Chapter 4: Network Layer and Routing

Introduction

In this chapter we begin our journey into the network core. We learn that one of the biggest challenges is routing datagrams through a network of millions of hosts and routers. We investigate how to solve this scaling problem by partitioning large networks, such as the Internet, into independent administrative domains called Autonomous Systems (ASs). We learn that routing is done on two levels--one level for within each of the ASs and another level for among the ASs. We examine the underlying principles of routing algorithms for intra-AS routing and inter-AS routing, including link-state routing algorithms and distance-vector routing algorithms. We then survey many of the Internet routing protocols for intra-AS and inter-AS routing. In this chapter we also examine the IP protocol in detail, covering IP addressing, IP datagram format, datagram fragmentation, and the ICMP (Internet Control Message Protocol). We also explore two more advanced topics, namely, IPv6 and multicast routing.

4.1: Introduction and Network Service Models

We saw in the previous chapter that the transport layer provides communication service between two processes running on two different hosts. In order to provide this service, the transport layer relies on the services of the network layer, which provides a communication service between hosts. In particular, the network layer moves transport-layer segments from one host to another. At the sending host, the transport-layer segment is passed to the network layer. It is then the job of the network layer to get the segment to the destination host and pass the segment up the protocol stack to the transport layer. Exactly how the network layer moves a segment from the transport layer of an origin host to the transport layer of the destination host is the subject of this chapter. We will see that unlike the transport layers, the network layer *involves each and every host and router in the network.* Because of this, network-layer protocols are among the most challenging (and therefore interesting!) in the protocol stack.

Figure 4.1 shows a simple network with two hosts (H1 and H2) and several routers on the path between H1 and H2. The role of the network layer in a sending host is to begin the packet on its journey to the receiving host. For example, if H1 is sending to H2, the network layer in host H1 transfers these packets to its nearby router R2. At the receiving host (for example, H2), the network layer receives the packet from its nearby router (in this case, R2) and delivers the packet up to the transport layer at H2. The primary role of the routers is to "switch" packets from input links to output links. Note that the routers in Figure 4.1 are shown with a truncated protocol stack, that is, with no upper layers above the network layer, because (except for control purposes) routers do not run transport- and application-layer protocols such as those we examined in Chapters 2 and 3.

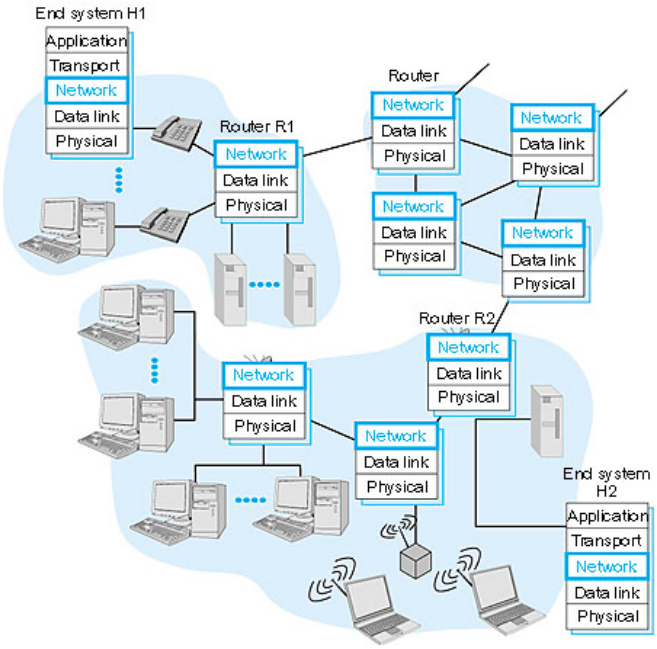


Figure 4.1: The network layer

The role of the network layer is thus deceptively simple--to transport packets from a sending host to a receiving host. To do so, three important network-layer functions can be identified:

- *Path determination.* The network layer must determine the route or path taken by packets as they flow from a sender to a receiver. The algorithms that calculate these paths are referred to as **routing algorithms.** A routing algorithm would determine, for example, the path along which packets flow from H1 to H2. Much of this chapter will focus on routing algorithms. In Section 4.2 we will study the theory of routing algorithms, concentrating on the two most prevalent classes of routing algorithms: link-state routing and distance vector routing. We'll see that the complexity of routing algorithms grows considerably as the number of routers in the network increases. This motivates the use of hierarchical routing, a topic we cover in Section 4.3. In Section 4.8 we cover multicast routing--the routing algorithms, switching functions, and call setup mechanisms that allow a packet that is sent just once by a sender to be delivered to multiple destinations.
- *Switching.* When a packet arrives at the input to a router, the router must move it to the appropriate output link. For example, a packet arriving from host H1 to router R2 must be forwarded to the next router on the path to H2. In Section 4.6, we look inside a router and examine how a packet is actually switched (moved) from an input link at a router to an output link.
- *Call setup.* Recall that in our study of TCP, a three-way handshake was required before data actually flowed from sender to receiver. This allowed the sender and receiver to set up the needed state information (for example, sequence number and initial flow-control window size). In an analogous manner, some network-layer architectures (for example, ATM) require that the routers along the chosen path from source to destination handshake with each other in order to setup state before data actually begins to flow. In the network layer, this process is referred to as **call setup.** The network layer of the Internet architecture does not perform any such call setup.

Before delving into the details of the theory and implementation of the network layer, however, let us first take the broader view and consider what different types of service might be offered by the network layer.

4.1.1: Network Service Model

When the transport layer at a sending host transmits a packet into the network (that is, passes it down to the network layer at the sending host), can the transport layer count on the network layer to deliver the packet to the destination? When multiple packets are sent, will they be delivered to the transport layer in the receiving host in the order in which they were sent? Will the amount of time between the sending of two sequential packet transmissions be the same as the amount of time between their reception? Will the network provide any feedback about congestion in the network? What is the abstract view (properties) of the channel connecting the transport layer in the sending and receiving hosts? The answers to these questions and others are determined by the service model provided by the network layer. The **network-service model** defines the characteristics of end-to-end transport of data between one "edge" of the network and the other, that is, between sending and receiving end systems.

Datagram or Virtual Circuit?

Perhaps the most important abstraction provided by the network layer to the upper layers is whether or not the network layer uses **virtual circuits (VCs).** You may recall from Chapter 1 that a virtual-circuit packet network behaves much like a telephone network, which uses "real circuits" as opposed to "virtual circuits." There are three identifiable phases in a virtual circuit:

- *VC setup.* During the setup phase, the sender contacts the network layer, specifies the receiver address, and waits for the network to set up the VC. The network layer determines the path between sender and receiver, that is, the series of links and packet switches through which all packets of the VC will travel. As discussed in Chapter 1, this typically involves updating tables in each of the packet switches in the path. During VC setup, the network layer may also reserve resources (for example, bandwidth) along the path of the VC.
- *Data transfer.* Once the VC has been established, data can begin to flow along the VC.
- *Virtual-circuit teardown.* This is initiated when the sender (or receiver) informs the network layer of its desire to terminate the VC. The network layer will then typically inform the end system on the other side of the network of the call termination and update the tables in each of the packet switches on the path to indicate that the VC no longer exists.

There is a subtle but important distinction between VC setup at the network layer and connection setup at the transport layer (for example, the TCP three-way handshake we studied in Chapter 3). Connection setup at the transport layer involves only the two end systems. The two end systems agree to communicate and together determine the parameters (for example, initial sequence number, flow-control window size) of their transport-layer connection before data actually begins to flow on the transport-level connection. Although the two end

systems are aware of the transport-layer connection, the switches within the network are completely oblivious to it. On the other hand, with a virtual-circuit network layer, *packet switches along the path between the two end systems are involved in virtual-circuit setup, and each packet switch is fully aware of all the VCs passing through it.*

The messages that the end systems send to the network to indicate the initiation or termination of a VC, and the messages passed between the switches to set up the VC (that is, to modify switch tables) are known as **signaling messages** and the protocols used to exchange these messages are often referred to as **signaling protocols.** VC setup is shown pictorially in Figure 4.2. ATM, frame relay and X.25, which will be covered in Chapter 5, are three other networking technologies that use virtual circuits.

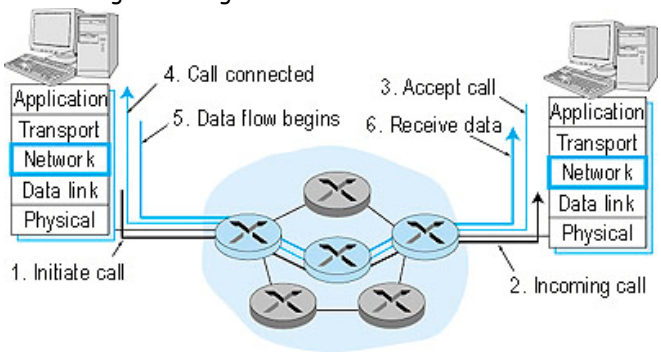


Figure 4.2: Virtual-circuit service model

With a **datagram network layer,** each time an end system wants to send a packet, it stamps the packet with the address of the destination end system, and then pops the packet into the network. As shown in Figure 4.3, this is done without any VC setup. Packet switches in a datagram network (called "routers" in the Internet) do not maintain any state information about VCs because there are no VCs! Instead, packet switches route a packet toward its destination by examining the packet's destination address, indexing a routing table with the destination address, and forwarding the packet in the direction of the destination. (As discussed in Chapter 1, datagram routing is similar to routing ordinary postal mail.) Because routing tables can be modified at any time, a series of packets sent from one end system to another may follow different paths through the network and may arrive out of order. The Internet uses a datagram network layer. [Paxson 1997] presents an interesting measurement study of packet reordering and other phenomena in the public Internet.

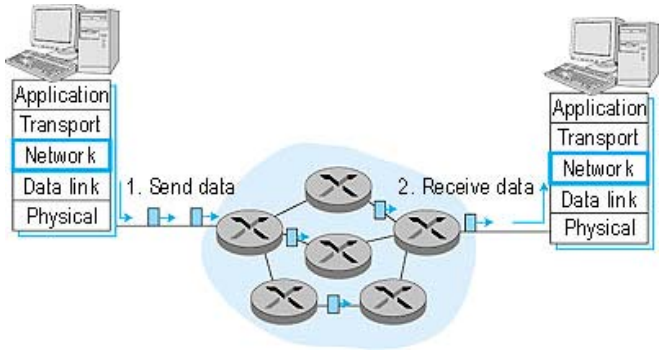


Figure 4.3: Datagram service model

You may recall from Chapter 1 that a packet-switched network typically offers either a VC service or a datagram service to the transport layer, but not both services. For example, we'll see in Chapter 5 that an ATM network offers only a VC service to the transport layer. The Internet offers only a datagram service to the transport layer.

An alternate terminology for VC service and datagram service is **network-layer connection-oriented service** and **network-layer connectionless service,** respectively. Indeed, VC service is a sort of connection-oriented service, as it involves setting up and tearing down a connection-like entity, and maintaining connection-state information in the packet switches. Datagram service is a sort of connectionless service in that it does not employ connection-like entities. Both sets of terminology have advantages and disadvantages, and both sets are commonly used in the networking literature. In this book we decided to use the "VC service" and "datagram service" terminology for the network layer, and reserve the "connection-oriented service" and "connectionless

service" terminology for the transport layer. We believe this distinction will be useful in helping the reader delineate the services offered by the two layers.

The key aspects of the service model of the Internet and ATM network architectures are summarized in Table 4.1. We do not want to delve deeply into the details of the service models here (it can be quite "dry" and detailed discussions can be found in the standards themselves [ATM Forum 1997]). A comparison between the Internet and ATM service models is, however, quite instructive.

Table 4.1: Internet and ATM Network Service Models

Network Architecture	Service Model	Bandwidth Guarantee	No Loss Guarantee	Ordering	Timing	Congestion Indication
Internet	Best Effort	None	None	Any order possible	Not maintained	None
ATM	CBR	Guaranteed constant rate	Yes	In order	Maintained	Congestion will not occur
ATM	VBR	Guaranteed Rate	Yes	In order	Maintained	Congestion will not occur
ATM	ABR	Guaranteed minimum	None	In order	Not maintained	Congestion indication provided
ATM	UBR	None	None	In order	Not maintained	None

The current Internet architecture provides only one service model, the datagram service, which is also known as "**best-effort service.**" From Table 4.1, it might appear that best effort service is a euphemism for "no service at all." With best-effort service, timing between packets is not guaranteed to be preserved, packets are not guaranteed to be received in the order in which they were sent, nor is the eventual delivery of transmitted packets guaranteed. Given this definition, a network that delivered *no* packets to the destination would satisfy the definition of best-effort delivery service. (Indeed, today's congested public Internet might sometimes appear to be an example of a network that does so!) As we will discuss shortly, however, there are sound reasons for such a minimalist network service model. The Internet's best-effort only service model is currently being extended to include so-called integrated services and differentiated service. We will cover these still-evolving service models later in Chapter 6.

Let us next turn to the ATM service models. We'll focus here on the service model standards being developed in the ATM Forum [ATM Forum 1997]. The ATM architecture provides for multiple service models (that is, the ATM standard has multiple service models). This means that within the same network, different connections can be provided with different classes of service.

Constant bit rate (CBR) network service was the first ATM service model to be standardized, probably reflecting the fact that telephone companies were the early prime movers behind ATM, and CBR network service is ideally suited for carrying real-time, constant-bit-rate audio (for example, a digitized telephone call) and video traffic. The goal of CBR service is conceptually simple--to make the network connection look like a dedicated copper or fiber connection between the sender and receiver. With CBR service, ATM packets (referred to as **cells** in ATM jargon) are carried across the network in such a way that the end-to-end delay experienced by a cell (the so-called cell-transfer delay, CTD), the variability in the end-end delay (often referred to as "jitter" or cell-delay variation, CDV), and the fraction of cells that are lost or delivered late (the so-called cell-loss rate, CLR) are guaranteed to be less than some specified values. Also, an allocated transmission rate (the peak cell rate, PCR) is defined for the connection and the sender is expected to offer data to the network at this rate. The values for the PCR, CTD, CDV, and CLR are agreed upon by the sending host and the ATM network when the CBR connection is first established.

A second conceptually simple ATM service class is **Unspecified bit rate (UBR) network service.** Unlike CBR service, which guarantees rate, delay, delay jitter, and loss, UBR makes no guarantees at all other than in-order delivery of cells (that is, cells that are fortunate enough to make it to the receiver). With the exception of in-order delivery, UBR service is thus equivalent to the Internet best-effort service model. As with the Internet best-effort service model, UBR also provides no feedback to the sender about whether or not a cell is dropped within the network. For reliable transmission of data over a UBR network, higher-layer protocols (such as those we studied in the previous chapter) are needed. UBR service might be well suited for noninteractive data transfer applications such as e-mail and newsgroups.

If UBR can be thought of as a "best-effort" service, then **available bit rate (ABR) network service** might best be characterized as a "better" best-effort service model. The two most important additional features of ABR service over UBR service are:

- A minimum cell transmission rate (MCR) is guaranteed to a connection using ABR service. If, however, the network has enough free resources at a given time, a sender may actually be able to successfully send traffic at a *higher* rate than the MCR.
- Congestion feedback from the network. We saw in Section 3.6.3 that an ATM network can provide feedback to the sender (in terms of a congestion notification bit, or a lower rate at which to send) that controls how the sender should adjust its rate between the MCR and the peak cell rate (PCR). ABR senders control their transmission rates based on such feedback.
  ABR provides a minimum bandwidth guarantee, but on the other hand will attempt to transfer data as fast as possible (up to the limit imposed by the PCR). As such, ABR is well suited for data transfer, where it is desirable to keep the transfer delays low (for example, Web browsing).

The final ATM service model is **variable bit rate (VBR) network service**. VBR service comes in two flavors (perhaps indicating a service class with an identity crisis!). In real-time VBR service, the acceptable cell-loss rate, delay, and delay jitter are specified as in CBR service. However, the actual source rate is allowed to vary according to parameters specified by the user to the network. The declared variability in rate may be used by the network (internally) to more efficiently allocate resources to its connections, but in terms of the loss, delay, and jitter seen by the sender, the service is essentially the same as CBR service. While early efforts in defining a VBR service model were clearly targeted toward real-time services (for example, as evidenced by the PCR, CTD, CDV, and CLR parameters), a second flavor of VBR service is now targeted toward non-real-time services and provides a cell-loss rate guarantee. An obvious question with VBR is what advantages it offers over CBR (for real-time applications) and over UBR and ABR for non-real-time applications. Currently, there is not enough (any?) experience with VBR service to answer these questions.

An excellent discussion of the rationale behind various aspects of the ATM Forum's Traffic Management Specification 4.0 [ATM Forum 1996] for CBR, VBR, ABR, and UBR service is [Garrett 1996].

4.1.2: Origins of Datagram and Virtual Circuit Service

The evolution of the Internet and ATM network service models reflects their origins. With the notion of a virtual circuit as a central organizing principle, and an early focus on CBR services, ATM reflects its roots in the telephony world (which uses "real circuits"). The subsequent definition of UBR and ABR service classes acknowledges the importance of data applications developed in the data networking community. Given the VC architecture and a focus on supporting real-time traffic with *guarantees* about the level of received performance (even with data-oriented services such as ABR), the network layer is *significantly more complex* than the best-effort Internet. This, too, is in keeping with the ATM's telephony heritage. Telephone networks, by necessity, had their "complexity" within the network, since they were connecting "dumb" end-system devices such as a rotary telephone. (For those too young to know, a rotary phone is a nondigital telephone with no buttons--only a dial.)

The Internet, on the other hand, grew out of the need to connect computers (that is, more sophisticated end devices) together. With sophisticated end-system devices, the Internet architects chose to make the network-service model (best effort) as simple as possible and to implement any additional functionality (for example, reliable data transfer), as well as any new application-level network services at a higher layer, at the end systems. This inverts the model of the telephone network, with some interesting consequences:

- The resulting Internet network-service model, which made minimal (no!) service guarantees (and hence posed minimal requirements on the network layer), also made it easier to *interconnect* networks that used very different link-layer technologies (for example, satellite, Ethernet, fiber, or radio) that had very different transmission rates and loss characteristics. We will address the interconnection of IP networks in detail in Section 4.4.
- As we saw in Chapter 2, applications such as e-mail, the Web, and even a network-layer-centric service such as the DNS are implemented in hosts (servers) at the edge of the network. The ability to add a new service simply by attaching a host to the network and defining a new higher-layer protocol (such as HTTP) has allowed new services such as the WWW to be adopted in the Internet in a breathtakingly short period of time.

As we will see in Chapter 6, however, there is considerable debate in the Internet community about how the network-layer architecture must evolve in order to support real-time services such as multimedia. An interesting comparison of the ATM and the proposed next generation Internet architecture is given in [Crowcroft 1995].

4.2: Routing Principles

In order to transfer packets from a sending host to the destination host, the network layer must determine the *path* or *route* that the packets are to follow. Whether the network layer provides a datagram

service (in which case different packets between a given host-destination pair may take different routes) or a virtual-circuit 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 for a packet. This is the job of the network layer **routing protocol**.

At the heart of any routing protocol is the algorithm (the **routing algorithm**) that determines the path for a packet. The purpose of a routing algorithm is simple: given a set of routers, with links connecting the routers, a routing algorithm finds a "good" path from source to destination. Typically, a "good" path is one that has "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.

The graph abstraction used to formulate routing algorithms is shown in Figure 4.4. To view some graphs representing real network maps, see [Dodge 1999]; for a discussion of how well different graph-based models model the Internet, see [Zegura 1997]. Here, nodes in the graph represent routers--the points at which packet routing decisions are made--and the lines ("edges" in graph theory terminology) connecting these nodes represent the physical links between these routers. A link also has a value representing the "cost" of sending a packet across the link. The cost may reflect the level of congestion on that link (for example, the current average delay for a packet across that link) or the physical distance traversed by that link (for example, a transoceanic link might have a higher cost than a short-haul terrestrial link). For our current purposes, we'll simply take the link costs as a given and won't worry about how they are determined.

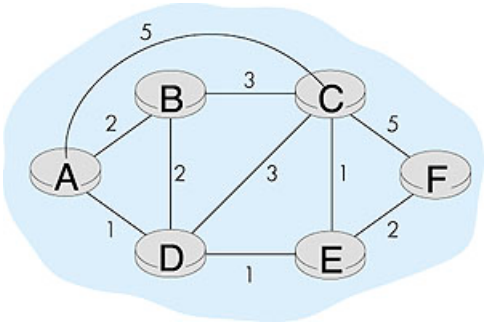


Figure 4.4: Abstract model of a network

Given the graph abstraction, the problem of finding the least-cost path from a source to a destination requires identifying a series of links such that:

- the first link in the path is connected to the source
- the last link in the path is connected to the destination
- for all *i*, the *i* and *i*-1st link in the path are connected to the same node
- for the **least-cost path**, the sum of the cost of the links on the path is the minimum over all possible paths between the source and destination. Note that if all link costs are the same, the least-cost path is also the **shortest path** (that is, the path crossing the smallest number of links between the source and the destination).

In Figure 4.4, for example, the least-cost path between nodes *A* (source) and *C* (destination) is along the path *ADEC*. (We will find it notationally easier to refer to the path in terms of the nodes on the path, rather than the links on the path.)

As a simple exercise, try finding the least-cost path from nodes *A* to *F*, and reflect for a moment on how you calculated that path. If you are like most people, you found the path from *A* to *F* by examining Figure 4.4, tracing a few routes from *A* to *F*, and somehow convincing yourself that the path you had chosen had the least cost among all possible paths. (Did you check all of the 12 possible paths between *A* and *F*? 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 algorithms,** since the algorithm must be aware of the cost of each link in the network. We will study a global link state algorithm in Section 4.2.1.

- 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. We will study a decentralized routing algorithm known as a **distance vector algorithm** in Section 4.2.2. It is called a distance vector algorithm because a node never actually knows a complete path from source to destination. Instead, it only knows the neighbor to which it should forward a packet in order to reach a given destination along the least-cost path, and the cost of that path from itself to the destination.

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, issues we will consider in Section 4.2.2.

Only two types of routing algorithms are typically used in the Internet: a dynamic global link state algorithm, and a dynamic decentralized distance vector algorithm. We cover these algorithms in Section 4.2.1 and 4.2.2, respectively. Other routing algorithms are surveyed briefly in Section 4.2.3.

4.2.1: *A Link State 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 link state algorithm. In practice this is accomplished by having each node **broadcast** the identities and costs of its attached links to *all* other routers in the network. This **link state broadcast** [Perlman 1999], can be accomplished without the nodes having to initially know the identities of all other nodes in the network. A node need only know the identities and costs to its directly attached neighbors; it will then learn about the topology of the rest of the network by receiving link state broadcasts from other nodes. (In Chapter 5, we will learn how a router learns the identities of its directly attached neighbors). The result of the nodes' link state broadcast is that all nodes have an identical and complete view of the network. Each node can then run the link state algorithm and compute the same set of least-cost paths as every other node.

The link state algorithm we present below is known as Dijkstra's algorithm, named after its inventor. A closely related algorithm is Prim's algorithm; see [Corman 1990] 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 A) to all other nodes in the network. Dijkstra's algorithm is iterative and has the property that after the kth iteration of the algorithm, the least-cost paths are known to k destination nodes, and among the least-cost paths to all destination nodes, these k paths will have the k smallest costs. Let us define the following notation:

- $c(i,j)$: link cost from node i to node j. If nodes i and j are not directly connected, then $c(i,j) = \infty$. We will assume for simplicity that $c(i,j)$ equals $c(j,i)$.
- $D(v)$: cost of the path from the source node to destination v that has currently (as of this iteration of the algorithm) the least cost
- $p(v)$: previous node (neighbor of v) along the current least-cost path from the source to v
- N: set of nodes whose least-cost path from the source is definitively known

The link state 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 to every other node in the network.

