is done by having a router send the packet to the gateway router that has the smallest router-to-gateway cost among all gateways with a path to the destination. In the context of the current example, hot-potato routing, running in 1d, would use information from the intra-AS routing protocol to determine the path costs to 1b and 1c, and then choose the path with the least cost. Once this path is chosen, router 1d adds an entry for subnet x in its forwarding table. Figure 4.33 summarizes the actions taken at router 1d for adding the new entry for x to the forwarding table.

When an AS learns about a destination from a neighboring AS, the AS can advertise this routing information to some of its other neighboring ASs. For example, suppose AS1 learns from AS2 that subnet x is reachable via AS2. AS1 could then tell AS3 that x is reachable via AS1. In this manner, if AS3 needs to route a packet destined to x , AS3 would forward the packet to AS1, which would in turn forward the packet to AS2. As we'll see in our discussion of BGP, an AS has quite a bit of

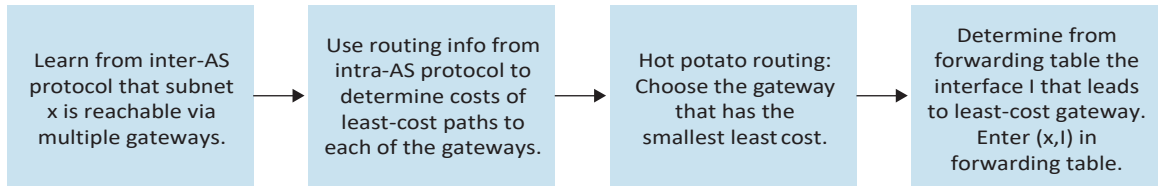


Figure 4.33 ♦ Steps in adding an outside-AS destination in a router’s forwarding table

flexibility in deciding which destinations it advertises to its neighboring ASs. This is a *policy* decision, typically depending more on economic issues than on technical issues.

Recall from Section 1.5 that the Internet consists of a hierarchy of interconnected ISPs. So what is the relationship between ISPs and ASs? You might think that the routers in an ISP, and the links that interconnect them, constitute a single AS. Although this is often the case, many ISPs partition their network into multiple ASs. For example, some tier-1 ISPs use one AS for their entire network; others break up their ISP into tens of interconnected ASs.

In summary, the problems of scale and administrative authority are solved by defining autonomous systems. Within an AS, all routers run the same intra-AS routing protocol. Among themselves, the ASs run the same inter-AS routing protocol. The problem of scale is solved because an intra-AS router need only know about routers within its AS. The problem of administrative authority is solved since an organization can run whatever intra-AS routing protocol it chooses; however, each pair of connected ASs needs to run the same inter-AS routing protocol to exchange reachability information.

In the following section, we’ll examine two intra-AS routing protocols (RIP and OSPF) and the inter-AS routing protocol (BGP) that are used in today’s Internet. These case studies will nicely round out our study of hierarchical routing.

4.6 Routing in the Internet

Having studied Internet addressing and the IP protocol, we now turn our attention to the Internet’s routing protocols; their job is to determine the path taken by a datagram between source and destination. We’ll see that the Internet’s routing protocols embody many of the principles we learned earlier in this chapter. The link-state and distance-vector approaches studied in Sections 4.5.1 and 4.5.2 and the notion of an autonomous system considered in Section 4.5.3 are all central to how routing is done in today’s Internet.

Recall from Section 4.5.3 that an autonomous system (AS) is a collection of routers under the same administrative and technical control, and that all run the same routing protocol among themselves. Each AS, in turn, typically contains multiple subnets (where we use the term subnet in the precise, addressing sense in Section 4.4.2).

4.6.1 Intra-AS Routing in the Internet: RIP

An intra-AS routing protocol is used to determine how routing is performed within an autonomous system (AS). Intra-AS routing protocols are also known as **interior gateway protocols**. Historically, two routing protocols have been used extensively for routing within an autonomous system in the Internet: the **Routing Information Protocol (RIP)** and **Open Shortest Path First (OSPF)**. A routing protocol closely related to OSPF is the **IS-IS** protocol [RFC 1142, Perlman 1999]. We first discuss RIP and then consider OSPF.

RIP was one of the earliest intra-AS Internet routing protocols and is still in widespread use today. It traces its origins and its name to the Xerox Network Systems (XNS) architecture. The widespread deployment of RIP was due in great part to its inclusion in 1982 in the Berkeley Software Distribution (BSD) version of UNIX supporting TCP/IP. RIP version 1 is defined in [RFC 1058], with a backward-compatible version 2 defined in [RFC 2453].

RIP is a distance-vector protocol that operates in a manner very close to the idealized DV protocol we examined in Section 4.5.2. The version of RIP specified in RFC 1058 uses hop count as a cost metric; that is, each link has a cost of 1. In the DV algorithm in Section 4.5.2, for simplicity, costs were defined between pairs of routers. In RIP (and also in OSPF), costs are actually from source router to a destination subnet. RIP uses the term *hop*, which is the number of subnets traversed along the shortest path from source router to destination subnet, including the destination subnet. Figure 4.34 illustrates an AS with six leaf subnets. The table in the figure indicates the number of hops from the source A to each of the leaf subnets.

The maximum cost of a path is limited to 15, thus limiting the use of RIP to autonomous systems that are fewer than 15 hops in diameter. Recall that in DV protocols, neighboring routers exchange distance vectors with each other. The distance vector for any one router is the current estimate of the shortest path distances from that router to the subnets in the AS. In RIP, routing updates are exchanged between neighbors approximately every 30 seconds using a **RIP response message**. The response message sent by a router or host contains a list of up to 25 destination subnets within the AS, as well as the sender's distance to each of those subnets. Response messages are also known as **RIP advertisements**.

Let's take a look at a simple example of how RIP advertisements work. Consider the portion of an AS shown in Figure 4.35. In this figure, lines connecting the routers denote subnets. Only selected routers (*A*, *B*, *C*, and *D*) and subnets (*w*, *x*, *y*,

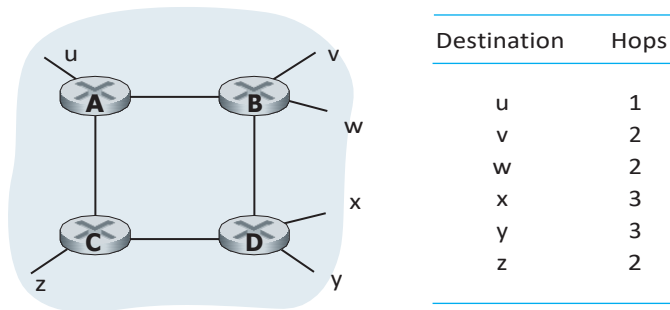


Figure 4.34 ♦ Number of hops from source router A to various subnets

and z) are labeled. Dotted lines indicate that the AS continues on; thus this autonomous system has many more routers and links than are shown.

Each router maintains a RIP table known as a **routing table**. A router's routing table includes both the router's distance vector and the router's forwarding table. Figure 4.36 shows the routing table for router D . Note that the routing table has three columns. The first column is for the destination subnet, the second column indicates the identity of the next router along the shortest path to the destination subnet, and the third column indicates the number of hops (that is, the number of subnets that have to be traversed, including the destination subnet) to get to the destination subnet along the shortest path. For this example, the table indicates that to send a datagram from router D to destination subnet w , the datagram should first be forwarded to neighboring router A ; the table also indicates that destination subnet w is two hops away along the shortest path. Similarly, the table indicates that subnet z is seven hops away via router B . In principle, a routing table will have one row for each subnet in the AS, although RIP version 2 allows subnet entries to be aggregated using route aggregation techniques similar to those we examined in

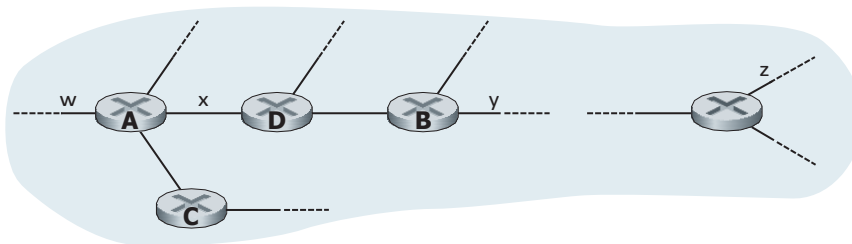


Figure 4.35 ♦ A portion of an autonomous system

Destination Subnet	Next Router	Number of Hops to Destination
w	A	2
y	B	2
z	B	7
x	—	1
...

Figure 4.36 ♦ Routing table in router D before receiving advertisement from router A

Section 4.4. The table in Figure 4.36, and the subsequent tables to come, are only partially complete.

Now suppose that 30 seconds later, router *D* receives from router *A* the advertisement shown in Figure 4.37. Note that this advertisement is nothing other than the routing table information from router *A*! This information indicates, in particular, that subnet *z* is only four hops away from router *A*. Router *D*, upon receiving this advertisement, merges the advertisement (Figure 4.37) with the old routing table (Figure 4.36). In particular, router *D* learns that there is now a path through router *A* to subnet *z* that is shorter than the path through router *B*. Thus, router *D* updates its routing table to account for the shorter shortest path, as shown in Figure 4.38. How is it, you might ask, that the shortest path to subnet *z* has become shorter? Possibly, the decentralized distance-vector algorithm is still in the process of converging (see Section 4.5.2), or perhaps new links and/or routers were added to the AS, thus changing the shortest paths in the AS.

