Data Communication

3. The Network Layer

4.1- Introduction

Figure 4.1 shows a simple network with two hosts, H1 and H2, and several routers on the path between H1 and H2. Suppose that H1 is sending information to H2, and consider the role of the network layer in these hosts and in the intervening routers. The network layer in H1 takes segments from the transport layer in H1, encapsulates each segment into a datagram (that is, a network-layer packet), and then sends the datagrams to its nearby router, R1. At the receiving host, H2, the network layer receives the datagrams from its nearby router R2, extracts the transport-layer segments, and delivers the segments up to the transport layer at H2. The primary role of the routers is to forward datagrams 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 application- and transport-layer protocols such as those we examined in Chapters 2 and 3.

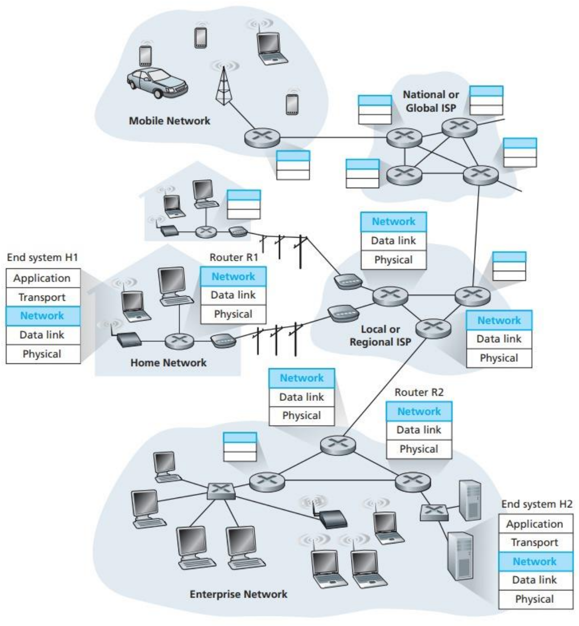


Figure 4.1 ♦ The network layer

FORWARDING AND ROUTING

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

→ Forwarding. When a packet arrives at a router's input link, the router must move the packet to the appropriate output link. For example, a packet arriving from Host H1 to Router R1 must be forwarded to the next router on a path to H2.

In Section 4.3, we'll look inside a router and examine how a packet is actually forwarded from an input link to an output link within a router.

→ Routing. 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.

The terms forwarding and routing are often used interchangeably by authors discussing the network layer. We'll use these terms much more precisely in this book. Forwarding refers to the router-local action of transferring a packet from an input link interface to the appropriate output link interface. Routing refers to the network-wide process that determines the end-to-end paths that packets take from source to destination. Using a driving analogy, consider the trip from Pennsylvania to Florida undertaken by our traveller back in Section 1.3.1. During this trip, our driver passes through many interchanges en route to Florida. We can think of forwarding as the process of getting through a single interchange: A car enters the interchange from one road and determines which road it should take to leave the interchange. We can think of routing as the process of planning the trip from Pennsylvania to Florida: Before embarking on the trip, the driver has consulted a map and chosen one of many paths possible, with each path consisting of a series of road segments connected at interchanges.

Every router has a **forwarding table**. A router forwards a packet by examining the value of a field in the arriving packet's header, and then using this header value to index into the router's forwarding table. The value stored in the forwarding table entry for that header indicates the router's outgoing link interface to which that packet is to be forwarded. Depending on the network-layer protocol, the header value could be the destination address of the packet or an indication of the connection to which the packet belongs. Figure 4.2 provides an example. In Figure 4.2, a packet with a header field value of 0111 arrives to a router. The router indexes into its forwarding table and determines that the output link interface for this packet is interface 2. The router then internally forwards the packet to interface 2. In Section 4.3, we'll look inside a router and examine the forwarding function in much greater detail.

You might now be wondering how the forwarding tables in the routers are configured. This is a crucial issue, one that exposes the important interplay between routing and forwarding. As shown in Figure 4.2, the routing algorithm determines the values that are inserted into the routers' forwarding tables. The routing algorithm may be centralized (e.g., with an algorithm executing on a central site and downloading routing information to each of the routers) or decentralized (i.e., with a piece of the distributed routing algorithm running in each router). In either case, a router receives routing protocol messages, which are used to configure its forwarding table. The distinct and different purposes of the forwarding and routing functions can be further illustrated by considering the hypothetical (and unrealistic, but technically feasible) case of a network in which all forwarding tables are configured directly by human network operators physically present at the routers. In this case, no routing protocols would be required! Of course, the human operators would need to interact with each other to ensure that the forwarding tables were configured in such a way that packets reached their intended destinations. It’s also likely that human configuration would be more error-prone and much slower to respond to changes in the network topology than a routing protocol. We’re thus fortunate that all networks have both a forwarding and a routing function!

While we’re on the topic of terminology, it’s worth mentioning two other terms that are often used interchangeably, but that we will use more carefully. We’ll reserve the term packet switch to mean a general packet-switching device that transfers a packet from input link interface to output link interface, according to the value in a field in the header of the packet. Some packet switches,

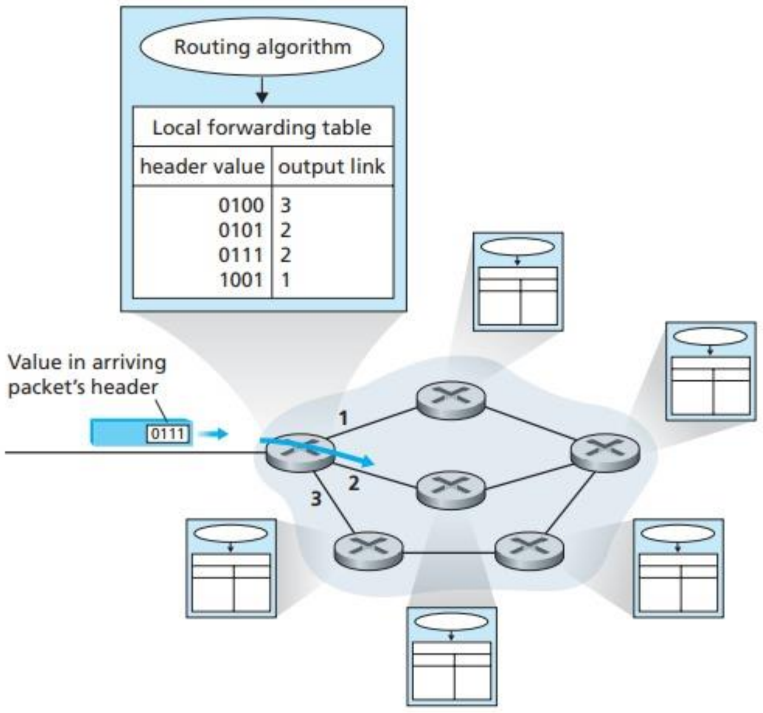


Figure 4.2 ♦ Routing algorithms determine values in forwarding tables

called link-layer switches (examined in Chapter 5), base their forwarding decision on values in the fields of the linklayer frame; switches are thus referred to as link-layer (layer 2) devices. Other packet switches, called routers, base their forwarding decision on the value in the networklayer field. Routers are thus network-layer (layer 3) devices, but must also implement layer 2 protocols as well, since layer 3 devices require the services of layer 2 to implement their (layer 3) functionality. (To fully appreciate this important distinction, you might want to review Section 1.5.2, where we discuss network-layer datagrams and link-layer frames and their relationship.) To confuse matters, marketing literature often refers to “layer 3 switches” for routers with Ethernet interfaces, but these are really layer 3 devices. Since our focus in this chapter is on the network layer, we use the term router in place of packet switch. We’ll even use the term router when talking about packet switches in virtual-circuit networks (soon to be discussed).

Connection Setup

We just said that the network layer has two important functions, forwarding and routing.

But we'll soon see that in some computer networks there is actually a third important network-layer function, namely, **connection setup**. Recall from our study of TCP that a three-way handshake is required before data can flow from sender to receiver. This allows 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, frame relay, and MPLS (which we will study in Section 5.8)—require the routers along the chosen path from source to destination to handshake with each other in order to set up state before network-layer data packets within a given source-to-destination connection can begin to flow. In the network layer, this process is referred to as connection setup. We'll examine connection setup in Section 4.2. 4.1.2

Network Service Models

Before delving into the network layer, let's take the broader view and consider the different types of service that might be offered by the network layer. 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 rely 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 packets between sending and receiving end systems.

Let's now consider some possible services that the network layer could provide. In the sending host, when the transport layer passes a packet to the network layer, specific services that could be provided by the network layer include:

→Guaranteed delivery. This service guarantees that the packet will eventually arrive at its destination.

→Guaranteed delivery with bounded delay. This service not only guarantees delivery of the packet, but delivery within a specified host-to-host delay bound (for example, within 100 msec). Furthermore, the following services could be provided to a flow of packets between a given source and destination:

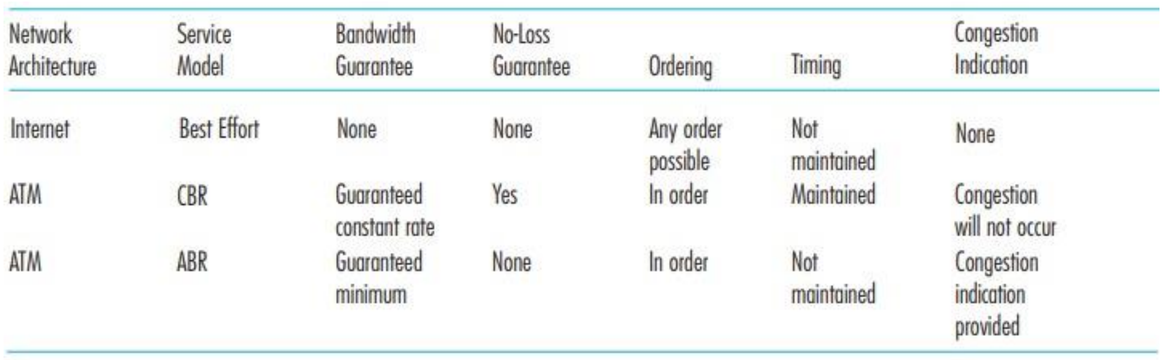
→In-order packet delivery. This service guarantees that packets arrive at the destination in the order that they were sent.