Link State (LS) Algorithm:

```
1 Initialization:
2  N = {A}
3  for all nodes v
4   if v adjacent to A
5    then D(v) = c(A,v)
6    else D(v) = ∞
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 all v adjacent to 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 shortest path cost to w plus cost from w to v */
15 **until** all nodes in N

As an example, let's consider the network in Figure 4.4 and compute the least-cost paths from A to all possible destinations. A tabular summary of the algorithm's computation is shown in Table 4.2, where each line in the table gives the values of the algorithm's variables at the end of the iteration.

Table 4.2: Running the link state algorithm on the network in Figure 4.4

step	*N*	*D(B),p(B)*	*D(C),p(C)*	*D(D),p(D)*	*D(E),p(E)*	*D(F),p(F)*
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D		2,D	∞
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					

Let's consider the few first steps in detail:

- **In the initialization step,** the currently known least-cost path from *A* to its directly attached neighbors, *B*, *C*, and *D* are initialized to 2, 5, and 1 respectively. Note in particular that the cost to *C* 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 *A* to *C*. The costs to *E* and *F* are set to infinity because they are not directly connected to *A*.
- **In the first iteration,** we look among those nodes not yet added to the set *N* and find that node with the least cost as of the end of the previous iteration. That node is *D*, with a cost of 1, and thus *D* is added to the set *N*. 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 4.2. The cost of the path to *B* is unchanged. The cost of the path to *C* (which was 5 at the end of the initialization) through node *D* is found to have a cost of 4. Hence this lower cost path is selected and *C*'s predecessor along the shortest path from *A* is set to *D*. Similarly, the cost to *E* (through *D*) is computed to be 2, and the table is updated accordingly.
- **In the second iteration,** nodes *B* and *E* are found to have the least path costs (2), and we break the tie arbitrarily and add *E* to the set *N* so that *N* now contains *A*, *D*, and *E*. The cost to the remaining nodes not yet in *N*, that is, nodes *B*, *C*, and *F*, are updated via line 12 of the LS algorithm, yielding the results shown in the third row in the Table 4.2.
- 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 have *its* predecessor, and so in this manner we can construct the entire path from the source to all destinations.

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 above implementation of the link state 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 4.5 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*(*A,B*) equals *c*(*B,A*) only if the load carried on both directions on the *AB* link is the same. In this example, node *D* originates a unit of traffic destined for *A*, node *B* also originates a unit of traffic destined for *A*, and node *C* injects an amount of

traffic equal to *e*, also destined for *A*. The initial routing is shown in Figure 4.5(a) with the link costs corresponding to the amount of traffic carried.

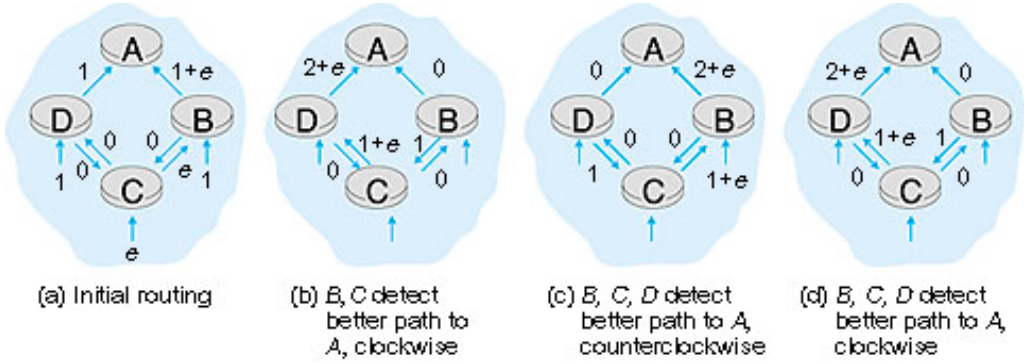


Figure 4.5: Oscillations with link state (LS) routing

When the LS algorithm is next run, node *C* determines (based on the link costs shown in Figure 4.5a) that the clockwise path to *A* has a cost of 1, while the counterclockwise path to *A* (which it had been using) has a cost of 1 + *e*. Hence *C*'s least-cost path to *A* is now clockwise. Similarly, *B* determines that its new least-cost path to *A* is also clockwise, resulting in costs shown in Figure 4.5b. When the LS algorithm is run next, nodes *B*, *C*, and *D* all detect a zero-cost path to *A* in the counterclockwise direction, and all route their traffic to the counterclockwise routes. The next time the LS algorithm is run, *B*, *C*, and *D* all then route their traffic to the clockwise routes.

What can be done to prevent such oscillations (which can occur in any 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 all routers do not run the LS algorithm at the same time. This seems a more reasonable solution, since we would hope that even if routers run the LS algorithm with the same periodicity, the execution instance of the algorithm would not be the same at each node. Interestingly, researchers have recently noted 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 to purposefully introduce randomization into the period between execution instants of the algorithm at each node.

Having now studied the link state algorithm, let's next consider the other major routing algorithm that is used in practice today--the distance vector routing algorithm.

4.2.2: A Distance Vector Routing Algorithm

While 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 may then distribute 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, we will see that the algorithm is 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 lock step with each other. We'll see that an asynchronous, iterative, self terminating, distributed algorithm is much more interesting and fun than a centralized algorithm!

The principal data structure in the DV algorithm is the **distance table** maintained at each node. Each node's distance table has a row for each destination in the network and a column for each of its directly attached neighbors. Consider a node *X* that is interested in routing to destination *Y* via its directly attached neighbor *Z*. Node *X*'s **distance table entry,** $D^X(Y,Z)$ is the sum of the cost of the direct one-hop link between *X* and *Z*, $c(X,Z)$, plus neighbor *Z*'s currently known minimum-cost path from itself (*Z*) to *Y*. That is:

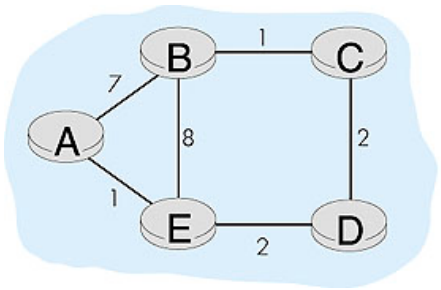
$$D^X(Y,Z) = c(X,Z) + \min_w\{D^Z(Y,w)\}$$

The $\min_w$ term in the equation is taken over all of *Z*'s directly attached neighbors (including *X*, as we shall soon see).

The equation suggests the form of the neighbor-to-neighbor communication that will take place in the DV algorithm--each node must know the cost of each of its neighbors' minimum-cost path to each destination.

Thus, whenever a node computes a new minimum cost to some destination, it must inform its neighbors of this new minimum cost.

Before presenting the DV algorithm, let's consider an example that will help clarify the meaning of entries in the distance table. Consider the network topology and the distance table shown for node *E* in Figure 4.6. This is the distance table in node *E* once the DV algorithm has converged. Let's first look at the row for destination *A*.



$D^E()$	cost to destination via A	B	D
A	①	14	5
B	7	8	⑤
C	6	9	④
D	4	11	②

destination

Figure 4.6: A distance table example

- Clearly the cost to get to *A* from *E* via the direct connection to *A* has a cost of 1. Hence $D^E(A,A) = 1$.
- Let's now consider the value of $D^E(A,D)$--the cost to get from *E* to *A*, given that the first step along the path is *D*. In this case, the distance table entry is the cost to get from *E* to *D* (a cost of 2) plus whatever the minimum cost it is to get from *D* to *A*. Note that the minimum cost from *D* to *A* is 3--a path that passes right back through *E*! Nonetheless, we record the fact that the minimum cost from *E* to *A* given that the first step is via *D* has a cost of 5. We're left, though, with an uneasy feeling that the fact that the path from *E* via *D* loops back through *E* may be the source of problems down the road (it will!).
- Similarly, we find that the distance table entry via neighbor *B* is $D^E(A,B) = 14$. Note that the cost is *not* 15. (Why?)

A circled entry in the distance table gives the cost of the least-cost path to the corresponding destination (row). The column with the circled entry identifies the next node along the least-cost path to the destination. Thus, a node's **routing table** (which indicates which outgoing link should be used to forward packets to a given destination) is easily constructed from the node's distance table.

In discussing the distance table entries for node *E* above, we informally took a global view, knowing the costs of all links in the network. The distance vector algorithm we will now present is *decentralized* and does not use such global information. Indeed, the only information a node will have are the costs of the links to its directly attached neighbors, and information it receives from these directly attached neighbors. The distance vector algorithm we will study is also known as the Bellman-Ford algorithm, after its inventors. It is used in many routing protocols in practice, including: Internet BGP, ISO IDRP, Novell IPX, and the original ARPAnet.

Distance Vector (DV) algorithm

At each node, *X*:

```
1 Initialization:
2  for all adjacent nodes v:
3   D^X (*,v) = ∞       /* the * operator means "for all rows" */
4   D^X (v,v) = c(X,v)
5  for all destinations, y
6   send min_w D(y,w) to each neighbor  /* w over all X's neighbors */
7
8 loop
9   wait (until I see a link cost change to neighbor V
10       or until I receive an update from neighbor V)
11
12  if (c(X,V) changes by d)
13    /* change cost to all dest's via neighbor v by d */
14    /* note: d could be positive or negative */
15    for all destinations y: D^X (y,V) = D^X (y,V) + d
16
17  else if (update received from V wrt destination Y)
```

18 /* shortest path from V to some Y has changed */
19 /* V has sent a new value for its $\min_w D^V(Y,w)$ */
20 /* call this received new value "newval" */
21 for the single destination y: $D^X(Y,V) = c(X,V) + newval$
22
23 **if** we have a new $\min_w D^X(Y,w)$for any destination Y
24 send new value of $\min_w D^X(Y,w)$ to all neighbors
25
26 **forever**

The key steps are lines 15 and 21, where a node updates its distance table entries in response to either a change of cost of an attached link or the receipt of an update message from a neighbor. The other key step is line 24, where a node sends an update to its neighbors if its minimum cost path to a destination has changed.

Figure 4.7 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 messages from their neighbors, compute new distance table entries, and inform their neighbors of any changes in their new least path costs. After studying this example, you should convince yourself that the algorithm operates correctly in an asynchronous manner as well, with node computations and update generation/reception occurring at any times.

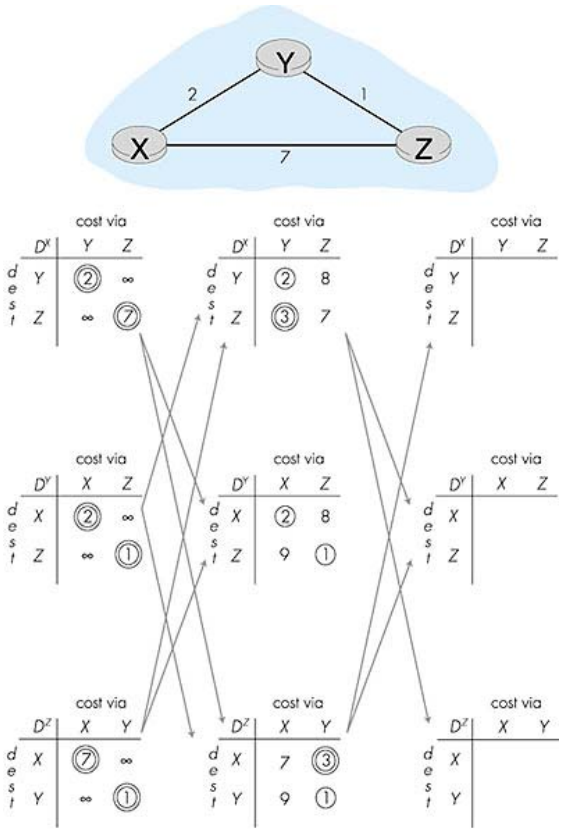


Figure 4.7: Distance vector (DV) algorithm: Example

The circled distance table entries in Figure 4.7 show the current minimum path cost to a destination. A double-circled entry indicates that a new minimum cost has been computed (in either line 4 of the DV algorithm (initialization) or line 21). In such cases an update message will be sent (line 24 of the DV algorithm) to the node's neighbors as represented by the arrows between columns in Figure 4.7.

The leftmost column in Figure 4.7 shows the distance table entries for nodes *X*, *Y*, and *Z* after the initialization step.

Let's now consider how node *X* computes the distance table shown in the middle column of Figure 4.7 after receiving updates from nodes *Y* and *Z*. As a result of receiving the updates from *Y* and *Z*, *X* computes in line 21 of the DV algorithm:

$$D^X(Y,Z) = c(X,Z) + \min_w D^Z(Y,w)$$
$$= 7 + 1$$
$$= 8$$

$$D^X(Z,Y) = c(X,Y) + \min_w D^Y(Z,w)$$
$$= 2 + 1$$
$$= 3$$

It is important to note that the only reason that X knows about the terms $\min_w D^Z(Y,w)$ and $\min_w D^Y(Z,w)$ is because nodes Z and Y have sent those values to X (and are received by X in line 10 of the DV algorithm). As an exercise, verify the distance tables computed by Y and Z in the middle column of Figure 4.7.

The value $D^X(Z,Y) = 3$ means that X's minimum cost to Z has changed from 7 to 3. Hence, X sends updates to Y and Z informing them of this new least cost to Z. Note that X need not update Y and Z about its cost to Y since this has not changed. Note also that although Y's recomputation of its distance table in the middle column of Figure 4.7 *does* result in new distance entries, it *does not* result in a change of Y's least-cost path to nodes X and Z. Hence Y does *not* send updates to X and Z.

The process of receiving updated costs from neighbors, recomputation of distance table entries, and updating 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 distance table calculations will occur and the algorithm enters a quiescent state; that is, all nodes are performing the wait in line 9 of the DV algorithm. The algorithm would remain in the quiescent state until a link cost changes, as discussed below.

The 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 (line 12), it updates its distance table (line 15) and, if there's a change in the cost of the least-cost path, updates its neighbors (lines 23 and 24). Figure 4.8 illustrates this behavior for 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 (row) X.

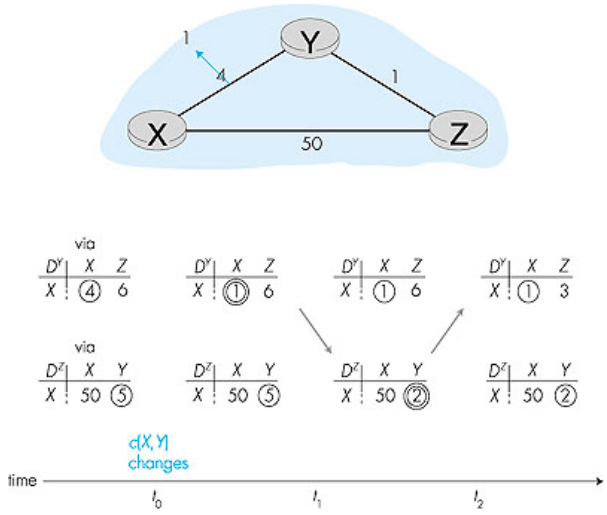


Figure 4.8: Link-cost change: Good news travels fast

- At time t_0, Y detects the link-cost change (the cost has changed from 4 to 1) and informs its neighbors of this change since the cost of the minimum cost path has changed.
- At time t_1, Z receives the update from Y and then updates its table. Since it computes a new least cost to X (it has decreased from a cost of 5 to a cost of 2), it informs its neighbors.
- At time t_2, Y receives Z's update and updates its distance table. Y's least costs have not changed (although its cost to X via Z has changed) and hence Y does *not* send any message to Z. The algorithm comes to a quiescent state.

In Figure 4.8, 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 fast 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 4.9.

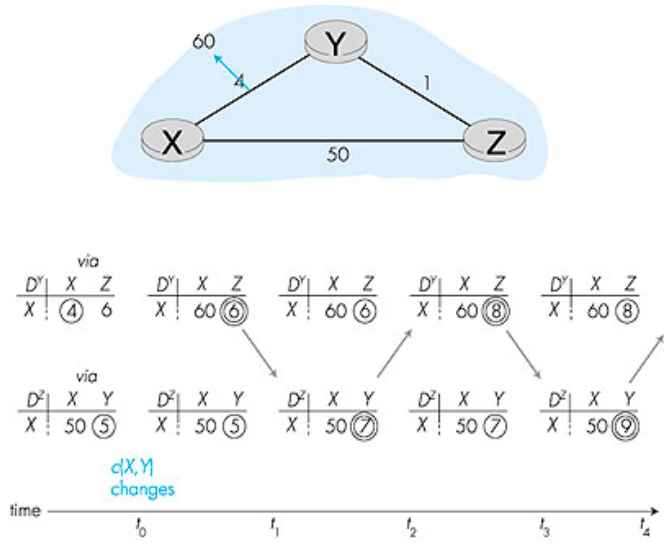


Figure 4.9: Link-cost changes: Bad news travels slowly and causes loops

- 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 6 via node Z. 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 arriving at Y or Z as of t_1 and destined for X, will bounce back and forth between these two nodes forever (or until the routing tables are changed).
- Since node Y has computed a new minimum cost to X, it informs Z of this new cost at time t_1.
- Sometime after t_1, Z receives the new least cost to X via Y (Y has told Z that Y's new minimum cost is 6). Z knows it can get to Y with a cost of 1 and hence computes a new least cost to X (still via Y) of 7. Since Z's least cost to X has increased, it then informs Y of its new cost at t_2.
- In a similar manner, Y then updates its table and informs Z of a new cost of 8. Z then updates its table and informs Y of a new cost of 9, etc.

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 illustrated in Figure 4.9 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 (Z's) distance to X is infinity. 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). Figure 4.10 illustrates how poisoned reverse solves the particular looping problem we encountered before in Figure 4.9. As a result of the poisoned reverse, Y's distance table indicates an infinite cost when routing to X via Z (the result of Z having informed Y that Z's cost to X was infinity). When the cost of the XY 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 this change in cost. After receiving the update at t_1, Z immediately shifts its route to X to be via the direct ZX link at a cost of 50. Since this is a new least-cost to X, and since the path no longer passes through Y, Z informs Y of this new least-cost path to X at t_2. After receiving the update from Z, Y updates its distance table to route to X via Z at a least cost of 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 it (Y) has an infinite cost to get to X. The algorithm becomes quiescent after t_4, with distance table entries for destination X shown in the rightmost column in Figure 4.10.

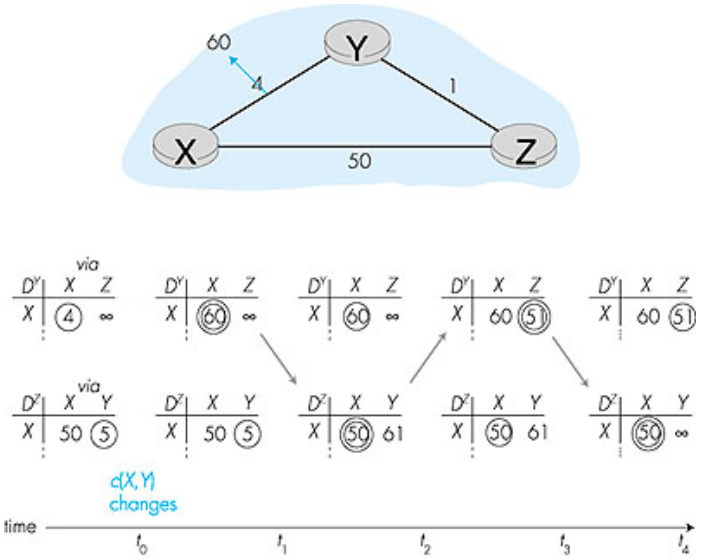


Figure 4.10: Poisoned reverse

Does poison 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, as we saw in Figure 4.10) will not be detected by the poison reverse technique.

A Comparison of Link State and Distance Vector Routing Algorithms

Let's conclude our study of link state and distance vector algorithms with a quick comparison of some of their attributes.

- *Message complexity.* We have seen that LS requires each node to know the cost of each link in the network. This requires $O(nE)$ messages to be sent, where n is the number of nodes in the network and E is the number of links. 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 $O(nE)$ messages, and that it potentially suffers from oscillations. The DV algorithm can converge slowly (depending on the relative path costs, as we saw in Figure 4.10) 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 LS broadcast packets it receives as part of a link state broadcast. But an LS node is only computing its own routing tables; other nodes are performing the 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/all destinations. (Indeed, in 1997, a malfunctioning router in a small ISP provided national backbone routers with erroneous routing tables. 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 a "winner" over the other; as we will see in Section 4.4, both algorithms are used in the Internet.

4.2.3: 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.

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. One of the simplest routing algorithms proposed is

hot potato routing. The algorithm derives its name from its behavior--a router tries to get rid of (forward) an outgoing packet as soon as it can. It does so by forwarding it on *any* outgoing link that is not congested, regardless of destination.

Another broad class of routing algorithms are 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]. Let us define λ_{ij} as the amount of traffic (for example, in packets/sec) entering the network for the first time at node *i* and destined for node *j*. The set of flows, $\{\lambda_{ij}\}$ for all *i,j*, is sometimes referred to as the network **traffic matrix**. In a network flow problem, traffic flows must be assigned to a set of network links subject to constraints such as:

- The sum of the flows between all source destination pairs passing though link *m* must be less than the capacity of link *m*.
- The amount of λ_{ij} traffic entering any router *r* (either from other routers, or directly entering that router from an attached host) must equal the amount of λ_{ij} traffic leaving the router either via one of *r*'s outgoing links or to an attached host at that router. This is a flow conservation constraint.

Let us define λ_{ij}^m as the amount of source *i*, destination *j* traffic passing through link *m*. The optimization problem then is to find the set of link flows, $\{\lambda_{ij}^m\}$ for all links *m* and all sources, *i*, and destinations, *j*, that satisfies the constraints above and optimizes a performance measure that is a function of $\{\lambda_{ij}^m\}$. The solution to this optimization problem then defines the routing used in the network. For example, if the solution to the optimization problem is such that $\lambda_{ij}^m = \lambda_{ij}$ for some link *m*, then all *i*-to-*j* traffic will be routed over link *m*. In particular, if link *m* is attached to node *i*, then *m* is the first hop on the optimal path from source *i* to destination *j*.

But what performance function should be optimized? There are many possible choices. If we make certain assumptions about the size of packets and the manner in which packets arrive at the various routers, we can use the so-called M/M/1 queuing theory formula [Kleinrock 1975] to express the average delay at link *m* as:

$$D_m = \frac{1}{R_m - \sum_i \sum_j \lambda_{ij}^m}$$

where R_m is link *m*'s capacity (measured in terms of the average number of packets/sec it can transmit) and $\Sigma_i \Sigma_j \lambda_{ij}^m$ is the total arrival rate of packets (in packets/ sec) that arrive to link *m*. The overall network-wide performance measure to be optimized might then be the sum of all link delays in the network, or some other suitable performance metric. A number of elegant distributed algorithms exist for computing the optimum link flows (and hence the routing paths, as discussed above). The reader is referred to [Bertsekas 1991] for a detailed study of these algorithms.

The final 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, we will see that there are a number of similarities, at least as far as the path finding algorithm (routing algorithm) is concerned. Our goal here is to provide a brief introduction for this class of routing algorithms. The reader is referred to [Ash 1998; Ross 1995; Girard 1990] for a detailed discussion of this active research area.

The circuit-switched routing problem formulation is illustrated in Figure 4.11. Each link has a certain amount of resources (for example, bandwidth). The easiest (and a quite accurate) way to visualize this is to consider the link to be a bundle of circuits, with each call that is routed over the link requiring the dedicated use of one of the link's circuits. A link is thus characterized both by its total number of circuits, as well as the number of these circuits currently in use. In Figure 4.11, all links except *AB* and *BD* have 20 circuits; the number to the left of the number of circuits indicates the number of circuits currently in use.

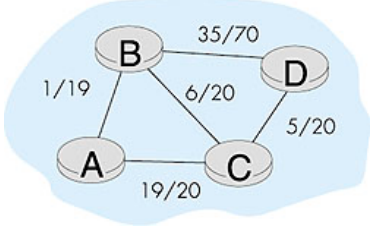


Figure 4.11: Circuit-switched routing

Suppose now that a call arrives at node *A*, destined to node *D*. What path should be taken? In **shortest path first (SPF)** routing, the shortest path (least number of links traversed) is taken. We have already seen how the Dijkstra LS algorithm can be used to find shortest-path routes. In Figure 4.11, either the *ABD* or *ACD* path would thus be taken. In **least loaded path** (LLP) routing, the load at a link is defined as the ratio of the number of used circuits at the link and the total number of circuits at that link. The path load is the maximum of the loads of all links in the path. In LLP routing, the path taken is that with the smallest path load. In Figure 4.11, the LLP path is *ABCD*. In **maximum free circuit (MFC)** routing, the number of free circuits associated with a path is the minimum of the number of free circuits at each of the links on a path. In MFC routing, the path with the maximum number of free circuits is taken. In Figure 4.11, the path *ABD* would be taken with MFC routing.

Given these examples from the circuit-switching world, we see that the path selection algorithms have much the same flavor as LS routing. All nodes have complete information about the network's link states. Note, however, that the potential consequences of old or inaccurate state information are more severe with circuit-oriented routing--a call may be routed along a path only to find that the circuits it had been expecting to be allocated are no longer available. In such a case, the call setup is blocked and another path must be attempted. Nonetheless, the main differences between connection-oriented, circuit-switched routing and connectionless packet-switched routing come not in the path-selection mechanism, but rather in the actions that must be taken when a connection is set up, or torn down, from source to destination.

4.3: Hierarchical Routing

In the previous section, we 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 homogenous 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 the routing table information (for example, link-state updates or least-cost path changes) becomes prohibitive. Today's public Internet consists of millions of interconnected routers and more than 50 million hosts. Storing routing table entries to each of these hosts and routers would clearly require enormous amounts of memory. The overhead required to broadcast link state updates among millions of routers would leave no bandwidth left for sending data packets! A distance vector algorithm that iterated among millions 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 engineers 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 the networks' 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 aggregating routers into regions or **autonomous systems** (**AS**s). 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 previous section. The routing algorithm running within an autonomous system is called an **intraautonomous 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 routing packets to destinations outside the AS. Routers in an AS that have the responsibility of routing packets to destinations outside the AS are called **gateway routers**. In order for gateway routers to route packets from one AS to another (possibly passing through multiple other ASs before reaching the destination AS), the gateways must know how to route (that is, determine routing paths) among themselves. The routing algorithm that gateways use to route among the various ASs is known as an **inter-autonomous system routing protocol**.

In summary, the problems of scale and administrative authority are solved by defining autonomous systems. Within an AS, all routers run the same intra-autonomous system routing protocol. Special gateway routers in the various ASs run an inter-autonomous system routing protocol that determines routing paths among the ASs. The problem of scale is solved since an intra-AS router need only know about routers within its AS and the gateway router(s) in its AS. The problem of administrative authority is solved since an organization can run whatever intra-AS routing protocol it chooses, as long as the AS's gateway(s) is able to run an inter-AS routing protocol that can connect the AS to other ASs.

Figure 4.12 illustrates this scenario. Here, there are three routing ASs, A, B, and C. Autonomous system A has four routers, A.a, A.b, A.c, and A.d, which run the intra-AS routing protocol used within autonomous system A. These four routers have complete information about routing paths within autonomous system A. Similarly, autonomous systems B and C have three and two routers, respectively. Note that the intra-AS routing protocols

running in A, B, and C need not be the same. The gateway routers are A.a, A.c, B.a, and C.b. In addition to running the intra-AS routing protocol in conjunction with other routers in their ASs, these four routers run an inter-AS routing protocol among themselves. The topological view they use for their inter-AS routing protocol is shown at the higher level, with "links" shown in blue. Note that a "link" at the higher layer may be an actual physical link, for example, the link connection A.c and B.a, or a logical link, such as the link connecting A.c and A.a. Figure 4.12 also illustrates that the gateway router A.c must run an intra-AS routing protocol with its neighbors A.b and A.d, as well as an inter-AS protocol with gateway router B.a.

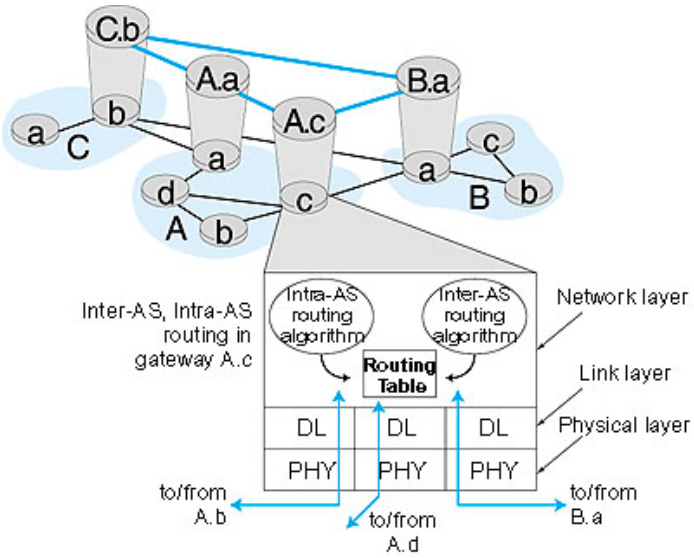


Figure 4.12: Intra-AS and Inter-AS routing

Suppose now that a host h1 attached to router A.d needs to route a packet to destination h2 in autonomous system B, as shown in Figure 4.13. Assuming that A.d's routing table indicates that router A.c is responsible for routing its (A.d's) packets outside the AS, the packet is first routed from A.d to A.c using A's intra-AS routing protocol. It is important to note that router A.d does not know about the internal structure of autonomous systems B and C and indeed need not even know about the topology connecting autonomous systems A, B, and C. Router A.c will receive the packet and see that it is destined to an autonomous system outside of A. A.c's routing table for the inter-AS protocol would indicate that a packet destined to autonomous system B should be routed along the A.c to B.a link. When the packet arrives at B.a, B.a's *inter-AS* routing sees that the packet is destined for autonomous system B. The packet is then "handed over" to the *intra-AS* routing protocol within B, which routes the packet to its final destination, h2. In Figure 4.13, the portion of the path routed using A's intra-AS protocol is shown on the lower plane with a dotted line, the portion using the inter-AS routing protocol is shown in the upper plane as a solid line, and the portion of the path routed using B's intra-AS protocol is shown on the lower plane with a dotted line. We will examine specific inter-AS and intra-AS routing protocols used in the Internet in Section 4.5.

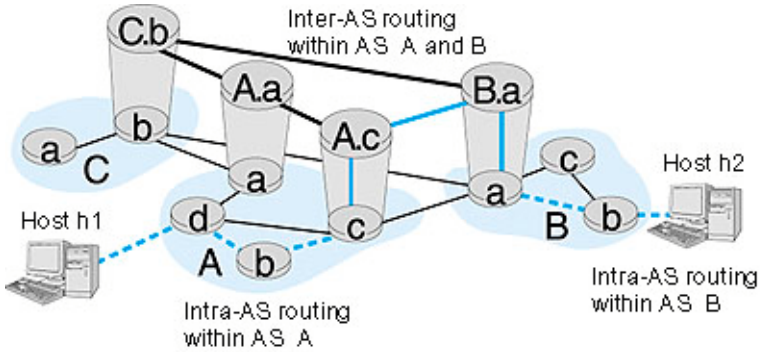


Figure 4.13: The route from A.d to B.b: intra-AS and inter-AS path segments

4.4: Internet Protocol

So far in this chapter we've focused on underlying principles of the network layer, without reference to any specific network architecture. We've discussed various network-layer service models, the routing algorithms commonly used to deter mine paths between source and destination, and the use of hierarchy to address the problem of scale. In this section, we turn our attention to the network layer of the Internet, the pieces of which are often collectively referred to as the IP layer (named after the Internet's IP protocol). We'll see, though, that the IP protocol itself is just one piece (albeit a very important piece) of the Internet's network layer.

As noted in Section 4.1, the Internet's network layer provides connectionless datagram service rather than virtual-circuit service. When the network layer at the sending host receives a segment from the transport layer, it encapsulates the segment within an IP datagram, writes the destination host address as well as other fields in the datagram, and sends the datagram to the first router on the path toward the destination host. Our analogy from Chapter 1 was that this process is similar to a person writing a letter, inserting the letter in an envelope, writing the destination address on the envelope, and dropping the envelope into a mailbox. Neither the Internet's network layer nor the postal service make any preliminary contact with the destination before moving its "parcel" (datagram or letter, respectively) toward the destination. Furthermore, as discussed in Section 4.1, the Internet's network-layer service and the postal delivery service both provide so-called best-effort service: neither guarantee that a parcel will arrive within a certain time at the destination, nor does either guarantee that a series of parcels will arrive in the order sent. Indeed, neither even guarantee that a parcel will ever arrive at its destination!

As shown in Figure 4.14, the network layer in a datagram-oriented network such as the Internet has three major components.

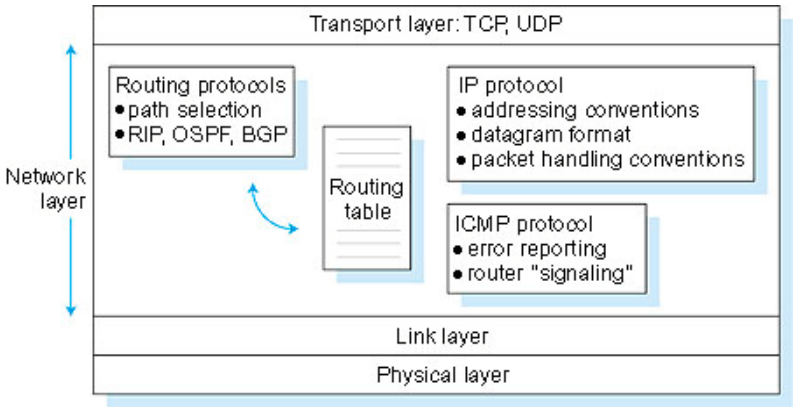


Figure 4.14: A look inside the Internet's network layer

- The first component is the network protocol, which defines network-layer addressing, the fields in the datagram (that is, the network-layer PDU), and the actions taken by routers and end systems on a datagram based on the values in these fields. The network protocol in the Internet is called the Internet Protocol, or more commonly, the **IP Protocol**. There are two versions of the IP protocol in use today. We'll examine the widely deployed Internet Protocol version 4, more commonly known simply as **IPv4** [RFC 791] in Sections 4.4.1 through 4.4.4. In Section 4.7 we'll examine IP version 6 [RFC 2373; RFC 2460], which has been proposed to replace IPv4 in upcoming years.
- The second major component of the network layer is the path determination component; it determines the route a datagram follows from source to destination. We saw in Section 4.2 that routing protocols compute the routing tables that are used to route packets through the network. We'll study the Internet's path determination component in Section 4.5.
- The final component of the network layer is a facility to report errors in datagrams and respond to requests for certain network-layer information. We'll cover the Internet's network-layer error and information reporting protocol, ICMP, in Section 4.4.5.

4.4.1: IPv4 Addressing

Let's begin our study of IPv4 by considering IPv4 addressing. Although addressing may seem a rather straightforward and perhaps tedious topic, the coupling between addressing and network-layer routing is both crucial and subtle. Excellent treatments of IPv4 addressing are [Semeria 1996] and the first chapter in [Stewart 1999].

Before discussing IP addressing, however, we'll need to say a few words about how hosts and routers are connected into the network. A host typically has only a single link into the network. When IP in the host wants to send a datagram, it will do so over this link. The boundary between the host and the physical link is called an **interface.** A router, on the other hand, is fundamentally different from a host. Because a router's job is to receive a datagram on an "incoming" link and forward the datagram on some "outgoing" link, a router necessarily has two or more links to which it is connected. The boundary between the router and any one of its links is also called an interface. A router thus has multiple interfaces, one for each of its links. Because every host and router is capable of sending and receiving IP datagrams, IP requires each interface to have an IP address. Thus, an IP address is technically associated with an interface, rather than with the host or router containing that interface

Each IP address is 32 bits long (equivalently, four bytes), and there are thus a total of 2^{32} possible IP addresses. These addresses are typically written in so-called **dotted-decimal notation,** in which each byte of the address is written in its decimal form and is separated by a period ("dot") from other bytes in the address. For example, consider the IP address 193.32.216.9. The 193 is the decimal equivalent of the first eight bits of the address; the 32 is the decimal equivalent of the second eight bits of the address, and so on. Thus, the address 193.32.216.9 in binary notation is:

11000001 00100000 11011000 00001001.

Each interface on every host and router in the global Internet must have an IP address that is globally unique. These addresses cannot be chosen in a willy-nilly manner, however. To a large extent, an interface's IP address will be determined by the "network" to which it is connected. In this context, the term "network" does not refer to the general infrastructure of hosts, routers, and links that make up a network. Instead, the term has a very precise meaning that is closely tied to IP addressing, as we will see. Figure 4.15 provides an example of IP addressing and interfaces. In this figure, one router (with three interfaces) is used to interconnect seven hosts. Take a close look at the IP addresses assigned to the host and router interfaces; there are several things to be noted. The three hosts in the upper-left portion of Figure 4.15, and the router interface to which they are connected all have an IP address of the form 223.1.1.xxx. That is, they share a common leftmost 24 bits of their IP address. They are also interconnected to each other by a single physical link (in this case, a broadcast link such as an Ethernet cable to which they are all physically attached) with no intervening routers. In the jargon of IP, the interfaces in these hosts and the upper-left interface in the router form an **IP network** or more simply a **network.** The 24 address bits that they share in common constitute the network portion of their IP address; the remaining eight bits are the host portion of the IP address. (We would prefer to use the terminology "interface part of the address" rather than "host part of the address" because an IP address is really for an interface rather than a host; but the terminology "host part" is commonly used in practice.) The network itself also has an address: 223.1.1.0/24, where the "/24" notation, sometimes known as a **network mask**, indicates that the leftmost 24 bits of the 32-bit quantity define the network address. These leftmost bits that define the network address are also often referred to as the **network prefix**. The network 223.1.1.0/24 thus consists of the three host interfaces (223.1.1.1, 223.1.1.2, and 223.1.1.3) and one router interface (223.1.1.4). Any additional hosts attached to the 223.1.1.0/24 network would be *required* to have an address of the form 223.1.1.xxx. There are two additional networks shown in Figure 4.15: the 223.1.2.0/24 network and the 223.1.3.0/24 network. Figure 4.16 illustrates the three IP networks present in Figure 4.15.