Let's next consider a few of the implementation aspects of RIP. Recall that RIP routers exchange advertisements approximately every 30 seconds. If a router does not hear from its neighbor at least once every 180 seconds, that neighbor is considered to be no longer reachable; that is, either the neighbor has died or the

Destination Subnet	Next Router	Number of Hops to Destination
z	C	4
w	—	1
x	—	1
...

Figure 4.37 ♦ Advertisement from router A

Destination Subnet	Next Router	Number of Hops to Destination
w	A	2
y	B	2
z	A	5
....

Figure 4.38 ♦ Routing table in router D after receiving advertisement from router A

connecting link has gone down. When this happens, RIP modifies the local routing table and then propagates this information by sending advertisements to its neighboring routers (the ones that are still reachable). A router can also request information about its neighbor's cost to a given destination using RIP's request message. Routers send RIP request and response messages to each other over UDP using port number 520. The UDP segment is carried between routers in a standard IP datagram. The fact that RIP uses a transport-layer protocol (UDP) on top of a network-layer protocol (IP) to implement network-layer functionality (a routing algorithm) may seem rather convoluted (it is!). Looking a little deeper at how RIP is implemented will clear this up.

Figure 4.39 sketches how RIP is typically implemented in a UNIX system, for example, a UNIX workstation serving as a router. A process called *routed* (pronounced “route dee”) executes RIP, that is, maintains routing information and exchanges messages with *routed* processes running in neighboring routers. Because RIP is implemented as an application-layer process (albeit a very special one that is able to

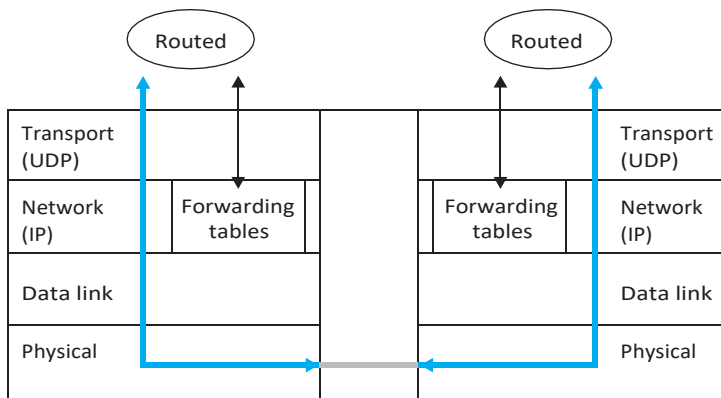


Figure 4.39 ♦ Implementation of RIP as the routed daemon

manipulate the routing tables within the UNIX kernel), it can send and receive messages over a standard socket and use a standard transport protocol. As shown, RIP is implemented as an application-layer protocol (see Chapter 2) running over UDP. If you're interested in looking at an implementation of RIP (or the OSPF and BGP protocols that we will study shortly), see [Quagga 2012].

4.6.2 Intra-AS Routing in the Internet: OSPF

Like RIP, OSPF routing is widely used for intra-AS routing in the Internet. OSPF and its closely related cousin, IS-IS, are typically deployed in upper-tier ISPs whereas RIP is deployed in lower-tier ISPs and enterprise networks. The Open in OSPF indicates that the routing protocol specification is publicly available (for example, as opposed to Cisco's EIGRP protocol). The most recent version of OSPF, version 2, is defined in RFC 2328, a public document.

OSPF was conceived as the successor to RIP and as such has a number of advanced features. At its heart, however, OSPF is a link-state protocol that uses flooding of link-state information and a Dijkstra least-cost path algorithm. With OSPF, a router constructs a complete topological map (that is, a graph) of the entire autonomous system. The 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 (see Principles and Practice: Setting OSPF Weights). 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 (that is the job of the network administrator), 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 routers. A router broadcasts link-state information whenever there is a change in a link's state (for example, a change in cost or a change in up/down status). It also broadcasts a link's state periodically (at least once every 30 minutes), even if the link's state has not changed. RFC 2328 notes that "this periodic updating of link state advertisements adds robustness to the link state algorithm." 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 (for example, link-state updates) can be authenticated. With authentication, only trusted routers can participate

in the OSPF protocol within an AS, thus preventing malicious intruders (or networking students taking their newfound knowledge out for a joyride) 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 (see Chapter 8 for a discussion on MD5 and authentication in general). With simple authentication, the same password is configured 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. (See the discussion of message authentication codes in Chapter 7.) Then the router includes the resulting hash value in the OSPF packet. The receiving router, using the preconfigured secret key, will compute an MD5 hash of the packet and compare it with the hash value that the packet carries, thus verifying the packet's authenticity. Sequence numbers are also used with MD5 authentication to protect against replay attacks.

- *Multiple same-cost paths.* When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used (that is, a single path need not be chosen for carrying all traffic when multiple equal-cost paths exist).
- *Integrated support for unicast and multicast routing.* Multicast OSPF (MOSPF) [RFC 1584] provides simple extensions to OSPF to provide for multicast routing (a topic we cover in more depth in Section 4.7.2). 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 routing domain.* Perhaps the most significant advance in OSPF is the ability to structure an autonomous system hierarchically. Section 4.5.3 has already looked at the many advantages of hierarchical routing structures. We cover the implementation of OSPF hierarchical routing in the remainder of this section.

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 nonborder 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.



PRINCIPLES IN PRACTICE

SETTING OSPF LINK WEIGHTS

Our discussion of link-state routing has implicitly assumed that link weights are set, a routing algorithm such as OSPF is run, and traffic flows according to the routing tables computed by the LS algorithm. In terms of cause and effect, the link weights are given (i.e., they come first) and result (via Dijkstra's algorithm) in routing paths that minimize overall cost. In this viewpoint, link weights reflect the cost of using a link (e.g., if link weights are inversely proportional to capacity, then the use of high-capacity links would have smaller weights and thus be more attractive from a routing standpoint) and Dijkstra's algorithm serves to minimize overall cost.

In practice, the cause and effect relationship between link weights and routing paths may be reversed, with network operators configuring link weights in order to obtain routing paths that achieve certain traffic engineering goals [Fortz 2000, Fortz 2002]. For example, suppose a network operator has an estimate of traffic flow entering the network at each ingress point and destined for each egress point. The operator may then want to put in place a specific routing of ingress-to-egress flows that minimizes the maximum utilization over all of the network's links. But with a routing algorithm such as OSPF, the operator's main "knobs" for tuning the routing of flows through the network are the link weights. Thus, in order to achieve the goal of minimizing the maximum link utilization, the operator must find the set of link weights that achieves this goal. This is a reversal of the cause and effect relationship—the desired routing of flows is known, and the OSPF link weights must be found such that the OSPF routing algorithm results in this desired routing of flows.

OSPF is a relatively complex protocol, and our coverage here has been necessarily brief; [Huitema 1998; Moy 1998; RFC 2328] provide additional details.

4.6.3 Inter-AS Routing: BGP

We just learned how ISPs use RIP and OSPF to determine optimal paths for source-destination pairs that are internal to the same AS. Let's now examine how paths are determined for source-destination pairs that span multiple ASs. The **Border Gateway Protocol** version 4, specified in RFC 4271 (see also [RFC 4274]), is the *de facto* standard inter-AS routing protocol in today's Internet. It is commonly referred to as BGP4 or simply as **BGP**. As an inter-AS routing protocol (see Section 4.5.3), BGP provides each AS a means to

1. Obtain subnet reachability information from neighboring ASs.
2. Propagate the reachability information to all routers internal to the AS.
3. Determine "good" routes to subnets based on the reachability information and on AS policy.

Most importantly, 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 ASs in the Internet know about the subnet and how to get there. If it weren’t for BGP, each subnet would be isolated—alone and unknown by the rest of the Internet.

BGP Basics

BGP is extremely complex; entire books have been devoted to the subject and many issues are still not well understood [Yannuzzi 2005]. Furthermore, even after having read the books and RFCs, you may find it difficult to fully master BGP without having practiced BGP for many months (if not years) as a designer or administrator of an upper-tier ISP. Nevertheless, because BGP is an absolutely critical protocol for the Internet—in essence, it is the protocol that glues the whole thing together—we need to acquire at least a rudimentary understanding of how it works. We begin by describing how BGP might work in the context of the simple example network we studied earlier in Figure 4.32. In this description, we build on our discussion of hierarchical routing in Section 4.5.3; we encourage you to review that material.