→Guaranteed minimal bandwidth. This network-layer service emulates the behavior of a transmission link of a specified bit rate (for example, 1 Mbps) between sending and receiving hosts. As long as the sending host transmits bits (as part of packets) at a rate below the specified bit rate, then no packet is lost and each packet arrives within a prespecified host-to-host delay (for example, within 40 msec).

→Guaranteed maximum jitter. This service guarantees that the amount of time between the transmission of two successive packets at the sender is equal to the amount of time between their receipt at the destination (or that this spacing changes by no more than some specified value).

→Security services. Using a secret session key known only by a source and destination host, the network layer in the source host could encrypt the payloads of all datagrams being sent to the destination host. The network layer in the destination host would then be responsible for decrypting the payloads. With such a service, confidentiality would be provided to all transport-layer segments (TCP and UDP) between the source and destination hosts. In addition to confidentiality, the network layer could provide data integrity and source authentication services.

This is only a partial list of services that a network layer could provide—there are countless variations possible. The Internet's network layer provides a single service, 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. As we'll discuss shortly, however, there are sound reasons for such a minimalist network-layer service model.



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	ABR	Guaranteed minimum	None	In order	Not maintained	Congestion indication provided

Table 4.1 ♦ Internet, ATM CBR, and ATM ABR service models

Other network architectures have defined and implemented service models that go beyond the Internet's best-effort service. For example, the ATM network architecture [MFA Forum 2012, Black 1995] provides for multiple service models, meaning that different connections can be provided with different classes of service within the same network. A discussion of how an ATM network provides such services is well beyond the scope of this book; our aim here is only to note that alternatives do exist to the Internet's best-effort model. Two of the more important ATM service models are constant bit rate and available bit rate service:

→**Constant bit rate (CBR) ATM network service**. This was the first ATM service model to be standardized, reflecting early interest by the telephone companies in ATM and the suitability of CBR service for carrying real-time, constant bit rate audio and video traffic.

The goal of CBR service is conceptually simple—to provide a flow of packets (known as cells in ATM terminology) with a virtual pipe whose properties are the same as if a dedicated fixed-bandwidth transmission link existed between sending and receiving hosts. With CBR service, a flow of ATM cells is carried across the network in such a way that a cell's end-to-end delay, the variability in a cell's end-to-end delay (that is, the jitter), and the fraction of cells that are lost or delivered late are all guaranteed to be less than specified values. These values are agreed upon by the sending host and the ATM network when the CBR connection is first established.

→**Available bit rate (ABR) ATM network service**. With the Internet offering socalled best-effort service, ATM's ABR might best be characterized as being a slightly-better-than-best-effort service. As with the Internet service model, cells may be lost under ABR service. Unlike in the Internet, however, cells cannot be reordered (although they may be lost), and a minimum cell transmission rate (MCR) is guaranteed to a connection using ABR service. If the network has enough free resources at a given time, a sender may also be able to send cells successfully at a higher rate than the MCR. Additionally, as we saw in Section 3.6, ATM ABR service can provide feedback to the sender (in terms of a congestion notification bit, or an explicit rate at which to send) that controls how the sender adjusts its rate between the MCR and an allowable peak cell rate.

4.2Virtual Circuit and Datagram Networks

Recall from Chapter 3 that a transport layer can offer applications connectionless service or connection-oriented service between two processes. For example, the Internet's transport layer provides each application a choice between two services: UDP, a connectionless service; or TCP, a connection-oriented service. In a similar manner, a network layer can provide connectionless service or connection service between two hosts. Network-layer connection and connectionless services in many ways parallel transport-layer connection-oriented and connectionless services. For example, a network-layer connection service begins with handshaking between the source and destination hosts; and a network-layer connectionless service does not have any handshaking preliminaries. Although the network-layer connection and connectionless services have some parallels with transport-layer connection-oriented and connectionless services, there are crucial differences:

→In the network layer, these services are host-to-host services provided by the network layer for the transport layer. In the transport layer these services are processto-process services provided by the transport layer for the application layer.

→In all major computer network architectures to date (Internet, ATM, frame relay, and so on), the network layer provides either a host-to-host connectionless service or a host-to-host connection service, but not both. Computer networks that provide only a connection service at the network layer are called **virtual-circuit (VC) networks**; computer networks that provide only a connectionless service at the network layer are called **datagram networks.**

→The implementations of connection-oriented service in the transport layer and the connection service in the network layer are fundamentally different. We saw in the previous chapter that the transport-layer connection-oriented service is implemented at the edge of the network in the end systems; we'll see shortly that the network-layer connection service is implemented in the routers in the network core as well as in the end systems.

Virtual-circuit and datagram networks are two fundamental classes of computer networks. They use very different information in making their forwarding decisions. Let's now take a closer look at their implementations.

4.2.1 Virtual-Circuit Networks

While the Internet is a datagram network, many alternative network architectures— including those of ATM and frame relay—are virtual-circuit networks and, therefore, use connections at the network layer. These network-layer connections are called **virtual circuits (VCs).** Let's now consider how a VC service can be implemented in a computer network.

A VC consists of (1) a path (that is, a series of links and routers) between the source and destination hosts, (2) VC numbers, one number for each link along the path, and (3) entries in the forwarding table in each router along the path. A packet belonging to a virtual circuit will carry a VC number in its header. Because a virtual circuit may have a different VC number on each link, each intervening router must replace the VC number of each traversing packet with a new VC number. The new VC number is obtained from the forwarding table.

To illustrate the concept, consider the network shown in Figure 4.3. The numbers next to the links of R1 in Figure 4.3 are the link interface numbers. Suppose now that Host A requests that the network establish a VC between itself and Host B. Suppose also that the network chooses the path A-R1-R2-B and assigns VC numbers 12, 22, and 32 to the three links in this path for this virtual circuit. In this case, when a packet in this VC leaves Host A, the value in the VC number field in the packet header is 12; when it leaves R1, the value is 22; and when it leaves R2, the value is 32. How does the router determine the replacement VC number for a packet traversing the router? For a VC network, each router's forwarding table includes VC

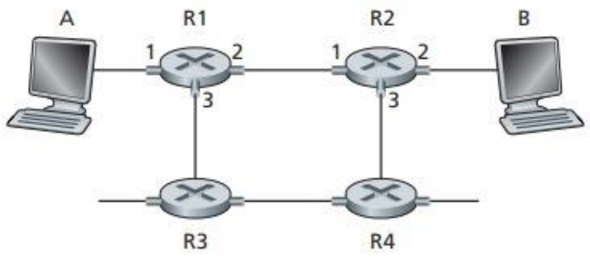


Figure 4.3 ♦ A simple virtual circuit network

number translation; for example, the forwarding table in R1 might look something like this:

Incoming Interface	Incoming VC #	Outgoing Interface	Outgoing VC #
1	12	2	22
2	63	1	18
3	7	2	17
1	97	3	87
...	...	...	...

Whenever a new VC is established across a router, an entry is added to the forwarding table. Similarly, whenever a VC terminates, the appropriate entries in each table along its path are removed.

You might be wondering why a packet doesn't just keep the same VC number on each of the links along its route. The answer is twofold. First, replacing the number from link to link reduces the length of the VC field in the packet header. Second, and more importantly, VC setup is considerably simplified by permitting a different VC number at each link along the path of the VC. Specifically, with multiple VC numbers, each link in the path can choose a VC number independently of the VC numbers chosen at other links along the path. If a common VC number were required for all links along the path, the routers would have to exchange and process a substantial number of messages to agree on a common VC number (e.g., one that is not being used by any other existing VC at these routers) to be used for a connection.

In a VC network, the network's routers must maintain **connection state information** for the ongoing connections. Specifically, each time a new connection is established across a router, a new connection entry must be added to the router's forwarding table; and each time a connection is released, an entry must be removed from the table. Note that even if there is no VC-number translation, it is still necessary to maintain connection state information that associates VC numbers with output interface numbers. The issue of whether or not a router maintains connection state information for each ongoing connection is a crucial one—one that we'll return to repeatedly in this book.

There are three identifiable phases in a virtual circuit:

→VC setup. During the setup phase, the sending transport layer contacts the network layer, specifies the receiver's 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 routers through which all packets of the VC will travel. The network layer also determines the VC number for each link along the path. Finally, the network layer adds an entry in the forwarding table in each router along the path. During VC setup, the network layer may also reserve resources (for example, bandwidth) along the path of the VC.

→Data transfer. As shown in Figure 4.4, once the VC has been established, packets can begin to flow along the VC.

→VC 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 forwarding tables in each of the packet routers 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.

During transport-layer connection setup, the two end systems alone determine the parameters (for example, initial sequence number and flow-control window size) of their transport-layer connection. Although the two end systems are aware of the transport-layer connection, the routers within the network are completely oblivious to it. On the other hand, with a VC network layer, *routers along the path between the two end systems are involved in VC setup, and each router is fully aware of all the VCs passing through it.*

The messages that the end systems send into the network to initiate or terminate a VC, and the messages passed between the routers to set up the VC (that is, to modify connection state in router 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 4.4. We'll not cover VC signaling protocols in this book; see [Black 1997] for a general discussion of signaling in connection-oriented networks and [ITU-T Q.2931 1995] for the specification of ATM's Q.2931 signaling protocol.