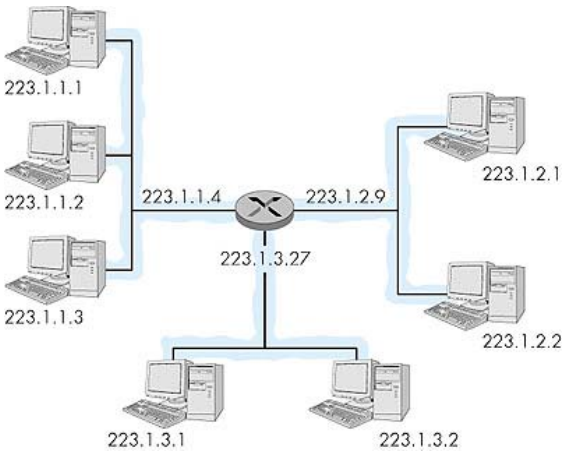


Figure 4.15: Interface addresses

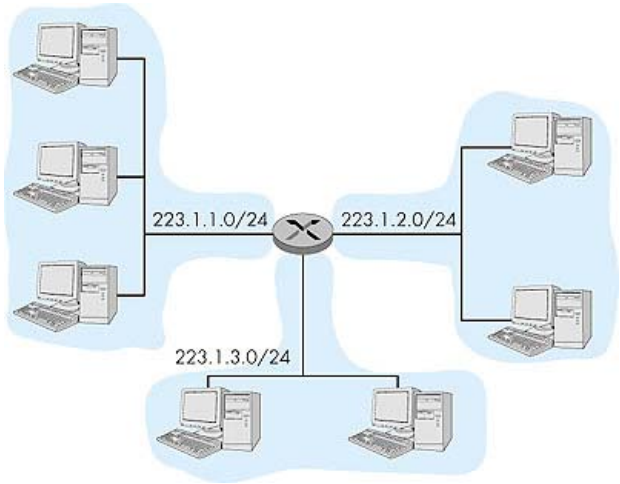


Figure 4.16: Network addresses

The IP definition of a "network" is not restricted to Ethernet segments that connect multiple hosts to a router interface. To get some insight here, consider Figure 4.17, which shows three routers that are interconnected with each other by point-to-point links. Each router has three interfaces, one for each point-to-point link, and one for the broadcast link that directly connects the router to a pair of hosts. What IP networks are present here? Three networks, 223.1.1.0/24, 223.1.2.0/24, and 223.1.3.0/24 are similar in spirit to the networks we encountered in Figure 4.15. But note that there are three additional networks in this example as well: one network, 223.1.9.0/24, for the interfaces that connect routers R1 and R2; another network, 223.1.8.0/24, for the interfaces that connect routers R2 and R3; and a third network, 223.1.7.0/24, for the interfaces that connect routers R3 and R1.

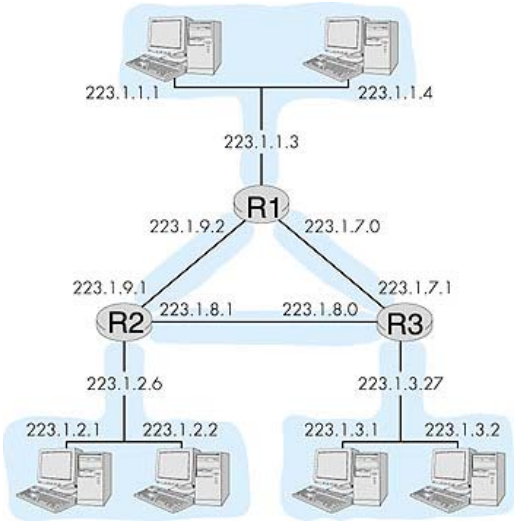


Figure 4.17: Three routers interconnecting six hosts

For a general interconnected system of routers and hosts, we can use the following recipe to define the networks in the system. We first detach each interface from its host or router. This creates islands of isolated networks, with interfaces terminating the endpoints of the isolated networks. We then call each of these isolated networks a network. If we apply this procedure to the interconnected system in Figure 4.17, we get six islands or networks. The current Internet consists of millions of such networks. The notion of a network and a network address is an important one, and plays a central role in the Internet's routing architecture.

Now that we have defined a network, we are ready to discuss IP addressing in more detail. The original Internet addressing architecture defined four classes of address, as shown in Figure 4.18. A fifth address class, beginning with 11110, was reserved for future use. For a class A address, the first eight bits identify the network, and the last 24 bits identify the interface within that network. Thus, within class A we can have up to 2^7 networks (the first of the eight bits is fixed as 0), each with up to 2^{24} interfaces. The class B address space allows for 2^{14} networks, with up to 2^{16} interfaces within each network. A class C address uses 21 bits to identify the network

and leaves only eight bits for the interface identifier. Class D addresses are reserved for so-called multicast addresses; we'll defer our discussion of class D addresses until Section 4.7.

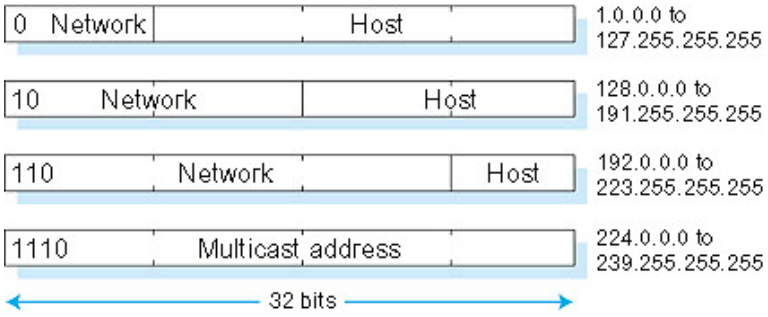


Figure 4.18: IPv4 address formats

The four address classes shown in Figure 4.18 (sometimes known as **classful addressing**) are no longer formally part of the IP addressing architecture. The requirement that the network portion of an IP address be exactly one, two, or three bytes long turned out to be problematic for supporting the rapidly growing number of organizations with small and medium-sized networks. A class C (/24) network could only accommodate up to 2^8 - 2 = 254 hosts (two of the 2^8 = 256 addresses are reserved for special use)--too small for many organizations. However, a class B (/16) network, which supports up 65,634 hosts was too large. Under classful addressing, an organization with, say, 2,000 hosts was typically allocated a class B (/16) network address. This led to a rapid depletion of the class B address space and poor utilization of the assigned address space. For example, the organization that used a class B address for its 2,000 hosts was allocated enough of the address space for up to 65,534 interfaces--leaving more than 63,000 unused addresses that could not be used by other organizations.

In 1993, the IETF standardized on **Classless Interdomain Routing (CIDR**--pronounced the same as "cider"**)** [RFC 1519]. With so-called CIDRized network addresses, the network part of an IP address can be any number of bits long, rather than being constrained to 8, 16, or 24 bits. A CIDRized network address has the dotted-decimal form *a.b.c.d/x*, where *x* indicates the number of leading bits in the 32-bit quantity that constitutes the network portion of the address. In our example above, the organization needing to support 2,000 hosts could be allocated a block of only 2,048 host addresses of the form *a.b.c.d*/21, allowing the approximately 63,000 addresses that would have been allocated and unused under classful addressing to be allocated to a different organization. In this case, the first 21 bits specify the organization's network address and are common in the IP addresses of all hosts in the organization. The remaining 11 bits then identify the specific hosts in the organization. In practice, the organization could further divide these 11 rightmost bits using a procedure known as **subnetting** [RFC 950] to create its own internal networks within the *a.b.c.d*/21 network.

Assigning Addresses

Having introduced IP addressing, a question that immediately comes to mind is how a host gets its own IP address. We have just learned that an IP address has two parts, a network part and a host part. The host part of the address can be assigned in several different ways, including:

- Manual configuration. The IP address is configured into the host (typically in a file) by the system administrator.
- **Dynamic Host Configuration Protocol (DHCP)** [RFC 2131]. DHCP is an extension of the BOOTP [RFC 1542] protocol, and is sometimes referred to as Plug and Play. With DHCP, a DHCP server in a network (for example, in a LAN) receives DHCP requests from a client and, in the case of dynamic address allocation, allocates an IP address back to the requesting client. DHCP is used extensively in LANs and in residential Internet access.

Obtaining a network address is not as simple. An organization's network administrator might first contact its ISP, which would provide addresses from a larger block of addressees that had already been allocated to the ISP. For example, the ISP may itself have been allocated the address block 200.23.16.0/20. The ISP, in turn could divide its address block into eight equal-size smaller address blocks and give one of these address blocks out to each of up to eight organizations that are supported by this ISP, as shown below. (We have underlined the network part of these addresses for visual convenience.)

ISP's block 11001000 00010111 00010000 00000000 200.23.16.0/20
Organization 0 11001000 00010111 00010000 00000000 200.23.16.0/23
Organization 1 11001000 00010111 00010010 00000000 200.23.18.0/23
Organization 2 11001000 00010111 00010100 00000000 200.23.20.0/23
...
Organization 7 11001000 00010111 00011110 00000000 200.23.30.0/23

Let's conclude our discussion of addressing by considering how an ISP itself gets a block of addresses. IP addresses are managed under the authority of The Internet Corporation for Assigned Names and Numbers (ICANN) [ICANN 2000] based on guidelines set forth in RFC 2050. The role of the nonprofit ICANN organization [NTIA 1998] is to allocate not only IP addresses, but also to manage the DNS root servers. It also has the very contentious job of assigning domain names and resolving domain name disputes. The actual assignment of addresses is now managed by regional Internet registries. As of mid-2000, there are three such regional registries: the American Registry for Internet Number (ARIN, which handles registrations for North and South America, as well as parts of Africa. ARIN has recently taken over a number of the functions previously provided by Network Solutions), the Reseaux IP Europeans (RIPE, which covers Europe and nearby countries), and the Asia Pacific Network Information Center (APNIC).

Before leaving our discussion of addressing, we want to mention that mobile hosts may change the network to which they are attached, either dynamically while in motion or on a longer time scale. Because routing is to a network first, and then to a host within the network, this means that the mobile host's IP address must change when the host changes networks. Techniques for handling such issues are now under development within the IETF and the research community [RFC 2002; RFC 2131].

Principles in Practice

This example of an ISP that connects eight organizations into the larger Internet also nicely illustrates how carefully allocated CIDRized addresses facilitate hierarchical routing. Suppose, as shown in Figure 4.19, that the ISP (which we'll call Fly-By-Night-ISP) advertises to the outside world that it should be sent any datagrams whose first 20 address bits match 200.23.16.0/20. The rest of the world need not know that within the address block 200.23.16.0/20 there are in fact eight other organizations, each with their own networks. This ability to use a single network prefix to advertise multiple networks is often referred to as **route aggregation** or **route summarization.**

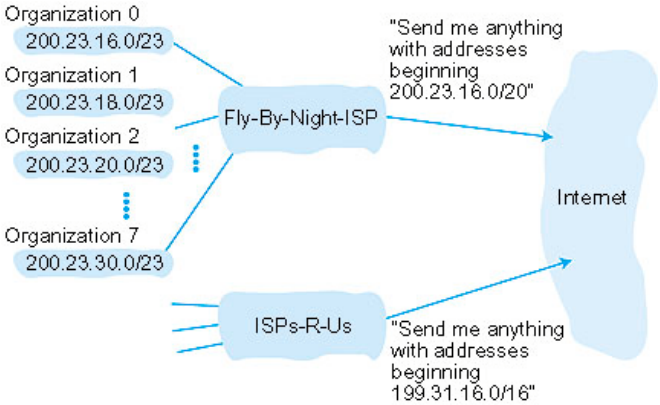


Figure 4.19: Hierarchical addressing and route aggregation

Route aggregation works extremely well when addresses are allocated in blocks to ISPs and then from ISPs to client organizations. But what happens when addresses are not allocated in such a hierarchical manner? What would happen, for example, if Organization 1 becomes discontent with the poor service provided by Fly-By-Night-ISP and decides to switch over to a new ISP, say, ISPs-R-Us? As shown in Figure 4.19, ISPs-R-Us owns the address block 199.31.0.0/16, but Organization 2's IP addresses are unfortunately outside of this address block. What should be done here? Certainly, Organization 1 could renumber all of its routers and hosts to have addresses within the ISPs-R-Us address block. But this is a costly solution, and Organization 1 might well choose to switch from ISPs-R-Us to yet another ISP in the future. The solution typically adopted is for Organization 1 to keep its IP addresses in 200.23.18.0/23. In this case, as shown in Figure 4.20, Fly-By-Night-ISP continues to advertise the address block 200.23.16.0/20 and ISPs-R-Us continues to advertise 199.31.0.0/16. However, ISPs-R-Us now *also* advertises the block of addresses for Organization 1, 200.23.18.0/23. When other routers in the larger Internet see the address blocks 200.23.16.0/20 (from Fly-By-Night-ISP) and 200.23.18.0/23 (from ISPs-R-Us) and want to route to an address in the block 200.23.18.0/23,

they will use a **longest prefix matching** rule, and route toward ISPs-R-Us, as it advertises the longest (more specific) address prefix that matches the destination address.

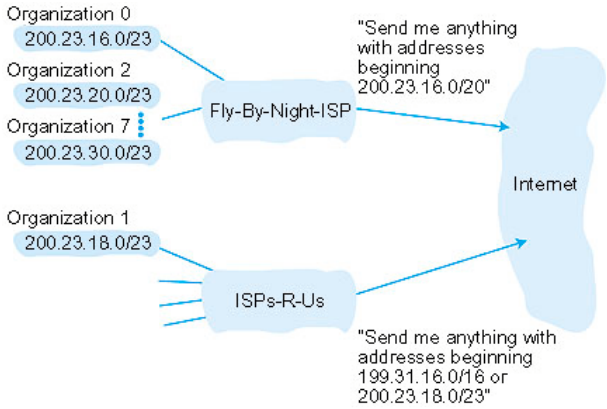


Figure 4.20: ISPs-R-Us has a more specific route to Organization 1

4.4.2: Transporting a Datagram from Source to Destination: Addressing and Routing

Now that we have defined interfaces and networks and have a basic understanding of IP addressing, we take a step back and examine how hosts and routers transport an IP datagram from source to destination. To this end, a high-level view of an IP datagram is shown in Figure 4.21. Every IP datagram has a source address field and a destination address field. The source host fills a datagram's source address field with its own 32-bit IP address. It fills the destination address field with the 32-bit IP address of the final destination host to which the datagram is being sent. The data field of the datagram is typically filled with a TCP or UDP segment. We'll discuss the remaining IP datagram fields a little later in this section.

Figure 4.21: The key fields of an IP datagram

Once the source host creates the IP datagram, how does the network layer transport the datagram from the source host to the destination host? The answer to this question depends on whether the source and destination reside on the same network (where the term "network" is used here in the precise, addressing sense discussed in Section 4.4.1). Let's consider this question in the context of the network shown in Figure 4.22. First suppose host *A* wants to send an IP datagram to host *B*, which resides on the same network, 223.1.1.0/24, as *A*. This is accomplished as follows. IP in host *A* first consults its internal routing table, shown in Figure 4.22, and finds an entry, 223.1.1.0/24, whose network address matches the leading bits in the IP address of host *B*. The routing table shows that the number of hops to network 223.1.1.0 is 1, indicating that *B* is on the very same network to which *A* itself is attached. Host *A* thus knows that destination host *B* can be reached directly via *A*'s outgoing interface, without the need for any intervening routers. Host *A* then passes the IP datagram to the link-layer protocol for the interface, which then has the responsibility of transporting the datagram to host *B*. (We'll study how the link layer transports a datagram between two interfaces on the same network in Chapter 5.)

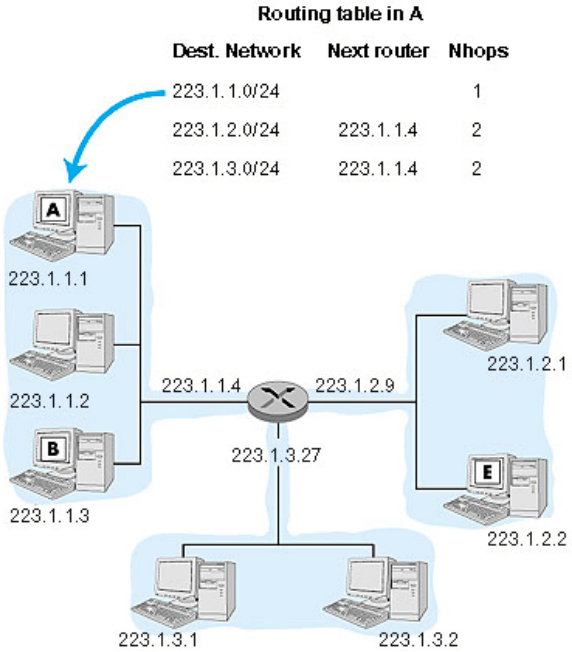


Figure 4.22: Routing table in host A

Let's next consider the more interesting case that host *A* wants to send a datagram to another host, say *E*, that is on a different network. Host *A* again consults its routing table and finds an entry, 223.1.2.0/24, whose network address matches the leading bits in the IP address of host *E*. Because the number of hops to the destination is 2, host *A* knows that the destination is on another network and thus an intervening router will necessarily be involved. The routing table also tells host *A* that in order to get the datagram to host *E*, host *A* should first send the datagram to IP address 223.1.1.4, the router interface to which *A*'s own interface is directly connected. IP in host *A* then passes the datagram down to the link layer and indicates to the link layer that it should send the datagram to IP address 223.1.1.4. It's important to note here that although the datagram is being sent (via the link layer) to the router's interface, the destination address of the datagram remains that of the ultimate destination (host *E*,) *not* that of the intermediate router interface.

The datagram is now in the router, and it is the job of the router to move the datagram toward its ultimate destination. As shown in Figure 4.23, the router consults it own routing table and finds an entry, 223.1.2.0/24, whose network address matches the leading bits in the IP address of host *E*. The routing table indicates that the datagram should be forwarded on router interface 223.1.2.9. Since the number of hops to the destination is 1, the router knows that destination host *E* is on the same network as its own interface, 223.1.2.9. The router thus moves the datagram to this interface, which then transmits the datagram to host *E*.

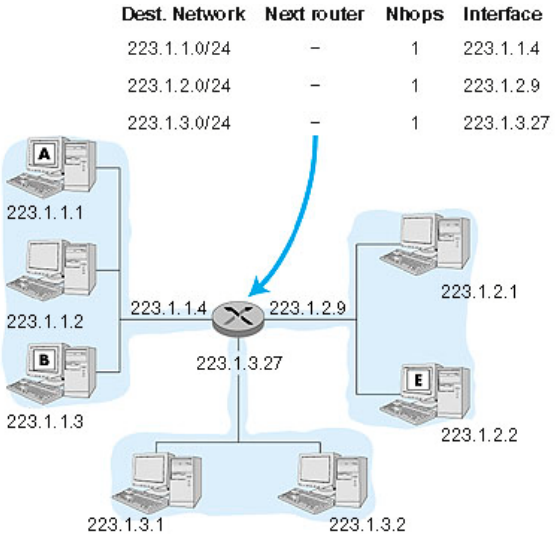


Figure 4.23: Routing table in router

In Figure 4.23, note that the entries in the "next router" column are all empty since each of the networks (223.1.1.0/24., 223.1.2.0/24, and 223.1.3.0/24) is directly attached to the router. In this case, there is no need to go through an intermediate router to get to a destination host. However, if host *A* and host *E* were separated by two routers, then within the routing table of the first router along the path from *A* to *B*, the appropriate row would indicate 2 hops to the destination and would specify the IP address of the second router along the path. The first router would then forward the datagram to the second router, using the link-layer protocol that connects the two routers. The second router would then forward the datagram to the destination host, using the link-layer protocol that connects the second router to the destination host.

You may recall from Chapter 1 that we said that routing a datagram in the Internet is similar to a person driving a car and asking gas station attendants at each intersection along the way how to get to the ultimate destination. It should now be clear why this an appropriate analogy for routing in the Internet. As a datagram travels from source to destination, it visits a series of routers. At each router in the series, it stops and asks the router how to get to its ultimate destination. Unless the router is on the same network as the ultimate destination, the routing table essentially says to the datagram: "I don't know exactly how to get to the ultimate destination, but I do know that the ultimate destination is in the direction of the link (analogous to a road) connected to one of my interfaces." The datagram then sets out on the link connected to this interface, arrives at a new router, and again asks for new directions.

From this discussion we see that the routing tables in the routers play a central role in routing datagrams through the Internet. But how are these routing tables configured and maintained for large networks with multiple paths between sources and destinations (such as in the Internet)? Clearly, these routing tables should be configured so that the datagrams follow "good" routes from source to destination. As you probably guessed, routing algorithms--like those studied in Section 4.2--have the job of configuring and maintaining the routing tables. We will discuss the Internet's routing algorithms in Section 4.5. But before moving on to routing algorithms, we cover three more important topics for the IP protocol, namely, the datagram format, datagram fragmentation, and the Internet Control Message Protocol (ICMP).

4.4.3: Datagram Format

The IPv4 datagram format is shown in Figure 4.24.

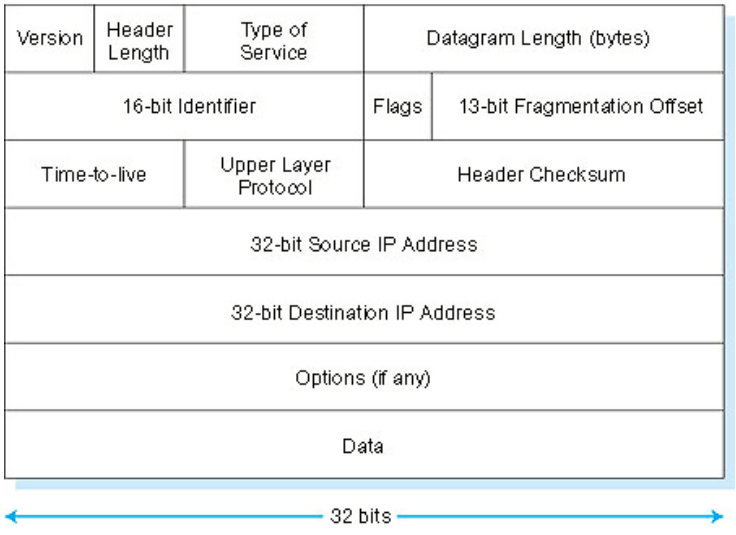


Figure 4.24: IPv4 datagram format

The key fields in the IPv4 datagram are the following:

- *Version Number.* These four bits specify the IP protocol version of the datagram. By looking at the version number, the router can then determine how to interpret the remainder of the IP datagram. Different versions of IP use different datagram formats. The datagram format for the current version of IP, IPv4, is shown in Figure 4.24. The datagram format for the new version of IP (IPv6) is discussed in Section 4.7.
- *Header Length.* Because an IPv4 datagram can contain a variable number of options (that are included in the IPv4 datagram header) these four bits are needed to determine where in the IP datagram the data

actually begins. Most IP datagrams do not contain options so the typical IP datagram has a 20-byte header.

- *TOS.* The type of service (TOS) bits were included in the IPv4 header to allow different "types" of IP datagrams to be distinguished from each other, presumably so that they could be handled differently in times of overload. When the network is overloaded, for example, it would be useful to be able to distinguish network-control datagrams (for example, see the ICMP discussion in Section 4.4.5) from datagrams carrying data (for example, HTTP messages). It would also be useful to distinguish real-time datagrams (for example, used by an IP telephony application) from non-real-time traffic (for example, FTP). More recently, one major routing vendor (Cisco) interprets the first three TOS bits as defining differential levels of service that can be provided by the router. The specific level of service to be provided is a policy issue determined by the router's administrator. We'll explore the topic of differentiated service in detail in Chapter 6.
- *Datagram Length.* This is the total length of the IP datagram (header plus data) measured in bytes. Since this field is 16 bits long, the theoretical maximum size of the IP datagram is 65,535 bytes. However, datagrams are rarely greater than 1,500 bytes and are often limited in size to 576 bytes.
- *Identifier, Flags, Fragmentation Offset.* These three fields have to do with so-called IP fragmentation, a topic we will consider in depth shortly. Interestingly, the new version of IP, IPv6, does not allow for fragmentation at routers.
- *Time-to-live.* The time-to-live (TTL) field is included to ensure that datagrams do not circulate forever (due to, for example, a long-lived router loop) in the network. This field is decremented by one each time the datagram is processed by a router. If the TTL field reaches 0, the datagram must be dropped.
- *Protocol.* This field is used only when an IP datagram reaches its final destination. The value of this field indicates the transport-layer protocol at the destination to which the data portion of this IP datagram will be passed. For example, a value of 6 indicates that the data portion is passed to TCP, while a value of 17 indicates that the data is passed to UDP. For a listing of all possible numbers, see RFC 1700. Note that the protocol number in the IP datagram has a role that is fully analogous to the role of the port number field in the transport-layer segment. The protocol number is the "glue" that binds the network and transport layers together, whereas the port number is the "glue" that binds the transport and application layers together. We will see in Chapter 5 that the link-layer frame also has a special field that binds the link layer to the network layer.
- *Header Checksum.* The header checksum aids a router in detecting bit errors in a received IP datagram. The header checksum is computed by treating each two bytes in the header as a number and summing these numbers using 1's complement arithmetic. As discussed in Section 3.3, the 1's complement of this sum, known as the Internet checksum, is stored in the checksum field. A router computes the Internet checksum for each received IP datagram and detects an error condition if the checksum carried in the datagram does not equal the computed checksum. Routers typically discard datagrams for which an error has been detected. Note that the checksum must be recomputed and restored at each router, as the TTL field, and possibly options fields as well, may change. An interesting discussion of fast algorithms for computing the Internet checksum is RFC 1071. A question often asked at this point is, why does TCP/IP perform error checking at both the transport and network layers? There are many reasons for this repetition. First, routers are not required to perform error checking, so the transport layer cannot count on the network layer to do the job. Second, TCP/UDP and IP do not necessarily both have to belong to the same protocol stack. TCP can, in principle, run over a different protocol (for example, ATM) and IP can carry data that will not be passed to TCP/UDP.
- *Source and Destination IP Address.* These fields carry the 32-bit IP address of the source and final destination for this IP datagram. The use and importance of the destination address is clear. Recall from Section 3.2 that the source IP address (along with the source and destination port numbers) is used at the destination host to direct the application data to the proper socket.
- *Options.* The options fields allow an IP header to be extended. Header options were meant to be used rarely--hence the decision to save overhead by not including the information in options fields in every datagram header. However, the mere existence of options does complicate matters--since datagram headers can be of variable length, one cannot determine *a priori* where the data field will start. Also, since some datagrams may require options processing and others may not, the amount of time needed to process an IP datagram at a router can vary greatly. These considerations become particularly important for IP processing in high-performance routers and hosts. For these reasons and others, IP options were dropped in the IPv6 header.
- *Data (payload).* Finally, we come to the last, and most important field--the *raison d'être* for the datagram in the first place! In most circumstances, the data field of the IP datagram contains the

transport-layer segment (TCP or UDP) to be delivered to the destination. However, the data field can carry other types of data, such as ICMP messages (discussed in Section 4.4.5).

Note that an IP datagram has a total of 20 bytes of header (assuming it has no options). If the datagram carries a TCP segment, then each (non-fragmented) datagram carries a total of 40 bytes of header (20 IP header bytes and 20 TCP header bytes) along with the application-layer message.

4.4.4: IP Fragmentation and Reassembly

We will see in Chapter 5 that not all link-layer protocols can carry packets of the same size. Some protocols can carry "big" packets, whereas other protocols can only carry "little" packets. For example, Ethernet packets can carry no more than 1,500 bytes of data, whereas packets for many wide-area links can carry no more than 576 bytes. The maximum amount of data that a link-layer packet can carry is called the **MTU (maximum transfer unit).** Because each IP datagram is encapsulated within the link-layer packet for transport from one router to the next router, the MTU of the link-layer protocol places a hard limit on the length of an IP datagram. Having a hard limit on the size of an IP datagram is not much of a problem. What is a problem is that each of the links along the route between sender and destination can use different link-layer protocols, and each of these protocols can have different MTUs.

To understand the problem better, imagine that *you* are a router that interconnects several links, each running different link-layer protocols with different MTUs. Suppose you receive an IP datagram from one link, you check your routing table to determine the outgoing link, and this outgoing link has an MTU that is smaller than the length of the IP datagram. Time to panic--how are you going to squeeze this oversized IP packet into the payload field of the link-layer packet? The solution to this problem is to "fragment" the data in the IP datagram among two or more smaller IP datagrams, and then send these smaller datagrams over the outgoing link. Each of these smaller datagrams is referred to as a **fragment.**

Fragments need to be reassembled before they reach the transport layer at the destination. Indeed, both TCP and UDP are expecting to receive complete, unfragmented segments from the network layer. The designers of IPv4 felt that reassembling (and possibly refragmenting) datagrams in the routers would introduce significant complication into the protocol and put a damper on router performance. (If you were a router, would you want to be reassembling fragments on top of everything else you have to do?) Sticking to the principle of keeping the network layer simple, the designers of IPv4 decided to put the job of datagram reassembly in the end systems rather than in network routers.