In BGP, pairs of routers exchange routing information over semipermanent TCP connections using port 179. The semi-permanent TCP connections for the network in Figure 4.32 are shown in Figure 4.40. There is typically one such BGP TCP connection for each link that directly connects two routers in two different ASs; thus, in Figure 4.40, there is a TCP connection between gateway routers 3a and 1c and another TCP connection between gateway routers 1b and 2a. There are also semipermanent BGP TCP connections between routers within an AS. In particular, Figure 4.40 displays a common configuration of one TCP connection for each pair of routers internal to an AS, creating a mesh of TCP connections within each AS. For each TCP connection, the two routers at the end of the connection are called **BGP peers**, and the TCP connection along with all the BGP messages sent over the

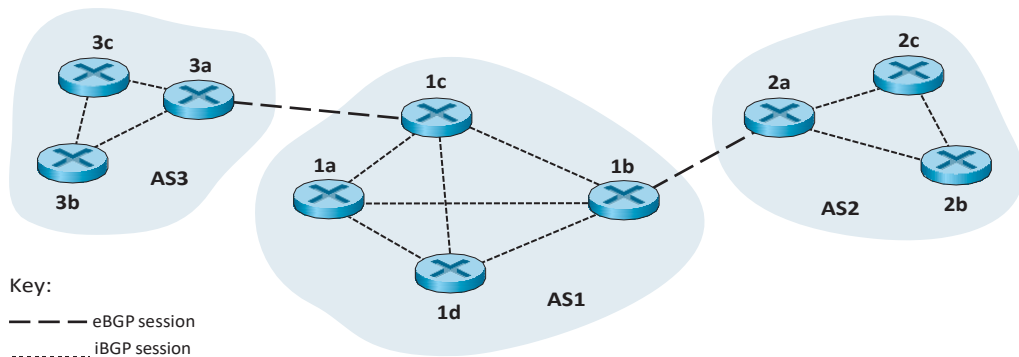


Figure 4.40 ♦ eBGP and iBGP sessions





PRINCIPLES IN PRACTICE

OBTAINING INTERNET PRESENCE: PUTTING THE PUZZLE TOGETHER

Suppose you have just created a small that has a number of servers, including a public Web server that describes your company's products and services, a mail server from which your employees obtain their email messages, and a DNS server. Naturally, you would like the entire world to be able to surf your Web site in order to learn about your exciting products and services. Moreover, you would like your employees to be able to send and receive email to potential customers throughout the world.

To meet these goals, you first need to obtain Internet connectivity, which is done by contracting with, and connecting to, a local ISP. Your company will have a gateway router, which will be connected to a router in your local ISP. This connection might be a DSL connection through the existing telephone infrastructure, a leased line to the ISP's router, or one of the many other access solutions described in Chapter 1. Your local ISP will also provide you with an IP address range, e.g., a /24 address range consisting of 256 addresses. Once you have your physical connectivity and your IP address range, you will assign one of the IP addresses (in your address range) to your Web server, one to your mail server, one to your DNS server, one to your gateway router, and other IP addresses to other servers and networking devices in your company's network.

In addition to contracting with an ISP, you will also need to contract with an Internet registrar to obtain a domain name for your company, as described in Chapter 2. For example, if your company's name is, say, Xanadu Inc., you will naturally try to obtain the domain name `xanadu.com`. Your company must also obtain presence in the DNS system. Specifically, because outsiders will want to contact your DNS server to obtain the IP addresses of your servers, you will also need to provide your registrar with the IP address of your DNS server. Your registrar will then put an entry for your DNS server (domain name and corresponding IP address) in the `.com` top-level-domain servers, as described in Chapter 2. After this step is completed, any user who knows your domain name (e.g., `xanadu.com`) will be able to obtain the IP address of your DNS server via the DNS system.

So that people can discover the IP addresses of your Web server, in your DNS server you will need to include entries that map the host name of your Web server (e.g., `www.xanadu.com`) to its IP address. You will want to have similar entries for other publicly available servers in your company, including your mail server. In this manner, if Alice wants to browse your Web server, the DNS system will contact your DNS server, find the IP address of your Web server, and give it to Alice. Alice can then establish a TCP connection directly with your Web server.

However, there still remains one other necessary and crucial step to allow outsiders from around the world access your Web server. Consider what happens when Alice, who knows the IP address of your Web server, sends an IP datagram (e.g., a TCP SYN segment) to that IP address. This datagram will be routed through the Internet, visiting a series of routers in many different ASes, and eventually reach your Web server. When

any one of the routers receives the datagram, it is going to look for an entry in its forwarding table to determine on which outgoing port it should forward the datagram. Therefore, each of the routers needs to know about the existence of your company's /24 prefix (or some aggregate entry). How does a router become aware of your company's prefix? As we have just seen, it becomes aware of it from BGP! Specifically, when your company contracts with a local ISP and gets assigned a prefix (i.e., an address range), your local ISP will use BGP to advertise this prefix to the ISPs to which it connects. Those ISPs will then, in turn, use BGP to propagate the advertisement. Eventually, all Internet routers will know about your prefix (or about some aggregate that includes your prefix) and thus be able to appropriately forward datagrams destined to your Web and mail servers.

connection is called a **BGP session**. Furthermore, a BGP session that spans two ASs is called an **external BGP (eBGP) session**, and a BGP session between routers in the same AS is called an **internal BGP (iBGP) session**. In Figure 4.40, the eBGP sessions are shown with the long dashes; the iBGP sessions are shown with the short dashes. Note that BGP session lines in Figure 4.40 do not always correspond to the physical links in Figure 4.32.

BGP allows each AS to learn which destinations are reachable via its neighboring ASs. In BGP, destinations are not hosts but instead are CIDRized **prefixes**, with each prefix representing a subnet or a collection of subnets. Thus, for example, suppose there are four subnets attached to AS2: 138.16.64/24, 138.16.65/24, 138.16.66/24, and 138.16.67/24. Then AS2 could aggregate the prefixes for these four subnets and use BGP to advertise the single prefix to 138.16.64/22 to AS1. As another example, suppose that only the first three of those four subnets are in AS2 and the fourth subnet, 138.16.67/24, is in AS3. Then, as described in the Principles and Practice in Section 4.4.2, because routers use longest-prefix matching for forwarding datagrams, AS3 could advertise to AS1 the more specific prefix 138.16.67/24 and AS2 could *still* advertise to AS1 the aggregated prefix 138.16.64/22.

Let's now examine how BGP would distribute prefix reachability information over the BGP sessions shown in Figure 4.40. As you might expect, using the eBGP session between the gateway routers 3a and 1c, AS3 sends AS1 the list of prefixes that are reachable from AS3; and AS1 sends AS3 the list of prefixes that are reachable from AS1. Similarly, AS1 and AS2 exchange prefix reachability information through their gateway routers 1b and 2a. Also as you may expect, when a gateway router (in any AS) receives eBGP-learned prefixes, the gateway router uses its iBGP sessions to distribute the prefixes to the other routers in the AS. Thus, all the routers in AS1 learn about AS3 prefixes, including the gateway router 1b. The gateway router 1b (in AS1) can therefore re-advertise AS3's prefixes to AS2. When a router (gateway or not) learns about a new prefix, it creates an entry for the prefix in its forwarding table, as described in Section 4.5.3.

Path Attributes and BGP Routes

Having now a preliminary understanding of BGP, let's get a little deeper into it (while still brushing some of the less important details under the rug!). In BGP, an autonomous system is identified by its globally unique **autonomous system number (ASN)** [RFC 1930]. (Technically, not every AS has an ASN. In particular, a so-called stub AS that carries only traffic for which it is a source or destination will not typically have an ASN; we ignore this technicality in our discussion in order to better see the forest for the trees.) AS numbers, like IP addresses, are assigned by ICANN regional registries [ICANN 2012].

When a router advertises a prefix across a BGP session, it includes with the prefix a number of **BGP attributes**. In BGP jargon, a prefix along with its attributes is called a **route**. Thus, BGP peers advertise routes to each other. Two of the more important attributes are AS-PATH and NEXT-HOP:

- **AS-PATH.** This attribute contains the ASs through which the advertisement for the prefix has passed. When a prefix is passed into an AS, the AS adds its ASN to the AS-PATH attribute. For example, consider Figure 4.40 and suppose that prefix 138.16.64/24 is first advertised from AS2 to AS1; if AS1 then advertises the prefix to AS3, AS-PATH would be AS2 AS1. Routers use the AS-PATH attribute to detect and prevent looping advertisements; specifically, if a router sees that its AS is contained in the path list, it will reject the advertisement. As we'll soon discuss, routers also use the AS-PATH attribute in choosing among multiple paths to the same prefix.
- 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 router interface that begins the AS-PATH.* To gain insight into this attribute, let's again refer to Figure 4.40. Consider what happens when the gateway router 3a in AS3 advertises a route to gateway router 1c in AS1 using eBGP. The route includes the advertised prefix, which we'll call x , and an AS-PATH to the prefix. This advertisement also includes the NEXT-HOP, which is the IP address of the router 3a interface that leads to 1c. (Recall that a router has multiple IP addresses, one for each of its interfaces.) Now consider what happens when router 1d learns about this route from iBGP. After learning about this route to x , router 1d may want to forward packets to x along the route, that is, router 1d may want to include the entry (x, l) in its forwarding table, where l is its interface that begins the least-cost path from 1d towards the gateway router 1c. To determine l , 1d provides the IP address in the NEXT-HOP attribute to its intra-AS routing module. Note that the intra-AS routing algorithm has determined the least-cost path to all subnets attached to the routers in AS1, including to the subnet for the link between 1c and 3a. From this least-cost path from 1d to the 1c-3a subnet, 1d determines its router interface l that begins this path and then adds the entry (x, l) to its forwarding table. Whew! In summary, the NEXT-HOP attribute is used by routers to properly configure their forwarding tables.
- Figure 4.41 illustrates another situation where the NEXT-HOP is needed. In this figure, AS1 and AS2 are connected by two peering links. A router in AS1 could learn

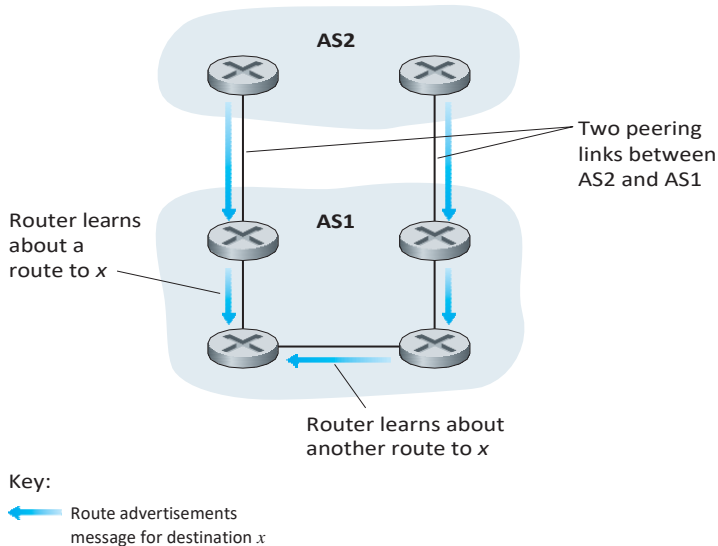


Figure 4.41 ♦ NEXT-HOP attributes in advertisements are used to determine which peering link to use

about two different routes to the same prefix x . These two routes could have the same AS-PATH to x , but could have different NEXT-HOP values corresponding to the different peering links. Using the NEXT-HOP values and the intra-AS routing algorithm, the router can determine the cost of the path to each peering link, and then apply hot-potato routing (see Section 4.5.3) to determine the appropriate interface.