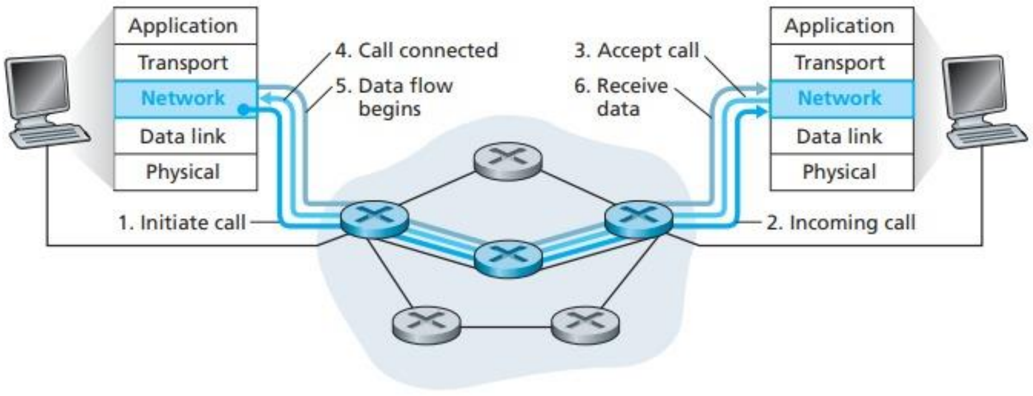


Figure 4.4 ♦ Virtual-circuit setup

4.2.2 Datagram Networks

In a **datagram network**, 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 4.5, there is no VC setup and routers do not maintain any VC state information (because there are no VCs!).

As a packet is transmitted from source to destination, it passes through a series of routers. Each of these routers uses the packet's destination address to forward the packet. Specifically, each router has a forwarding table that maps destination addresses to link interfaces; when a packet arrives at the router, the router uses the packet's destination address to look up the appropriate output link interface in the forwarding table. The router then intentionally forwards the packet to that output link interface.

To get some further insight into the lookup operation, let's look at a specific example. Suppose that all destination addresses are 32 bits (which just happens to be the length of the destination address in an IP datagram). A brute-force implementation of the forwarding table would have one entry for every possible destination address. Since there are more than 4 billion possible addresses, this option is totally out of the question.

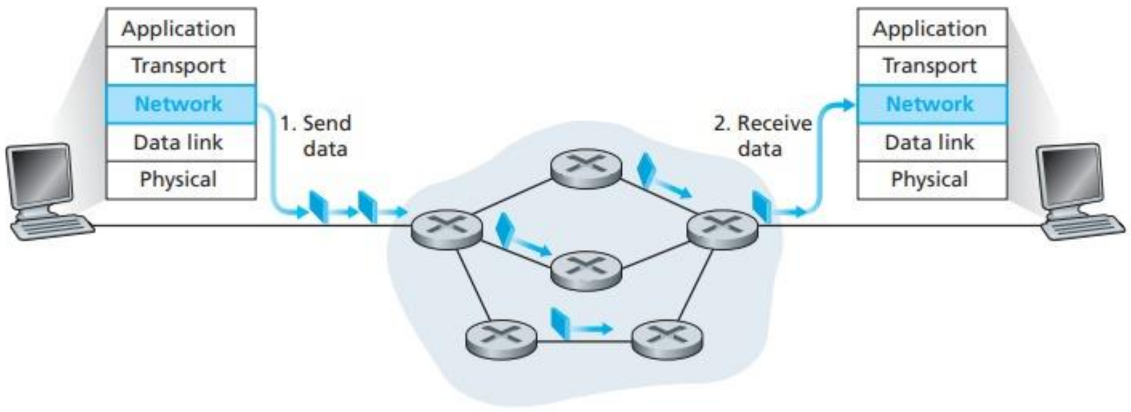


Figure 4.5 ♦ Datagram network

Now let's further suppose that our router has four links, numbered 0 through 3, and that packets are to be forwarded to the link interfaces as follows:

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Clearly, for this example, it is not necessary to have 4 billion entries in the router's forwarding table. We could, for example, have the following forwarding table with just four entries:

Prefix Match	Link Interface
11001000 00010111 00010	0
11001000 00010111 00011000	1
11001000 00010111 00011	2
otherwise	3

With this style of forwarding table, the router matches a prefix of the packet's destination address with the entries in the table; if there's a match, the router forwards the packet to a link associated with the match. For example, suppose the packet's destination address is 11001000 00010111 00010110 10100001; because the 21-bit prefix of this address matches the first entry in the table, the router forwards the packet to link interface 0. If a prefix doesn't match any of the first three entries, then the router forwards the packet to interface 3. Although this sounds simple enough, there's an important subtlety here. You may have noticed that it is possible for a destination address to match more than one entry. For example, the first 24 bits of the address 11001000 00010111 00011000 10101010 match the second entry in the table, and the first 21 bits of the address match the third entry in the table. When there are multiple matches, the router uses the longest prefix matching rule; that is, it finds the longest matching entry in the table and forwards the packet to the link interface associated with the longest prefix match. We'll see exactly why this longest prefix-matching rule is used when we study Internet addressing in more detail in Section 4.4.

Although routers in datagram networks maintain no connection state information, they nevertheless maintain forwarding state information in their forwarding tables. However, the time scale at which this forwarding state information changes is relatively slow. Indeed, in a datagram network the forwarding tables are modified by the routing algorithms, which typically update a forwarding table every one-to five minutes or so. In a VC network, a forwarding table in a router is modified whenever a new connection is set up through the router or whenever an existing connection through the router is torn down. This could easily happen at a microsecond timescale in a backbone, tier-1 router.

Because forwarding tables in datagram networks 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. [Paxson 1997] and [Jaiswal 2003] present interesting measurement studies of packet reordering and other phenomena in the public Internet.

4.2.3 Origin of VC and Datagram Networks

The evolution of datagram and VC networks reflects their origins. The notion of a virtual circuit as a central organizing principle has its roots in the telephony world, which uses real circuits. With call setup and per-call state being maintained at the routers within the network, a VC network is arguably more complex than a datagram network (although see [Molinero-Fernandez 2002] for an interesting comparison of the complexity of circuit- versus packet-switched networks). This, too, is in keeping with its telephony heritage. Telephone networks, by necessity, had their complexity within the network, since they were connecting dumb end-system devices such as rotary telephones. (For those too young to know, a rotary phone is an analog telephone with no buttons—only a dial.)

The Internet as a datagram network, on the other hand, grew out of the need to connect computers together. Given more sophisticated end-system devices, the Internet architects chose to make the network-layer service model as simple as possible. As we have already seen in Chapters 2 and 3, additional functionality (for example, in-order delivery, reliable data transfer, congestion control, and DNS name resolution) is then implemented at a higher layer, in the end systems. This inverts the model of the telephone network, with some interesting consequences:

→Since the resulting Internet network-layer service model makes minimal (no!) service guarantees, it imposes minimal requirements on the network layer. This makes it easier to interconnect networks that use very different link-layer technologies (for example, satellite, Ethernet, fiber, or radio) that have 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 some network infrastructure services such as the DNS are implemented in hosts (servers) at the network edge. The ability to add a new service simply by attaching a host to the network and defining a new application-layer protocol (such as HTTP) has allowed new Internet applications such as the Web to be deployed in a remarkably short period of time.

4.3 What's Inside a Router ?

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

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

→ **Input ports.** An input port performs several key functions. It performs the physical layer function of terminating an incoming physical link at a router; this is shown in the leftmost box of the input port and the rightmost box of the output port in Figure 4.6. An input port also performs link-layer functions needed to interoperate with the link layer at the other side of the incoming link; this is represented by the middle boxes in the input and output ports. Perhaps most crucially, the lookup function is also performed at the input port; this will occur in the rightmost box of the input port. It is here that the forwarding table is consulted to determine the router output port to which an arriving packet will be forwarded via the switching fabric. Control packets (for example, packets carrying routing protocol information) are forwarded from an input port to the routing processor. Note that the term port here— referring to the physical input and output router interfaces—is distinctly different from the software ports associated with network applications and sockets discussed in Chapters 2 and 3.

→ **Switching fabric**. The switching fabric connects the router's input ports to its output ports. This switching fabric is completely contained within the router— a network inside of a network router!

→ **Output ports.** An output port stores packets received from the switching fabric and transmits these packets on the outgoing link by performing the necessary link-layer and physical-layer functions. When a link is bidirectional (that is, carries traffic in both directions), an output port will typically be paired with the input port for that link on the same line card (a printed circuit board containing one or more input ports, which is connected to the switching fabric).

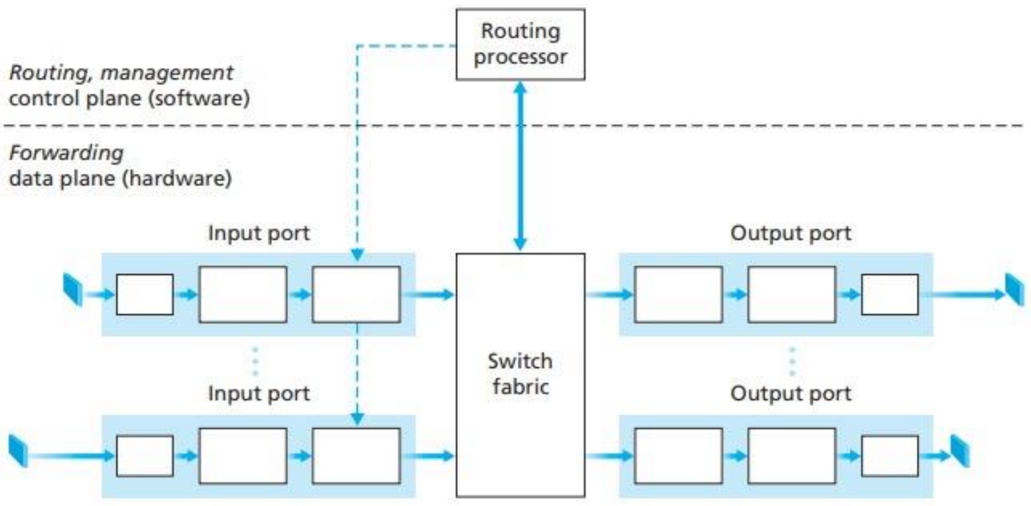


Figure 4.6 ♦ Router architecture

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

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

While the forwarding plane operates at the nanosecond time scale, a router's control functions—executing the routing protocols, responding to attached links that go up or down, and performing management functions such as those we'll study in Chapter 9—operate at the millisecond or second timescale. These router control plane functions are usually implemented in software and execute on the routing processor (typically a traditional CPU).