When a destination host receives a series of datagrams from the same source, it needs to determine if any of these datagrams are fragments of some original larger datagram. If it does determine that some datagrams are fragments, it must further determine when it has received the last fragment and how the fragments it has received should be pieced back together to form the original datagram. To allow the destination host to perform these reassembly tasks, the designers of IP (version 4) put *identification, flag,* and *fragmentation* fields in the IP datagram. When a datagram is created, the sending host stamps the datagram with an identification number as well as a source and destination address. The sending host increments the identification number for each datagram it sends. When a router needs to fragment a datagram, each resulting datagram (that is, "fragment") is stamped with the source address, destination address, and identification number of the original datagram. When the destination receives a series of datagrams from the same sending host, it can examine the identification numbers of the datagrams to determine which of the datagrams are actually fragments of the same, larger datagram. Because IP is an unreliable service, one or more of the fragments may never arrive at the destination. For this reason, in order for the destination host to be absolutely sure it has received the last fragment of the original datagram, the last fragment has a flag bit set to 0 whereas all the other fragments have this flag bit set to 1. Also, in order for the destination host to determine if a fragment is missing (and also to be able to reassemble the fragments in their proper order), the offset field is used to specify where the fragment fits within the original IP datagram.

Figure 4.25 illustrates an example. A datagram of 4,000 bytes arrives at a router, and must be forwarded to a link with an MTU of 1,500 bytes. This implies that the 3,980 data bytes in the original datagram must be allocated to three separate fragments (each of which are also IP datagrams). Suppose that the original datagram is stamped with an identification number of 777. The characteristics of the three fragments are shown in Table 4.3.

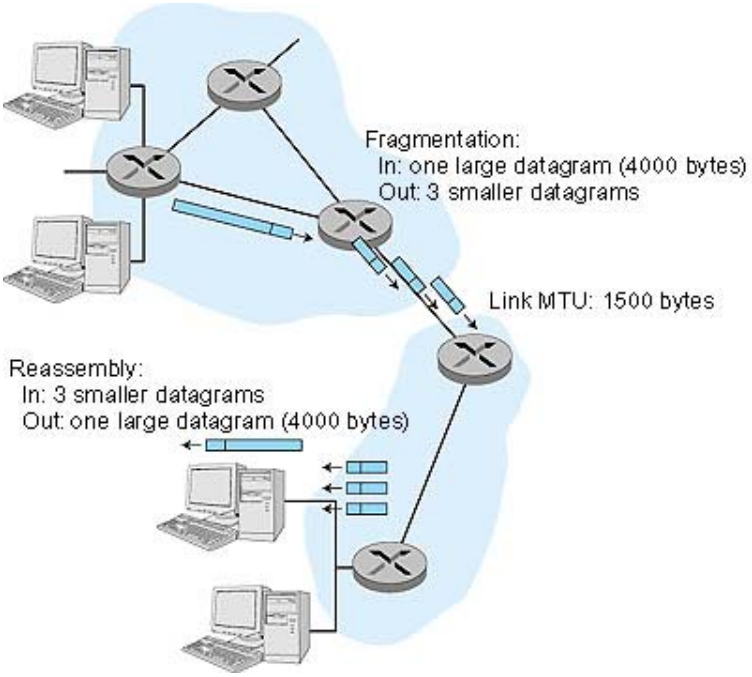


Figure 4.25: IP fragmentation and reassembly

Table 4.3: IP Fragments

Fragment	Bytes	ID	Offset	Flag
1st fragment	1480 bytes in the data field of the IP datagram	identification=777	Offset=0 (meaning the data should be inserted beginning at byte 0)	flag=1 (meaning there is more)
2nd fragment	1480 byte information field	identification=777	offset=1,480 (meaning the data should be inserted beginning at byte 1,480)	flag=1 (meaning there is more)
3rd fragment	1020 byte (=3980 - 1480 - 1480) information field	identification=777	offset=2,960 (meaning the data should be inserted beginning at byte 2,960)	flag=0 (meaning this is the last fragment)

The payload of the datagram is only passed to the transport layer at the destination once the IP layer has fully reconstructed the original IP datagram. If one or more of the fragments does not arrive to the destination, the datagram is discarded and not passed to the transport layer. But, as we learned in the previous chapter, if TCP is being used at the transport layer, then TCP will recover from this loss by having the source retransmit the data in the original datagram.

Fragmentation and reassembly puts an additional burden on Internet routers (the additional effort to create fragments out of a datagram) and on the destination hosts (the additional effort to reassemble fragments). For this reason it is desirable to keep fragmentation to a minimum. This is often done by limiting the TCP and UDP segments to a relatively small size, so that fragmentation of the corresponding datagrams is

unlikely. Because all data-link protocols supported by IP are supposed to have MTUs of at least 576 bytes, fragmentation can be entirely eliminated by using an MSS of 536 bytes, 20 bytes of TCP segment header and 20 bytes of IP datagram header. This is why most TCP segments for bulk data transfer (such as with HTTP) are 512-536 bytes long. (You may have noticed while surfing the Web that 500 or so bytes of data often arrive at a time.)

4.4.5: ICMP: Internet Control Message Protocol

We conclude this section with a discussion of the Internet Control Message Protocol, ICMP, which is used by hosts, routers, and gateways to communicate network layer information to each other. ICMP is specified in RFC 792. The most typical use of ICMP is for error reporting. For example, when running a Telnet, FTP, or HTTP session, you may have encountered an error message such as "Destination network unreachable." This message had its origins in ICMP. At some point, an IP router was unable to find a path to the host specified in your Telnet, FTP, or HTTP application. That router created and sent a type-3 ICMP message to your host indicating the error. Your host received the ICMP message and returned the error code to the TCP code that was attempting to connect to the remote host. TCP, in turn, returned the error code to your application.

ICMP is often considered part of IP, but architecturally lies just above IP, as ICMP messages are carried inside IP packets. That is, ICMP messages are carried as IP payload, just as TCP or UDP segments are carried as IP payload. Similarly, when a host receives an IP packet with ICMP specified as the upper-layer protocol, it demultiplexes the packet to ICMP, just as it would demultiplex a packet to TCP or UDP.

ICMP messages have a type and a code field, and also contain the first eight bytes of the IP datagram that caused the ICMP message to be generated in the first place (so that the sender can determine the packet that caused the error). Selected ICMP messages are shown below in Figure 4.26. Note that ICMP messages are used not only for signaling error conditions. The well-known ping program sends an ICMP type 8 code 0 message to the specified host. The destination host, seeing the echo request, sends back a type 0 code 0 ICMP echo reply. Another interesting ICMP message is the source quench message. This message is seldom used in practice. Its original purpose was to perform congestion control--to allow a congested router to send an ICMP source quench message to a host to force that host to reduce its transmission rate. We have seen in Chapter 3 that TCP has its own congestion-control mechanism that operates at the transport layer, without the use of network-layer feedback such as the ICMP source quench message.

ICMP Type	Code	Description
0	0	echo reply (to ping)
3	0	destination network unreachable
3	1	destination host unreachable
3	2	destination protocol unreachable
3	3	destination port unreachable
3	6	destination network unknown
3	7	destination host unknown
4	0	source quench (congestion control)
8	0	echo request
9	0	router advertisement
10	0	router discovery
11	0	TTL expired
12	0	IP header bad

Figure 4.26: IP Fragmentation

In Chapter 1 we introduced the Traceroute program, which enabled you to trace the route from a few given hosts to any host in the world. Interestingly, Traceroute also uses ICMP messages. To determine the names and addresses of the routers between source and destination, Traceroute in the source sends a series of ordinary IP datagrams to the destination. The first of these datagrams has a TTL of 1, the second of 2, the third of 3, etc. The source also starts timers for each of the datagrams. When the *n*th datagram arrives at the *n*th router, the *n*th router observes that the TTL of the datagram has just expired. According to the rules of the IP protocol, the router discards the datagram and sends an ICMP warning message to the source (type 11 code 0). This warning message includes the name of the router and its IP address. When this ICMP message arrives at the source, the source obtains the round-trip time from the timer and the name and IP address of the *n*th router from the ICMP message. Now that you understand how Traceroute works, you may want to go back and play with it some more.

4.5: 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 Section 4.2, and the notion of an autonomous system (AS) considered in Section 4.3 are all central to how routing is done in today's Internet.

Informally, we know that the Internet is a loose confederation of interconnected "networks" of local, regional, national, and international ISPs. We'll now need to make this understanding a bit more precise, given that we've seen that the notion of "network" has a very precise meaning in terms of IP addressing. Recall from Section 4.3 that a collection of routers that are under the same administrative and technical control, and that all run the same routing protocol amongst themselves, is known as an autonomous system (AS). Each AS, in turn, is typically comprised of multiple networks (where we use the term "network" in the precise, addressing sense of Section 4.4). The most important distinction to be made among Internet routing protocols is whether they are used to route datagrams *within* a single AS, or *among* multiple ASs. We consider the first class of protocols, known as intra-autonomous system routing protocols, in Section 4.5.1. We consider the second class of protocols, inter-autonomous system routing protocols, in Section 4.5.2.

4.5.1: Intra-Autonomous-System Routing in the Internet

An intra-AS routing protocol is used to configure and maintain the routing tables within an autonomous system (AS). Intra-AS routing protocols are also known as **interior gateway protocols.** Historically, three routing protocols have been used extensively for routing within an autonomous system in the Internet: RIP (the Routing Information Protocol), OSPF (Open Shortest Path First), and EIGRP (Cisco's propriety Enhanced Interior Gateway Routing Protocol).

RIP: Routing Information Protocol

The Routing Information Protocol (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 of the Berkeley Software Distribution (BSD) version of UNIX supporting TCP/IP. RIP version 1 is defined in RFC 1058, with a backwards compatible version 2 defined in RFC 1723.

RIP is a distance vector protocol that operates in a manner very close to the idealized protocol we examined in Section 4.2.3. The version of RIP specified in RFC 1058 uses hop count as a cost metric; that is, each link has a cost of 1. The maximum cost of a path is limited to 15, thus limiting the use of RIP to autonomous systems that are less than 15 hops in diameter. Recall that in distance vector protocols, neighboring routers exchange routing information with each other. In RIP, routing tables are exchanged between neighbors approximately every 30 seconds using a so-called **RIP response message.** The response message sent by a router or host contains the sender's routing table entries for up to 25 destination networks within the AS. 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 4.27. In this figure, lines connecting the routers denote networks. Only selected routers (*A,B,C,* and *D*) and networks (*w,x,y,z*) are labeled for visual convenience. Dotted lines indicate that the AS continues on, and thus this autonomous system has many more routers and links than are shown in the Figure 4.27.

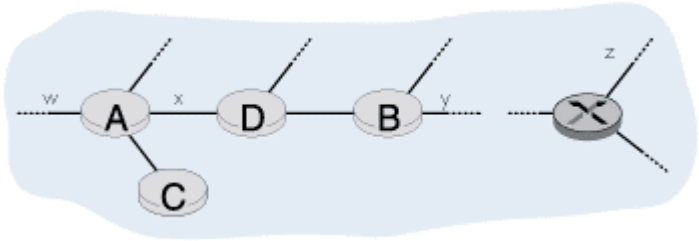


Figure 4.27: A portion of an autonomous system

Now suppose that the routing table for router *D* is as shown in Figure 4.28. Note that the routing table has three columns. The first column is for the destination network, the second column indicates the identity of the next router along the shortest path to the destination network, and the third column indicates the number of hops (that is, the number of networks that have to be traversed, including the destination network) to get to the destination network along the shortest path. For this example, the table indicates that to send a datagram

from router *D* to destination network *w*, the datagram should be first sent to neighboring router *A*; the table also indicates that destination network *w* is two hops away along the shortest path. Similarly, the table indicates that network *z* is seven hops away via router *B*. In principle, a routing table will have one row for each network in the AS, although RIP version 2 allows routes to networks to be aggregated using the route aggregation techniques similar to those we examined in Section 4.4.1. The routing table will also have at least one row for routing to networks that are outside of the AS. The table in Figure 4.28, and the subsequent tables to come, are thus only partially complete.

Destination network	Next Router	Number of Hops to Destination
W	A	2
Y	B	2
Z	B	7
X	--	1
....	....	....

Figure 4.28: Routing table in router *D* before receiving advertisement from router *A*

Now suppose that 30 seconds later, router *D* receives from router *A* the advertisement shown in Figure 4.29. Note that this advertisement is nothing other but the routing table in router *A*! This routing table says, in particular, that network *z* is only 4 hops away from router *A.* Router *D*, upon receiving this advertisement, merges the advertisement (Figure 4.29) with the "old" routing table (Figure 4.28). In particular, router *D* learns that there is now a path through router *A* to network *z* that is shorter than the path through router *B*.

Destination network	Next Router	Number of Hops to Destination
Z	C	4
W	--	1
X	--	1
....	....	....

Figure 4.29: Advertisement from router *A*

Thus, router *D* updates its routing table to account for the "shorter" shortest path, as shown in Figure 4.30. How is it, you might ask, that the shortest path to network *z* has become shorter? Possibly, the decentralized distance vector algorithm was still in the process of converging (see Section 4.2), or perhaps new links and/or routers were added to the AS, thus changing the shortest paths in the network.

Destination network	Next Router	Number of Hops to Destination
W	A	2
Y	B	2
Z	A	5
....	....	....

Figure 4.30: Routing table in router *D* after receiving advertisement from router *A*

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 connecting link has gone down. When this happens, RIP modifies its 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 packet is carried between routers in a standard IP packet. 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 4.31 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 the RIP protocol, that is, maintains the routing table 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 manipulate the routing tables within the UNIX kernel), it can send and receive messages over a standard socket

and use a standard transport protocol. Thus, RIP is an application-layer protocol (see Chapter 2) running over UDP.

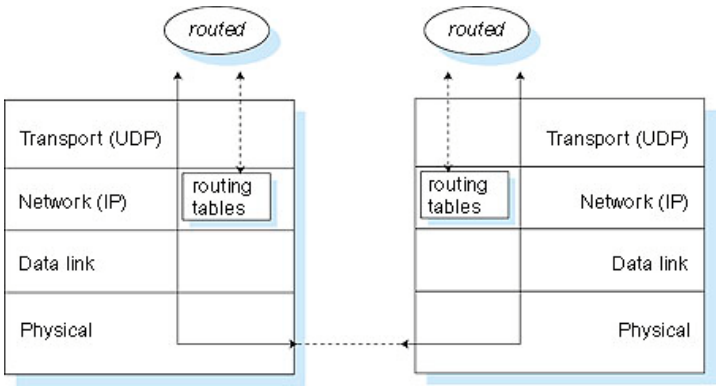


Figure 4.31: Implementation of RIP as the *routed* daemon

Finally, let us take a quick look at a RIP routing table. The RIP routing table in Figure 4.32 is taken from a UNIX router giroflee.eurecom.fr. If you give a netstat -rn command on a UNIX system, you can view the routing table for that host or router. Performing a netstat command on giroflee.eurecom.fr yields the routing table in Figure 4.32.

Destination	Gateway	Flags	Ref	Use	Interface
127.0.0.1	127.0.0.1	UH	0	26492	lo0
192.168.2.	192.168.2.5	U	2	13	fa0
193.55.114.	193.55.114.6	U	3	58503	le0
192.168.3.	192.168.3.5	U	2	25	qaa0
224.0.0.0	193.55.114.6	U	3	0	le0
default	193.55.114.129	UG	0	143454	

Figure 4.32: RIP routing table from giroflee.eurecom.fr

The router giroflee is connected to three networks. The second, third, and fourth rows in the table tell us that these three networks are attached to giroflee via giroflee's network interfaces fa0, le0 and qaa0. These giroflee interfaces have IP addresses 192.168.2.5, 193.55.114.6, and 192.168.3.5, respectively. To transmit a packet to any host belonging to one of these three networks, giroflee will simply send the outgoing IP datagram over the appropriate interface. Of particular interest to us is the **default route**. Any IP datagram that is not destined for one of the networks explicitly listed in the routing table will be forwarded to the router with IP address 193.55.114.129; this router is reached by sending the datagram over the default network interface. The first entry in the routing table is the so-called loopback interface. When IP sends a datagram to the loopback interface, the packet is simply returned back to IP; this is useful for debugging purposes. The address 224.0.0.0 is a special multicast (Class D) IP address. We will examine IP multicast in Section 4.8.

OSPF: Open Shortest Path First

Like RIP, Open Shortest Path First (OSPF) routing is used for intra-AS routing. 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 2178--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 directed graph) of the entire autonomous system. The router then locally runs Dijkstra's shortest-path algorithm to determine a shortest-path tree to all networks with itself as the root node. The router's routing table is then obtained from this shortest-path tree. Individual link costs are configured by the network administrator.

Let us now contrast and compare the advertisements sent by RIP and OSPF. With OSPF, a router periodically broadcasts routing information to *all* other routers in the autonomous system, not just to its neighboring routers. For an excellent treatment of link state broadcast algorithms, see [Perlman 1999]. This routing information sent by a router has one entry for each of the router's neighbors; the entry gives the

distance (that is, link state) from the router to the neighbor. On the other hand, a RIP advertisement sent by a router contains information about all the networks in the autonomous system, although this information is only sent to its neighboring routers. In a sense, the advertising techniques of RIP and OSPF are duals of each other. Some of the advances embodied in OSPF include the following:

- *Security.* All exchanges between OSPF routers (for example, link-state updates) are authenticated. This means that only trusted routers can participate in the OSPF protocol within a domain, thus preventing malicious intruders (or networking students taking their newfound knowledge out for a joyride) from injecting incorrect information into router tables.
- *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).
- *Different cost metrics for different TOS traffic.* OSPF allows each link to have different costs for different TOS (type of service) IP packets. For example, a high-bandwidth satellite link might be configured to have a low cost (and hence be attractive) for non-time-critical traffic, but a very high cost metric for delay-sensitive traffic. In essence, OSPF sees different network topologies for different classes of traffic and hence can compute different routes for each type of traffic.
- *Integrated support for unicast and multicast routing.* Multicast OSPF [RFC 1584] provides simple extensions to OSPF to provide for multicast routing (a topic we cover in more depth in Section 4.8). 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 hierarchically structure an autonomous system. Section 4.3 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.

As OSPF autonomous system can be configured 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. The internal details of an area thus remain invisible to all routers outside the area. Intra-area routing involves only those routers within the same area.

Within each area, one or more **area border routers** are responsible for routing packets outside the area. 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.

A diagram of a hierarchically structured OSPF network is shown in Figure 4.33. We can identify four types of OSPF routers in Figure 4.33:

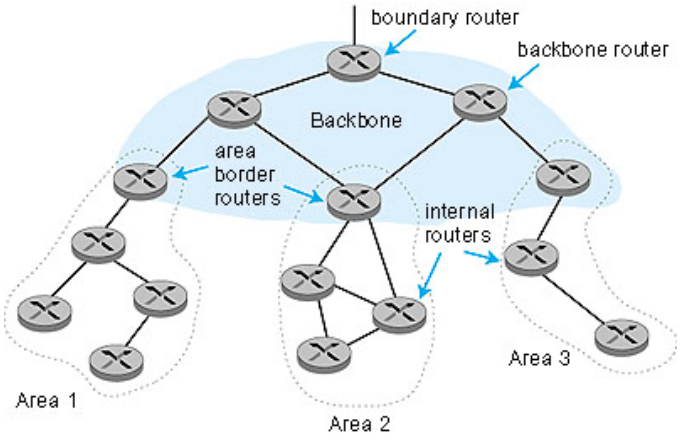


Figure 4.33: Hierarchically structured OSPF AS with four areas

- *Internal routers.* These routers are in nonbackbone areas and only perform intra-AS routing.
- *Area border routers.* These routers belong to both an area and the backbone.
- *Backbone routers (nonborder routers).* These routers perform routing within the backbone but themselves are not area border routers. Within a nonbackbone area, internal routers learn of the existence of routes to other areas from information (essentially a link-state advertisement, but

advertising the cost of a route to another area, rather than a link cost) broadcast within the area by its backbone routers.

- *Boundary routers.* A boundary router exchanges routing information with routers belonging to other autonomous systems. This router might, for example, use BGP to perform inter-AS routing. It is through such a boundary router that other routers learn about paths to external networks.

EIGRP: Enhanced Internal Gateway Routing Protocol

The Enhanced Interior Gateway Routing Protocol (EIGRP) [Cisco IGRP 1997] is a proprietary routing algorithm developed by Cisco Systems, Inc., as a successor for RIP. EIGRP is a distance vector protocol. Several cost metrics (including delay, bandwidth, reliability, and load) can be used in making routing decisions, with the weight given to each of the metrics being determined by the network administrator. This ability to use administrator-defined costs in making route selections is an important difference from RIP; we will see shortly that so-called policy-based inter-AS Internet routing protocols such as BGP also allow administratively defined routing decisions to be made. Other important differences from RIP include the use of a reliable transport protocol to communicate routing information, the use of update messages that are sent only when routing table costs change (rather than periodically), route aggregation, and the use of a distributed diffusing update routing algorithm [Garcia-Luna 1993] to quickly compute loop-free routing paths.

4.5.2: Inter-Autonomous System Routing

The Border Gateway Protocol version 4, specified in RFC 1771 (see also RFC 1772; RFC 1773), is the *de facto* standard interdomain routing protocol in today's Internet. It is commonly referred to as BGP4 or simply as BGP. As an inter-autonomous system routing protocol, it provides for routing between autonomous systems (that is, administrative domains).

While BGP has the same general flavor as the distance vector protocol that we studied in Section 4.2, it is more appropriately characterized as a **path vector protocol**. This is because BGP in a router does not propagate cost information (for example, number of hops to a destination), but instead propagates path information, such as the sequence of ASs on a route to a destination AS. We will examine the path information in detail shortly. We note though that although this information includes the names of the ASs on a route to the destination, it does *not* contain cost information. Nor does BGP specify how a specific route to a particular destination should be chosen among the routes that have been advertised. That decision is a *policy* decision that is left up to the domain's administrator. Each domain can thus choose its routes according to whatever criteria it chooses (and need not even inform its neighbors of its policy!)--allowing a significant degree of autonomy in route selection. In essence, BGP provides the *mechanisms* to distribute path information among the interconnected autonomous systems, but leaves the policy for making the actual route selections up to the network administrator.

Let's begin with a grossly simplified description of how BGP works. This will help us see the forest through the trees. As discussed in Section 4.3, as far as BGP is concerned, the whole Internet is a graph of ASs, and each AS is identified by an AS number. At any instant of time, a given AS X may, or may not, know of a path of ASs that lead to a given destination AS Z. As an example, suppose X has a path $XY_1Y_2Y_3Z$ from itself to Z listed in its BGP table. This means that X knows that it can send datagrams to Z through the ASs X, Y_1, Y_2, and Y_3, Z. When X sends updates to its BGP neighbors (that is, the neighbors in the graph), X actually sends the entire path information, $XY_1Y_2Y_3Z$, to its neighbors (as well as other paths to other ASs). If, for example, W is a neighbor of X, and W receives an advertisement that includes the path $XY_1Y_2Y_3Z$, then W can list a new entry $WXY_1Y_2Y_3Z$ in its BGP table. However, we should keep in mind that W may decide to *not* create this new entry for one of several reasons. For example, W would not create this entry if W is equal to (say) Y_2, thereby creating an undesirable loop in the routing; or if W already has a path to Z in its tables, and this existing path is preferable (with respect to the metric used by BGP at W) to $WXY_1Y_2Y_3Z$; or, finally, if W has a policy decision to not forward datagrams through (say) Y_2.

In BGP jargon, the immediate neighbors in the graph of ASs are called **peers**. BGP information is propagated through the network by exchanges of BGP mes -sages between peers. The BGP protocol defines the four types of messages: OPEN, UPDATE, NOTIFICATION, and KEEPALIVE.

- *OPEN.* BGP peers communicate using the TCP protocol and port number 179. TCP thus provides for reliable and congestion-controlled message exchange between peers. In contrast, recall that we earlier saw that two RIP peers, for example, the *routed*'s in Figure 4.31 communicate via unreliable UDP. When a BGP gateway wants to first establish contact with a BGP peer (for example, after the gateway itself or a connecting link has just been booted), an OPEN message is sent to the peer. The OPEN message allows a BGP gateway to identify and authenticate itself, and provide timer information. If the OPEN is acceptable to the peer, it will send back a KEEPALIVE message.

- *UPDATE.* A BGP gateway uses the UPDATE message to advertise a path to a given destination (for example, $XY_1Y_2Y_3Z$) to the BGP peer. The UPDATE message can also be used to *withdraw* routes that had previously been advertised (that is, to tell a peer that a route that it had previously advertised is no longer a valid route).
- *KEEPALIVE.* This BGP message is used to let a peer know that the sender is alive but that the sender doesn't have other information to send. It also serves as an acknowledgment to a received OPEN message.
- *NOTIFICATION.* This BGP message is used to inform a peer that an error has been detected (for example, in a previously transmitted BGP message) or that the sender is about to close the BGP session.

