Chapter 5. The Network Layer

The network layer is concerned with getting packets from the source all the way to the destination. Getting to the destination may require making many hops at intermediate routers along the way. This function clearly contrasts with that of the data link layer, which has the more modest goal of just moving frames from one end of a wire to the other. Thus, the network layer is the lowest layer that deals with end-to-end transmission.

To achieve its goals, the network layer must know about the topology of the communication subnet (i.e., the set of all routers) and choose appropriate paths through it. It must also take care to choose routes to avoid overloading some of the communication lines and routers while leaving others idle. Finally, when the source and destination are in different networks, new problems occur. It is up to the network layer to deal with them. In this chapter we will study all these issues and illustrate them, primarily using the Internet and its network layer protocol, IP, although wireless networks will also be addressed.

5.1 Network Layer Design Issues

In the following sections we will provide an introduction to some of the issues that the designers of the network layer must grapple with. These issues include the service provided to the transport layer and the internal design of the subnet.

5.1.1 Store-and-Forward Packet Switching

But before starting to explain the details of the network layer, it is probably worth restating the context in which the network layer protocols operate. This context can be seen in Fig. 5-1. The major components of the system are the carrier's equipment (routers connected by transmission lines), shown inside the shaded oval, and the customers' equipment, shown outside the oval. Host H1 is directly connected to one of the carrier's routers, A, by a leased line. In contrast, H2 is on a LAN with a router, F, owned and operated by the customer. This router also has a leased line to the carrier's equipment. We have shown F as being outside the oval because it does not belong to the carrier, but in terms of construction, software, and protocols, it is probably no different from the carrier's routers. Whether it belongs to the subnet is arguable, but for the purposes of this chapter, routers on customer premises are considered part of the subnet because they run the same algorithms as the carrier's routers (and our main concern here is algorithms).

Figure 5-1. The environment of the network layer protocols.
This equipment is used as follows. A host with a packet to send transmits it to the nearest router, either on its own LAN or over a point-to-point link to the carrier. The packet is stored there until it has fully arrived so the checksum can be verified. Then it is forwarded to the next router along the path until it reaches the destination host, where it is delivered. This mechanism is store-and-forward packet switching, as we have seen in previous chapters.

5.1.2 Services Provided to the Transport Layer

The network layer provides services to the transport layer at the network layer/transport layer interface. An important question is what kind of services the network layer provides to the transport layer. The network layer services have been designed with the following goals in mind.

1. The services should be independent of the router technology.

2. The transport layer should be shielded from the number, type, and topology of the routers present.

3. The network addresses made available to the transport layer should use a uniform numbering plan, even across LANs and WANs.

Given these goals, the designers of the network layer have a lot of freedom in writing detailed specifications of the services to be offered to the transport layer. This freedom often degenerates into a raging battle between two warring factions. The discussion centers on whether the network layer should provide connection-oriented service or connectionless service.

One camp (represented by the Internet community) argues that the routers' job is moving packets around and nothing else. In their view (based on 30 years of actual experience with a real, working computer network), the subnet is inherently unreliable, no matter how it is designed. Therefore, the hosts should accept the fact that the network is unreliable and do error control (i.e., error detection and correction) and flow control themselves.

This viewpoint leads quickly to the conclusion that the network service should be connectionless, with primitives SEND PACKET and RECEIVE PACKET and little else. In particular, no packet ordering and flow control should be done, because the hosts are going to do that anyway, and there is usually little to be gained by doing it twice. Furthermore, each packet must carry the full destination address, because each packet sent is carried independently of its predecessors, if any.

The other camp (represented by the telephone companies) argues that the subnet should provide a reliable, connection-oriented service. They claim that 100 years of successful experience with the worldwide telephone system is an excellent guide. In this view, quality of service is the dominant factor, and without connections in the subnet, quality of service is very difficult to achieve, especially for real-time traffic such as voice and video.
These two camps are best exemplified by the Internet and ATM. The Internet offers connectionless network-layer service; ATM networks offer connection-oriented network-layer service. However, it is interesting to note that as quality-of-service guarantees are becoming more and more important, the Internet is evolving. In particular, it is starting to acquire properties normally associated with connection-oriented service, as we will see later. Actually, we got an inkling of this evolution during our study of VLANs in Chap. 4.

### 5.1.3 Implementation of Connectionless Service

Having looked at the two classes of service the network layer can provide to its users, it is time to see how this layer works inside. Two different organizations are possible, depending on the type of service offered. If connectionless service is offered, packets are injected into the subnet individually and routed independently of each other. No advance setup is needed. In this context, the packets are frequently called **datagrams** (in analogy with telegrams) and the subnet is called a **datagram subnet**. If connection-oriented service is used, a path from the source router to the destination router must be established before any data packets can be sent. This connection is called a **VC (virtual circuit)**, in analogy with the physical circuits set up by the telephone system, and the subnet is called a **virtual-circuit subnet**. In this section we will examine datagram subnets; in the next one we will examine virtual-circuit subnets.

Let us now see how a datagram subnet works. Suppose that the process $P1$ in Fig. 5-2 has a long message for $P2$. It hands the message to the transport layer with instructions to deliver it to process $P2$ on host $H2$. The transport layer code runs on $H1$, typically within the operating system. It prepends a transport header to the front of the message and hands the result to the network layer, probably just another procedure within the operating system.

![Figure 5-2. Routing within a datagram subnet.](image)

Let us assume that the message is four times longer than the maximum packet size, so the network layer has to break it into four packets, 1, 2, 3, and 4 and sends each of them in turn to router $A$ using some point-to-point protocol, for example, PPP. At this point the carrier takes over. Every router has an internal table telling it where to send packets for each possible destination.
Each table entry is a pair consisting of a destination and the outgoing line to use for that destination. Only directly-connected lines can be used. For example, in Fig. 5-2, A has only two outgoing lines—to B and C—so every incoming packet must be sent to one of these routers, even if the ultimate destination is some other router. A’s initial routing table is shown in the figure under the label "initially."

As they arrived at A, packets 1, 2, and 3 were stored briefly (to verify their checksums). Then each was forwarded to C according to A’s table. Packet 1 was then forwarded to E and then to F. When it got to F, it was encapsulated in a data link layer frame and sent to H2 over the LAN. Packets 2 and 3 follow the same route.

However, something different happened to packet 4. When it got to A it was sent to router B, even though it is also destined for F. For some reason, A decided to send packet 4 via a different route than that of the first three. Perhaps it learned of a traffic jam somewhere along the ACE path and updated its routing table, as shown under the label "later." The algorithm that manages the tables and makes the routing decisions is called the routing algorithm. Routing algorithms are one of the main things we will study in this chapter.