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

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

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

4.3.1 Input Processing

A more detailed view of input processing is given in Figure 4.7. As discussed above, the input port’s line termination function and link-layer processing implement the physical and link layers for that individual input link. The lookup performed in the input port is central to the router’s operation—it is here that the router uses the forwarding table to look up the output port to which an arriving packet will be forwarded via the switching fabric. The forwarding table is computed and updated by the routing processor, with a shadow copy typically stored at each input port. The forwarding table is copied from the routing processor to the line cards over a separate bus (e.g., a PCI bus) indicated by the dashed line from the routing processor to the input line cards in Figure 4.6. With a shadow copy, forwarding decisions can be made locally, at each input port, without invoking the centralized routing processor on a per-packet basis and thus avoiding a centralized processing bottleneck.

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

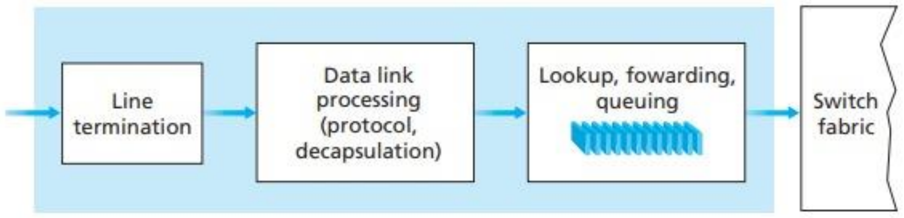

Line termination
Data link processing (protocol, decapsulation)
Lookup, fowarding, queuing
Switch fabric

Figure 4.7 ♦ Input port processing

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

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

4.3.2 Switching

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

→Switching via memory. The simplest, earliest routers were traditional computers, with switching between input and output ports being done under direct control of the CPU (routing processor). Input and output ports functioned as traditional I/O devices in a traditional operating system.

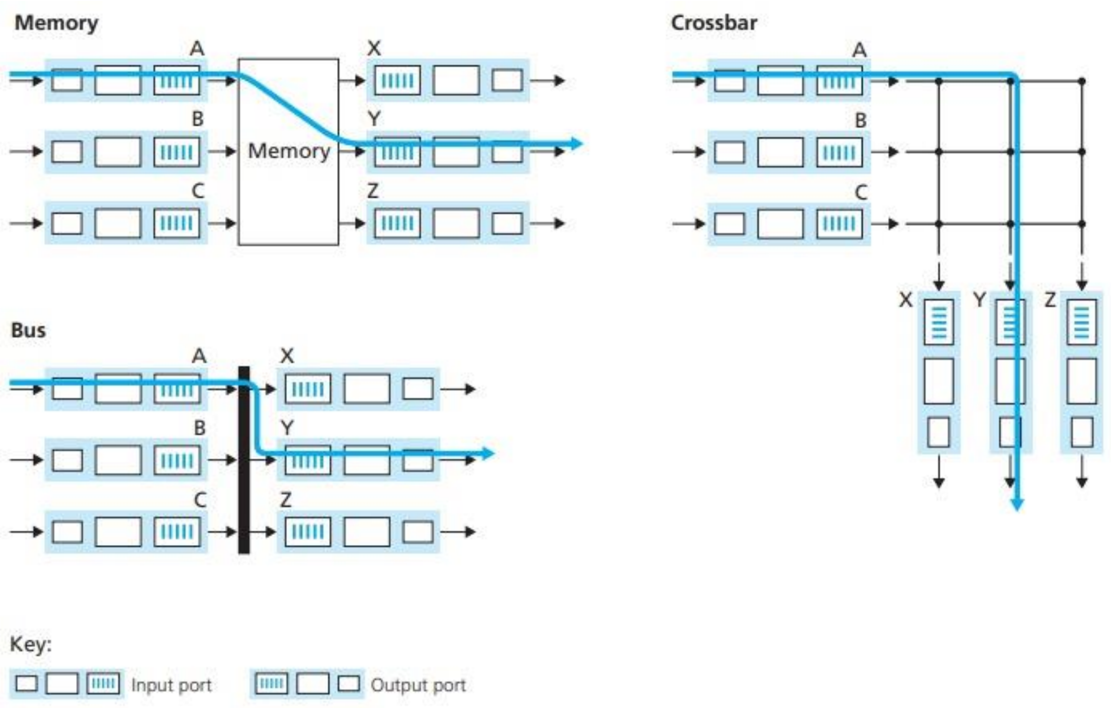


Figure 4.8 ♦ Three switching techniques

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

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

→ **Switching via a bus.** In this approach, an input port transfers a packet directly to the output port over a shared bus, without intervention by the routing processor. This is typically done by having the input port pre-pend a switch-internal label (header) to the packet indicating the local output port to which this packet is being transferred and transmitting the packet onto the bus. The packet is received by all output ports, but only the port that matches the label will keep the packet. The label is then removed at the output port, as this label is only used within the switch to cross the bus. If multiple packets arrive to the router at the same time, each at a different input port, all but one must wait since only one packet can cross the bus at a time. Because every packet must cross the single bus, the switching speed of the router is limited to the bus speed; in our roundabout analogy, this is as if the roundabout could only contain one car at a time. Nonetheless, switching via a bus is often sufficient for routers that operate in small local area and enterprise networks. The Cisco 5600 [Cisco Switches 2012] switches packets over a 32 Gbps backplane bus.

→ **Switching via an interconnection network**. One way to overcome the bandwidth limitation of a single, shared bus is to use a more sophisticated interconnection network, such as those that have been used in the past to interconnect processors in a multiprocessor computer architecture. A crossbar switch is an interconnection network consisting of 2N buses that connect N input ports to N output ports, as shown in Figure 4.8. Each vertical bus intersects each horizontal bus at a crosspoint, which can be opened or closed at any time by the switch fabric controller (whose logic is part of the switching fabric itself). When a packet arrives from port A and needs to be forwarded to port Y, the switch controller closes the crosspoint at the intersection of busses A and Y, and port A then sends the packet onto its bus, which is picked up (only) by bus Y. Note that a packet from port B can be forwarded to port X at the same time, since the A-to-Y and B-to-X packets use different input and output busses. Thus, unlike the previous two switching approaches, crossbar networks are capable of forwarding multiple packets in parallel. However, if two packets from two different input ports are destined to the same output port, then one will have to wait at the input, since only one packet can be sent over any given bus at a time. More sophisticated interconnection networks use multiple stages of switching elements to allow packets from different input ports to proceed towards the same output port at the same time through the switching fabric. See [Tobagi 1990] for a survey of switch architectures. Cisco 12000 family switches [Cisco 12000 2012] use an interconnection network.

Output Processing

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

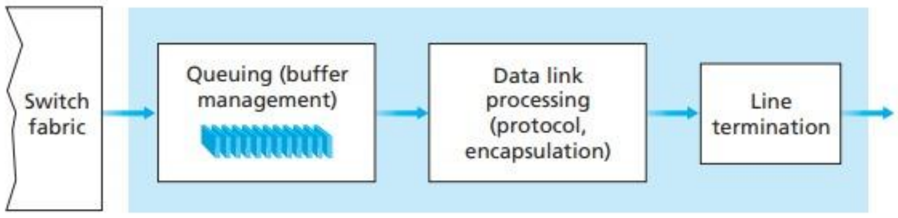

Switch fabric
Queuing (buffer management)
Data link processing (protocol, encapsulation)
Line termination

Figure 4.9 ♦ Output port processing

Where Does Queueing Occur?

If we consider input and output port functionality and the configurations shown in Figure 4.8,

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

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

But what can happen at the output ports? Let's suppose that R_{switch} is still N times faster than R_{line}. Once again, packets arriving at each of the N input ports are destined to the same output port. In this case, in the time it takes to send a single packet onto the outgoing link, N new packets will arrive at this output port. Since the output port can transmit only a single packet in a unit of time (the packet transmission time), the N arriving packets will have to queue (wait) for transmission over the outgoing link. Then N more packets can possibly arrive in the time it takes to transmit just one of the N packets that had just previously been queued. And so on. Eventually, the number of queued packets can grow large enough to exhaust available memory at the output port, in which case packets are dropped.

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

Given that router buffers are needed to absorb the fluctuations in traffic load, the natural question to ask is how much buffering is required. For many years, the rule of thumb [RFC 3439] for buffer sizing was that the amount of buffering (B) should be equal to an average round-trip time (RTT, say 250 msec) times the link capacity (C). This result is based on an analysis of the queueing dynamics of a relatively small number of TCP flows [Villamizar 1994]. Thus, a 10 Gbps link with an RTT of 250 msec would need an amount of buffering equal to B = RTT · C = 2.5 Gbits of buffers. Recent theoretical and experimental efforts [Appenzeller 2004], however, suggest that when there are a large number of TCP flows (N) passing through a link, the amount of buffering needed is B = RTT -C/√N —. With a large number of flows typically passing through large backbone router links (see, e.g., [Fraleigh 2003]), the value of N can be large, with the decrease in needed buffer size becoming quite significant. [Appenzellar 2004; Wischik 2005; Beheshti 2008] provide very readable discussions of the buffer sizing problem from a theoretical, implementation, and operational standpoint.