Recall from our discussion above that BGP provides mechanisms for distributing path information but does not mandate policies for selecting a route from those available. Within this framework, it is thus possible for an AS such as Hatfield.net to implement a policy such as "traffic from my AS should not cross the AS McCoy.net," since it knows the identities of all AS's on the path. (The Hatfields and the McCoys are two famous feuding families in the U.S.) But what about a policy that would prevent the McCoys from sending traffic through the Hatfield's network? The only means for an AS to control the traffic it passes through its AS (known as "transit" traffic--traffic that neither originates in, nor is destined for, the network, but instead is "just passing through") is by controlling the paths that it advertises. For example, if the McCoys are immediate neighbors of the Hatfields, the Hatfields could simply not advertise any routes to the McCoys that contain the Hatfield network. But restricting transit traffic by controlling an AS's route advertisement can only be partially effective. For example, if the Joneses are between the Hatfields and the McCoys, and the Hatfields advertise routes to the Joneses that pass through the Hatfields, then the Hatfields cannot prevent (using BGP mechanisms) the Joneses from advertising these routes to the McCoys.

Very often an AS will have multiple gateway routers that provide connections to other ASs. Even though BGP is an inter-AS protocol, it can still be used inside an AS as a pipe to exchange BGP updates among gateway routers belonging to the same AS. BGP connections inside an AS are called Internal **BGP (IBGP),** whereas BGP connections between ASs are called External **BGP (EBGP).**

As noted above, BGP is becoming the *de facto* standard for inter-AS routing for the public Internet. BGP is used, for example, at the major network access points (NAPs) where major Internet carriers connect to each other and exchange traffic. To see the contents of today's (less than four hours out of date) BGP routing table (large!) at one of the major NAPs in the U.S. (which include Chicago and San Francisco), see [IPMA 2000].

This completes our brief introduction of BGP. Although BGP is complex, it plays a central role in the Internet. We encourage readers to see the references [Stewart 1999; Labovitz 1997; Halabi 1997; Huitema 1995] to learn more about BGP.

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 specifically 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 specifically 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 preference or costs associated with routes. Within a single AS, however, such policy concerns can be ignored, allowing routing to focus more on the level of performance realized on a route.

4.6: What's Inside a Router?

Our study of the network layer so far has focused on network-layer service models, the routing algorithms that control the routes taken by packets through the network, and the protocols that embody these routing algorithms. These topics, however, are only part (albeit important ones) of what goes on in the network layer. We have yet to consider the **switching function** of a router--the actual transfer of datagrams from a router's incoming links to the appropriate outgoing links. Studying just the control and service aspects of the network layer is like studying a company and considering only its management (which controls the company but typically performs very little of the actual "grunt" work that makes a company run!) and its public relations ("Our product will provide this wonderful service to you!"). To fully appreciate what really goes on within a company, one needs to consider the workers. In the network layer, the real work (that is, the reason the network layer exists in the first place) is the forwarding of datagrams. A key component in this forwarding process is the transfer of a datagram from a router's incoming link to an outgoing link. In this section, we study how this is accomplished. Our coverage here is necessarily brief, as an entire course would be needed to cover router design in depth. Consequently, we'll make a special effort in this section to provide pointers to material that covers this topic in more depth.

A high-level view of a generic router architecture is shown in Figure 4.34. Four components of a router can be identified:

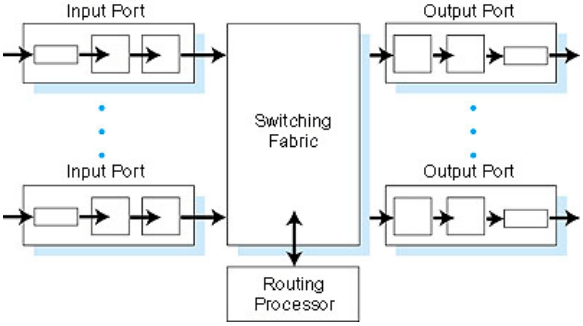


Figure 4.34: Router architecture

- *Input ports.* The input port performs several functions. It performs the physical layer functionality (the leftmost box of the input port and the rightmost box of the output port in Figure 4.34) of terminating an incoming physical link to a router. It performs the data-link layer functionality (shown in the middle boxes in the input and output ports) needed to interoperate with the data link layer functionality (see Chapter 5) on the other side of the incoming link. It also performs a lookup and forwarding function (the rightmost box of the input port and the leftmost box of the output port) so that a packet forwarded into the switching fabric of the router emerges at the appropriate output port. Control packets (for example, packets carrying routing protocol information for RIP, OSPF or BGP) are forwarded from the input port to the routing processor. In practice, multiple ports are often gathered together on a single **line card** within a router.
- *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 the packets that have been forwarded to it through the switching fabric, and then transmits the packets on the outgoing link. The output port thus performs the reverse data link and physical layer functionality as the input port.
- *Routing processor.* The routing processor executes the routing protocols (for example, the protocols we studied in Section 4.5), maintains the routing tables, and performs network management functions (see Chapter 8), within the router. Since we cover these topics elsewhere in this book, we defer discussion of these topics to elsewhere.

In the following, we'll take a look at input ports, the switching fabric, and output ports in more detail. [Turner 1988; Giacopelli 1990; McKeown 1997a; Partridge 1998] provide a discussion of some specific router architectures. [McKeown 1997b] provides a particularly readable overview of modern router architectures, using the Cisco 12000 router as an example.

4.6.1: Input Ports

A more detailed view of input port functionality is given in Figure 4.35. As discussed above, the input port's line termination function and data link processing implement the physical and data link layers associated with an individual input link to the router. The lookup/forwarding function of the input port is central to the switching function of the router. In many routers, it is here that the router determines the output port to which an arriving packet will be forwarded via the switching fabric. The choice of the output port is made using the information contained in the routing table. Although the routing table is computed by the routing processor, a shadow copy of the routing table is typically stored at each input port and updated, as needed, by the routing processor. With local copies of the routing table, the switching decision can be made locally, at each input port, without invoking the centralized routing processor. Such *decen tralized* switching avoids creating a forwarding bottle neck at a single point within the router.

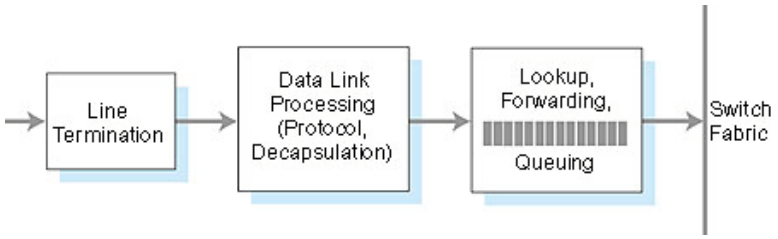


Figure 4.35: Input port processing

In routers with limited processing capabilities at the input port, the input port may simply forward the packet to the centralized routing processor, which will then perform the routing table lookup and forward the packet to the appropriate output port. This is the approach taken when a workstation or server serves as a router; here, the routing processor is really just the workstation's CPU and the input port is really just a network interface card (for example, an Ethernet card).

Given the existence of a routing table, the routing table lookup is concep tually simple--we just search through the routing table, looking for a destination entry that best matches the destination network address of the packet, or a default route if the destination entry is missing. (Recall from our discussion in Section 4.4.1 that the best match is the routing table entry with the longest network prefix that matches the packet's des tination address.) In practice, however, life is not so simple. Perhaps the most important complicating factor is that backbone routers must operate at high speeds, being capable of performing millions of lookups per second. Indeed, it is desirable for the input port processing to be able to proceed at **line speed,** that is, that a lookup can be done in less than the amount of time needed to receive a packet at the input port. In this case, input processing of a received packet can be completed before the next receive operation is complete. To get an idea

of the performance requirements for a lookup, consider that a so-called OC48 link runs at 2.5 Gbps. With 256-byte-long packets, this implies a lookup speed of approximately a million lookups per second.

Given the need to operate at today's high-link speeds, a linear search through a large routing table is impossible. A more reasonable technique is to store the routing table entries in a tree data structure. Each level in the tree can be thought of as corresponding to a bit in the destination address. To look up an address, one simply starts at the root node of the tree. If the first address bit is a zero, then the left subtree will contain the routing table entry for a destination address; otherwise it will be in the right subtree. The appropriate subtree is then traversed using the remaining address bits--if the next address bit is a zero, the left subtree of the initial subtree is chosen; otherwise, the right subtree of the initial subtree is chosen. In this manner, one can look up the routing table entry in N steps, where N is the number of bits in the address. (The reader will note that this is essentially a binary search through an address space of size 2^N.) Refinements of this approach are discussed in [Doeringer 1996]. An improvement over binary search techniques is described in [Srinivasan 1999].

But even with $N = 32$ (for example, a 32-bit IP address) steps, the lookup speed via binary search is not fast enough for today's backbone routing requirements. For example, assuming a memory access at each step, less than a million address lookups/sec could be performed with 40 ns memory access times. Several techniques have thus been explored to increase lookup speeds. Content addressable memories (CAMs) allow a 32-bit IP address to be presented to the CAM, which then returns the content of the routing table entry for that address in essentially constant time. The Cisco 8500 series router [Cisco 8500 1999] has a 64K CAM for each input port. Another technique for speeding lookup is to keep recently accessed routing table entries in a cache [Feldmeier 1988]. Here, the potential concern is the size of the cache. Measurements in [Thomson 1997] suggest that even for an OC-3 speed link, approximately 256,000 source-destination pairs might be seen in one minute in a backbone router. Most recently, even faster data structures, which allow routing table entries to be located in $\log(N)$ steps [Waldvogel 1997], or which compress routing tables in novel ways [Brodnik 1997], have been proposed. A hardware-based approach to lookup that is optimized for the common case that the address being looked up has 24 or fewer significant bits is discussed in [Gupta 1998].

Once the output port for a packet has been determined via the lookup, the packet can be forwarded into the switching fabric. However, as we'll see below, a packet may be temporarily **blocked** from entering the switching fabric (due to the fact that packets from other input ports are currently using the fabric). A blocked packet must thus be queued at the input port and then scheduled to cross the switching 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) within a router in Section 4.6.4.

Case History

Cisco Systems: Dominating the Network Core

As of this writing (January 2000), Cisco Systems employs 23,000 people and has a market capitalization of $360 billion. Cisco currently dominates the Internet router market, and in recent years has quickly moved into the Internet telephony market, where it will compete head-to-head with the telephone equipment companies, such as Lucent, Alcatel, Nothern Telecom, and Siemens. How did this gorilla of a networking company come to be? It all started in 1984 (only 16 years ago) in the living room of a house in 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. 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 originally in their living room, and they originally financed the project with credit cards and moonlighting consulting jobs. At the end of 1986, Cisco's revenues reached $250,000 a month--not bad for a business financed on credit cards and without any venture capital. At the end of 1987, Cisco finally succeeded in attracting venture capital--$2 million dollars 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 strained. Cisco went public in 1990, but in the same year Cisco fired Lerner, and Bosak resigned.

4.6.2: Switching Fabrics

The switching fabric is at the very heart of a router. It is through this switching fabric that the packets are actually moved from an input port to an output port. Switching can be accomplished in a number of ways, as indicated in Figure 4.36.

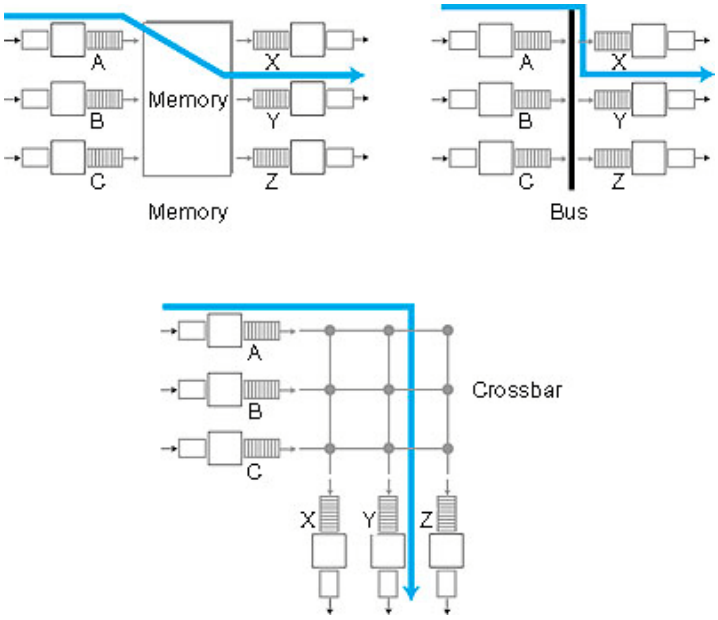


Figure 4.36: Three switching techniques

- *Switching via memory.* The simplest, earliest routers were often traditional computers, with switching between input and output port, being done under direct control of 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 routing table, and copied the packet to the output port's buffers. Note that if the memory bandwidth is such that *B* packets/sec can be written into, or read from, memory, then the overall switch throughput (the total rate at which packets are transferred from input ports to output ports) must be less than *B*/2.
  Many modern routers also switch via memory. A major difference from early routers, however, is that the lookup of the destination address and the storing (switching) of the packet into the appropriate memory location is performed by processors on the input line cards. In some ways, routers that switch via memory look very much like shared memory multiprocessors, with the processors on a line card storing packets into the memory of the appropriate output port. Cisco's Catalyst 8500 series switches [Cisco 8500 1999] and Bay Networks Accelar 1200 Series routers switch packets via a shared memory.
- *Switching via a bus.* In this approach, the input ports transfer a packet directly to the output port over a shared bus, without intervention by the routing processor (Note that when switching via memory, the packet must also cross the system bus going to/from memory). Although the routing processor is not involved in the bus transfer, since the bus is shared, only one packet at a time can be transferred over the bus. A packet arriving at an input port and finding the bus busy with the transfer of another packet is blocked from passing through the switching fabric and is queued at the input port. Because every packet must cross the single bus, the switching bandwidth of the router is limited to the bus speed.
  Given that bus bandwidths of over a gigabit per second are possible in today's technology, switching via a bus is often sufficient for routers that operate in access and enterprise networks (for example, local area and corporate networks). Bus-based switching has been adopted in a number of current router products, including the Cisco 1900 [Cisco Switches 1999], which switches packets over a 1 Gbps Packet Exchange Bus. 3Com's CoreBuilder 5000 systems [Kapoor 1997] interconnects ports that reside on different switch modules over its PacketChannel data bus, with a bandwidth of 2 Gbps.
- *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 $2N$ busses that connect N input ports to N output ports, as shown in Figure 4.36. A packet arriving at an input port travels along the horizontal bus attached to the input port until it intersects with the vertical bus leading to the desired output port. If the vertical bus leading to the output port is free, the packet is transferred to the output port. If the vertical bus is being used to transfer a packet from another input port to this same output port, the arriving packet is blocked and must be queued at the input port.

Delta and Omega switching fabrics have also been proposed as an interconnection network between input and output ports. See [Tobagi 1990] for a survey of switch architectures. Cisco 12000 Family switches [Cisco 12000 1998] use an interconnection network, providing up to 60 Gbps through the switching fabric. One current trend in interconnection network design [Keshav 1998] is to fragment a variable length IP datagram into fixed-length cells, and then tag and switch the fixed-length cells through the interconnection network. The cells are then reassembled into the original datagram at the output port. The fixed-length cell and internal tag can considerably simplify and speed up the switching of the packet through the interconnection network.

4.6.3: Output Ports

Output port processing, shown in Figure 4.37, takes the datagrams that have been stored in the output port's memory and transmits them over the outgoing link. The data-link protocol processing and line termination are the send-side link- and physical-layer functionality that interact with the input port on the other end of the outgoing link, as discussed above in Section 4.6.1. The queuing and buffer management functionality are needed when the switch fabric delivers packets to the output port at a rate that exceeds the output link rate; we'll cover output port queuing below.

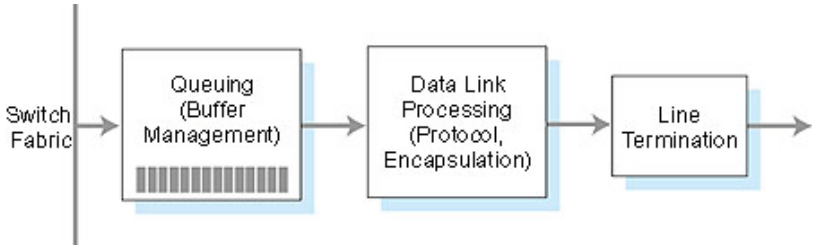


Figure 4.37: Output port processing

4.6.4: Where Does Queuing Occur?

If we look at the input and output port functionality and the configurations shown in Figure 4.36, it is evident that packet queues can form at both the input ports *and* the output ports. It is important to consider these queues in a bit more detail, since as these queues grow large, the router's buffer space will eventually be exhausted and **packet loss** will occur. Recall that in our earlier discussions, we said rather vaguely 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 dropped and lost. The actual location of packet loss (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, as discussed below.

Suppose that the input line speeds and output line speeds are all identical, and that there are n input ports and n output ports. If the switching fabric speed is at least n times as fast as the input line speed, then no queuing can occur at the input ports. This is because even in the worst case that all n input lines are receiving packets, the switch will be able to transfer n packets from input port to output port in the time it takes each of the n input ports to (simultaneously) receive a *single* packet. But what can happen at the output ports? Let us suppose still that the switching fabric is at least n times as fast as the line speeds. In the worst case, the packets arriving at each of the n input ports will be destined to the *same* output port. In this case, in the time it takes to receive (or send) a single packet, n packets will arrive at this output port. Since the output port can only transmit 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. n more packets can then possibly arrive in the time it takes to transmit just one of the n packets that had previously been queued. And so on. Eventually, the number of queued packets can grow large enough to exhaust the memory space at the output port, in which case packets are dropped.

Output port queuing is illustrated in Figure 4.38. 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.

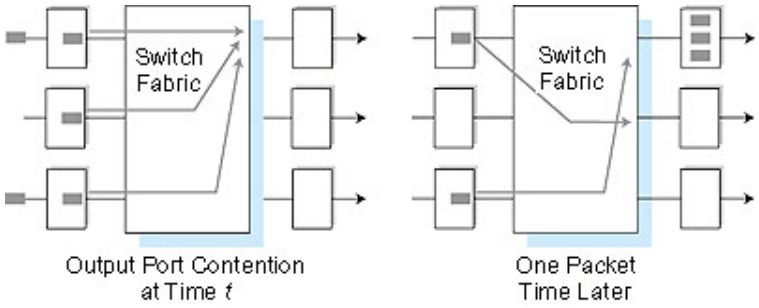


Figure 4.38: Output port queuing

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 will cover this topic extensively in Section 6.6. A discussion of output port packet scheduling disciplines used in today's routers is [Cisco Queue 1995].

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 will 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 a 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 to the same output queue, then one of the packets will be blocked and must wait at the input queue--the switching fabric can only transfer one packet to a given output port at a time.

Figure 4.39 shows an example where two packets (shaded black) 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 black packet in the lower-left queue must wait. But not only must this black packet wait, but so too must the white 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 white packet). This phenomenon is known as **head-of-the-line (HOL) blocking** in an 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].

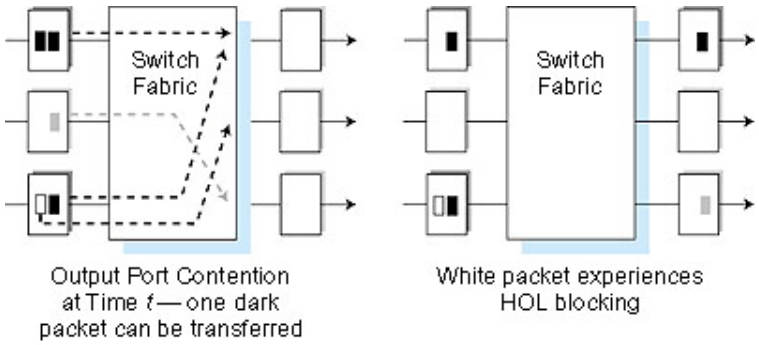


Figure 4.39: HOL blocking at an input queued switch

4.7: IPv6

In the early 1990s, the Internet Engineering Task Force began an effort to develop a successor to the IPv4 protocol. A prime motivation for this effort was the realization that the 32-bit IP address space was beginning to be used up, with new networks and IP nodes being attached to the Internet (and being allocated unique IP addresses) at a breathtaking rate. To respond to this need for a large IP address space, a new IP protocol, IPv6, was developed. The designers of IPv6 also took this opportunity to tweak and augment other aspects of IPv4, based on the accumulated operational experience with IPv4.

The point in time when IPv4 addresses would have been completely allocated (and hence no new networks could have attached to the Internet) was the subject of considerable debate. Based on current trends in address allocation at the time, the estimates of the two leaders of the IETF's Address Lifetime Expectations working group were that addresses would become exhausted in 2008 and 2018, respectively [Solensky 1996]. In 1996, the American Registry for Internet Numbers (ARIN) reported that all of the IPv4 class A addresses had been assigned, 62 percent of the class B addresses had been assigned, and 37 percent of the class C addresses had been assigned [ARIN 1996]. Although these estimates and numbers suggested that a considerable amount of time might be left until the IPv4 address space became exhausted, it was realized that considerable time would be needed to deploy a new technology on such an extensive scale, and so the "Next Generation IP" (IPng) effort [Bradner 1996; RFC 1752], was begun. An excellent online source of information about IPv6 is The IP Next Generation Homepage [Hinden 1999]. An excellent book is also available on the subject [Huitema 1997].

4.7.1: IPv6 Packet Format

The format of the IPv6 packet is shown in Figure 4.40. The most important changes introduced in IPv6 are evident in the packet format:

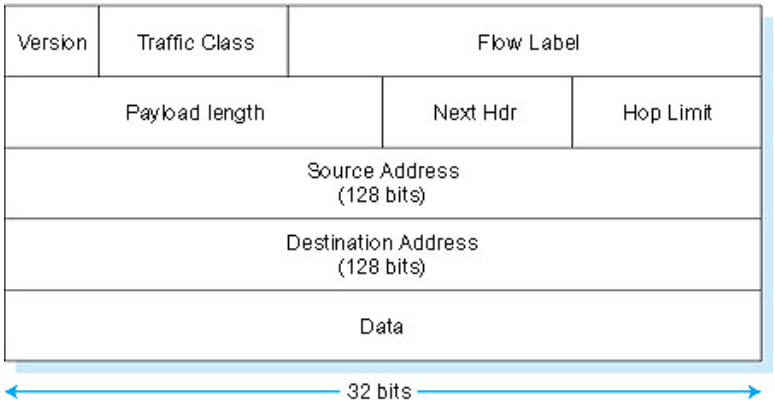


Figure 4.40: IPv6 packet format

- *Expanded addressing capabilities.* IPv6 increases the size of the IP address from 32 to 128 bits. This ensures that the world won't run out of IP addresses. Now, every grain of sand on the planet can be IP-addressable. In addition to unicast and multicast addresses, a new type of address, called an **anycast address**, has also been introduced, which allows a packet addressed to an anycast address to be delivered to any one of a group of hosts. (This feature could be used, for example, to send an HTTP GET to the nearest of a number of mirror sites that contain a given document.)
- *A streamlined 40-byte header.* As discussed below, a number of IPv4 fields have been dropped or made optional. The resulting 40-byte fixed-length header allows for faster processing of the IP datagram. A new encoding of options allows for more flexible options processing.
- *Flow labeling and priority.* IPv6 has an elusive definition of a "**flow.**" RFC 1752 and RFC 2460 state that this allows "labeling of packets belonging to particular flows for which the sender requests special handling, such as a non-default quality of service or real-time service." For example, audio and video transmission might likely be treated as a flow. On the other hand, the more traditional applications, such as file transfer and e-mail might not be treated as flows. It is possible that the traffic carried by a high-priority user (for example, someone paying for better service for their traffic) might also be treated as a flow. What is clear, however, is that the designers of IPv6 foresee the eventual need to be able to differentiate among the "flows," even if the exact meaning of a flow has not yet been determined. The IPv6 header also has eight-bit Traffic Class field. This field, like the TOS field in IPv4, can be used to give priority to certain packets within a flow, or it can be used to give priority to

datagrams from certain applications (for example, ICMP packets) over datagrams from other applications (for example, network news).

The IPv6 datagram format is shown in Figure 4.40. As noted above, a comparison of Figure 4.40 with Figure 4.24 reveals the simpler, more streamlined structure of the IPv6 datagram. The following fields are defined in IPv6:

- *Version.* This four-bit field identifies the IP version number. Not surprisingly, IPv6 carries a value of "6" in this field. Note that putting a "4" in this field does not create a valid IPv4 datagram. (If it did, life would be a lot simpler--see the discussion below regarding the transition from IPv4 to IPv6.)
- *Traffic class.* This eight-bit field is similar in spirit to the ToS field we saw in IP version 4.
- *Flow label.* As discussed above, this 20-bit field is used to identify a "flow" of datagrams.
- *Payload length.* This 16-bit value is treated as an unsigned integer giving the number of bytes in the IPv6 datagram following the fixed-length, 40-byte packet header.
- *Next header.* This field identifies the protocol to which the contents (data field) of this datagram will be delivered (for example, to TCP or UDP). The field uses the same values as the Protocol field in the IPv4 header.
- *Hop limit.* The contents of this field are decremented by one by each router that forwards the datagram. If the hop limit count reaches zero, the datagram is discarded.
- *Source and destination address.* The various formats of the IPv6 128-bit address are described in RFC 2373.
- *Data.* This is the payload portion of the IPv6 datagram. When the datagram reaches its destination, the payload will be removed from the IP datagram and passed on to the protocol specified in the next header field.

The discussion above identified the purpose of the fields that *are* included in the IPv6 datagram. Comparing the IPv6 datagram format in Figure 4.40 with the IPv4 datagram format that we saw earlier in Figure 4.24, we notice that several fields appearing in the IPv4 datagram are no longer present in the IPv6 datagram:

- *Fragmentation/Reassembly.* IPv6 does not allow for fragmentation and reassembly at intermediate routers; these operations can be performed only by the source and destination. If an IPv6 datagram received by a router is too large to be forwarded over the outgoing link, the router simply drops the datagram and sends a "Packet Too Big" ICMP error message (see below) back to the sender. The sender can then resend the data, using a smaller IP datagram size. Fragmentation and reassembly is a time-consuming operation; removing this functionality from the routers and placing it squarely in the end systems considerably speeds up IP forwarding within the network.