BGP also includes attributes that allow routers to assign preference metrics to the routes, and an attribute that indicates how the prefix was inserted into BGP at the origin AS. For a full discussion of route attributes, see [Griffin 2012; Stewart 1999; Halabi 2000; Feamster 2004; RFC 4271].

When a gateway router receives a route advertisement, it uses its **import policy** to decide whether to accept or filter the route and whether to set certain attributes such as the router preference metrics. The import policy may filter a route because the AS may not want to send traffic over one of the ASs in the route's AS-PATH. The gateway router may also filter a route because it already knows of a preferable route to the same prefix.

BGP Route Selection

As described earlier in this section, BGP uses eBGP and iBGP to distribute routes to all the routers within ASs. From this distribution, a router may learn about more than one route to any one prefix, in which case the router must select one of the

possible routes. The input into this route selection process is the set of all routes that have been learned and accepted by the router. If there are two or more routes to the same prefix, then BGP sequentially invokes the following elimination rules until one route remains:

- Routes are assigned a local preference value as one of their attributes. The local preference of a route could have been set by the router or could have been learned by another router in the same AS. This is a policy decision that is left up to the AS's network administrator. (We will shortly discuss BGP policy issues in some detail.) The routes with the highest local preference values are selected.
- From the remaining routes (all with the same 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.
- From the remaining routes (all with the same local preference value and the same AS-PATH length), the route with the closest NEXT-HOP router is selected. Here, closest means the router for which the cost of the least-cost path, determined by the intra-AS algorithm, is the smallest. As discussed in Section 4.5.3, this process is called hot-potato routing.
- If more than one route still remains, the router uses BGP identifiers to select the route; see [Stewart 1999].

The elimination rules are even more complicated than described above. To avoid nightmares about BGP, it's best to learn about BGP selection rules in small doses!



PRINCIPLES IN PRACTICE

PUTTING IT ALL TOGETHER: HOW DOES AN ENTRY GET INTO A ROUTER'S FORWARDING TABLE?

Recall that an entry in a router's forwarding table consists of a prefix (e.g., 138.16.64/22) and a corresponding router output port (e.g., port 7). When a packet arrives to the router, the packet's destination IP address is compared with the prefixes in the forwarding table to find the one with the longest prefix match. The packet is then forwarded (within the router) to the router port associated with that prefix. Let's now summarize how a routing entry (prefix and associated port) gets entered into a forwarding table. This simple exercise will tie together a lot of what we just learned about routing and forwarding. To make things interesting, let's assume that the prefix is a "foreign prefix," that is, it does not belong to the router's AS but to some other AS.

In order for a prefix to get entered into the router's forwarding table, the router has to first become aware of the prefix (corresponding to a subnet or an aggregation of subnets). As we have just learned, the router becomes aware of the prefix via a BGP route

advertisement. Such an advertisement may be sent to it over an eBGP session (from a router in another AS) or over an iBGP session (from a router in the same AS).

After the router becomes aware of the prefix, it needs to determine the appropriate output port to which datagrams destined to that prefix will be forwarded, before it can enter that prefix in its forwarding table. If the router receives more than one route advertisement for this prefix, the router uses the BGP route selection process, as described earlier in this subsection, to find the “best” route for the prefix. Suppose such a best route has been selected. As described earlier, the selected route includes a NEXT-HOP attribute, which is the IP address of the first router outside the router’s AS along this best route. As described above, the router then uses its intra-AS routing protocol (typically OSPF) to determine the shortest path to the NEXT-HOP router. The router finally determines the port number to associate with the prefix by identifying the first link along that shortest path. The router can then (finally!) enter the prefix-port pair into its forwarding table! The forwarding table computed by the routing processor (see Figure 4.6) is then pushed to the router’s input port line cards.

Routing Policy

Let’s illustrate some of the basic concepts of BGP routing policy with a simple example. Figure 4.42 shows six interconnected autonomous systems: A, B, C, W, X, and Y. It is important to note that A, B, C, W, X, and Y are ASs, not routers. Let’s assume that autonomous systems W, X, and Y are stub networks and that A, B, and C are backbone provider networks. We’ll also assume that A, B, and C, all peer with each other, and provide full BGP information to their customer networks. All traffic entering a **stub network** must be destined for that network, and all traffic leaving a stub network must have originated in that network. W and Y are clearly stub networks. X is a **multi-homed stub network**, since it is connected to the rest of the network via two different providers (a scenario that is becoming increasingly common in practice). However, like W and Y, X itself must be the source/destination of all traffic leaving/entering X. But how will this stub network behavior be implemented and enforced? How will X be prevented from forwarding traffic between B and C? This can easily be

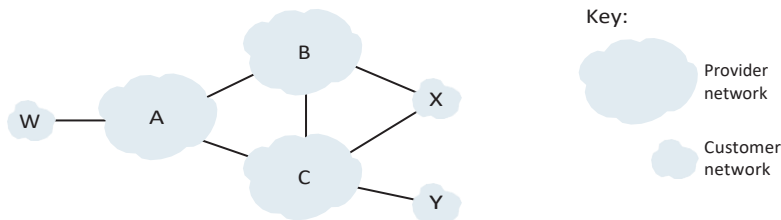


Figure 4.42 ♦ A simple BGP scenario



PRINCIPLES IN PRACTICE

WHY ARE THERE DIFFERENT INTER-AS AND INTRA-AS ROUTING PROTOCOLS?

Having now studied the details of specific inter-AS and intra-AS routing protocols deployed in today's Internet, let's conclude by considering perhaps the most fundamental question we could ask about these protocols in the first place (hopefully, you have been wondering this all along, and have not lost the forest for the trees!): Why are different inter-AS and intra-AS routing protocols used?

The answer to this question gets at the heart of the differences between the goals of routing within an AS and among ASs:

- **Policy.** Among ASs, policy issues dominate. It may well be important that traffic originating in a given AS not be able to pass through another specific AS. Similarly, a given AS may well want to control what transit traffic it carries between other ASs. We have seen that BGP carries path attributes and provides for controlled distribution of routing information so that such policy-based routing decisions can be made. Within an AS, everything is nominally under the same administrative control, and thus policy issues play a much less important role in choosing routes within the AS.
- **Scale.** The ability of a routing algorithm and its data structures to scale to handle routing to/among large numbers of networks is a critical issue in inter-AS routing. Within an AS, scalability is less of a concern. For one thing, if a single administrative domain becomes too large, it is always possible to divide it into two ASs and perform inter-AS routing between the two new ASs. (Recall that OSPF allows such a hierarchy to be built by splitting an AS into areas.)
- **Performance.** Because inter-AS routing is so policy oriented, the quality (for example, performance) of the routes used is often of secondary concern (that is, a longer or more costly route that satisfies certain policy criteria may well be taken over a route that is shorter but does not meet that criteria). Indeed, we saw that among ASs, there is not even the notion of cost (other than AS hop count) associated with routes. Within a single AS, however, such policy concerns are of less importance, allowing routing to focus more on the level of performance realized on a route.

accomplished by controlling the manner in which BGP routes are advertised. In particular, X will function as a stub network if it advertises (to its neighbors B and C) that it has no paths to any other destinations except itself. That is, even though X may know of a path, say XCY, that reaches network Y, it will *not* advertise this path to B. Since B is unaware that X has a path to Y, B would never forward traffic destined to Y (or C) via X. This simple example illustrates how a selective route advertisement policy can be used to implement customer/provider routing relationships.

Let's next focus on a provider network, say AS B. Suppose that B has learned (from A) that A has a path AW to W. B can thus install the route BAW into its routing information base. Clearly, B also wants to advertise the path BAW to its customer, X, so that X knows that it can route to W via B. But should B advertise the path BAW to C? If it does so, then C could route traffic to W via CBAW. If A, B, and C are all backbone providers, then B might rightly feel that it should not have to shoulder the burden (and cost!) of carrying transit traffic between A and C. B might rightly feel that it is A's and C's job (and cost!) to make sure that C can route to/from A's customers via a direct connection between A and C. 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. Individual peering agreements (that would govern questions such as those raised above) are typically negotiated between pairs of ISPs and are often confidential; [Huston 1999a] provides an interesting discussion of peering agreements. For a detailed description of how routing policy reflects commercial relationships among ISPs, see [Gao 2001; Dimitropoulos 2007]. For a discussion of BGP routing policies from an ISP standpoint, see [Caesar 2005b].