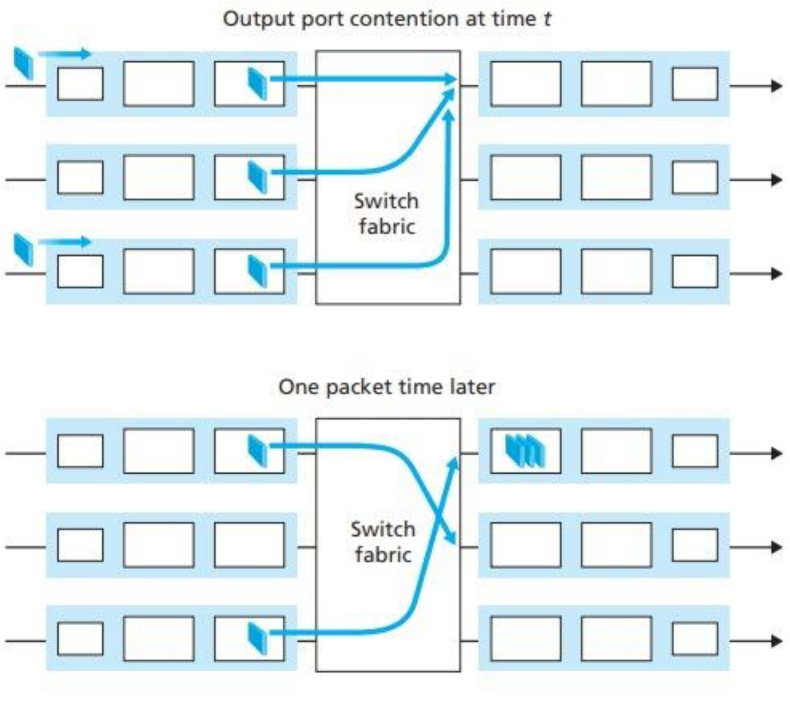


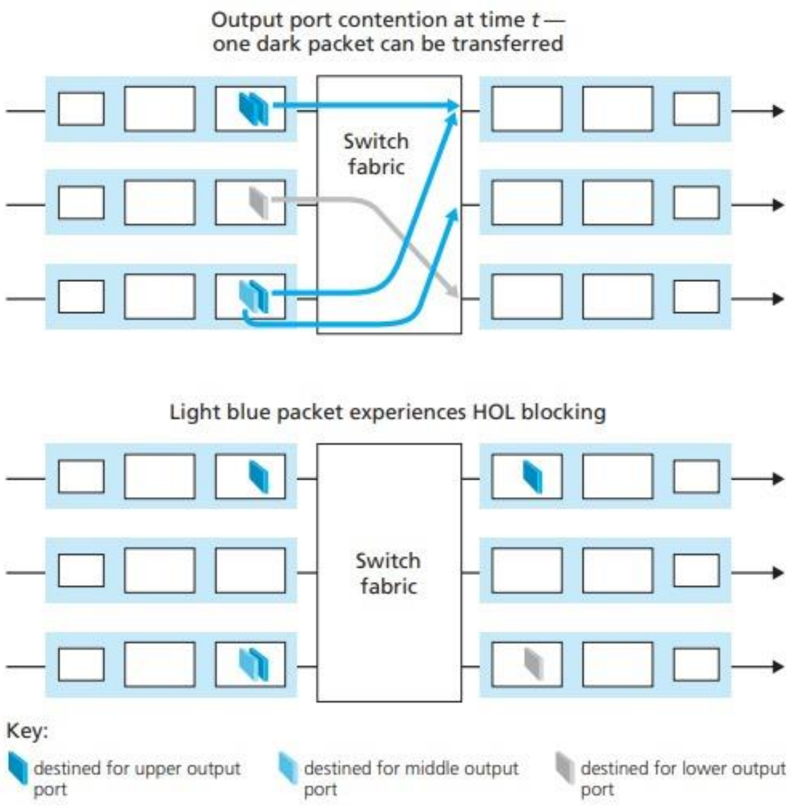
Figure 4.10 ♦ Output port queuing

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'll thus cover packet scheduling extensively in Chapter 7. A discussion of output port packet scheduling disciplines is [Cisco Queue 2012].

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

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

Figure 4.11 shows an example in which two packets (darkly shaded) at the front of their input queues are destined for the same upper-right output port. Suppose that the switch fabric chooses to transfer the packet from the front of the upper-left queue.



Output port contention at time t—
one dark packet can be transferred
Switch fabric
Light blue packet experiences HOL blocking
Switch fabric
Key:
destined for upper output port
destined for middle output port
destined for lower output port

Figure 4.11 ♦ HOL blocking at an input queued switch

In this case, the darkly shaded packet in the lower-left queue must wait. But not only must this darkly shaded packet wait, so too must the lightly shaded packet that is queued behind that packet in the lower-left queue, even though there is no contention for the middle-right output port (the destination for the lightly shaded packet). This phenomenon is known as head-of-the-line (HOL) blocking in an input-queued switch—a queued packet in an input queue must wait for transfer through the fabric (even though its output port is free) because it is blocked by another packet at the head of the line. [Karol 1987] shows that due to HOL blocking, the input queue will grow to unbounded length (informally, this is equivalent to saying that significant packet loss will occur) under certain assumptions as soon as the packet arrival rate on the input links reaches only 58 percent of their capacity. A number of solutions to HOL blocking are discussed in [McKeown 1997b].

4.3.5 The Routing Control Plane

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

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

The Internet Protocol(IP):Forwarding and Addressing in the Internet

Our discussion of network-layer addressing and forwarding thus far has been without reference to any specific computer network. In this section, we'll turn our attention to how addressing and forwarding are done in the Internet. We'll see that Internet addressing and forwarding are important components of the Internet Protocol (IP). There are two versions of IP in use today. We'll first examine the widely deployed IP protocol version 4, which is usually referred to simply as IPv4 [RFC 791]. We'll examine IP version 6 [RFC 2460; RFC 4291], which has been proposed to replace IPv4, at the end of this section.

But before beginning our foray into IP, let's take a step back and consider the components that make up the Internet's network layer. As shown in Figure 4.12, the Internet's network layer has three major components. The first component is the IP protocol, the topic of this section. The second major component is the routing component, which determines the path a datagram follows from source to destination. We mentioned earlier that routing protocols compute the forwarding tables that are used to forward packets through the network. We'll study the Internet's routing protocols in Section 4.6. 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, the Internet Control Message Protocol (ICMP), in Section 4.4.3.

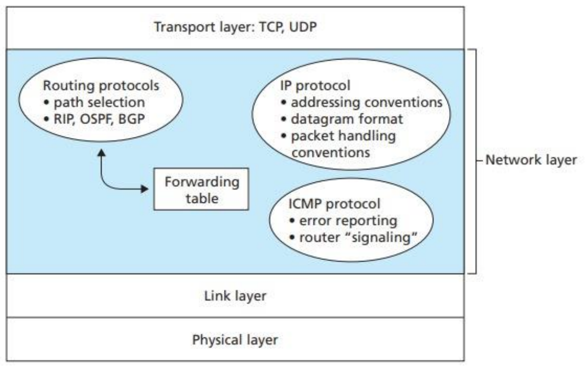


Figure 4.12 ♦ A look inside the Internet's network layer

Datagram Format

Recall that a network-layer packet is referred to as a datagram. We begin our study of IP with an overview of the syntax and semantics of the IPv4 datagram. You might be thinking that nothing could be drier than the syntax and semantics of a packet's bits. Nevertheless, the datagram plays a central role in the Internet—every networking student and professional needs to see it, absorb it, and master it. The IPv4 datagram format is shown in Figure 4.13. The key fields in the IPv4 datagram are the following:

→ Version number. These 4 bits specify the IP protocol version of the datagram. By looking at the version number, the router can 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 4.13. The datagram format for the new version of IP (IPv6) is discussed at the end of this section.

→ Header length. Because an IPv4 datagram can contain a variable number of options (which are included in the IPv4 datagram header), these 4 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

→ Type of service. The type of service (TOS) bits were included in the IPv4 header to allow different types of IP datagrams (for example, datagrams particularly requiring low delay, high throughput, or reliability) to be distinguished from each other. For example, it might be useful to distinguish real-time datagrams (such as those used by an IP telephony application) from non-real-time traffic (for example, FTP). The specific level of service to be provided is a

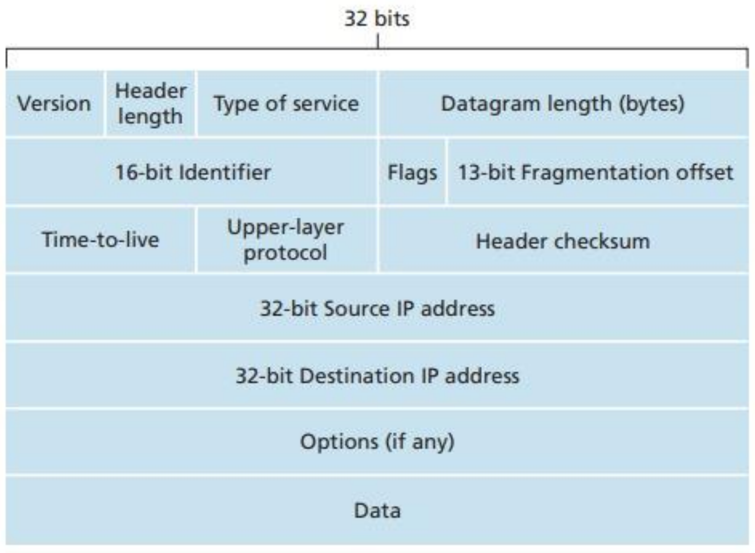


Figure 4.13 ♦ IPv4 datagram format

policy issue determined by the router's administrator. We'll explore the topic of differentiated service in Chapter 7.

- 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 larger than 1,500 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 routing 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 specific transport-layer protocol to which the data portion of this IP datagram should 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 list of all possible values, see [IANA Protocol Numbers 2012]. Note that the protocol number in the IP datagram has a role that is 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'll 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 2 bytes in the header as a number and summing these numbers using 1s complement arithmetic. As discussed in Section 3.3, the 1s complement of this sum, known as the Internet checksum, is stored in the checksum field. A router computes the header checksum for each received IP datagram and detects an error condition if the checksum carried in the datagram header does not equal the computed check sum. Routers typically discard datagrams for which an error has been detected. Note that the check sum must be recomputed and stored again at each router, as the TTL field, and possibly the options field 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 several reasons for this repetition. First, note that only the IP header is check summed at the IP layer, while the TCP/UDP checksum is computed over the entire TCP/UDP segment. 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 addresses. When a source creates a datagram, it inserts its IP address into the source IP address field and inserts the address of the ultimate destination into the destination IP address field. Often the source host determines the destination address via a DNS lookup, as discussed in Chapter 2. We'll discuss IP addressing in detail in Section 4.4.2.
- 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, as discussed in Section 4.4.4.