- *Checksum.* Because the transport layer (for example, TCP and UDP) and data link (for example, Ethernet) protocols in the Internet layers perform checksumming, the designers of IP probably felt that this functionality was sufficiently redundant in the network layer that it could be removed. Once again, fast processing of IP packets was a central concern. Recall from our discussion of IPv4 in Section 4.4.1, that since the IPv4 header contains a TTL field (similar to the hop limit field in IPv6), the IPv4 header checksum needed to be recomputed at every router. As with fragmentation and reassembly, this too was a costly operation in IPv4.
- *Options.* An options field is no longer a part of the standard IP header. However, it has not gone away. Instead, the options field is one of the possible "next headers" pointed to from within the IPv6 header. That is, just as TCP or UDP protocol headers can be the next header within an IP packet, so too can an options field. The removal of the options field results in a fixed length, 40-byte IP header.

A New ICMP for IPv6

Recall from our discussion in Section 4.4, that the ICMP protocol is used by IP nodes to report error conditions and provide limited information (for example, the echo reply to a ping message) to an end system. A new version of ICMP has been defined for IPv6 in RFC 2463. In addition to reorganizing the existing ICMP type and code definitions, ICMPv6 also added new types and codes required by the new IPv6 functionality. These include the "Packet Too Big" type, and an "unrecognized IPv6 options" error code. In addition, ICMPv6 subsumes the functionality of the Internet Group Management Protocol (IGMP) that we will study in Section 4.8. IGMP, which is used to manage a host's joining and leaving of so-called multicast groups, was previously a separate protocol from ICMP in IPv4.

4.7.2: Transitioning from IPv4 to IPv6

Now that we have seen the technical details of IPv6, let us consider a very practical matter: how will the public Internet, which is based on IPv4, be transitioned to IPv6? The problem is that while new IPv6-capable systems can be made "backwards compatible," that is, can send, route, and receive IPv4 datagrams, already deployed IPv4-capable systems are not capable of handling IPv6 datagrams. Several options are possible.

One option would be to declare a "flag day"--a given time and date when all Internet machines would be turned off and upgraded from IPv4 to IPv6. The last major technology transition (from using NCP to using TCP for reliable transport service) occurred almost 20 years ago. Even back then [RFC 801], when the Internet was tiny and still being administered by a small number of "wizards," it was realized that such a flag day was not possible. A flag day involving hundreds of millions of machines and millions of network administrators and users is even more unthinkable today. RFC 1933 describes two approaches (which can be used either alone or together) for gradually integrating IPv6 hosts and routers into an IPv4 world (with the long-term goal, of course, of having all IPv4 nodes eventually transition to IPv6).

Probably the most straightforward way to introduce IPv6-capable nodes is a **dual-stack** approach, where IPv6 nodes also have a complete IPv4 implementation as well. Such a node, referred to as an IPv6/IPv4 node in RFC 1933, has the ability to send and receive both IPv4 and IPv6 datagrams. When interoperating with an IPv4 node, an IPv6/IPv4 node can use IPv4 datagrams; when interoperating with an IPv6 node, it can speak IPv6. IPv6/IPv4 nodes must have both IPv6 and IPv4 addresses. They must furthermore be able to determine whether another node is IPv6-capable or IPv4-only. This problem can be solved using the DNS (see Chapter 2), which can return an IPv6 address if the node name being resolved is IPv6-capable, or otherwise return an IPv4 address. Of course, if the node issuing the DNS request is only IPv4-capable, the DNS returns only an IPv4 address.

In the dual-stack approach, if either the sender or the receiver is only IPv4-capable, an IPv4 datagram must be used. As a result, it is possible that two IPv6-capable nodes can end up, in essence, sending IPv4 datagrams to each other. This is illustrated in Figure 4.41. Suppose node A is IPv6 capable and wants to send an IP datagram to node F, which is also IPv6-capable. Nodes A and B can exchange an IPv6 packet. However, node B must create an IPv4 datagram to send to C. Certainly, the data field of the IPv6 packet can be copied into the data field of the IPv4 datagram and appropriate address mapping can be done. However, in performing the conversion from IPv6 to IPv4, there will be IPv6-specific fields in the IPv6 datagram (for example, the flow identifier field) that have no counterpart in IPv4. The information in these fields will be lost. Thus, even though E and F can exchange IPv6 datagrams, the arriving IPv4 datagrams at E from D do not contain all of the fields that were in the original IPv6 datagram sent from A.

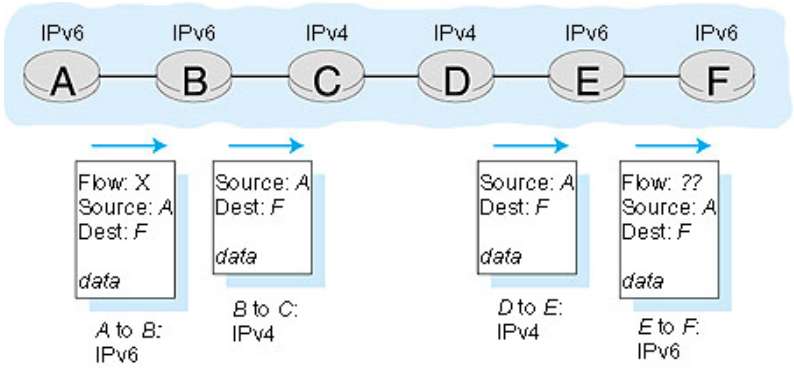


Figure 4.41: A dual-stack approach

An alternative to the dual-stack approach, also discussed in RFC 1933, is known as **tunneling.** Tunneling can solve the problem noted above, allowing, for example, E to receive the IPv6 datagram originated by A. The basic idea behind tunneling is the following. Suppose two IPv6 nodes (for example, B and E in Figure 4.41) want to interoperate using IPv6 datagrams, but are connected to each other by intervening IPv4 routers. We refer to the intervening set of IPv4 routers between two IPv6 routers as a **tunnel,** as illustrated in Figure 4.42. With tunneling, the IPv6 node on the sending side of the tunnel (for example, B) takes the *entire* IPv6 datagram and puts it in the data (payload) field of an IPv4 datagram. This IPv4 datagram is then addressed to the IPv6 node on the receiving side of the tunnel (for example, E) and sent to the first node in the tunnel (for example, C). The intervening IPv4 routers in the tunnel route this IPv4 datagram among themselves, just as they would any other datagram, blissfully unaware that the IPv4 datagram itself contains a complete IPv6 datagram. The IPv6 node on the receiving side of the tunnel eventually receives the IPv4 datagram (it is the destination of the IPv4 datagram!), determines that the IPv4 datagram contains an IPv6 datagram, extracts the IPv6 datagram, and then routes the IPv6 datagram exactly as it would if it had received the IPv6 datagram from a directly connected IPv6 neighbor.

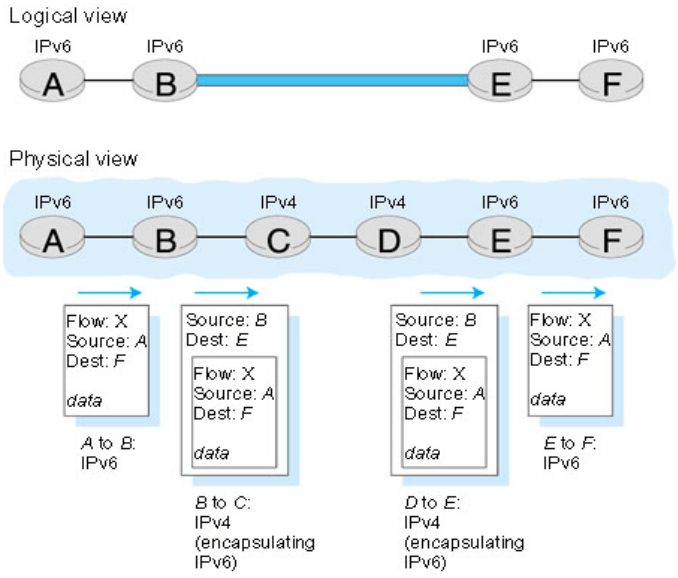


Figure 4.42: Tunneling

We end this section by mentioning that there is currently some doubt about whether IPv6 will make significant inroads into the Internet in the near future (2000-2002) or even ever at all [Garber 1999]. Indeed, at the time of this writing, a number of North American ISPs have said they don't plan to buy IPv6-enabled networking equipment. These ISPs say that there is little customer demand for IPv6's capabilities when IPv4, with some patches (such as CIDR, see Section 4.4.1, and network address translator [RFC 1631] boxes), is working well enough. On the other hand, there appears to be more interest in IPv6 in Europe and Asia.

One important lesson that we can learn from the IPv6 experience is that it is enormously difficult to change network-layer protocols. Since the early 1990s, numerous new network-layer protocols have been trumpeted as the next major revolution for the Internet, but most of these protocols have had limited penetration to date. These protocols include IPv6, multicast protocols (Section 4.8), and resource reservation protocols (Section 6.9). Indeed, introducing new protocols into the network layer is like replacing the foundation of a house--it is difficult to do without tearing the whole house down or at least temporarily relocated the house's residents. On the other hand, the Internet has witnessed rapid deployment of new protocols at the application layer. The classic example, of course, is HTTP and the Web; other examples include audio and video streaming and chat. Introducing new application layer protocols is like adding a new layer of paint to a house--it is relatively easy to do, and if you choose an attractive color, others in the neighborhood will copy you. In summary, in the future we can expect to see changes in the Internet's network layer, but these changes will likely occur on a time scale that is much slower than the changes that will occur at the application layer.

4.8: Multicast Routing

The transport- and network-layer protocols we have studied so far provide for the delivery of packets from a single source to a single destination. Protocols involving just one sender and one receiver are often referred to as **unicast protocols**.

A number of emerging network applications require the delivery of packets from one or more senders to a *group of receivers*. These applications include bulk data transfer (for example, the transfer of a software upgrade from the software developer to users needing the upgrade), streaming continuous media (for example, the transfer of the audio, video, and text of a live lecture to a set of distributed lecture participants), shared data applications (for example, a whiteboard or teleconferencing application that is shared among many distributed participants), data feeds (for example, stock quotes), WWW cache updating, and interactive gaming (for example, distributed interactive virtual environments or multiplayer games such as Quake). For each of these applications, an extremely useful abstraction is the notion of a **multicast:** the sending of a packet from one sender to multiple receivers with a single send operation.

In this section we consider the network-layer aspects of multicast. We continue our primary focus on the Internet here, as multicast is much more mature (although it is still undergoing significant development and evolution) in the Internet than in ATM networks. We will see that as in the unicast case, routing algorithms again

play a central role in the network layer. We will also see, however, that unlike the unicast case, Internet multicast is *not* a connectionless service--state information for a multicast connection must be established and maintained in routers that handle multicast packets sent among hosts in a so-called multicast group. This, in turn, will require a combination of signaling and routing protocols in order to set up, maintain, and tear down connection state in the routers.

4.8.1: Introduction: The Internet Multicast Abstraction and Multicast Groups

From a networking standpoint, the multicast abstraction--a single send operation that results in copies of the sent data being delivered to many receivers--can be implemented in many ways. One possibility is for the sender to use a separate unicast transport connection to each of the receivers. An application-level data unit that is passed to the transport layer is then duplicated at the sender and transmitted over each of the individual connections. This approach implements a one-sender-to-many-receivers multicast abstraction using an underlying unicast network layer [Talpade 1995; Chu 2000]. It requires no explicit multicast support from the network layer to implement the multicast abstraction; multicast is emulated using multiple point-to-point unicast connections. This is shown in the left of Figure 4.43, with network routers shaded in grey to indicate that they are not actively involved in supporting the multicast. Here, the multicast sender uses three *separate* unicast connections to reach the three receivers.

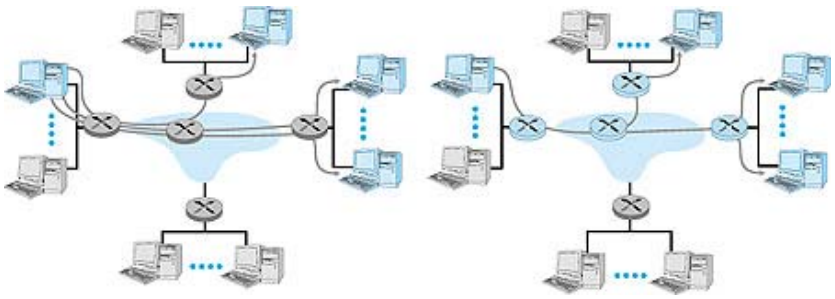


Figure 4.43: Two approaches toward implementing the multicast abstraction

A second alternative is to provide explicit multicast support at the network layer. In this latter approach, a *single* datagram is transmitted from the sending host. This datagram (or a copy of this datagram) is then replicated at a network router whenever it must be forwarded on multiple outgoing links in order to reach the receivers. The right side of Figure 4.43 illustrates this second approach, with certain routers shaded in color to indicate that they are actively involved in supporting the multicast. Here, a single datagram is transmitted by the sender. That datagram is then duplicated by the router within the network; one copy is forwarded to the uppermost receiver and another copy is forwarded toward the rightmost receivers. At the rightmost router, the multicast datagram is broadcast over the Ethernet that connects the two receivers to the rightmost router. Clearly, this second approach toward multicast makes more efficient use of network bandwidth in that only a *single* copy of a datagram will ever traverse a link. On the other hand, considerable network layer support is needed to implement a multicast-aware network layer. For the remainder of this section we will focus on a multicast-aware network layer, as this approach is implemented in the Internet and poses a number of interesting challenges.

With multicast communication, we are immediately faced with two problems that are much more complicated than in the case of unicast--how to identify the receivers of a multicast datagram and how to address a datagram sent to these receivers.

In the case of unicast communication, the IP address of the receiver (destination) is carried in each IP unicast datagram and identifies the single recipient. But in the case of multicast, we now have multiple receivers. Does it make sense for each multicast datagram to carry the IP addresses of all of the multiple recipients? While this approach might be workable with a small number of recipients, it would not scale well to the case of hundreds or thousands of receivers; the amount of addressing information in the datagram would swamp the amount of data actually carried in the datagram's payload field. Explicit identification of the receivers by the sender also requires that the sender know the identities and addresses of all of the receivers. We will see shortly that there are cases where this requirement might be undesirable.

For these reasons, in the Internet architecture (and the ATM architecture as well), a multicast datagram is addressed using **address indirection**. That is, a single identifier is used for the group of receivers, and a copy of the datagram that is addressed to the group using this single identifier is delivered to all of the multicast receivers associated with that group. In the Internet, the single identifier that represents a group of receivers is a Class D multicast address, as we saw earlier in Section 4.4. The group of receivers associated with

a class D address is referred to as a **multicast group**. The multicast group abstraction is illustrated in Figure 4.44. Here, four hosts (shown with shaded color screens) are associated with the multicast group address of 226.17.30.197 and will receive all datagrams addressed to that multicast address. The difficulty that we must still address is the fact that each host has a unique IP unicast address that is completely independent of the address of the multicast group in which it is participating.

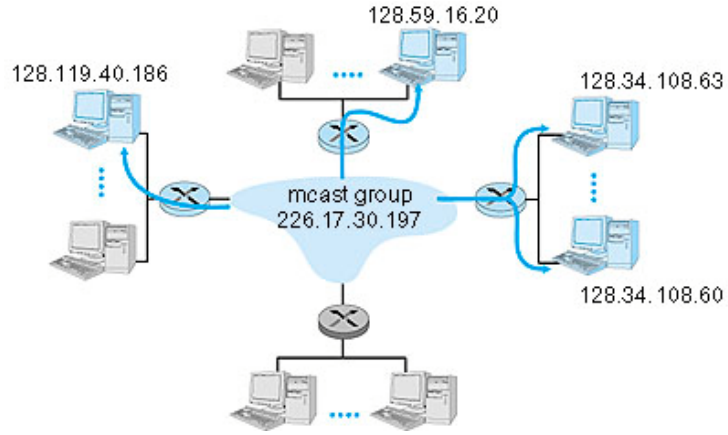


Figure 4.44: The multicast group: a datagram addressed to the group is delivered to all members of the multicast group

While the multicast group abstraction is simple, it raises a host (pun intended) of questions. How does a group get started and how does it terminate? How is the group address chosen? How are new hosts added to the group (either as senders or receivers)? Can anyone join a group (and send to, or receive from, that group) or is group membership restricted and if so, by whom? Do group members know the identities of the other group members as part of the network-layer protocol? How do the network routers interoperate with each other to deliver a multicast datagram to all group members? For the Internet, the answers to all of these questions involve the Internet Group Management Protocol [RFC 2236]. So, let us next consider the IGMP protocol and then return to these broader questions.

4.8.2: The IGMP Protocol

The **Internet Group Management Protocol, IGMP** version 2 [RFC 2236], operates between a host and its directly attached router (informally, think of the directly attached router as the "first-hop" router that a host would see on a path to any other host outside its own local network, or the "last-hop" router on any path to that host), as shown in Figure 4.45. Figure 4.45 shows three first-hop multicast routers, each connected to its attached hosts via one outgoing local interface. This local interface is attached to a LAN in this example, and while each LAN has multiple attached hosts, at most a few of these hosts will typically belong to a given multicast group at any given time.

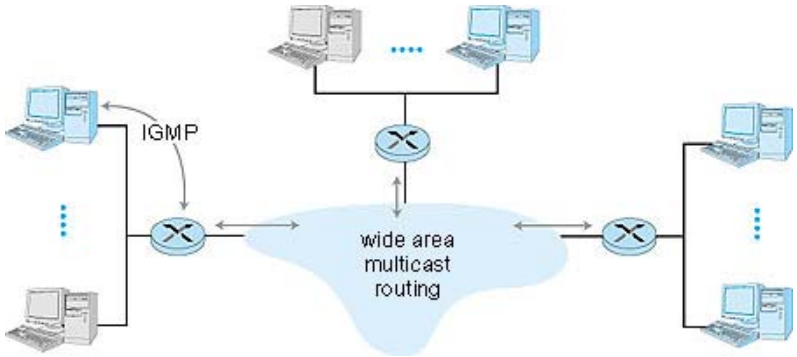


Figure 4.45: The two components of network-layer multicast: IGMP and multicast routing protocols

IGMP provides the means for a host to inform its attached router that an application running on the host wants to join a specific multicast group. Given that the scope of IGMP interaction is limited to a host and its attached router, another protocol is clearly required to coordinate the multicast routers (including the attached routers) throughout the Internet, so that multicast datagrams are routed to their final destinations. This latter

functionalit is accomplished by the **network-layer multicast routing algorithms** such as PIM, DVMRP and MOSFP. We will study multicast routing algorithms in Sections 4.8.3 and 4.8.4. Network-layer multicast in the Internet thus consists of two complementary components: IGMP and multicast routing protocols.

Although IGMP is referred to as a "group membership protocol," the term is a bit misleading since IGMP operates locally, between a host and an attached router. Despite its name, IGMP is *not* a protocol that operates among all the hosts that have joined a multicast group, hosts that may be spread around the world. Indeed, there is *no* network-layer multicast group membership protocol that operates among all the Internet hosts in a group. There is no network-layer protocol, for example, that allows a host to determine the identities of all of the other hosts, network-wide, that have joined the multicast group. (See the homework problems for a further exploration of the consequences of this design choice.)

IGMP version 2 [RFC 2236] has only three message types, as shown in Table 4.4. A general membership_query message is sent by a router to all hosts on an attached interface (for example, to all hosts on a local area network) to determine the set of all multicast groups that have been joined by the hosts on that interface. A router can also determine if a specific multicast group has been joined by hosts on an attached interface using a specific membership_query. The specific query includes the multicast address of the group being queried in the multicast group address field of the IGMP membership_query message, as shown in Figure 4.47.

Table 4.4: IGMP v2 Message types

IGMP Message Types	Sent by	Purpose
Membership query: general	router	Query multicast groups joined by attached hosts
Membership query: specific	router	Query if specific multicast group joined by attached hosts
Membership report	host	Report host wants to join or is joined to given multicast group
Leave group	host	Report leaving given multicast group

Hosts respond to a membership_query message with an IGMP membership_report message, as illustrated in Figure 4.46. Membership_report messages can also be generated by a host when an application first joins a multicast group without waiting for a membership_query message from the router. Membership_report messages are received by the router, as well as all hosts on the attached interface (for example, in the case of a LAN). Each membership_report contains the multicast address of a single group that the responding host has joined. Note that an attached router doesn't really care *which* hosts have joined a given multicast group or even *how many* hosts on the same LAN have joined the same group. (In either case, the router's work is the same--it must run a multicast routing protocol together with other routers to ensure that it receives the multicast datagrams for the appropriate multicast groups.) Since a router really only cares about whether one or more of its attached hosts belong to a given multicast group, it would ideally like to hear from only one of the attached hosts that belongs to each group (why waste the effort to receive identical responses from multiple hosts?). IGMP thus provides an explicit mechanism aimed at decreasing the number of membership_report messages generated when multiple attached hosts belong to the same multicast group.

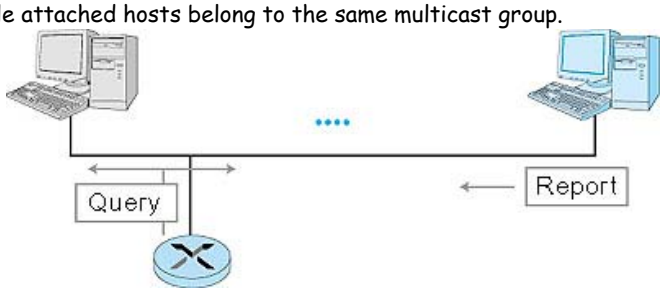


Figure 4.46: IGMP member query and membership report

Specifically, each membership_query message sent by a router also includes a "maximum response time" value field, as shown in Figure 4.47. After receiving a membership_query message and before sending a membership_report message for a given multicast group, a host waits a random amount of time between zero and the maximum response time value. If the host observes a membership_report message from some *other* attached host for that given multicast group, it *suppresses* (discards) its own pending membership_report message, since

the host now knows that the attached router already knows that one or more hosts are joined to that multicast group. This form of **feedback suppression** is thus a performance optimization--it avoids the transmission of unnecessary membership_report messages. Similar feedback suppression mechanisms have been used in a number of Internet protocols, including reliable multicast transport protocols [Floyd 1997].

Figure 4.47: IGMP message format

The final type of IGMP message is the leave_group message. Interestingly, this message is optional! But if it is optional, how does a router detect that there are no longer any hosts on an attached interface that are joined to a given multicast group? The answer to this question lies in the use of the IGMP membership_query message. The router infers that no hosts are joined to a given multicast group when no host responds to a membership_query message with the given group address. This is an example of what is sometimes called **soft state** in an Internet protocol. In a soft-state protocol, the state (in this case of IGMP, the fact that there are hosts joined to a given multicast group) is removed via a timeout event (in this case, via a periodic membership_query message from the router) if it is not explicitly refreshed (in this case, by a membership_report message from an attached host). It has been argued that soft-state protocols result in simpler control than hard-state protocols, which not only require state to be explicitly added and removed, but also require mechanisms to recover from the situation where the entity responsible for removing state has terminated prematurely or failed [Sharma 1997]. An excellent discussion of soft state can be found in [Raman 1999].

The IGMP message format is summarized in Figure 4.47. Like ICMP, IGMP messages are carried (encapsulated) within an IP datagram, with an IP protocol number of 2.

Having examined the protocol for joining and leaving multicast groups, we are now in a better position to reflect on the current Internet multicast service model, which is based on the work of Steve Deering [RFC 1112; Deering 1990]. In this multicast service model, any host can join a multicast group at the network layer. A host simply issues a membership_report IGMP message to its attached router. That router, working in concert with other Internet routers, will soon begin delivering multicast datagrams to the host. Joining a multicast group is thus receiver-driven. A sender need not be concerned with explicitly adding receivers to the multicast group but neither can it control who joins the group and therefore who receives datagrams sent to that group. Similarly, there is no control over who sends to the multicast group. Datagrams sent by different hosts can be arbitrarily interleaved at the various receivers (with different interleavings possible at different receivers). A malicious sender can inject datagrams into the multicast group datagram flow. Even with benign senders, since there is no network-layer coordination of the use of multicast addresses, it is possible that two different multicast groups will choose to use the same multicast address. From a multicast application viewpoint, this will result in interleaved extraneous multicast traffic.

These problems may seem to be insurmountable drawbacks for developing multicast applications. All is not lost, however. Although the network layer does not provide for filtering, ordering, or privacy of multicast datagrams, these mechanisms can all be implemented at the application layer. There is also ongoing work aimed at adding some of this functionality into the network layer [Cain 1999]. In many ways, the current Internet multicast service model reflects the same philosophy as the Internet unicast service model--an extremely simple network layer with additional functionality being provided in the upper-layer protocols in the hosts at the edges of the network. This philosophy has been unquestionably successful for the unicast case; whether the minimalist network layer philosophy will be equally successful for the multicast service model still remains an open question. An interesting discussion of an alternate multicast service model is [Holbrook 1999].

4.8.3: Multicast Routing: The General Case

In the previous section we have seen how the IGMP protocol operates at the edge of the network between a router and its attached hosts, allowing a router to determine what multicast group traffic it needs to receive for forwarding to its attached hosts. We can now focus our attention on just the multicast routers: how should they route packets amongst themselves in order to ensure that each router receives the multicast group traffic that it needs?

Figure 4.48 illustrates the setting for the **multicast routing problem**. Let us consider a single multicast group and assume that any router that has an attached host that has joined this group may either send or receive traffic addressed to this group. In Figure 4.48, hosts joined to the multicast group are shaded in color; their immediately attached router is also shaded in color. As shown in Figure 4.48, among the population of multicast routers, only a subset of these routers (those with attached hosts that are joined to the multicast group) actually need to receive the multicast traffic. In Figure 4.48, only routers A, B, E and F need to receive the multicast traffic. Since none of the attached hosts to router D are joined to the multicast group and since router C has no attached hosts, neither C nor D need to receive the multicast group traffic.

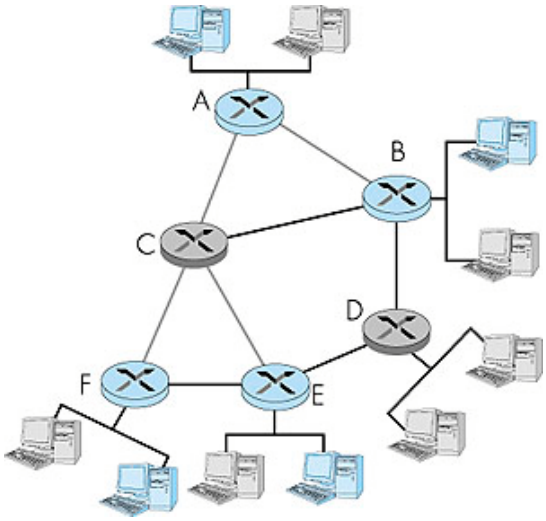