As noted above, BGP is the *de facto* standard for inter-AS routing for the public Internet. To see the contents of various BGP routing tables (large!) extracted from routers in tier-1 ISPs, see <http://www.routeviews.org>. BGP routing tables often contain tens of thousands of prefixes and corresponding attributes. Statistics about the size and characteristics of BGP routing tables are presented in [Potaroo 2012].

This completes our brief introduction to BGP. Understanding BGP is important because it plays a central role in the Internet. We encourage you to see the references [Griffin 2012; Stewart 1999; Labovitz 1997; Halabi 2000; Huitema 1998; Gao 2001; Feamster 2004; Caesar 2005b; Li 2007] to learn more about BGP.

4.7 Summary

In this chapter, we began our journey into the network core. We learned that the network layer involves each and every host and router in the network. Because of this, network-layer protocols are among the most challenging in the protocol stack.

We learned that a router may need to process millions of flows of packets between different source-destination pairs at the same time. To permit a router to process such a large number of flows, network designers have learned over the years that the router's tasks should be as simple as possible. Many measures can be taken

to make the router's job easier, including using a datagram network layer rather than a virtual-circuit network layer, using a streamlined and fixed-sized header (as in IPv6), eliminating fragmentation (also done in IPv6), and providing the one and only best-effort service. Perhaps the most important trick here is *not* to keep track of individual flows, but instead base routing decisions solely on hierarchically structured destination addresses in the datagrams. It is interesting to note that the postal service has been using this approach for many years.

In this chapter, we also looked at the underlying principles of routing algorithms. We learned how routing algorithms abstract the computer network to a graph with nodes and links. With this abstraction, we can exploit the rich theory of shortest-path routing in graphs, which has been developed over the past 40 years in the operations research and algorithms communities. We saw that there are two broad approaches: a centralized (global) approach, in which each node obtains a complete map of the network and independently applies a shortest-path routing algorithm; and a decentralized approach, in which individual nodes have only a partial picture of the entire network, yet the nodes work together to deliver packets along the shortest routes. We also studied how hierarchy is used to deal with the problem of scale by partitioning large networks into independent administrative domains called autonomous systems (ASs). Each AS independently routes its datagrams through the AS, just as each country independently routes its postal mail through the country. We learned how centralized, decentralized, and hierarchical approaches are embodied in the principal routing protocols in the Internet: RIP, OSPF, and BGP. We concluded our study of routing algorithms by considering broadcast and multicast routing.

Having completed our study of the network layer, our journey now takes us one step further down the protocol stack, namely, to the link layer. Like the network layer, the link layer is also part of the network core. But we will see in the next chapter that the link layer has the much more localized task of moving packets between nodes on the same link or LAN. Although this task may appear on the surface to be trivial compared with that of the network layer's tasks, we will see that the link layer involves a number of important and fascinating issues that can keep us busy for a long time.



Homework Problems and Questions

Chapter 4 Review Questions

SECTIONS 4.1–4.2

- R1. Let's review some of the terminology used in this textbook. Recall that the name of a transport-layer packet is *segment* and that the name of a link-layer packet is *frame*. What is the name of a network-layer packet? Recall that both routers and link-layer switches are called *packet switches*. What is the fundamental difference between a router and link-layer switch? Recall that we use the term *routers* for both datagram networks and VC networks.

- R2. What are the two most important network-layer functions in a datagram network? What are the three most important network-layer functions in a virtual-circuit network?
- R3. What is the difference between routing and forwarding?
- R4. Do the routers in both datagram networks and virtual-circuit networks use forwarding tables? If so, describe the forwarding tables for both classes of networks.
- R5. Describe some hypothetical services that the network layer can provide to a single packet. Do the same for a flow of packets. Are any of your hypothetical services provided by the Internet's network layer? Are any provided by ATM's CBR service model? Are any provided by ATM's ABR service model?
- R6. List some applications that would benefit from ATM's CBR service model.

SECTION 4.3

- R7. Discuss why each input port in a high-speed router stores a shadow copy of the forwarding table.
- R8. Three types of switching fabrics are discussed in Section 4.3. List and briefly describe each type. Which, if any, can send multiple packets across the fabric in parallel?
- R9. Describe how packet loss can occur at input ports. Describe how packet loss at input ports can be eliminated (without using infinite buffers).
- R10. Describe how packet loss can occur at output ports. Can this loss be prevented by increasing the switch fabric speed?
- R11. What is HOL blocking? Does it occur in input ports or output ports?

SECTION 4.4

- R12. Do routers have IP addresses? If so, how many?
- R13. What is the 32-bit binary equivalent of the IP address 223.1.3.27?
- R14. Visit a host that uses DHCP to obtain its IP address, network mask, default router, and IP address of its local DNS server. List these values.
- R15. Suppose there are three routers between a source host and a destination host. Ignoring fragmentation, an IP datagram sent from the source host to the destination host will travel over how many interfaces? How many forwarding tables will be indexed to move the datagram from the source to the destination?
- R16. Suppose an application generates chunks of 40 bytes of data every 20 msec, and each chunk gets encapsulated in a TCP segment and then an IP datagram. What percentage of each datagram will be overhead, and what percentage will be application data?
- R17. Suppose Host A sends Host B a TCP segment encapsulated in an IP datagram. When Host B receives the datagram, how does the network layer in Host B

know it should pass the segment (that is, the payload of the datagram) to TCP rather than to UDP or to something else?

- R18. Suppose you purchase a wireless router and connect it to your cable modem. Also suppose that your ISP dynamically assigns your connected device (that is, your wireless router) one IP address. Also suppose that you have five PCs at home that use 802.11 to wirelessly connect to your wireless router. How are IP addresses assigned to the five PCs? Does the wireless router use NAT? Why or why not?
- R19. Compare and contrast the IPv4 and the IPv6 header fields. Do they have any fields in common?
- R20. It has been said that when IPv6 tunnels through IPv4 routers, IPv6 treats the IPv4 tunnels as link-layer protocols. Do you agree with this statement? Why or why not?

SECTION 4.5

- R21. Compare and contrast link-state and distance-vector routing algorithms.
- R22. Discuss how a hierarchical organization of the Internet has made it possible to scale to millions of users.
- R23. Is it necessary that every autonomous system use the same intra-AS routing algorithm? Why or why not?

SECTION 4.6

- R24. Consider Figure 4.37. Starting with the original table in *D*, suppose that *D* receives from *A* the following advertisement:

Destination Subnet	Next Router	Number of Hops to Destination
z	C	10
w	—	1
x	—	1
....

Will the table in *D* change? If so how?

- R25. Compare and contrast the advertisements used by RIP and OSPF.
- R26. Fill in the blank: RIP advertisements typically announce the number of hops to various destinations. BGP updates, on the other hand, announce the _____ to the various destinations.
- R27. Why are different inter-AS and intra-AS protocols used in the Internet?
- R28. Why are policy considerations as important for intra-AS protocols, such as OSPF and RIP, as they are for an inter-AS routing protocol like BGP?

- R29. Define and contrast the following terms: *subnet*, *prefix*, and *BGP route*.
- R30. How does BGP use the NEXT-HOP attribute? How does it use the AS-PATH attribute?
- R31. Describe how a network administrator of an upper-tier ISP can implement policy when configuring BGP.

SECTION 4.7

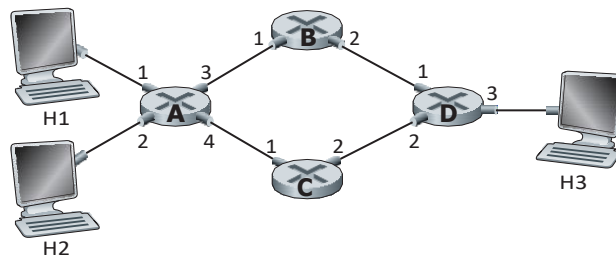
- R32. What is an important difference between implementing the broadcast abstraction via multiple unicasts, and a single network- (router-) supported broadcast?
- R33. For each of the three general approaches we studied for broadcast communication (uncontrolled flooding, controlled flooding, and spanning-tree broadcast), are the following statements true or false? You may assume that no packets are lost due to buffer overflow and all packets are delivered on a link in the order in which they were sent.
- A node may receive multiple copies of the same packet.
 - A node may forward multiple copies of a packet over the same outgoing link.
- R34. When a host joins a multicast group, must it change its IP address to that of the multicast group it is joining?
- R35. What are the roles played by the IGMP protocol and a wide-area multicast routing protocol?
- R36. What is the difference between a group-shared tree and a source-based tree in the context of multicast routing?



Problems

- P1. In this question, we consider some of the pros and cons of virtual-circuit and datagram networks.
- Suppose that routers were subjected to conditions that might cause them to fail fairly often. Would this argue in favor of a VC or datagram architecture? Why?
 - Suppose that a source node and a destination require that a fixed amount of capacity always be available at all routers on the path between the source and destination node, for the exclusive use of traffic flowing between this source and destination node. Would this argue in favor of a VC or datagram architecture? Why?
 - Suppose that the links and routers in the network never fail and that routing paths used between all source/destination pairs remains constant. In this scenario, does a VC or datagram architecture have more control traffic overhead? Why?