→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 4.4.3).

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

IPV4 Addressing

We now turn our attention to IPv4 addressing. Although you may be thinking that addressing must be a straightforward topic, hopefully by the end of this chapter you'll be convinced that Internet addressing is not only a juicy, subtle, and interesting topic but also one that is of central importance to the Internet. Excellent treatments of IPv4 addressing are [3Com Addressing 2012] 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 does so over this link. The boundary between the host and the physical link is called an interface. Now consider a router and its **interfaces**. Because a router's job is to receive a datagram on one link and forward the datagram on some other 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 host and router interface to have its own 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, 4 bytes), and there are thus a total of 232 possible IP addresses. By approximating 210 by 103, it is easy to see that there are about 4 billion 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 8 bits of the address; the 32 is the decimal equivalent of the second 8 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 (except for interfaces behind NATs, as discussed at the end of this section). These addresses cannot be chosen in a willy-nilly manner, however. A portion of an interface's IP address will be determined by the subnet to which it is connected.

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, as there are several things to notice. 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 all have the same leftmost 24 bits in their IP address. The four interfaces are also interconnected to each other by a network that contains no routers.

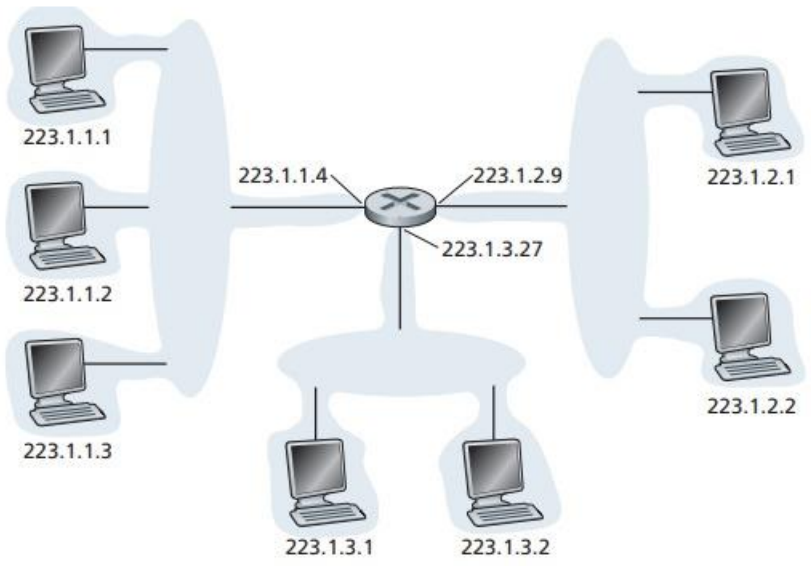


Figure 4.15 ♦ Interface addresses and subnets

This Network could be interconnected by an Ethernet LAN, in which case the interfaces would be interconnected by an Ethernet switch (as we'll discuss in Chapter 5), or by a wireless access point (as we'll discuss in Chapter 6). We'll represent this router less network connecting these hosts as a cloud for now, and dive into the internals of such networks in Chapters 5 and 6.

In IP terms, this network interconnecting three host interfaces and one router interface forms a **subnet** [RFC 950]. (A subnet is also called an IP network or simply a network in the Internet literature.) IP addressing assigns an address to this subnet: 223.1.1.0/24, where the /24 notation, sometimes known as a **subnet mask**, indicates that the leftmost 24 bits of the 32-bit quantity define the subnet address. The subnet 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 subnet would be required to have an address of the form 223.1.1.xxx. There are two additional subnets shown in Figure 4.15: the 223.1.2.0/24 network and the 223.1.3.0/24 subnet. Figure 4.16 illustrates the three IP subnets present in Figure 4.15.

The IP definition of a subnet 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 subnets are present here? Three subnets, 223.1.1.0/24, 223.1.2.0/24, and 223.1.3.0/24, are similar to the subnets we encountered in Figure 4.15. But note that there are three

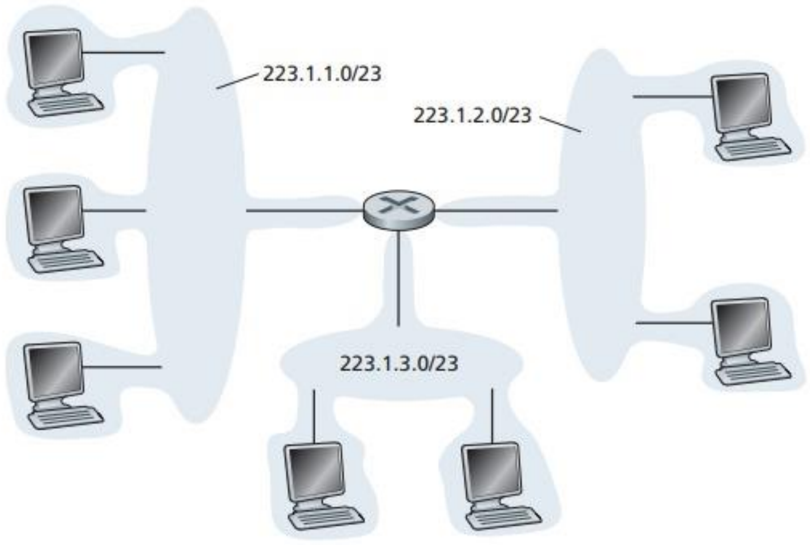


Figure 4.16 ♦ Subnet addresses

additional subnets in this example as well: one subnet, 223.1.9.0/24, for the interfaces that connect routers R1 and R2; another subnet, 223.1.8.0/24, for the interfaces that connect routers R2 and R3; and a third subnet, 223.1.7.0/24, for the interfaces that connect routers R3 and R1. For a general interconnected system of routers and hosts, we can use the following recipe to define the subnets in the system:

To determine the subnets, detach each interface from its host or router, creating islands of isolated networks, with interfaces terminating the end points of the isolated networks. Each of these isolated networks is called a ***subnet****.*

If we apply this procedure to the interconnected system in Figure 4.17, we get six islands or subnets.

From the discussion above, it's clear that an organization (such as a company or academic institution) with multiple Ethernet segments and point-to-point links will have multiple subnets, with all of the devices on a given subnet having the same subnet address. In principle, the different subnets could have quite different subnet addresses. In practice, however, their subnet addresses often have much in common. To understand why, let's next turn our attention to how addressing is handled in the global Internet.

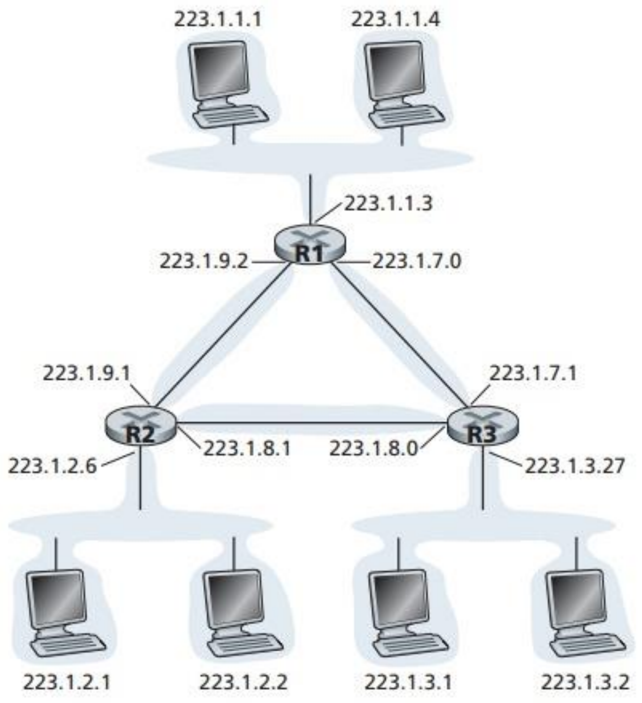


Figure 4.17 ♦ Three routers interconnecting six subnets

PRINCIPLES IN PRACTICE

This example of an ISP that connects eight organizations to the Internet nicely illustrates how

carefully allocated CIDRized addresses facilitate routing. Suppose, as shown in Figure 4.18, 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 its own subnets. This ability to use a single prefix to advertise multiple networks is often referred to as address aggregation (also route aggregation or route summarization).

Address 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 Fly-ByNight-ISP acquires ISPs-R-Us and then has Organization 1 connect to the Internet through its subsidiary ISPs-R-Us? As shown in Figure 4.18, the subsidiary ISPs-R-Us owns the address block 199.31.0.0/16, but Organization 1'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 be reassigned to another subsidiary 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 4.19, 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.

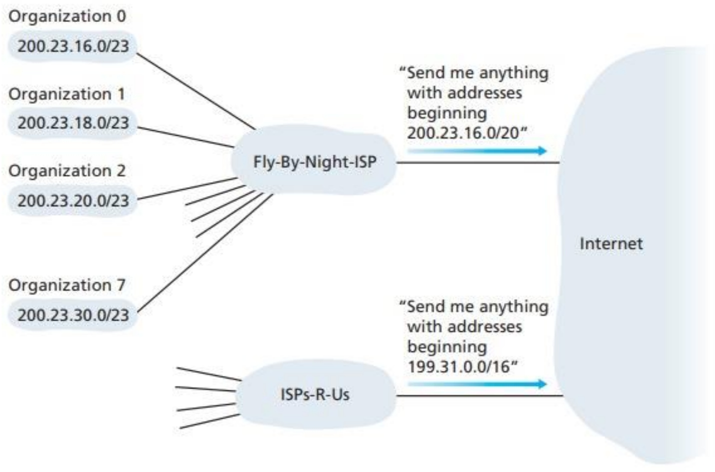


Figure 4.18 ♦ Hierarchical addressing and route aggregation

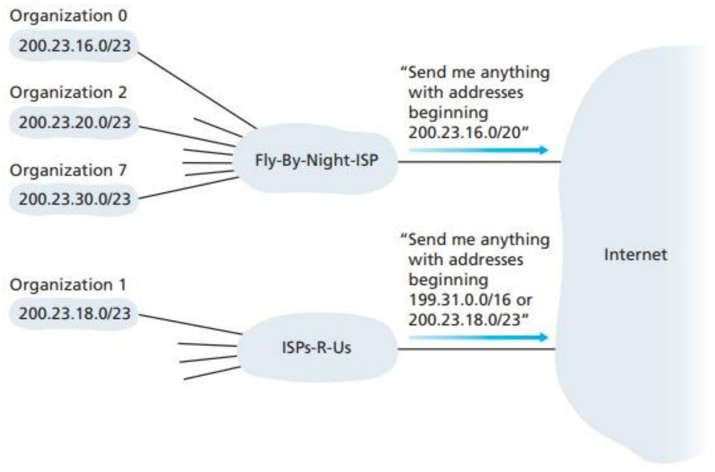


Figure 4.19 ♦ ISPs-R-Us has a more specific route to Organization 1

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 longest prefix matching (see Section 4.2.2), and route toward ISPs-R-Us, as it advertises the longest (most specific) address prefix that matches the destination address.