Figure 4.48: Multicast hosts, their attached routers, and other routers

The goal of multicast routing then is to find a tree of links that connects all of the routers that have attached hosts belonging to the multicast group. Multicast packets will then be routed along this tree from the sender to all of the hosts belonging to the multicast tree. Of course, the tree may contain routers that do not have attached hosts belonging to the multicast group (for example, in Figure 4.48, it is impossible to connect routers A, B, E, and F in a tree without involving either routers C and/or D).

In practice, two approaches have been adopted for determining the multicast routing tree. The two approaches differ according to whether a single tree is used to distribute the traffic for *all* senders in the group, or whether a source-specific routing tree is constructed for each individual sender:

- *Group-shared tree.* In the group-shared tree approach, only a *single* routing tree is constructed for the entire multicast group. For example, the single multicast tree shown with thicker shaded lines in the left of Figure 4.49, connects routers A, B, C, E, and F. (Following our conventions from Figure 4.48, router C is not shaded. Although it participates in the multicast tree, it has no attached hosts that are members of the multicast group). Multicast packets will flow only over those links shaded. Note that the links are bi-directional, since packets can flow in either direction on a link.

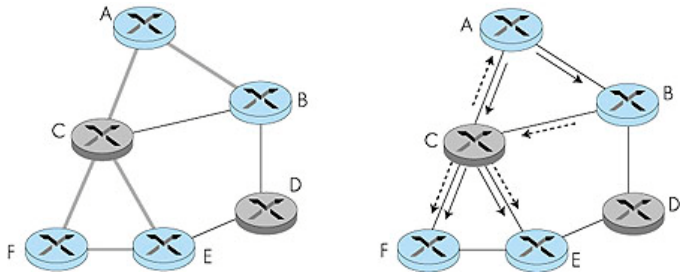


Figure 4.49: A single, shared tree (left), and two source-based trees (right)

- *Source-based trees.* In a source-based approach, an individual routing tree is constructed for *each* sender in the multicast group. In a multicast group with *N* hosts, *N* different routing trees will be constructed for that *single* multicast group. Packets will be routed to multicast group members in a source-specific manner. In the right of Figure 4.49, two source-specific multicast trees are shown, one

rooted at A and another rooted at B. Note that not only are there different links than in the group-shared tree case (for example, the link from BC is used in the source-specific tree routed at B, but not in the group-shared tree in the left of Figure 4.49), but that some links may also be used only in a single direction.

Multicast Routing Using a Group-Shared Tree

Let us first consider the case where all packets sent to a multicast group are to be routed along the same single multicast tree, regardless of the sender. In this case, the multicast routing problem appears quite simple: find a tree within the network that connects all routers having an attached host belonging to that multicast group. In Figure 4.49 (left), the tree composed of thick links is one such tree. Note that the tree contains routers that have attached hosts belonging to the multicast group (that is, routers A, B, E and F) as well as routers that have no attached hosts belonging to the multicast group. Ideally, one might also want the tree to have minimal "cost." If we assign a "cost" to each link (as we did for unicast routing in Section 4.2) then an optimal multicast routing tree is one having the smallest sum of the tree link costs. For the link costs given in Figure 4.50, the optimum multicast tree (with a cost of 7) is shown with thick lines.

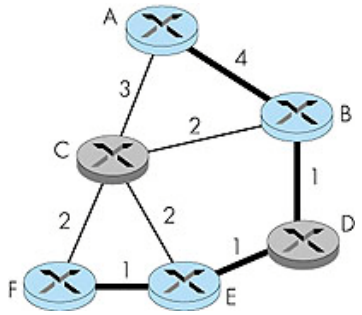


Figure 4.50: A minimum-cost multicast tree

The problem of finding a minimum cost tree is known as the **Steiner tree problem** [Hakimi 1971]. Solving this problem has been shown to be NP-complete [Garey 1978], but the approximation algorithm in [Kou 1981] has been proven to be within a constant of the optimal solution. Other studies have shown that, in general, approximation algorithms for the Steiner tree problem do quite well in practice [Wall 1980; Waxman 1988; Wei 1993].

Even though good heuristics exist for the Steiner tree problem, it is interesting to note that none of the existing Internet multicast routing algorithms have been based on this approach. Why? One reason is that information is needed about all links in the network. Another reason is that in order for a minimum-cost tree to be maintained, the algorithm needs to be rerun whenever link costs change. Finally, we will see that other considerations, such as the ability to leverage the routing tables that have already been computed for unicast routing, play an important role in judging the suitability of a multicast routing algorithm. In the end, performance (and optimality) is but one of many concerns.

An alternate approach toward determining the group-shared multicast tree, one that is used in practice by several Internet multicast routing algorithms, is based on the notion of defining a center node (also known as a **rendezvous point** or a **core**) in the single shared multicast routing tree. In the **center-based approach**, a center node is first identified for the multicast group. Routers with attached hosts belonging to the multicast group then unicast so-called join messages addressed to the center node. A join message is forwarded using unicast routing toward the center until it either arrives at a router that already belongs to the multicast tree or arrives at the center. In either case, the path that the join message has followed defines the branch of the routing tree between the edge router that initiated the join message and the center. One can think of this new path as being grafted onto the existing multicast tree for the group.

Figure 4.51 illustrates the construction of a center-based multicast routing tree. Suppose that router *E* is selected as the center of the tree. Node *F* first joins the multicast group and forwards a join message to *E*. The single link *EF* becomes the initial multicast tree. Node *B* then joins the multicast tree by sending its join message to *E*. Suppose that the unicast path route to *E* from *B* is via *D*. In this case, the join message results in the path *BDE* being grafted onto the multicast tree. Finally, node *A* joins the multicast group by forwarding its join message toward *E*. Let us assume that *A*'s unicast path to *E* is through *B*. Since *B* has already joined the multicast tree, the arrival of *A*'s join message at *B* will result in the *AB* link being immediately grafted onto the multicast tree.

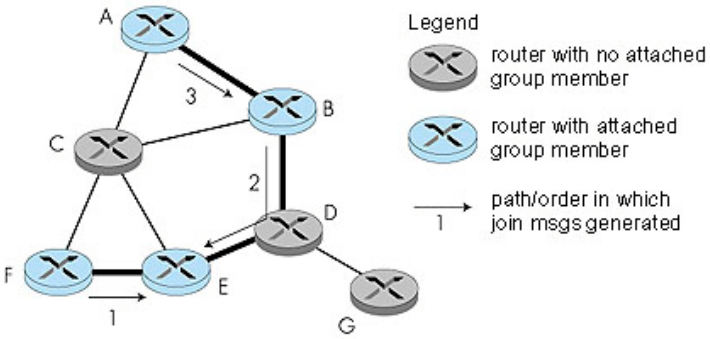


Figure 4.51: Constructing a center-based tree

A critical question for center-based tree multicast routing is the process used to select the center. Center-selection algorithms are discussed in [Wall 1980; Thaler 1997; Estrin 1997]. [Wall 1980] shows that centers can be chosen so that the resulting tree is within a constant factor of optimum (the solution to the Steiner tree problem).

Multicast Routing Using a Source-Based Tree

While the multicast routing algorithms we have studied above construct a single, shared routing tree that is used to route packets from *all* senders, the second broad class of multicast routing algorithms construct a multicast routing tree for *each* source in the multicast group.

We have already studied an algorithm (Dijkstra's link-state routing algorithm, in Section 4.2.1) that computes 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). Figure 4.52 shows the construction of the least cost path tree rooted at *A*. By comparing the tree in Figure 4.52 with that of Figure 4.50, it is evident that the least-cost path tree is *not* the same as the minimum overall cost tree computed as the solution to the Steiner tree problem. The reason for this difference is that the goals of these two algorithms are different: least unicast-cost path tree minimizes the cost from the source to each of the destinations (that is, there is no other tree that has a shorter distance path from the source to any of the destinations), while the Steiner tree minimizes the sum of the link costs in the tree. You might also want to convince yourself that the least unicast-cost path tree often differs from one source to another (for example, the source tree rooted at A is different from the source tree rooted at E in Figure 4.52).

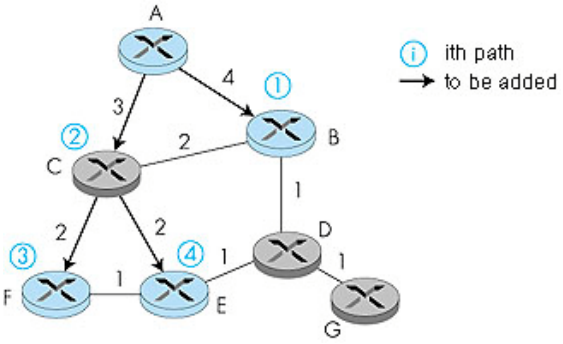


Figure 4.52: Construction of a least-cost path routing tree

The least-cost path multicast routing algorithm is a link-state algorithm. It requires that each router know the state of each link in the network in order to be able to compute the least-cost path tree from the source to all destinations. A simpler multicast routing algorithm, one that requires much less link state information than the least-cost path routing algorithm, is the **reverse path forwarding (RPF)** algorithm.

The idea behind reverse path forwarding is simple, yet elegant. When a router receives a multicast packet with a given source address, it transmits the packet on all of its outgoing links (except the one on which it was received) only if the packet arrived on the link that is on its own shortest path back to the sender. Otherwise, the router simply discards the incoming packet without forwarding it on any of its outgoing links. Such a packet can be dropped because the router knows it either will receive, or has already received, a copy of this packet on the link that is on its own shortest path back to the sender. (You might want to convince yourself that this will, in fact, happen.) Note that reverse path forwarding does not require that a router know the

complete shortest path from itself to the source; it need only know the next hop on its unicast shortest path to the sender.

Figure 4.53 illustrates RPF. Suppose that the links with thicker lines represent the least cost paths from the receivers to the source (*A*). Router *A* initially multicasts a source-S packet to routers *C* and *B*. Router *B* will forward the source-S packet it has received from *A* (since *A* is on its least-cost path to *A*) to both *C* and *D*. *B* will ignore (drop, without forwarding) any source-S packets it receives from any other routers (for example, from routers *C* or *D*).

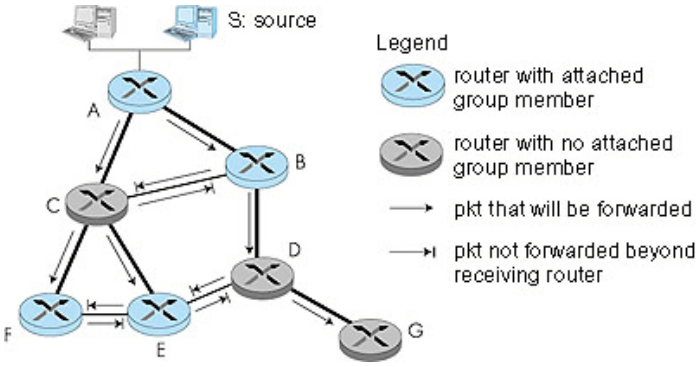


Figure 4.53: Reverse path forwarding

Let us now consider router *C*, which will receive a source-S packet directly from *A* as well as from *B*. Since *B* is not on *C*'s own shortest path back to *A*, *C* will ignore (drop) any source-S packets it receives from *B*. On the other hand, when *C* receives a source-S packet directly from *A*, it will forward the packet to routers *B*, *E*, and *F*.

RPF is a nifty idea. But consider what happens at router *D* in Figure 4.53. It will forward packets to router *G*, even though router *G* has no attached hosts that are joined to the multicast group. While this is not so bad for this case where *D* has only a single downstream router, *G*, imagine what would happen if there were thousands of routers downstream from *D*! Each of these thousands of routers would receive unwanted multicast packets. (This scenario is not as far-fetched as it might seem. The initial MBone [Casner 1992; Macedonia 1994], the first global multicast network suffered from precisely this problem at first!)

The solution to the problem of receiving unwanted multicast packets under RPF is known as **pruning**. A multicast router that receives multicast packets and has no attached hosts joined to that group will send a prune message to its upstream router. If a router receives prune messages from each of its downstream routers, then it can forward a prune message upstream. Pruning is illustrated in Figure 4.54.

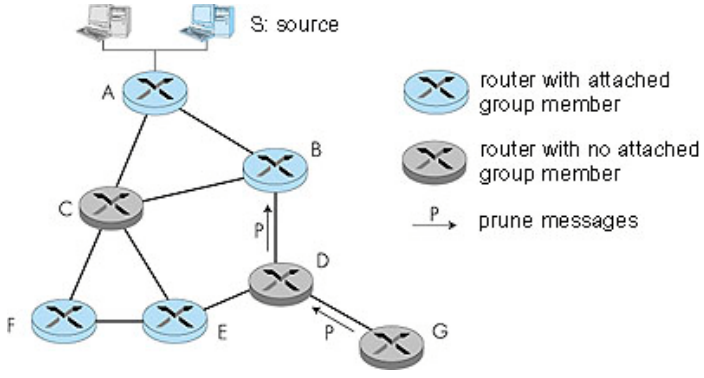


Figure 4.54: Pruning an RPF tree

While pruning is conceptually straightforward, two subtle issues arise. First, pruning requires that a router know which routers downstream are dependent on it for receiving their multicast packets. This requires additional information beyond that required for RPF alone. A second complication is more fundamental: if a router sends a prune message upstream, then what should happen if a router later needs to join that multicast group? Recall that under RPF, multicast packets are "pushed" down the RPF tree to all routers. If a prune message removes a branch from that tree, then some mechanism is needed to restore that branch. One possibility is to add a graft message that allows a router to "unprune" a branch. Another option is to allow pruned branches to

time-out and be added again to the multicast RPF tree; a router can then re-prune the added branch if the multicast traffic is still not wanted.

4.8.4: Multicast Routing in the Internet

Having now studied multicast routing algorithms in the abstract, let's now consider how these algorithms are put into practice in today's Internet by examining the three currently standardized Internet multicast routing protocols: DVMRP, MOSPF, CBT, and PIM.

DVMRP: Distance Vector Multicast Routing Protocol

The first multicast routing protocol used in the Internet and the most widely supported multicast routing algorithm [IP Multicast Initiative 1998] is the Distance Vector Multicast Routing Protocol (DVMRP) [RFC 1075]. DVMRP implements source-based trees with reverse path forwarding, pruning, and grafting. DVMRP uses a distance vector algorithm (see Section 4.2) that allows each router to compute the outgoing link (next hop) that is on its shortest path back to each possible source. This information is then used in the RPF algorithm, as discussed above. A public copy of DVMRP software is available at [mrouted 1996].

In addition to computing next-hop information, DVMRP also computes a list of dependent downstream routers for pruning purposes. When a router has received a prune message from all of its dependent downstream routers for a given group, it will propagate a prune message upstream to the router from which it receives its multicast traffic for that group. A DVMRP prune message contains a prune lifetime (with a default value of two hours) that indicates how long a pruned branch will remain pruned before being automatically restored. DVMRP graft messages are sent by a router to its upstream neighbor to force a previously pruned branch to be added back on to the multicast tree.

Before examining other multicast routing algorithms, let us consider how multicast routing can be deployed in the Internet. The crux of the problem is that only a small fraction of the Internet routers are multicast-capable. If one router is multicast-capable but all of its immediate neighbors are not, is this lone island of multicast-routing lost in a sea of unicast routers? Most decidedly not! Tunneling, a technique we examined earlier in the context of IP version 6 (Section 4.7), can be used to create a virtual network of multicast-capable routers on top of a physical network that contains a mix of unicast and multicast routers. This is the approach taken in the Internet MBone.

Multicast tunnels are illustrated in Figure 4.55. Suppose that multicast router *A* wants to forward a multicast datagram to multicast router *B*. Suppose that *A* and *B* are not physically connected to each other and that the intervening routers between *A* and *B* are not multicast capable. To implement tunneling, router *A* takes the multicast datagram and "encapsulates" it [RFC 2003] inside a standard unicast datagram. That is, the entire multicast datagram (including source and multicast address fields) is carried as the payload of an IP unicast datagram--a complete multicast IP datagram inside of a unicast IP datagram! The unicast datagram is then addressed to the unicast address of router *B* and forwarded toward *B* by router *A*. The unicast routers between *A* and *B* dutifully forward the unicast packet to *B*, blissfully unaware that the unicast datagram itself contains a multicast datagram. When the unicast datagram arrives at *B*, *B* then extracts the multicast datagram. *B* may then forward the multicast datagram on to one of its attached hosts, forward the packet to a directly attached neighboring router that is multicast-capable, or forward the multicast datagram to another logical multicast neighbor via another tunnel.

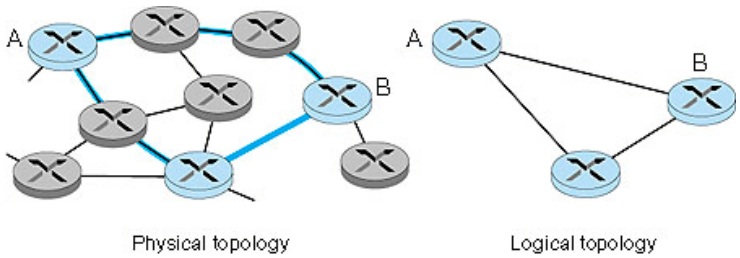


Figure 4.55: Multicast tunnels

MOSPF: Multicast Open Shortest Path First

The Multicast Open Shortest Path First protocol (MOSPF) [RFC 1584] operates in an autonomous system (AS) that uses the OSPF protocol (see Section 4.5) for unicast routing. MOSPF extends OSPF by having routers add their multicast group membership to the link state advertisements that are broadcast by routers as part of the OSPF protocol. With this extension, all routers have not only complete topology information, but also know which edge routers have attached hosts belonging to various multicast groups. With this information, the routers within the AS can build source-specific, pre-pruned, shortest-path trees for each multicast group.

CBT: Core-Based Trees

The core-based tree (CBT) multicast routing protocol [RFC 2201; RFC 2189] builds a bi-directional, group-shared tree with a single "core" (center). A CBT edge router unicasts a JOIN_REQUEST message toward the tree core. The core, or the first router that receives this JOIN_REQUEST and itself has already successfully joined the tree, will respond with a JOIN_ACK message to the edge router. Once a multicast routing tree has been built, it is maintained by having a downstream router send keepalive messages (ECHO_REQUEST) to its immediate upstream router. The immediate upstream router responds with an ECHO_REPLY message. These messages are exchanged at a time granularity of minutes. If a downstream router receives no reply to its ECHO_REQUEST, it will retry sending the ECHO_REQUEST for a small number of times. If no ECHO_REPLY is received, the router will dissolve the downstream tree by sending a FLUSH_TREE message downstream.

PIM: Protocol Independent Multicast

The Protocol Independent Multicast (PIM) routing protocol [Deering 1996; RFC 2362; Estrin 1998b] explicitly envisions two different multicast distribution scenarios. In so-called **dense mode,** multicast group members are densely located, that is, many or most of the routers in the area need to be involved in routing multicast datagrams. In **sparse mode,** the number of routers with attached group members is small with respect to the total number of routers; group members are widely dispersed.

The PIM designers noted several consequences of the sparse-dense dichotomy. In dense mode, since most routers will be involved in multicast (for example, have attached group members), it is reasonable to assume that each and every router should be involved in multicast. Thus, an approach like RPF that floods datagrams to every multicast router (unless a router explicitly prunes itself), is well-suited to this scenario. On the other hand, in sparse mode, the routers that need to be involved in multicast forwarding are few and far between. In this case, a data-driven multicast technique like RPF, which *forces* a router to constantly do work (prune) simply to avoid receiving multicast traffic, is much less satisfactory. In sparse mode, the default assumption should be that a router is not involved in a multicast distribution; the router should *not* have to do any work unless it wants to join a multicast group. This argues for a center-based approach, where routers send explicit join messages, but are otherwise uninvolved in multicast forwarding. One can think of the sparse-mode approach as being receiver-driven (that is, nothing happens until a receiver explicitly joins a group) versus the dense-mode approach as being data-driven (that is, that datagrams are multicast everywhere, unless explicitly pruned).

PIM accommodates this dense versus sparse dichotomy by offering two explicit modes of operation: dense mode and sparse mode. PIM Dense Mode is a flood-and-prune reverse-path-forwarding technique similar in spirit to DVMRP. Recall that PIM is protocol-independent, that is, independent of the underlying unicast routing protocol. A better description might be that it can interoperate with any underlying unicast routing protocol. Because PIM makes no assumptions about the underlying routing protocol, its reverse path forwarding algorithm is slightly simpler, although slightly less efficient than DVMRP.

PIM Sparse Mode is a center-based approach. PIM routers send JOIN messages toward a rendezvous point (center) to join the tree. As with CBT, intermediate routers set up multicast state and forward the JOIN message toward the rendezvous point. Unlike CBT, there is no acknowledgment generated in response to a JOIN message. JOIN messages are periodically sent upstream to refresh/maintain the PIM routing tree. One novel feature of PIM is the ability to switch from a group-shared tree to a source-specific tree after joining the rendezvous point. A source-specific tree may be preferred due to the decreased traffic concentration that occurs when multiple source-specific trees are used (see homework problems).

In PIM Sparse Mode, the router that receives a datagram to send from one of its attached hosts will unicast the datagram to the rendezvous point. The rendezvous point then multicasts the datagram via the group-shared tree. A sender is notified by the RP that it must stop sending to the RP whenever there are no routers joined to the tree (that is, no one is listening!).

PIM is implemented in numerous router platforms [IP Multicast Initiative 1998] and has been deployed in UUnet as part of their streaming multimedia delivery effort [LaPolla 1997].

Inter-Autonomous System Multicast Routing

In our discussion above, we have implicitly assumed that all routers are running the same multicast routing protocol. As we saw with unicasting, this will typically be the case within a single autonomous system (AS). However, different ASs may choose to run different multicast routing protocols. One AS might choose to run PIM within its autonomous system, while another may choose to run MOSPF. Interoperability rules have been defined for all of the major Internet multicast routing protocols. (This is a particularly messy issue due to the very different approaches taken to multicast routing by sparse and dense mode protocols.) What is still missing, however, is an *inter-AS* multicast routing protocol to route multicast datagrams among different AS's.

DVMRP has been the *de facto* inter-AS multicast routing protocol. However, as a dense-mode protocol, it is not particularly well-suited to the rather sparse set of routers participating in today's Internet MBone. The

development of an inter-AS multicast protocol is an active area of research and development being carried out by the *idmr* working group of the IETF [IDMR 1998].

Having now considered the multicast routing problem and a number of multicast protocols embodying the group-shared tree and source-based tree approaches, let us conclude by enumerating some of the factors involved in evaluating a multicast protocol:

- *Scalability.* What is the amount of state required in the routers by a multicast routing protocol? How does the amount of state change as the number of groups, or number of senders in a group, change?
- *Reliance on underlying unicast routing.* To what extent does a multicast protocol rely on information maintained by an underlying unicast routing protocol? We have seen solutions that range from reliance on one specific underlying unicast routing protocol (MOSPF), to a solution that is completely independent of the underlying unicast routing (PIM), to a solution that implements much of the same distance vector functionality that we saw earlier for the unicast case (DVMRP).
- *Excess (un-needed) traffic received.* We have seen solutions where a router receives data only if it has an attached host in the multicast group (MOSPF, PIM-Sparse Mode) to solutions where the default is for a router to receive all traffic for all multicast groups (DVMRP, PIM Dense Mode).
- *Traffic concentration.* The group-shared tree approach tends to concentrate traffic on a smaller number of links (those in the single tree), whereas source-specific trees tend to distribute multicast traffic more evenly.
- *Optimality of forwarding paths.* We have seen that determining the minimum cost multicast tree (that is, solving the Steiner problem) is difficult and that this approach has not been adopted in practice. Instead, heuristic approaches, based on either using the tree of shortest paths, or selecting a center router from which to "grow" the routing multicast tree, have been adopted in practice.

4.9: 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 one of the biggest challenges in the network layer is routing datagrams through a network of millions of hosts and routers. We saw that this scaling problem is solved 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. In the Internet, two popular protocols for intra-AS routing are currently RIP and OSPF. To route packets among ASs, an inter-AS routing protocol is needed. The dominant inter-AS protocol today is BGP4.

Performing routing on two levels--one level for within each of the ASs and another level for among the ASs--is referred to as hierarchical routing. The scaling problem is largely solved by a hierarchical organization of the routing infrastructure. This is a general principle we should keep in mind when designing protocols, particularly for network-layer protocols: scaling problems can often be solved by hierarchical organization. It is interesting to note that this principle has been applied throughout the ages to many other disciplines besides computer networking, including corporate, government, religious, and military organizations.

In this chapter, we also learned about a second scaling issue: For large computer networks, 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 to *not* keep track of individual flows, but instead base routing decisions solely on hierarchical-structured destination addresses in the packets. It is interesting to note that the postal service has been using this same trick for many years.

In this chapter, we also looked at the underlying principles of routing algorithms. We learned that designers of 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 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 only have a partial picture of the entire network, yet the nodes work together to deliver packets along the shortest routes. Routing algorithms in computer networks have been an active research area for many years, and will undoubtedly remain so.

At the end of this chapter, we examined two advanced subjects, reflecting current trends in computer networking and the Internet. The first subject is IPv6, which provides a streamlined network layer and resolves the IPv4 address space problem. The second subject is multicast routing, which can potentially save tremendous amounts of bandwidth, router, and server resources in a computer network. It will be interesting to see how the deployment of IPv6 and multicast routing protocols play out over the next decade.

Having completed our study of the network layer, our journey now takes us one further step 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 to 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.