- P2. Consider a virtual-circuit network. Suppose the VC number is an 8-bit field.
- What is the maximum number of virtual circuits that can be carried over a link?
 - Suppose a central node determines paths and VC numbers at connection setup. Suppose the same VC number is used on each link along the VC's path. Describe how the central node might determine the VC number at connection setup. Is it possible that there are fewer VCs in progress than the maximum as determined in part (a) yet there is no common free VC number?
 - Suppose that different VC numbers are permitted in each link along a VC's path. During connection setup, after an end-to-end path is determined, describe how the links can choose their VC numbers and configure their forwarding tables in a decentralized manner, without reliance on a central node.
- P3. A bare-bones forwarding table in a VC network has four columns. What is the meaning of the values in each of these columns? A bare-bones forwarding table in a datagram network has two columns. What is the meaning of the values in each of these columns?
- P4. Consider the network below.
- Suppose that this network is a datagram network. Show the forwarding table in router A, such that all traffic destined to host H3 is forwarded through interface 3.
 - Suppose that this network is a datagram network. Can you write down a forwarding table in router A, such that all traffic from H1 destined to host H3 is forwarded through interface 3, while all traffic from H2 destined to host H3 is forwarded through interface 4? (Hint: this is a trick question.)
 - Now suppose that this network is a virtual circuit network and that there is one ongoing call between H1 and H3, and another ongoing call between H2 and H3. Write down a forwarding table in router A, such that all traffic from H1 destined to host H3 is forwarded through interface 3, while all traffic from H2 destined to host H3 is forwarded through interface 4.
 - Assuming the same scenario as (c), write down the forwarding tables in nodes B, C, and D.



- P5. Consider a VC network with a 2-bit field for the VC number. Suppose that the network wants to set up a virtual circuit over four links: link A, link B,

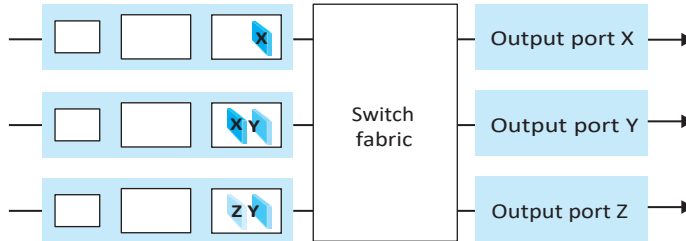
link C, and link D. Suppose that each of these links is currently carrying two other virtual circuits, and the VC numbers of these other VCs are as follows:

Link A	Link B	Link C	Link D
00	01	10	11
01	10	11	00

In answering the following questions, keep in mind that each of the existing VCs may only be traversing one of the four links.

- a. If each VC is required to use the same VC number on all links along its path, what VC number could be assigned to the new VC?
 - b. If each VC is permitted to have different VC numbers in the different links along its path (so that forwarding tables must perform VC number translation), how many different combinations of four VC numbers (one for each of the four links) could be used?
- P6. In the text we have used the term *connection-oriented service* to describe a transport-layer service and *connection service* for a network-layer service. Why the subtle shades in terminology?
- P7. Suppose two packets arrive to two different input ports of a router at exactly the same time. Also suppose there are no other packets anywhere in the router.
- a. Suppose the two packets are to be forwarded to two *different* output ports. Is it possible to forward the two packets through the switch fabric at the same time when the fabric uses a *shared bus*?
 - b. Suppose the two packets are to be forwarded to two *different* output ports. Is it possible to forward the two packets through the switch fabric at the same time when the fabric uses a *crossbar*?
 - c. Suppose the two packets are to be forwarded to the *same* output port. Is it possible to forward the two packets through the switch fabric at the same time when the fabric uses a *crossbar*?
- P8. In Section 4.3, we noted that the maximum queuing delay is $(n-1)D$ if the switching fabric is n times faster than the input line rates. Suppose that all packets are of the same length, n packets arrive at the same time to the n input ports, and all n packets want to be forwarded to *different* output ports. What is the maximum delay for a packet for the (a) memory, (b) bus, and (c) crossbar switching fabrics?
- P9. Consider the switch shown below. Suppose that all datagrams have the same fixed length, that the switch operates in a slotted, synchronous manner, and that in one time slot a datagram can be transferred from an input port to an output port. The switch fabric is a crossbar so that at most one datagram can

be transferred to a given output port in a time slot, but different output ports can receive datagrams from different input ports in a single time slot. What is the minimal number of time slots needed to transfer the packets shown from input ports to their output ports, assuming any input queue scheduling order you want (i.e., it need not have HOL blocking)? What is the largest number of slots needed, assuming the worst-case scheduling order you can devise, assuming that a non-empty input queue is never idle?



P10. Consider a datagram network using 32-bit host addresses. Suppose a router has four links, numbered 0 through 3, and packets are to be forwarded to the link interfaces as follows:

Destination Address Range	Link Interface
11100000 00000000 00000000 00000000 through 11100000 00111111 11111111 11111111	0
11100000 01000000 00000000 00000000 through 11100000 01000000 11111111 11111111	1
11100000 01000001 00000000 00000000 through 11100001 01111111 11111111 11111111	2
otherwise	3

- Provide a forwarding table that has five entries, uses longest prefix matching, and forwards packets to the correct link interfaces.
- Describe how your forwarding table determines the appropriate link interface for datagrams with destination addresses:

```

11001000 10010001 01010001 01010101
11100001 01000000 11000011 00111100
11100001 10000000 00010001 01110111

```

- P11. Consider a datagram network using 8-bit host addresses. Suppose a router uses longest prefix matching and has the following forwarding table:

Prefix Match	Interface
00	0
010	1
011	2
10	2
11	3

For each of the four interfaces, give the associated range of destination host addresses and the number of addresses in the range.

- P12. Consider a datagram network using 8-bit host addresses. Suppose a router uses longest prefix matching and has the following forwarding table:

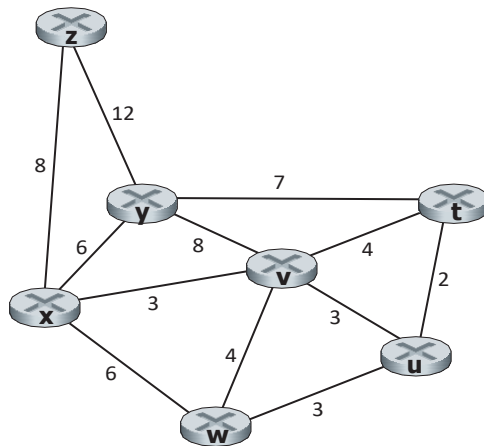
Prefix Match	Interface
1	0
10	1
111	2
otherwise	3

For each of the four interfaces, give the associated range of destination host addresses and the number of addresses in the range.

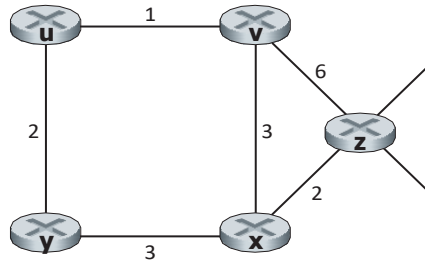
- P13. Consider a router that interconnects three subnets: Subnet 1, Subnet 2, and Subnet 3. Suppose all of the interfaces in each of these three subnets are required to have the prefix 223.1.17/24. Also suppose that Subnet 1 is required to support at least 60 interfaces, Subnet 2 is to support at least 90 interfaces, and Subnet 3 is to support at least 12 interfaces. Provide three network addresses (of the form a.b.c.d/x) that satisfy these constraints.
- P14. In Section 4.2.2 an example forwarding table (using longest prefix matching) is given. Rewrite this forwarding table using the a.b.c.d/x notation instead of the binary string notation.
- P15. In Problem P10 you are asked to provide a forwarding table (using longest prefix matching). Rewrite this forwarding table using the a.b.c.d/x notation instead of the binary string notation.

- P16. Consider a subnet with prefix 128.119.40.128/26. Give an example of one IP address (of form xxx.xxx.xxx.xxx) that can be assigned to this network. Suppose an ISP owns the block of addresses of the form 128.119.40.64/26. Suppose it wants to create four subnets from this block, with each block having the same number of IP addresses. What are the prefixes (of form a.b.c.d/x) for the four subnets?
- P17. Consider the topology shown in Figure 4.17. Denote the three subnets with hosts (starting clockwise at 12:00) as Networks A, B, and C. Denote the subnets without hosts as Networks D, E, and F.
- Assign network addresses to each of these six subnets, with the following constraints: All addresses must be allocated from 214.97.254/23; Subnet A should have enough addresses to support 250 interfaces; Subnet B should have enough addresses to support 120 interfaces; and Subnet C should have enough addresses to support 120 interfaces. Of course, subnets D, E and F should each be able to support two interfaces. For each subnet, the assignment should take the form a.b.c.d/x or a.b.c.d/x – e.f.g.h/y.
 - Using your answer to part (a), provide the forwarding tables (using longest prefix matching) for each of the three routers.
- P18. Use the whois service at the American Registry for Internet Numbers (<http://www.arin.net/whois>) to determine the IP address blocks for three universities. Can the whois services be used to determine with certainty the geographical location of a specific IP address? Use www.maxmind.com to determine the locations of the Web servers at each of these universities.
- P19. Consider sending a 2400-byte datagram into a link that has an MTU of 700 bytes. Suppose the original datagram is stamped with the identification number 422. How many fragments are generated? What are the values in the various fields in the IP datagram(s) generated related to fragmentation?
- P20. Suppose datagrams are limited to 1,500 bytes (including header) between source Host A and destination Host B. Assuming a 20-byte IP header, how many datagrams would be required to send an MP3 consisting of 5 million bytes? Explain how you computed your answer.
- P21. Consider the network setup in Figure 4.22. Suppose that the ISP instead assigns the router the address 24.34.112.235 and that the network address of the home network is 192.168.1/24.
- Assign addresses to all interfaces in the home network.
 - Suppose each host has two ongoing TCP connections, all to port 80 at host 128.119.40.86. Provide the six corresponding entries in the NAT translation table.