5.1.4 Implementation of Connection-Oriented Service

For connection-oriented service, we need a virtual-circuit subnet. Let us see how that works. The idea behind virtual circuits is to avoid having to choose a new route for every packet sent, as in Fig. 5-2. Instead, when a connection is established, a route from the source machine to the destination machine is chosen as part of the connection setup and stored in tables inside the routers. That route is used for all traffic flowing over the connection, exactly the same way that the telephone system works. When the connection is released, the virtual circuit is also terminated. With connection-oriented service, each packet carries an identifier telling which virtual circuit it belongs to.

As an example, consider the situation of Fig. 5-3. Here, host H1 has established connection 1 with host H2. It is remembered as the first entry in each of the routing tables. The first line of A’s table says that if a packet bearing connection identifier 1 comes in from H1, it is to be sent to router C and given connection identifier 1. Similarly, the first entry at C routes the packet to E, also with connection identifier 1.

Figure 5-3. Routing within a virtual-circuit subnet.
Now let us consider what happens if \( H3 \) also wants to establish a connection to \( H2 \). It chooses connection identifier 1 (because it is initiating the connection and this is its only connection) and tells the subnet to establish the virtual circuit. This leads to the second row in the tables. Note that we have a conflict here because although \( A \) can easily distinguish connection 1 packets from \( H1 \) from connection 1 packets from \( H3 \), \( C \) cannot do this. For this reason, \( A \) assigns a different connection identifier to the outgoing traffic for the second connection. Avoiding conflicts of this kind is why routers need the ability to replace connection identifiers in outgoing packets. In some contexts, this is called label switching.

### 5.1.5 Comparison of Virtual-Circuit and Datagram Subnets

Both virtual circuits and datagrams have their supporters and their detractors. We will now attempt to summarize the arguments both ways. The major issues are listed in Fig. 5-4, although purists could probably find a counterexample for everything in the figure.

#### Figure 5-4. Comparison of datagram and virtual-circuit subnets.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Datagram subnet</th>
<th>Virtual-circuit subnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit setup</td>
<td>Not needed</td>
<td>Required</td>
</tr>
<tr>
<td>Addressing</td>
<td>Each packet contains the full source and destination address</td>
<td>Each packet contains a short VC number</td>
</tr>
<tr>
<td>State information</td>
<td>Routers do not hold state information about connections</td>
<td>Each VC requires router table space per connection</td>
</tr>
<tr>
<td>Routing</td>
<td>Each packet is routed independently</td>
<td>Route chosen when VC is set up; all packets follow it</td>
</tr>
<tr>
<td>Effect of router failures</td>
<td>None, except for packets lost during the crash</td>
<td>All VCs that passed through the failed router are terminated</td>
</tr>
<tr>
<td>Quality of service</td>
<td>Difficult</td>
<td>Easy if enough resources can be allocated in advance for each VC</td>
</tr>
<tr>
<td>Congestion control</td>
<td>Difficult</td>
<td>Easy if enough resources can be allocated in advance for each VC</td>
</tr>
</tbody>
</table>

Inside the subnet, several trade-offs exist between virtual circuits and datagrams. One trade-off is between router memory space and bandwidth. Virtual circuits allow packets to contain circuit numbers instead of full destination addresses. If the packets tend to be fairly short, a full destination address in every packet may represent a significant amount of overhead and hence, wasted bandwidth. The price paid for using virtual circuits internally is the table space within the routers. Depending upon the relative cost of communication circuits versus router memory, one or the other may be cheaper.

Another trade-off is setup time versus address parsing time. Using virtual circuits requires a setup phase, which takes time and consumes resources. However, figuring out what to do with a data packet in a virtual-circuit subnet is easy: the router just uses the circuit number to index into a table to find out where the packet goes. In a datagram subnet, a more complicated lookup procedure is required to locate the entry for the destination.

Yet another issue is the amount of table space required in router memory. A datagram subnet needs to have an entry for every possible destination, whereas a virtual-circuit subnet just needs
an entry for each virtual circuit. However, this advantage is somewhat illusory since connection setup packets have to be routed too, and they use destination addresses, the same as datagrams do.

Virtual circuits have some advantages in guaranteeing quality of service and avoiding congestion within the subnet because resources (e.g., buffers, bandwidth, and CPU cycles) can be reserved in advance, when the connection is established. Once the packets start arriving, the necessary bandwidth and router capacity will be there. With a datagram subnet, congestion avoidance is more difficult.

For transaction processing systems (e.g., stores calling up to verify credit card purchases), the overhead required to set up and clear a virtual circuit may easily dwarf the use of the circuit. If the majority of the traffic is expected to be of this kind, the use of virtual circuits inside the subnet makes little sense. On the other hand, permanent virtual circuits, which are set up manually and last for months or years, may be useful here.

Virtual circuits also have a vulnerability problem. If a router crashes and loses its memory, even if it comes back up a second later, all the virtual circuits passing through it will have to be aborted. In contrast, if a datagram router goes down, only those users whose packets were queued in the router at the time will suffer, and maybe not even all those, depending upon whether they have already been acknowledged. The loss of a communication line is fatal to virtual circuits using it but can be easily compensated for if datagrams are used. Datagrams also allow the routers to balance the traffic throughout the subnet, since routes can be changed partway through a long sequence of packet transmissions.

5.2 Routing Algorithms

The main function of the network layer is routing packets from the source machine to the destination machine. In most subnets, packets will require multiple hops to make the journey. The only notable exception is for broadcast networks, but even here routing is an issue if the source and destination are not on the same network. The algorithms that choose the routes and the data structures that they use are a major area of network layer design.

The routing algorithm is that part of the network layer software responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagrams internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits internally, routing decisions are made only when a new virtual circuit is being set up. Thereafter, data packets just follow the previously-established route. The latter case is sometimes called session routing because a route remains in force for an entire user session (e.g., a login session at a terminal or a file transfer).

It is sometimes useful to make a distinction between routing, which is making the decision which routes to use, and forwarding, which is what happens when a packet arrives. One can think of a router as having two processes inside it. One of them handles each packet as it arrives, looking up the outgoing line to use for it in the routing tables. This process is forwarding. The other process is responsible for filling in and updating the routing tables. That is where the routing algorithm comes into play.

Regardless of whether routes are chosen independently for each packet or only when new connections are established, certain properties are desirable in a routing algorithm: correctness, simplicity, robustness, stability, fairness, and optimality. Correctness and simplicity hardly require comment, but the need for robustness may be less obvious at first. Once a major network comes
on the air, it may be expected to run continuously for years without systemwide failures. During that period there will be hardware and software failures of all kinds. Hosts, routers, and lines will fail repeatedly, and the topology will change many times. The routing algorithm should be able to cope with changes in the topology and traffic without requiring all jobs in all hosts to be aborted and the network to be rebooted every time some router crashes.