Routing protocol in Section 4.6, we'll see that only these x leading prefix bits are considered by routers outside the organization's network. That is, when a router outside the organization forwards a datagram whose destination address is inside the organization, only the leading x bits of the address need be considered. This considerably reduces the size of the forwarding table in these routers, since a single entry of the form a.b.c.d/x will be sufficient to forward packets to any destination within the organization.

The remaining 32-x bits of an address can be thought of as distinguishing among the devices within the organization, all of which have the same network prefix. These are the bits that will be considered when forwarding packets at routers within the organization. These lower-order bits may (or may not) have an additional subnetting structure, such as that discussed above. For example, suppose the first 21 bits of the CIDRized address a.b.c.d/21 specify the organization's network prefix and are common to the IP addresses of all devices in that organization. The remaining 11 bits then identify the specific hosts in the organization. The organization's internal structure might be such that these 11 rightmost bits are used for subnetting within the organization, as discussed above. For example, a.b.c.d/24 might refer to a specific subnet within the organization.

Before CIDR was adopted, the network portions of an IP address were constrained to be 8, 16, or 24 bits in length, an addressing scheme known as **classfull addressing**, since subnets with 8-, 16-, and 24-bit subnet addresses were known as class A, B, and C networks, respectively. The requirement that the subnet portion of an IP address be exactly 1, 2, or 3 bytes long turned out to be problematic for supporting the rapidly growing number of organizations with small and medium-sized subnets. A class C (/24) subnet could accommodate only up to 28 – 2 = 254 hosts (two of the 28 = 256 addresses are reserved for special use)—too small for many organizations. However, a class B (/16) subnet, which supports up to 65,634 hosts, was too large. Under classful addressing, an organization with, say, 2,000 hosts was typically allocated a class B (/16) subnet 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 addresses that could not be used by other organizations.

We would be remiss if we did not mention yet another type of IP address, the IP broadcast address 255.255.255.255. When a host sends a datagram with destination address 255.255.255.255, the message is delivered to all hosts on the same subnet. Routers optionally forward the message into neighboring subnets as well (although they usually don't).

Having now studied IP addressing in detail, we need to know how hosts and subnets get their addresses in the first place. Let's begin by looking at how an organization gets a block of addresses for its devices, and then look at how a device (such as a host) is assigned an address from within the organization's block of addresses.

Obtaining a Block of Addresses

In order to obtain a block of IP addresses for use within an organization's subnet, a network administrator might first contact its ISP, which would provide addresses from a larger block of addresses 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-sized contiguous 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 subnet part of these addresses for your convenience.)

ISP's block	200.23.16.0/20	11001000 00010111 00010000 00000000

Organization 0	200.23.16.0/23	11001000	00010111	00010000	00000000
Organization 1	200.23.18.0/23	11001000	00010111	00010010	00000000
Organization 2	200.23.20.0/23	11001000	00010111	00010100	00000000
. . . . .	. . . .				
Organization 7	200.23.30.0/23	11001000	00010111	00011110	00000000

While obtaining a set of addresses from an ISP is one way to get a block of addresses, it is not the only way. Clearly, there must also be a way for the ISP itself to get a block of addresses. Is there a global authority that has ultimate responsibility for managing the IP address space and allocating address blocks to ISPs and other organizations? Indeed there is! IP addresses are managed under the authority of the Internet Corporation for Assigned Names and Numbers (ICANN) [ICANN 2012], based on guidelines set forth in [RFC 2050]. The role of the nonprofit ICANN organization [NTIA 1998] is not only to allocate 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 ICANN allocates addresses to regional Internet registries (for example, ARIN, RIPE, APNIC, and LACNIC, which together form the Address Supporting Organization of ICANN [ASO-ICANN 2012]), and handle the allocation/management of addresses within their regions.

Obtaining a Host Address: the Dynamic Host Configuration Protocol

Once an organization has obtained a block of addresses, it can assign individual IP addresses to the host and router interfaces in its organization. A system administrator will typically manually configure the IP addresses into the router (often remotely, with a network management tool). Host addresses can also be configured manually, but more often this task is now done using the **Dynamic Host Configuration Protocol** (DHCP) [RFC 2131]. DHCP allows a host to obtain (be allocated) an IP address automatically. A network administrator can configure DHCP so that a given host receives the same IP address each time it connects to the network, or a host may be assigned a **temporary IP address** that will be different each time the host connects to the network. In addition to host IP address assignment, DHCP also allows a host to learn additional information, such as its subnet mask, the address of its first-hop router (often called the default gateway), and the address of its local DNS server.

Because of DHCP's ability to automate the network-related aspects of connecting a host into a network, it is often referred to as a **plug-and-play protocol.** This capability makes it very attractive to the network administrator who would otherwise have to perform these tasks manually! DHCP is also enjoying widespread use in residential Internet access networks and in wireless LANs, where hosts join and leave the network frequently. Consider, for example, the student who carries a laptop from a dormitory room to a library to a classroom. It is likely that in each location, the student will be connecting into a new subnet and hence will need a new IP address at each location. DHCP is ideally suited to this situation, as there are many users coming and going, and addresses are needed for only a limited amount of time. DHCP is similarly useful in residential ISP access networks.

Consider, for example, a residential ISP that has 2,000 customers, but no more than 400 customers are ever online at the same time. In this case, rather than needing a block of 2,048 addresses, a DHCP server

that assigns addresses dynamically needs only a block of 512 addresses (for example, a block of the form a.b.c.d/23). As the hosts join and leave, the DHCP server needs to update its list of available IP addresses. Each time a host joins, the DHCP server allocates an arbitrary address from its current pool of available addresses; each time a host leaves, its address is returned to the pool.

DHCP is a client-server protocol. A client is typically a newly arriving host wanting to obtain network configuration information, including an IP address for itself. In the simplest case, each subnet (in the addressing sense of Figure 4.17) will have a DHCP server. If no server is present on the subnet, a DHCP relay agent (typically a router) that knows the address of a DHCP server for that network is needed. Figure 4.20 shows a DHCP server attached to subnet 223.1.2/24, with the router serving as the relay agent for arriving clients attached to subnets 223.1.1/24 and 223.1.3/24. In our discussion below, we'll assume that a DHCP server is available on the subnet. For a newly arriving host, the DHCP protocol is a four-step process, as shown in Figure 4.21 for the network setting shown in Figure 4.20. In this figure, yiaddr (as in "your Internet address") indicates the address being allocated to the newly arriving client. The four steps are:

→ DHCP server discovery. The first task of a newly arriving host is to find a DHCP server with which to interact. This is done using a DHCP discover message, which a client sends within a UDP packet to port 67. The UDP packet is encapsulated in an IP datagram. But to whom should this datagram be sent? The host doesn't even know the IP address of the network to which it is attaching, much

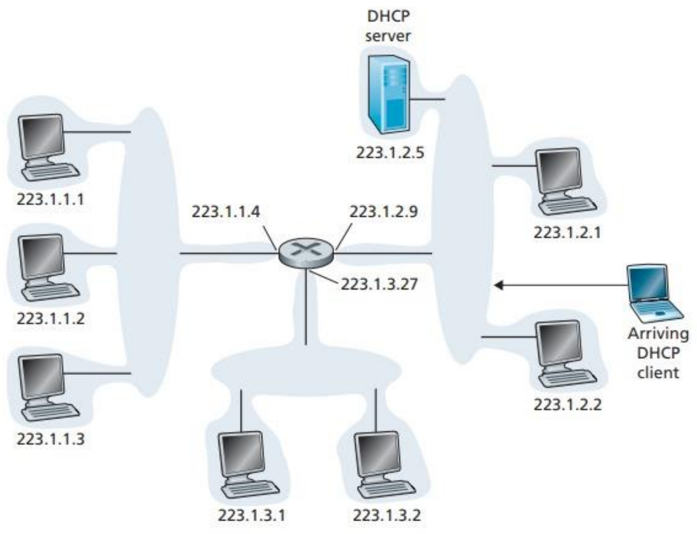


Figure 4.20 ♦ DHCP client-server scenario

less the address of a DHCP server for this network. Given this, the DHCP client creates an IP datagram containing its DHCP discover message along with the broadcast destination IP address of 255.255.255.255 and a “this host” source IP address of 0.0.0.0. The DHCP client passes the IP datagram to the link layer, which then broadcasts this frame to all nodes attached to the subnet (we will cover the details of link-layer broadcasting in Section 5.4).

→DHCP server offer(s). A DHCP server receiving a DHCP discover message responds to the client with a DHCP offer message that is broadcast to all nodes on the subnet, again using the IP broadcast address of 255.255.255.255. (You might want to think about why this server reply must also be broadcast). Since several DHCP servers can be present on the subnet, the client may find itself in the enviable position of being able to choose from among several offers. Each server offer message contains the transaction ID of the received discover message, the proposed IP address for the client, the network mask, and an IP address lease time—the amount of time for which the IP address will be valid. It is common for the server to set the lease time to several hours or days [Droms 2002].