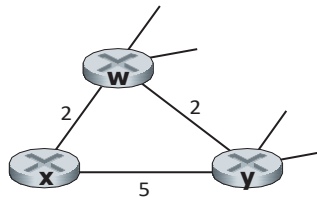
- P22. Suppose you are interested in detecting the number of hosts behind a NAT. You observe that the IP layer stamps an identification number sequentially on each IP packet. The identification number of the first IP packet generated by a host is a random number, and the identification numbers of the subsequent IP packets are sequentially assigned. Assume all IP packets generated by hosts behind the NAT are sent to the outside world.
- Based on this observation, and assuming you can sniff all packets sent by the NAT to the outside, can you outline a simple technique that detects the number of unique hosts behind a NAT? Justify your answer.
 - If the identification numbers are not sequentially assigned but randomly assigned, would your technique work? Justify your answer.
- P23. In this problem we'll explore the impact of NATs on P2P applications. Suppose a peer with username Arnold discovers through querying that a peer with username Bernard has a file it wants to download. Also suppose that Bernard and Arnold are both behind a NAT. Try to devise a technique that will allow Arnold to establish a TCP connection with Bernard without application-specific NAT configuration. If you have difficulty devising such a technique, discuss why.
- P24. Looking at Figure 4.27, enumerate the paths from y to u that do not contain any loops.
- P25. Repeat Problem P24 for paths from x to z , z to u , and z to w .
- P26. Consider the following network. With the indicated link costs, use Dijkstra's shortest-path algorithm to compute the shortest path from x to all network nodes. Show how the algorithm works by computing a table similar to Table 4.3.



- P27. Consider the network shown in Problem P26. Using Dijkstra's algorithm, and showing your work using a table similar to Table 4.3, do the following:
- Compute the shortest path from t to all network nodes.
 - Compute the shortest path from u to all network nodes.
 - Compute the shortest path from v to all network nodes.
 - Compute the shortest path from w to all network nodes.
 - Compute the shortest path from y to all network nodes.
 - Compute the shortest path from z to all network nodes.
- P28. Consider the network shown below, and assume that each node initially knows the costs to each of its neighbors. Consider the distance-vector algorithm and show the distance table entries at node z .

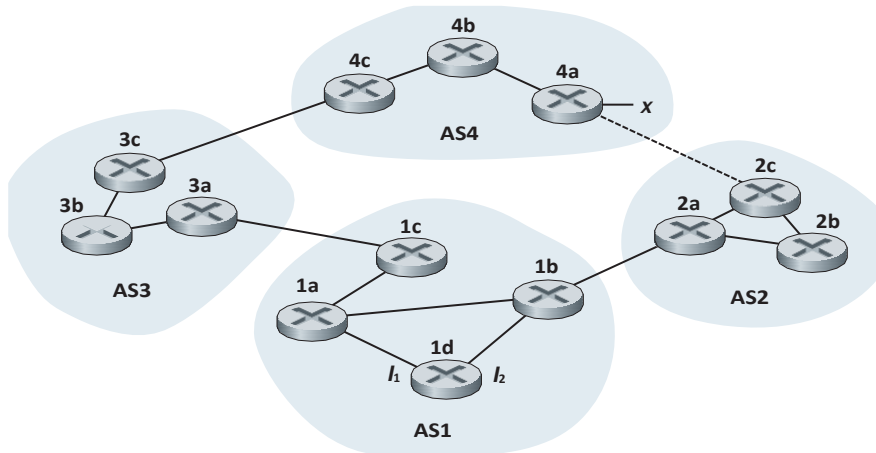


- P29. Consider a general topology (that is, not the specific network shown above) and a synchronous version of the distance-vector algorithm. Suppose that at each iteration, a node exchanges its distance vectors with its neighbors and receives their distance vectors. Assuming that the algorithm begins with each node knowing only the costs to its immediate neighbors, what is the maximum number of iterations required before the distributed algorithm converges? Justify your answer.
- P30. Consider the network fragment shown below. x has only two attached neighbors, w and y . w has a minimum-cost path to destination u (not shown) of 5, and y has a minimum-cost path to u of 6. The complete paths from w and y to u (and between w and y) are not shown. All link costs in the network have strictly positive integer values.



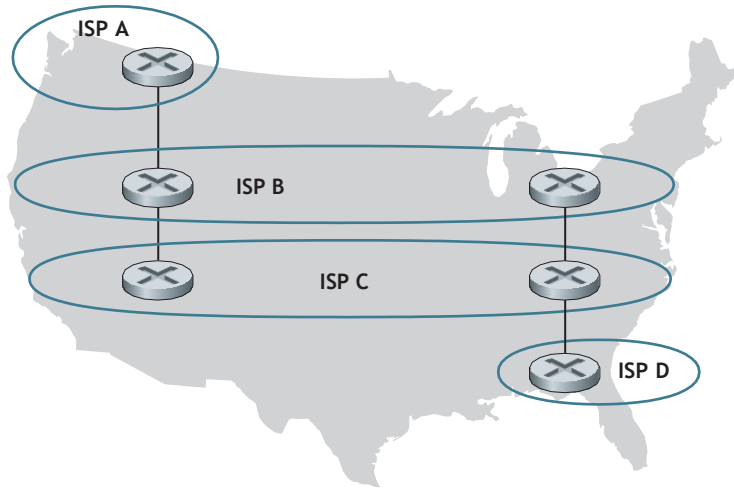
- a. Give x 's distance vector for destinations w , y , and u .
 - b. Give a link-cost change for either $c(x,w)$ or $c(x,y)$ such that x will inform its neighbors of a new minimum-cost path to u as a result of executing the distance-vector algorithm.
 - c. Give a link-cost change for either $c(x,w)$ or $c(x,y)$ such that x will *not* inform its neighbors of a new minimum-cost path to u as a result of executing the distance-vector algorithm.
- P31. Consider the three-node topology shown in Figure 4.30. Rather than having the link costs shown in Figure 4.30, the link costs are $c(x,y) = 3$, $c(y,z) = 6$, $c(z,x) = 4$. Compute the distance tables after the initialization step and after each iteration of a synchronous version of the distance-vector algorithm (as we did in our earlier discussion of Figure 4.30).
- P32. Consider the count-to-infinity problem in the distance vector routing. Will the count-to-infinity problem occur if we decrease the cost of a link? Why? How about if we connect two nodes which do not have a link?
- P33. Argue that for the distance-vector algorithm in Figure 4.30, each value in the distance vector $D(x)$ is non-increasing and will eventually stabilize in a finite number of steps.
- P34. Consider Figure 4.31. Suppose there is another router w , connected to router y and z . The costs of all links are given as follows: $c(x,y) = 4$, $c(x,z) = 50$, $c(y,w) = 1$, $c(z,w) = 1$, $c(y,z) = 3$. Suppose that poisoned reverse is used in the distance-vector routing algorithm.
- a. When the distance vector routing is stabilized, router w , y , and z inform their distances to x to each other. What distance values do they tell each other?
 - b. Now suppose that the link cost between x and y increases to 60. Will there be a count-to-infinity problem even if poisoned reverse is used? Why or why not? If there is a count-to-infinity problem, then how many iterations are needed for the distance-vector routing to reach a stable state again? Justify your answer.
 - c. How do you modify $c(y,z)$ such that there is no count-to-infinity problem at all if $c(y,x)$ changes from 4 to 60?
- P35. Describe how loops in paths can be detected in BGP.
- P36. Will a BGP router always choose the loop-free route with the shortest AS-path length? Justify your answer.
- P37. Consider the network shown below. Suppose AS3 and AS2 are running OSPF for their intra-AS routing protocol. Suppose AS1 and AS4 are running RIP for their intra-AS routing protocol. Suppose eBGP and iBGP are used for the inter-AS routing protocol. Initially suppose there is *no* physical link between AS2 and AS4.

- Router 3c learns about prefix x from which routing protocol: OSPF, RIP, eBGP, or iBGP?
- Router 3a learns about x from which routing protocol?
- Router 1c learns about x from which routing protocol?
- Router 1d learns about x from which routing protocol?