Robustness is also an important goal for the routing algorithm. There exist routing algorithms that never converge to equilibrium, no matter how long they run. A stable algorithm reaches equilibrium and stays there. Fairness and optimality may sound obvious—surely no reasonable person would oppose them—but as it turns out, they are often contradictory goals. As a simple example of this conflict, look at Fig. 5.5. Suppose that there is enough traffic between A and A’, between B and B’, and between C and C’ to saturate the horizontal links. To maximize the total flow, the X to X’ traffic should be shut off altogether. Unfortunately, X and X’ may not see it that way. Evidently, some compromise between global efficiency and fairness to individual connections is needed.

Figure 5-5. Conflict between fairness and optimality.

Before we can even attempt to find trade-offs between fairness and optimality, we must decide what it is we seek to optimize. Minimizing mean packet delay is an obvious candidate, but so is maximizing total network throughput. Furthermore, these two goals are also in conflict, since operating any queueing system near capacity implies a long queueing delay. As a compromise, many networks attempt to minimize the number of hops a packet must make, because reducing the number of hops tends to improve the delay and also reduce the amount of bandwidth consumed, which tends to improve the throughput as well.

Routing algorithms can be grouped into two major classes: nonadaptive and adaptive. Nonadaptive algorithms do not base their routing decisions on measurements or estimates of the current traffic and topology. Instead, the choice of the route to use to get from I to J (for all I and J) is computed in advance, off-line, and downloaded to the routers when the network is booted. This procedure is sometimes called static routing.

Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get their information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every ∆T sec, when the load changes or when the topology changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time). In the following sections we will discuss a variety of routing algorithms, both static and dynamic.

5.2.1 The Optimality Principle

Before we get into specific algorithms, it may be helpful to note that one can make a general statement about optimal routes without regard to network topology or traffic. This statement is
known as the **optimality principle**. It states that if router $J$ is on the optimal path from router $I$ to router $K$, then the optimal path from $J$ to $K$ also falls along the same route. To see this, call the part of the route from $I$ to $Jr_1$ and the rest of the route $r_2$. If a route better than $r_2$ existed from $J$ to $K$, it could be concatenated with $r_1$ to improve the route from $I$ to $K$, contradicting our statement that $r_1r_2$ is optimal.

As a direct consequence of the optimality principle, we can see that the set of optimal routes from all sources to a given destination form a tree rooted at the destination. Such a tree is called a **sink tree** and is illustrated in **Fig. 5-6**, where the distance metric is the number of hops. Note that a sink tree is not necessarily unique; other trees with the same path lengths may exist. The goal of all routing algorithms is to discover and use the sink trees for all routers.

**Figure 5-6. (a) A subnet. (b) A sink tree for router $B$.**

Since a sink tree is indeed a tree, it does not contain any loops, so each packet will be delivered within a finite and bounded number of hops. In practice, life is not quite this easy. Links and routers can go down and come back up during operation, so different routers may have different ideas about the current topology. Also, we have quietly finessed the issue of whether each router has to individually acquire the information on which to base its sink tree computation or whether this information is collected by some other means. We will come back to these issues shortly. Nevertheless, the optimality principle and the sink tree provide a benchmark against which other routing algorithms can be measured.

### 5.2.2 Shortest Path Routing

Let us begin our study of feasible routing algorithms with a technique that is widely used in many forms because it is simple and easy to understand. The idea is to build a graph of the subnet, with each node of the graph representing a router and each arc of the graph representing a communication line (often called a link). To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.

The concept of a **shortest path** deserves some explanation. One way of measuring path length is the number of hops. Using this metric, the paths $ABC$ and $ABE$ in **Fig. 5-7** are equally long. Another metric is the geographic distance in kilometers, in which case $ABC$ is clearly much longer than $ABE$ (assuming the figure is drawn to scale).

**Figure 5-7. The first five steps used in computing the shortest path from $A$ to $D$. The arrows indicate the working node.**
However, many other metrics besides hops and physical distance are also possible. For example, each arc could be labeled with the mean queueing and transmission delay for some standard test packet as determined by hourly test runs. With this graph labeling, the shortest path is the fastest path rather than the path with the fewest arcs or kilometers.

In the general case, the labels on the arcs could be computed as a function of the distance, bandwidth, average traffic, communication cost, mean queue length, measured delay, and other factors. By changing the weighting function, the algorithm would then compute the "shortest" path measured according to any one of a number of criteria or to a combination of criteria.

Several algorithms for computing the shortest path between two nodes of a graph are known. This one is due to Dijkstra (1959). Each node is labeled (in parentheses) with its distance from the source node along the best known path. Initially, no paths are known, so all nodes are labeled with infinity. As the algorithm proceeds and paths are found, the labels may change, reflecting better paths. A label may be either tentative or permanent. Initially, all labels are tentative. When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent and never changed thereafter.

To illustrate how the labeling algorithm works, look at the weighted, undirected graph of Fig. 5-7(a), where the weights represent, for example, distance. We want to find the shortest path from A to D. We start out by marking node A as permanent, indicated by a filled-in circle. Then we examine, in turn, each of the nodes adjacent to A (the working node), relabeling each one with the distance to A. Whenever a node is relabeled, we also label it with the node from which the probe was made so that we can reconstruct the final path later. Having examined each of the nodes adjacent to A, we examine all the tentatively labeled nodes in the whole graph and make the one with the smallest label permanent, as shown in Fig. 5-7(b). This one becomes the new working node.

We now start at B and examine all nodes adjacent to it. If the sum of the label on B and the distance from B to the node being considered is less than the label on that node, we have a shorter path, so the node is relabeled.
After all the nodes adjacent to the working node have been inspected and the tentative labels changed if possible, the entire graph is searched for the tentatively-labeled node with the smallest value. This node is made permanent and becomes the working node for the next round. Figure 5-7 shows the first five steps of the algorithm.

To see why the algorithm works, look at Fig. 5-7(c). At that point we have just made $E$ permanent. Suppose that there were a shorter path than $ABE$, say $AXYZE$. There are two possibilities: either node $Z$ has already been made permanent, or it has not been. If it has, then $E$ has already been probed (on the round following the one when $Z$ was made permanent), so the $AXYZE$ path has not escaped our attention and thus cannot be a shorter path.

Now consider the case where $Z$ is still tentatively labeled. Either the label at $Z$ is greater than or equal to that at $E$, in which case $AXYZE$ cannot be a shorter path than $ABE$, or it is less than that of $E$, in which case $Z$ and not $E$ will become permanent first, allowing $E$ to be probed from $Z$.

This algorithm is given in Fig. 5-8. The global variables $n$ and $dist$ describe the graph and are initialized before $shortest_path$ is called. The only difference between the program and the algorithm described above is that in Fig. 5-8, we compute the shortest path starting at the terminal node, $t$, rather than at the source node, $s$. Since the shortest path from $t$ to $s$ in an undirected graph is the same as the shortest path from $s$ to $t$, it does not matter at which end we begin (unless there are several shortest paths, in which case reversing the search might discover a different one). The reason for searching backward is that each node is labeled with its predecessor rather than its successor. When the final path is copied into the output variable, $path$, the path is thus reversed. By reversing the search, the two effects cancel, and the answer is produced in the correct order.