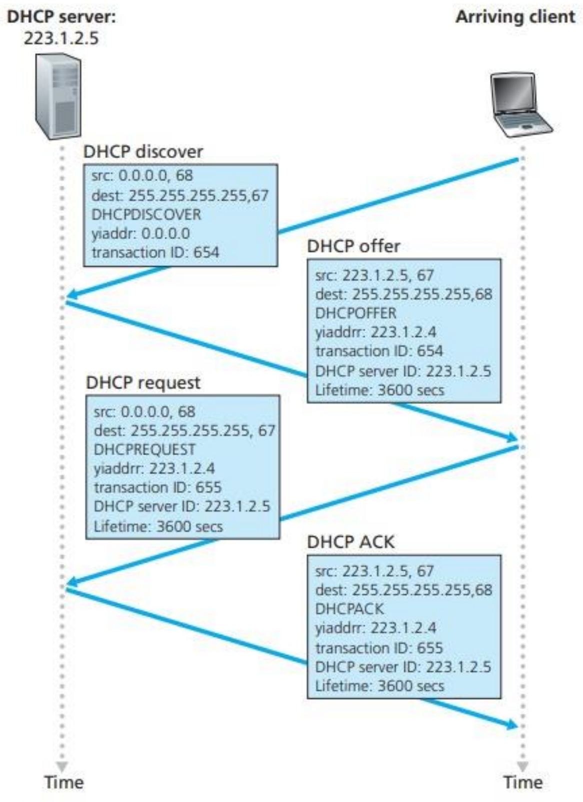


Figure 4.21 ♦ DHCP client-server interaction

→DHCP request. The newly arriving client will choose from among one or more server offers and respond to its selected offer with a DHCP request message, echoing back the configuration parameters.

→DHCP ACK. The server responds to the DHCP request message with a DHCP ACK message, confirming the requested parameters.

Once the client receives the DHCP ACK, the interaction is complete and the client can use the DHCP-allocated IP address for the lease duration. Since a client may want to use its address beyond the lease's expiration, DHCP also provides a mechanism that allows a client to renew its lease on an IP address.

→The value of DHCP's plug-and-play capability is clear, considering the fact that the alternative is to manually configure a host's IP address. Consider the student who moves from classroom to library to dorm room with a laptop, joins a new subnet, and thus obtains a new IP address at each location. It is unimaginable that a system administrator would have to reconfigure laptops at each location, and few students (except those taking a computer networking class!) would have the expertise to configure their laptops manually. From a mobility aspect, however, DHCP does have shortcomings. Since a new IP address is obtained from DHCP each time a node connects to a new subnet, a TCP connection to a remote application cannot be maintained as a mobile node moves between subnets. In Chapter 6, we will examine mobile IP—a recent extension to the IP infrastructure that allows a mobile node to use a single permanent address as it moves between subnets. Additional details about DHCP can be found in [Droms 2002] and [dhc 2012]. An open source reference implementation of DHCP is available from the Internet Systems Consortium [ISC 2012].

Internet Control Message Protocol(ICMP)

Recall that the network layer of the Internet has three main components: the IP protocol, discussed in the previous section; the Internet routing protocols (including RIP, OSPF, and BGP), which are covered in Section 4.6; and ICMP, which is the subject of this section.

ICMP, specified in [RFC 792], is used by hosts and routers to communicate network-layer information to each other. 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.

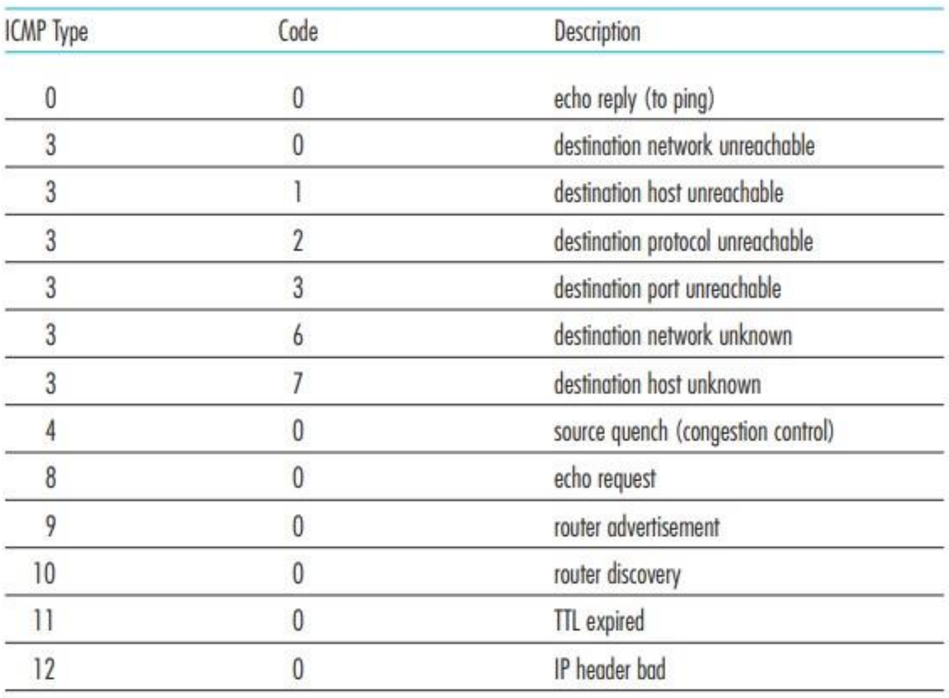
ICMP is often considered part of IP but architecturally it lies just above IP, as ICMP messages are carried inside IP datagrams. That is, ICMP messages are carried as IP payload, just as TCP or UDP segments are carried as IP payload. Similarly, when a host receives an IP datagram with ICMP specified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

ICMP messages have a type and a code field, and contain the header and the first 8 bytes of the IP datagram that caused the ICMP message to be generated in the first place (so that the sender can determine the datagram that caused the error). Selected ICMP message types are shown in Figure 4.23. 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. Most TCP/IP implementations support the ping server directly in the operating system; that is, the server is not a process. Chapter 11 of [Stevens 1990] provides the source code for the ping client program. Note that the client program needs to be able to instruct the operating system to generate an ICMP message of type 8 code 0.

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.

In Chapter 1 we introduced the Traceroute program, which allows us to trace a route from a host to any other host in the world. Interestingly, Traceroute is implemented with 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. Each of these datagrams carries a UDP segment with an unlikely UDP port number. The first of these datagrams has a TTL of 1, the second of 2, the third of 3, and so on.



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 4.23 ♦ ICMP message types

The source also starts timers for each of the datagrams. When the nth datagram arrives at the nth router, the nth 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 back at the source, the source obtains the round-trip time from the timer and the name and IP address of the nth router from the ICMP message.

How does a Traceroute source know when to stop sending UDP segments? Recall that the source increments the TTL field for each datagram it sends. Thus, one of the datagrams will

eventually make it all the way to the destination host. Because this datagram contains a UDP segment with an unlikely port number, the destination host sends a port unreachable ICMP message (type 3 code 3) back to the source. When the source host receives this particular ICMP message, it knows it does not need to send additional probe packets. (The standard Traceroute program actually sends sets of three packets with the same TTL; thus the Traceroute output provides three results for each TTL.

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 subnets 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 be completely allocated (and hence no new networks could attach to the Internet) was the subject of considerable debate. 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 February 2011, IANA allocated out the last remaining pool of unassigned IPv4 addresses to a regional registry. While these registries still have available IPv4 addresses within their pool, once these addresses are exhausted, there are no more available address blocks that can be allocated from a central pool [Huston 2011a]. Although the mid-1990s estimates of IPv4 address depletion suggested that a considerable amount of time might be left until the IPv4 address space was 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. The result of this effort was the specification of IP version 6 (IPv6) [RFC 2460] which we'll discuss below. (An often-asked question is what happened to IPv5? It was initially envisioned that the ST-2 protocol would become IPv5, but ST-2 was later dropped.) Excellent sources of information about IPv6 are [Huitema 1998, IPv6 2012].

IPv6 Datagram Format

The format of the IPv6 datagram is shown in Figure 4.24. The most important changes introduced in IPv6 are evident in the datagram 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, IPv6 has introduced a new type of address, called an anycast address, which allows a datagram 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.

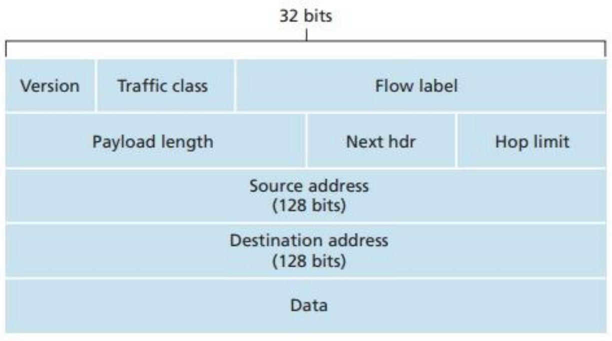


Figure 4.24 ♦ IPv6 datagram format

→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 nondefault 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 an 8-bit traffic class field. This field, like the TOS field in IPv4, can be used to give priority to certain datagrams within a flow, or it can be used to give priority to datagrams from certain applications (for example, ICMP) over datagrams from other applications (for example, network news).

As noted above, a comparison of Figure 4.24 with Figure 4.13 reveals the simpler, more streamlined structure of the IPv6 datagram. The following fields are defined in IPv6:

- Version. This 4-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 8-bit field is similar in spirit to the TOS field we saw in IPv4.
- 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 datagram 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 addresses. The various formats of the IPv6 128-bit address are described in RFC 4291.
- 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 4.24 with the IPv4 datagram format that we saw in Figure 4.13, 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.
- Header checksum. Because the transport-layer (for example, TCP and UDP) and link-layer (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

Recall from our discussion in Section 4.4.3 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 4443. 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’ll study in Section 4.7. IGMP, which is used to manage a host’s joining and leaving of multicast groups, was previously a separate protocol from ICMP in IPv4.