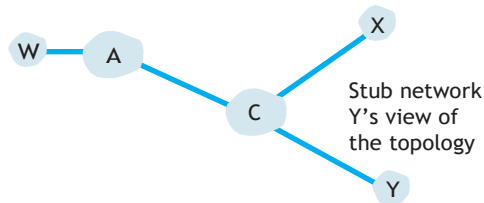


- P38. Referring to the previous problem, once router 1d learns about x it will put an entry (x, I) in its forwarding table.
- Will I be equal to I_1 or I_2 for this entry? Explain why in one sentence.
 - Now suppose that there is a physical link between AS2 and AS4, shown by the dotted line. Suppose router 1d learns that x is accessible via AS2 as well as via AS3. Will I be set to I_1 or I_2 ? Explain why in one sentence.
 - Now suppose there is another AS, called AS5, which lies on the path between AS2 and AS4 (not shown in diagram). Suppose router 1d learns that x is accessible via AS2 AS5 AS4 as well as via AS3 AS4. Will I be set to I_1 or I_2 ? Explain why in one sentence.
- P39. Consider the following network. ISP B provides national backbone service to regional ISP A. ISP C provides national backbone service to regional ISP D. Each ISP consists of one AS. B and C peer with each other in two places using BGP. Consider traffic going from A to D. B would prefer to hand that traffic over to C on the West Coast (so that C would have to absorb the cost of carrying the traffic cross-country), while C would prefer to get the traffic via its East Coast peering point with B (so that B would have carried the traffic across the country). What BGP mechanism might C use, so that B would hand over A-to-D traffic at its East Coast

peering point? To answer this question, you will need to dig into the BGP specification.



P40. In Figure 4.42, consider the path information that reaches stub networks W, X, and Y. Based on the information available at W and X, what are their respective views of the network topology? Justify your answer. The topology view at Y is shown below.



- P41. Consider Figure 4.42. B would never forward traffic destined to Y via X based on BGP routing. But there are some very popular applications for which data packets go to X first and then flow to Y. Identify one such application, and describe how data packets follow a path not given by BGP routing.
- P42. In Figure 4.42, suppose that there is another stub network V that is a customer of ISP A. Suppose that B and C have a peering relationship, and A is a customer of both B and C. Suppose that A would like to have the traffic destined to W to come from B only, and the traffic destined to V from either B or C. How should A advertise its routes to B and C? What AS routes does C receive?
- P43. Suppose ASs X and Z are not directly connected but instead are connected by AS Y. Further suppose that X has a peering agreement with Y, and that Y has

- a peering agreement with Z. Finally, suppose that Z wants to transit all of Y's traffic but does not want to transit X's traffic. Does BGP allow Z to implement this policy?
- P44. Consider the seven-node network (with nodes labeled t to z) in Problem P26. Show the minimal-cost tree rooted at z that includes (as end hosts) nodes u , v , w , and y . Informally argue why your tree is a minimal-cost tree.
- P45. Consider the two basic approaches identified for achieving broadcast, unicast emulation and network-layer (i.e., router-assisted) broadcast, and suppose spanning-tree broadcast is used to achieve network-layer broadcast. Consider a single sender and 32 receivers. Suppose the sender is connected to the receivers by a binary tree of routers. What is the cost of sending a broadcast packet, in the cases of unicast emulation and network-layer broadcast, for this topology? Here, each time a packet (or copy of a packet) is sent over a single link, it incurs a unit of cost. What topology for interconnecting the sender, receivers, and routers will bring the cost of unicast emulation and true network-layer broadcast as far apart as possible? You can choose as many routers as you'd like.
- P46. Consider the operation of the reverse path forwarding (RPF) algorithm in Figure 4.44. Using the same topology, find a set of paths from all nodes to the source node A (and indicate these paths in a graph using thicker-shaded lines as in Figure 4.44) such that if these paths were the least-cost paths, then node B would receive a copy of A 's broadcast message from nodes A , C , and D under RPF.
- P47. Consider the topology shown in Figure 4.44. Suppose that all links have unit cost and that node E is the broadcast source. Using arrows like those shown in Figure 4.44 indicate links over which packets will be forwarded using RPF, and links over which packets will not be forwarded, given that node E is the source.
- P48. Repeat Problem P47 using the graph from Problem P26. Assume that z is the broadcast source, and that the link costs are as shown in Problem P26.
- P49. Consider the topology shown in Figure 4.46, and suppose that each link has unit cost. Suppose node C is chosen as the center in a center-based multicast routing algorithm. Assuming that each attached router uses its least-cost path to node C to send join messages to C , draw the resulting center-based routing tree. Is the resulting tree a minimum-cost tree? Justify your answer.
- P50. Repeat Problem P49, using the graph from Problem P26. Assume that the center node is v .
- P51. In Section 4.5.1 we studied Dijkstra's link-state routing algorithm for computing the unicast paths that are individually the least-cost paths from the source to all destinations. The union of these paths might be thought of as forming a **least-unicast-cost path tree** (or a shortest unicast path tree, if all link costs are identical). By constructing a counterexample, show that the least-cost path tree is *not* always the same as a minimum spanning tree.

- P52. Consider a network in which all nodes are connected to three other nodes. In a single time step, a node can receive all transmitted broadcast packets from its neighbors, duplicate the packets, and send them to all of its neighbors (except to the node that sent a given packet). At the next time step, neighboring nodes can receive, duplicate, and forward these packets, and so on. Suppose that uncontrolled flooding is used to provide broadcast in such a network. At time step t , how many copies of the broadcast packet will be transmitted, assuming that during time step 1, a single broadcast packet is transmitted by the source node to its three neighbors.
- P53. We saw in Section 4.7 that there is no network-layer protocol that can be used to identify the hosts participating in a multicast group. Given this, how can multicast applications learn the identities of the hosts that are participating in a multicast group?
- P54. Design (give a pseudocode description of) an application-level protocol that maintains the host addresses of all hosts participating in a multicast group. Specifically identify the network service (unicast or multicast) that is used by your protocol, and indicate whether your protocol is sending messages in-band or out-of-band (with respect to the application data flow among the multicast group participants) and why.
- P55. What is the size of the multicast address space? Suppose now that two multicast groups randomly choose a multicast address. What is the probability that they choose the same address? Suppose now that 1,000 multicast groups are ongoing at the same time and choose their multicast group addresses at random. What is the probability that they interfere with each other?



Socket Programming Assignment

At the end of Chapter 2, there are four socket programming assignments. Below, you will find a fifth assignment which employs ICMP, a protocol discussed in this chapter.

Assignment 5: ICMP Ping

Ping is a popular networking application used to test from a remote location whether a particular host is up and reachable. It is also often used to measure latency between the client host and the target host. It works by sending ICMP “echo request” packets (i.e., ping packets) to the target host and listening for ICMP “echo response” replies (i.e., pong packets). Ping measures the RRT, records packet loss, and calculates a statistical summary of multiple ping-pong exchanges (the minimum, mean, max, and standard deviation of the round-trip times).

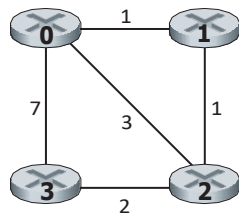
In this lab, you will write your own Ping application in Python. Your application will use ICMP. But in order to keep your program simple, you will not exactly follow the official specification in RFC 1739. Note that you will only need to write the client side of the program, as the functionality needed on the server side is built into almost all operating systems. You can find full details of this assignment, as well as important snippets of the Python code, at the Web site <http://www.awl.com/kurose-ross>.



Programming Assignment

In this programming assignment, you will be writing a “distributed” set of procedures that implements a distributed asynchronous distance-vector routing for the network shown below.

You are to write the following routines that will “execute” asynchronously within the emulated environment provided for this assignment. For node 0, you will write the routines:



- *rtinit0()*. This routine will be called once at the beginning of the emulation. *rtinit0()* has no arguments. It should initialize your distance table in node 0 to reflect the direct costs of 1, 3, and 7 to nodes 1, 2, and 3, respectively. In the figure above, all links are bidirectional and the costs in both directions are identical. After initializing the distance table and any other data structures needed by your node 0 routines, it should then send its directly connected neighbors (in this case, 1, 2, and 3) the cost of its minimum-cost paths to all other network nodes. This minimum-cost information is sent to neighboring nodes in a routing update packet by calling the routine *tolayer2()*, as described in the full assignment. The format of the routing update packet is also described in the full assignment.
- *rtupdate0(struct rtpkt *rcvdpkt)*. This routine will be called when node 0 receives a routing packet that was sent to it by one of its directly connected neighbors. The parameter **rcvdpkt* is a pointer to the packet that was received. *rtupdate0()* is the “heart” of the distance-vector algorithm. The values it receives in a routing update packet from some other node *i* contain *i*’s current shortest-path costs to all other network nodes. *rtupdate0()* uses these received

values to update its own distance table (as specified by the distance-vector algorithm). If its own minimum cost to another node changes as a result of the update, node 0 informs its directly connected neighbors of this change in minimum cost by sending them a routing packet. Recall that in the distance-vector algorithm, only directly connected nodes will exchange routing packets. Thus, nodes 1 and 2 will communicate with each other, but nodes 1 and 3 will not communicate with each other.

Similar routines are defined for nodes 1, 2, and 3. Thus, you will write eight procedures in all: *rtinit0()*, *rtinit1()*, *rtinit2()*, *rtinit3()*, *rtupdate0()*, *rtupdate1()*, *rtupdate2()*, and *rtupdate3()*. These routines will together implement a distributed, asynchronous computation of the distance tables for the topology and costs shown in the figure on the preceding page.

You can find the full details of the programming assignment, as well as C code that you will need to create the simulated hardware/software environment, at <http://www.awl.com/kurose-ross>. A Java version of the assignment is also available.



Wireshark Labs

In the companion Web site for this textbook, <http://www.awl.com/kurose-ross>, you'll find two Wireshark lab assignments. The first lab examines the operation of the IP protocol, and the IP datagram format in particular. The second lab explores the use of the ICMP protocol in the ping and traceroute commands.