**Figure 5-8. Dijkstra's algorithm to compute the shortest path through a graph.**
5.2.3 Flooding

Another static algorithm is flooding, in which every incoming packet is sent out on every outgoing line except the one it arrived on. Flooding obviously generates vast numbers of duplicate packets, in fact, an infinite number unless some measures are taken to damp the process. One such measure is to have a hop counter contained in the header of each packet, which is decremented at each hop, with the packet being discarded when the counter reaches zero. Ideally, the hop counter should be initialized to the length of the path from source to destination. If the sender does not know how long the path is, it can initialize the counter to the worst case, namely, the full diameter of the subnet.

An alternative technique for damming the flood is to keep track of which packets have been flooded, to avoid sending them out a second time. One way to achieve this goal is to have the source router put a sequence number in each packet it receives from its hosts. Each router then needs a list per source router telling which sequence numbers originating at that source have already been seen. If an incoming packet is on the list, it is not flooded.
To prevent the list from growing without bound, each list should be augmented by a counter, $k$, meaning that all sequence numbers through $k$ have been seen. When a packet comes in, it is easy to check if the packet is a duplicate; if so, it is discarded. Furthermore, the full list below $k$ is not needed, since $k$ effectively summarizes it.

A variation of flooding that is slightly more practical is **selective flooding**. In this algorithm the routers do not send every incoming packet out on every line, only on those lines that are going approximately in the right direction. There is usually little point in sending a westbound packet on an eastbound line unless the topology is extremely peculiar and the router is sure of this fact.

Flooding is not practical in most applications, but it does have some uses. For example, in military applications, where large numbers of routers may be blown to bits at any instant, the tremendous robustness of flooding is highly desirable. In distributed database applications, it is sometimes necessary to update all the databases concurrently, in which case flooding can be useful. In wireless networks, all messages transmitted by a station can be received by all other stations within its radio range, which is, in fact, flooding, and some algorithms utilize this property. A fourth possible use of flooding is as a metric against which other routing algorithms can be compared. Flooding always chooses the shortest path because it chooses every possible path in parallel. Consequently, no other algorithm can produce a shorter delay (if we ignore the overhead generated by the flooding process itself).

### 5.2.4 Distance Vector Routing

Modern computer networks generally use dynamic routing algorithms rather than the static ones described above because static algorithms do not take the current network load into account. Two dynamic algorithms in particular, distance vector routing and link state routing, are the most popular. In this section we will look at the former algorithm. In the following section we will study the latter algorithm.

**Distance vector routing** algorithms operate by having each router maintain a table (i.e., a vector) giving the best known distance to each destination and which line to use to get there. These tables are updated by exchanging information with the neighbors.

The distance vector routing algorithm is sometimes called by other names, most commonly the distributed **Bellman-Ford** routing algorithm and the **Ford-Fulkerson** algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the name RIP.

In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in the subnet. This entry contains two parts: the preferred outgoing line to use for that destination and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

The router is assumed to know the "distance" to each of its neighbors. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special ECHO packets that the receiver just timestamps and sends back as fast as it can.

As an example, assume that delay is used as a metric and that the router knows the delay to each of its neighbors. Once every $T$ msec each router sends to each neighbor a list of its estimated delays to each destination. It also receives a similar list from each neighbor. Imagine that one of these tables has just come in from neighbor $X$, with $X_i$ being $X$’s estimate of how long it takes to get to router $i$. If the router knows that the delay to $X$ is $m$ msec, it also knows that it can reach
router $i$ via $X$ in $X_i + m$ msec. By performing this calculation for each neighbor, a router can find out which estimate seems the best and use that estimate and the corresponding line in its new routing table. Note that the old routing table is not used in the calculation.

This updating process is illustrated in Fig. 5-9. Part (a) shows a subnet. The first four columns of part (b) show the delay vectors received from the neighbors of router $J$. $A$ claims to have a 12-msec delay to $B$, a 25-msec delay to $C$, a 40-msec delay to $D$, etc. Suppose that $J$ has measured or estimated its delay to its neighbors, $A$, $I$, $H$, and $K$ as 8, 10, 12, and 6 msec, respectively.

![Figure 5-9. (a) A subnet. (b) Input from $A$, $I$, $H$, $K$, and the new routing table for $J$.](image)

Consider how $J$ computes its new route to router $G$. It knows that it can get to $A$ in 8 msec, and $A$ claims to be able to get to $G$ in 18 msec, so $J$ knows it can count on a delay of 26 msec to $G$ if it forwards packets bound for $G$ to $A$. In one vector exchange, the good news is processed. To see how fast good news propagates, consider the five-node (linear) subnet of Fig. 5-10, where the delay metric is the number of hops. Suppose $A$ is down initially and all the other routers know this. In other words, they have all recorded the delay to $A$ as infinity.

![Figure 5-10. The count-to-infinity problem.](image)

**The Count-to-Infinity Problem**

Distance vector routing works in theory but has a serious drawback in practice: although it converges to the correct answer, it may do so slowly. In particular, it reacts rapidly to good news, but leisurely to bad news. Consider a router whose best route to destination $X$ is large. If on the next exchange neighbor $A$ suddenly reports a short delay to $X$, the router just switches over to using the line to $A$ to send traffic to $X$. In one vector exchange, the good news is processed.

To see how fast good news propagates, consider the five-node (linear) subnet of Fig. 5-10, where the delay metric is the number of hops. Suppose $A$ is down initially and all the other routers know this. In other words, they have all recorded the delay to $A$ as infinity.
When A comes up, the other routers learn about it via the vector exchanges. For simplicity we will assume that there is a gigantic gong somewhere that is struck periodically to initiate a vector exchange at all routers simultaneously. At the time of the first exchange, B learns that its left neighbor has zero delay to A. B now makes an entry in its routing table that A is one hop away to the left. All the other routers still think that A is down. At this point, the routing table entries for A are as shown in the second row of Fig. 5-10(a). On the next exchange, C learns that B has a path of length 1 to A, so it updates its routing table to indicate a path of length 2, but D and E do not hear the good news until later. Clearly, the good news is spreading at the rate of one hop per exchange. In a subnet whose longest path is of length N hops, within N exchanges everyone will know about newly-revived lines and routers.

Now let us consider the situation of Fig. 5-10(b), in which all the lines and routers are initially up. Routers B, C, D, and E have distances to A of 1, 2, 3, and 4, respectively. Suddenly A goes down, or alternatively, the line between A and B is cut, which is effectively the same thing from B's point of view.

At the first packet exchange, B does not hear anything from A. Fortunately, C says: Do not worry; I have a path to A of length 2. Little does B know that C's path runs through B itself. For all B knows, C might have ten lines all with separate paths to A of length 2. As a result, B thinks it can reach A via C, with a path length of 3. D and E do not update their entries for A on the first exchange.

On the second exchange, C notices that each of its neighbors claims to have a path to A of length 3. It picks one of the them at random and makes its new distance to A 4, as shown in the third row of Fig. 5-10(b). Subsequent exchanges produce the history shown in the rest of Fig. 5-10(b).

From this figure, it should be clear why bad news travels slowly: no router ever has a value more than one higher than the minimum of all its neighbors. Gradually, all routers work their way up to infinity, but the number of exchanges required depends on the numerical value used for infinity. For this reason, it is wise to set infinity to the longest path plus 1. If the metric is time delay, there is no well-defined upper bound, so a high value is needed to prevent a path with a long delay from being treated as down. Not entirely surprisingly, this problem is known as the count-to-infinity problem. There have been a few attempts to solve it (such as split horizon with poisoned reverse in RFC 1058), but none of these work well in general. The core of the problem is that when X tells Y that it has a path somewhere, Y has no way of knowing whether it itself is on the path.

5.2.5 Link State Routing

Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing. Two primary problems caused its demise. First, since the delay metric was queue length, it did not take line bandwidth into account when choosing routes. Initially, all the lines were 56 kbps,
so line bandwidth was not an issue, but after some lines had been upgraded to 230 kbps and others to 1.544 Mbps, not taking bandwidth into account was a major problem. Of course, it would have been possible to change the delay metric to factor in line bandwidth, but a second problem also existed, namely, the algorithm often took too long to converge (the count-to-infinity problem). For these reasons, it was replaced by an entirely new algorithm, now called **link state routing**. Variants of link state routing are now widely used.

The idea behind link state routing is simple and can be stated as five parts. Each router must do the following:

1. Discover its neighbors and learn their network addresses.
2. Measure the delay or cost to each of its neighbors.
3. Construct a packet telling all it has just learned.
4. Send this packet to all other routers.
5. Compute the shortest path to every other router.

In effect, the complete topology and all delays are experimentally measured and distributed to every router. Then Dijkstra's algorithm can be run to find the shortest path to every other router. Below we will consider each of these five steps in more detail.

**Learning about the Neighbors**

When a router is booted, its first task is to learn who its neighbors are. It accomplishes this goal by sending a special HELLO packet on each point-to-point line. The router on the other end is expected to send back a reply telling who it is. These names must be globally unique because when a distant router later hears that three routers are all connected to \( F \), it is essential that it can determine whether all three mean the same \( F \).

When two or more routers are connected by a LAN, the situation is slightly more complicated. **Fig. 5-11(a)** illustrates a LAN to which three routers, \( A \), \( C \), and \( F \), are directly connected. Each of these routers is connected to one or more additional routers, as shown.

**Figure 5-11. (a) Nine routers and a LAN. (b) A graph model of (a).**

One way to model the LAN is to consider it as a node itself, as shown in **Fig. 5-11(b)**. Here we have introduced a new, artificial node, \( N \), to which \( A \), \( C \), and \( F \) are connected. The fact that it is possible
to go from $A$ to $C$ on the LAN is represented by the path $ANC$ here.

**Measuring Line Cost**

The link state routing algorithm requires each router to know, or at least have a reasonable estimate of, the delay to each of its neighbors. The most direct way to determine this delay is to send over the line a special ECHO packet that the other side is required to send back immediately. By measuring the round-trip time and dividing it by two, the sending router can get a reasonable estimate of the delay. For even better results, the test can be conducted several times, and the average used. Of course, this method implicitly assumes the delays are symmetric, which may not always be the case.

An interesting issue is whether to take the load into account when measuring the delay. To factor the load in, the round-trip timer must be started when the ECHO packet is queued. To ignore the load, the timer should be started when the ECHO packet reaches the front of the queue.

Arguments can be made both ways. Including traffic-induced delays in the measurements means that when a router has a choice between two lines with the same bandwidth, one of which is heavily loaded all the time and one of which is not, the router will regard the route over the unloaded line as a shorter path. This choice will result in better performance.

Unfortunately, there is also an argument against including the load in the delay calculation. Consider the subnet of Fig. 5-12, which is divided into two parts, East and West, connected by two lines, $CF$ and $EI$.

![Figure 5-12. A subnet in which the East and West parts are connected by two lines.](image)

Suppose that most of the traffic between East and West is using line $CF$, and as a result, this line is heavily loaded with long delays. Including queueing delay in the shortest path calculation will make $EI$ more attractive. After the new routing tables have been installed, most of the East-West traffic will now go over $EI$, overloading this line. Consequently, in the next update, $CF$ will appear to be the shortest path. As a result, the routing tables may oscillate wildly, leading to erratic routing and many potential problems. If load is ignored and only bandwidth is considered, this problem does not occur. Alternatively, the load can be spread over both lines, but this solution does not fully utilize the best path. Nevertheless, to avoid oscillations in the choice of best path, it may be wise to distribute the load over multiple lines, with some known fraction going over each line.

**Building Link State Packets**
Once the information needed for the exchange has been collected, the next step is for each router to build a packet containing all the data. The packet starts with the identity of the sender, followed by a sequence number and age (to be described later), and a list of neighbors. For each neighbor, the delay to that neighbor is given. An example subnet is given in Fig. 5-13(a) with delays shown as labels on the lines. The corresponding link state packets for all six routers are shown in Fig. 5-13(b).

Figure 5-13. (a) A subnet. (b) The link state packets for this subnet.

Building the link state packets is easy. The hard part is determining when to build them. One possibility is to build them periodically, that is, at regular intervals. Another possibility is to build them when some significant event occurs, such as a line or neighbor going down or coming back up again or changing its properties appreciably.

Distributing the Link State Packets

The trickiest part of the algorithm is distributing the link state packets reliably. As the packets are distributed and installed, the routers getting the first ones will change their routes. Consequently, the different routers may be using different versions of the topology, which can lead to inconsistencies, loops, unreachable machines, and other problems.

First we will describe the basic distribution algorithm. Later we will give some refinements. The fundamental idea is to use flooding to distribute the link state packets. To keep the flood in check, each packet contains a sequence number that is incremented for each new packet sent. Routers keep track of all the (source router, sequence) pairs they see. When a new link state packet comes in, it is checked against the list of packets already seen. If it is new, it is forwarded on all lines except the one it arrived on. If it is a duplicate, it is discarded. If a packet with a sequence number lower than the highest one seen so far ever arrives, it is rejected as being obsolete since the router has more recent data.

This algorithm has a few problems, but they are manageable. First, if the sequence numbers wrap around, confusion will reign. The solution here is to use a 32-bit sequence number. With one link state packet per second, it would take 137 years to wrap around, so this possibility can be ignored.

Second, if a router ever crashes, it will lose track of its sequence number. If it starts again at 0, the next packet will be rejected as a duplicate.

Third, if a sequence number is ever corrupted and 65,540 is received instead of 4 (a 1-bit error), packets 5 through 65,540 will be rejected as obsolete, since the current sequence number is thought to be 65,540.

The solution to all these problems is to include the age of each packet after the sequence number and decrement it once per second. When the age hits zero, the information from that router is discarded. Normally, a new packet comes in, say, every 10 sec, so router information only times...
out when a router is down (or six consecutive packets have been lost, an unlikely event). The Age field is also decremented by each router during the initial flooding process, to make sure no packet can get lost and live for an indefinite period of time (a packet whose age is zero is discarded).

Some refinements to this algorithm make it more robust. When a link state packet comes in to a router for flooding, it is not queued for transmission immediately. Instead it is first put in a holding area to wait a short while. If another link state packet from the same source comes in before the first packet is transmitted, their sequence numbers are compared. If they are equal, the duplicate is discarded. If they are different, the older one is thrown out. To guard against errors on the router-router lines, all link state packets are acknowledged. When a line goes idle, the holding area is scanned in round-robin order to select a packet or acknowledgement to send.

The data structure used by router B for the subnet shown in Fig. 5-13(a) is depicted in Fig. 5-14. Each row here corresponds to a recently-arrived, but as yet not fully-processed, link state packet. The table records where the packet originated, its sequence number and age, and the data. In addition, there are send and acknowledgement flags for each of B’s three lines (to A, C, and F, respectively). The send flags mean that the packet must be sent on the indicated line. The acknowledgement flags mean that it must be acknowledged there.

**Figure 5-14. The packet buffer for router B in Fig. 5-13.**

<table>
<thead>
<tr>
<th>Source</th>
<th>Seq.</th>
<th>Age</th>
<th>Send flags</th>
<th>ACK flags</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>A</td>
<td>21</td>
<td>60</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>21</td>
<td>60</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>21</td>
<td>59</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
<td>60</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>21</td>
<td>59</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

In Fig. 5-14, the link state packet from A arrives directly, so it must be sent to C and F and acknowledged to A, as indicated by the flag bits. Similarly, the packet from F has to be forwarded to A and C and acknowledged to F.

However, the situation with the third packet, from E, is different. It arrived twice, once via EAB and once via EFB. Consequently, it has to be sent only to C but acknowledged to both A and F, as indicated by the bits.

If a duplicate arrives while the original is still in the buffer, bits have to be changed. For example, if a copy of C’s state arrives from F before the fourth entry in the table has been forwarded, the six bits will be changed to 100011 to indicate that the packet must be acknowledged to F but not sent there.

**Computing the New Routes**

Once a router has accumulated a full set of link state packets, it can construct the entire subnet graph because every link is represented. Every link is, in fact, represented twice, once for each direction. The two values can be averaged or used separately.

Now Dijkstra’s algorithm can be run locally to construct the shortest path to all possible destinations. The results of this algorithm can be installed in the routing tables, and normal
operation resumed.

For a subnet with \( n \) routers, each of which has \( k \) neighbors, the memory required to store the input data is proportional to \( kn \). For large subnets, this can be a problem. Also, the computation time can be an issue. Nevertheless, in many practical situations, link state routing works well.

However, problems with the hardware or software can wreak havoc with this algorithm (also with other ones). For example, if a router claims to have a line it does not have or forgets a line it does have, the subnet graph will be incorrect. If a router fails to forward packets or corrupts them while forwarding them, trouble will arise. Finally, if it runs out of memory or does the routing calculation wrong, bad things will happen. As the subnet grows into the range of tens or hundreds of thousands of nodes, the probability of some router failing occasionally becomes nonnegligible. The trick is to try to arrange to limit the damage when the inevitable happens. Perlman (1988) discusses these problems and their solutions in detail.

Link state routing is widely used in actual networks, so a few words about some example protocols using it are in order. The OSPF protocol, which is widely used in the Internet, uses a link state algorithm. We will describe OSPF in Sec. 5.6.4.

Another link state protocol is IS-IS (Intermediate System-Intermediate System), which was designed for DECnet and later adopted by ISO for use with its connectionless network layer protocol, CLNP. Since then it has been modified to handle other protocols as well, most notably, IP. IS-IS is used in some Internet backbones (including the old NSFNET backbone) and in some digital cellular systems such as CDPD. Novell NetWare uses a minor variant of IS-IS (NLSP) for routing IPX packets.

Basically IS-IS distributes a picture of the router topology, from which the shortest paths are computed. Each router announces, in its link state information, which network layer addresses it can reach directly. These addresses can be IP, IPX, AppleTalk, or any other addresses. IS-IS can even support multiple network layer protocols at the same time.

Many of the innovations designed for IS-IS were adopted by OSPF (OSPF was designed several years after IS-IS). These include a self-stabilizing method of flooding link state updates, the concept of a designated router on a LAN, and the method of computing and supporting path splitting and multiple metrics. As a consequence, there is very little difference between IS-IS and OSPF. The most important difference is that IS-IS is encoded in such a way that it is easy and natural to simultaneously carry information about multiple network layer protocols, a feature OSPF does not have. This advantage is especially valuable in large multiprotocol environments.

5.2.6 Hierarchical Routing

As networks grow in size, the router routing tables grow proportionally. Not only is router memory consumed by ever-increasing tables, but more CPU time is needed to scan them and more bandwidth is needed to send status reports about them. At a certain point the network may grow to the point where it is no longer feasible for every router to have an entry for every other router, so the routing will have to be done hierarchically, as it is in the telephone network.

When hierarchical routing is used, the routers are divided into what we will call regions, with each router knowing all the details about how to route packets to destinations within its own region, but knowing nothing about the internal structure of other regions. When different networks are interconnected, it is natural to regard each one as a separate region in order to free the routers in one network from having to know the topological structure of the other ones.

For huge networks, a two-level hierarchy may be insufficient; it may be necessary to group the
regions into clusters, the clusters into zones, the zones into groups, and so on, until we run out of names for aggregations. As an example of a multilevel hierarchy, consider how a packet might be routed from Berkeley, California, to Malindi, Kenya. The Berkeley router would know the detailed topology within California but would send all out-of-state traffic to the Los Angeles router. The Los Angeles router would be able to route traffic to other domestic routers but would send foreign traffic to New York. The New York router would be programmed to direct all traffic to the router in the destination country responsible for handling foreign traffic, say, in Nairobi. Finally, the packet would work its way down the tree in Kenya until it got to Malindi.

Figure 5-15 gives a quantitative example of routing in a two-level hierarchy with five regions. The full routing table for router 1A has 17 entries, as shown in Fig. 5-15(b). When routing is done hierarchically, as in Fig. 5-15(c), there are entries for all the local routers as before, but all other regions have been condensed into a single router, so all traffic for region 2 goes via the 1B -2A line, but the rest of the remote traffic goes via the 1C -3B line. Hierarchical routing has reduced the table from 17 to 7 entries. As the ratio of the number of regions to the number of routers per region grows, the savings in table space increase.

Unfortunately, these gains in space are not free. There is a penalty to be paid, and this penalty is in the form of increased path length. For example, the best route from 1A to 5C is via region 2, but with hierarchical routing all traffic to region 5 goes via region 3, because that is better for most destinations in region 5.

When a single network becomes very large, an interesting question is: How many levels should the hierarchy have? For example, consider a subnet with 720 routers. If there is no hierarchy, each router needs 720 routing table entries. If the subnet is partitioned into 24 regions of 30 routers each, each router needs 30 local entries plus 23 remote entries for a total of 53 entries. If a three-level hierarchy is chosen, with eight clusters, each containing 9 regions of 10 routers, each router needs 10 entries for local routers, 8 entries for routing to other regions within its own cluster, and 7 entries for distant clusters, for a total of 25 entries. Kamoun and Kleinrock (1979) discovered that the optimal number of levels for an \( N \) router subnet is \( \ln N \), requiring a total of \( e \ln N \) entries per router. They have also shown that the increase in effective mean path length caused by
hierarchical routing is sufficiently small that it is usually acceptable.

5.2.7 Broadcast Routing

In some applications, hosts need to send messages to many or all other hosts. For example, a service distributing weather reports, stock market updates, or live radio programs might work best by broadcasting to all machines and letting those that are interested read the data. Sending a packet to all destinations simultaneously is called broadcasting; various methods have been proposed for doing it.

One broadcasting method that requires no special features from the subnet is for the source to simply send a distinct packet to each destination. Not only is the method wasteful of bandwidth, but it also requires the source to have a complete list of all destinations. In practice this may be the only possibility, but it is the least desirable of the methods.

Flooding is another obvious candidate. Although flooding is ill-suited for ordinary point-to-point communication, for broadcasting it might rate serious consideration, especially if none of the methods described below are applicable. The problem with flooding as a broadcast technique is the same problem it has as a point-to-point routing algorithm: it generates too many packets and consumes too much bandwidth.

A third algorithm is multidestination routing. If this method is used, each packet contains either a list of destinations or a bit map indicating the desired destinations. When a packet arrives at a router, the router checks all the destinations to determine the set of output lines that will be needed. (An output line is needed if it is the best route to at least one of the destinations.) The router generates a new copy of the packet for each output line to be used and includes in each packet only those destinations that are to use the line. In effect, the destination set is partitioned among the output lines. After a sufficient number of hops, each packet will carry only one destination and can be treated as a normal packet. Multidestination routing is like separately addressed packets, except that when several packets must follow the same route, one of them pays full fare and the rest ride free.

A fourth broadcast algorithm makes explicit use of the sink tree for the router initiating the broadcast—or any other convenient spanning tree for that matter. A spanning tree is a subset of the subnet that includes all the routers but contains no loops. If each router knows which of its lines belong to the spanning tree, it can copy an incoming broadcast packet onto all the spanning tree lines except the one it arrived on. This method makes excellent use of bandwidth, generating the absolute minimum number of packets necessary to do the job. The only problem is that each router must have knowledge of some spanning tree for the method to be applicable. Sometimes this information is available (e.g., with link state routing) but sometimes it is not (e.g., with distance vector routing).

Our last broadcast algorithm is an attempt to approximate the behavior of the previous one, even when the routers do not know anything at all about spanning trees. The idea, called reverse path forwarding, is remarkably simple once it has been pointed out. When a broadcast packet arrives at a router, the router checks to see if the packet arrived on the line that is normally used for sending packets to the source of the broadcast. If so, there is an excellent chance that the broadcast packet itself followed the best route from the router and is therefore the first copy to arrive at the router. This being the case, the router forwards copies of it onto all lines except the one it arrived on. If, however, the broadcast packet arrived on a line other than the preferred one for reaching the source, the packet is discarded as a likely duplicate.

An example of reverse path forwarding is shown in Fig. 5-16. Part (a) shows a subnet, part (b) shows a sink tree for router $I$ of that subnet, and part (c) shows how the reverse path algorithm
works. On the first hop, I sends packets to F, H, J, and N, as indicated by the second row of the tree. Each of these packets arrives on the preferred path to I (assuming that the preferred path falls along the sink tree) and is so indicated by a circle around the letter. On the second hop, eight packets are generated, two by each of the routers that received a packet on the first hop. As it turns out, all eight of these arrive at previously unvisited routers, and five of these arrive along the preferred line. Of the six packets generated on the third hop, only three arrive on the preferred path (at C, E, and K); the others are duplicates. After five hops and 24 packets, the broadcasting terminates, compared with four hops and 14 packets had the sink tree been followed exactly.

**Figure 5-16. Reverse path forwarding. (a) A subnet. (b) A sink tree. (c) The tree built by reverse path forwarding.**

The principal advantage of reverse path forwarding is that it is both reasonably efficient and easy to implement. It does not require routers to know about spanning trees, nor does it have the overhead of a destination list or bit map in each broadcast packet as does multidestination addressing. Nor does it require any special mechanism to stop the process, as flooding does (either a hop counter in each packet and a priori knowledge of the subnet diameter, or a list of packets already seen per source).

### 5.2.8 Multicast Routing

Some applications require that widely-separated processes work together in groups, for example, a group of processes implementing a distributed database system. In these situations, it is frequently necessary for one process to send a message to all the other members of the group. If the group is small, it can just send each other member a point-to-point message. If the group is large, this strategy is expensive. Sometimes broadcasting can be used, but using broadcasting to inform 1000 machines on a million-node network is inefficient because most receivers are not interested in the message (or worse yet, they are definitely interested but are not supposed to see it). Thus, we need a way to send messages to well-defined groups that are numerically large in size but small compared to the network as a whole.

Sending a message to such a group is called **multicasting**, and its routing algorithm is called **multicast routing**. In this section we will describe one way of doing multicast routing. For additional information, see (Chu et al., 2000; Costa et al. 2001; Kasera et al., 2000; Madruga and Garcia-Luna-Aceves, 2001; Zhang and Ryu, 2001).

Multicasting requires group management. Some way is needed to create and destroy groups, and to allow processes to join and leave groups. How these tasks are accomplished is not of concern to the routing algorithm. What is of concern is that when a process joins a group, it informs its host of this fact. It is important that routers know which of their hosts belong to which groups. Either hosts must inform their routers about changes in group membership, or routers must query their hosts periodically. Either way, routers learn about which of their hosts are in which groups. Routers tell
their neighbors, so the information propagates through the subnet.

To do multicast routing, each router computes a spanning tree covering all other routers. For example, in Fig. 5-17(a) we have two groups, 1 and 2. Some routers are attached to hosts that belong to one or both of these groups, as indicated in the figure. A spanning tree for the leftmost router is shown in Fig. 5-17(b).

Figure 5-17. (a) A network. (b) A spanning tree for the leftmost router. (c) A multicast tree for group 1. (d) A multicast tree for group 2.

When a process sends a multicast packet to a group, the first router examines its spanning tree and prunes it, removing all lines that do not lead to hosts that are members of the group. In our example, Fig. 5-17(c) shows the pruned spanning tree for group 1. Similarly, Fig. 5-17(d) shows the pruned spanning tree for group 2. Multicast packets are forwarded only along the appropriate spanning tree.

Various ways of pruning the spanning tree are possible. The simplest one can be used if link state routing is used and each router is aware of the complete topology, including which hosts belong to which groups. Then the spanning tree can be pruned, starting at the end of each path, working toward the root, and removing all routers that do not belong to the group in question.

With distance vector routing, a different pruning strategy can be followed. The basic algorithm is reverse path forwarding. However, whenever a router with no hosts interested in a particular group and no connections to other routers receives a multicast message for that group, it responds with a PRUNE message, telling the sender not to send it any more multicasts for that group. When a router with no group members among its own hosts has received such messages on all its lines, it, too, can respond with a PRUNE message. In this way, the subnet is recursively pruned.

One potential disadvantage of this algorithm is that it scales poorly to large networks. Suppose that a network has \( n \) groups, each with an average of \( m \) members. For each group, \( m \) pruned spanning trees must be stored, for a total of \( mn \) trees. When many large groups exist, considerable storage is needed to store all the trees.
An alternative design uses core-based trees (Ballardie et al., 1993). Here, a single spanning tree per group is computed, with the root (the core) near the middle of the group. To send a multicast message, a host sends it to the core, which then does the multicast along the spanning tree. Although this tree will not be optimal for all sources, the reduction in storage costs from $m$ trees to one tree per group is a major saving.

5.2.9 Routing for Mobile Hosts

Millions of people have portable computers nowadays, and they generally want to read their e-mail and access their normal file systems wherever in the world they may be. These mobile hosts introduce a new complication: to route a packet to a mobile host, the network first has to find it. The subject of incorporating mobile hosts into a network is very young, but in this section we will sketch some of the issues and give a possible solution.

The model of the world that network designers typically use is shown in Fig. 5-18. Here we have a WAN consisting of routers and hosts. Connected to the WAN are LANs, MANs, and wireless cells of the type we studied in Chap. 2.

**Figure 5-18. A WAN to which LANs, MANs, and wireless cells are attached.**

Hosts that never move are said to be stationary. They are connected to the network by copper wires or fiber optics. In contrast, we can distinguish two other kinds of hosts. Migratory hosts are basically stationary hosts who move from one fixed site to another from time to time but use the network only when they are physically connected to it. Roaming hosts actually compute on the run and want to maintain their connections as they move around. We will use the term mobile hosts to mean either of the latter two categories, that is, all hosts that are away from home and still want to be connected.

All hosts are assumed to have a permanent home location that never changes. Hosts also have a permanent home address that can be used to determine their home locations, analogous to the way the telephone number 1-212-5551212 indicates the United States (country code 1) and Manhattan (212). The routing goal in systems with mobile hosts is to make it possible to send packets to mobile hosts using their home addresses and have the packets efficiently reach them wherever they may be. The trick, of course, is to find them.

In the model of Fig. 5-18, the world is divided up (geographically) into small units. Let us call them areas, where an area is typically a LAN or wireless cell. Each area has one or more foreign agents, which are processes that keep track of all mobile hosts visiting the area. In addition, each area has a home agent, which keeps track of hosts whose home is in the area, but who are currently visiting another area.
When a new host enters an area, either by connecting to it (e.g., plugging into the LAN) or just wandering into the cell, his computer must register itself with the foreign agent there. The registration procedure typically works like this:

1. Periodically, each foreign agent broadcasts a packet announcing its existence and address. A newly-arrived mobile host may wait for one of these messages, but if none arrives quickly enough, the mobile host can broadcast a packet saying: Are there any foreign agents around?

2. The mobile host registers with the foreign agent, giving its home address, current data link layer address, and some security information.

3. The foreign agent contacts the mobile host's home agent and says: One of your hosts is over here. The message from the foreign agent to the home agent contains the foreign agent's network address. It also includes the security information to convince the home agent that the mobile host is really there.

4. The home agent examines the security information, which contains a timestamp, to prove that it was generated within the past few seconds. If it is happy, it tells the foreign agent to proceed.

5. When the foreign agent gets the acknowledgement from the home agent, it makes an entry in its tables and informs the mobile host that it is now registered.

Ideally, when a host leaves an area, that, too, should be announced to allow deregistration, but many users abruptly turn off their computers when done.

When a packet is sent to a mobile host, it is routed to the host's home LAN because that is what the address says should be done, as illustrated in step 1 of Fig. 5-19. Here the sender, in the northwest city of Seattle, wants to send a packet to a host normally across the United States in New York. Packets sent to the mobile host on its home LAN in New York are intercepted by the home agent there. The home agent then looks up the mobile host's new (temporary) location and finds the address of the foreign agent handling the mobile host, in Los Angeles.

**Figure 5-19. Packet routing for mobile hosts.**
The home agent then does two things. First, it encapsulates the packet in the payload field of an outer packet and sends the latter to the foreign agent (step 2 in Fig. 5-19). This mechanism is called tunneling; we will look at it in more detail later. After getting the encapsulated packet, the foreign agent removes the original packet from the payload field and sends it to the mobile host as a data link frame.

Second, the home agent tells the sender to henceforth send packets to the mobile host by encapsulating them in the payload of packets explicitly addressed to the foreign agent instead of just sending them to the mobile host's home address (step 3). Subsequent packets can now be routed directly to the host via the foreign agent (step 4), bypassing the home location entirely.

The various schemes that have been proposed differ in several ways. First, there is the issue of how much of this protocol is carried out by the routers and how much by the hosts, and in the latter case, by which layer in the hosts. Second, in a few schemes, routers along the way record mapped addresses so they can intercept and redirect traffic even before it gets to the home location. Third, in some schemes each visitor is given a unique temporary address; in others, the temporary address refers to an agent that handles traffic for all visitors.

Fourth, the schemes differ in how they actually manage to arrange for packets that are addressed to one destination to be delivered to a different one. One choice is changing the destination address and just retransmitting the modified packet. Alternatively, the whole packet, home address and all, can be encapsulated inside the payload of another packet sent to the temporary address. Finally, the schemes differ in their security aspects. In general, when a host or router gets a message of the form "Starting right now, please send all of Stephany's mail to me," it might have a couple of questions about whom it was talking to and whether this is a good idea. Several mobile host protocols are discussed and compared in (Hac and Guo, 2000; Perkins, 1998a; Snoeren and Balakrishnan, 2000; Solomon, 1998; and Wang and Chen, 2001).

5.2.10 Routing in Ad Hoc Networks

We have now seen how to do routing when the hosts are mobile but the routers are fixed. An even more extreme case is one in which the routers themselves are mobile. Among the possibilities are:

1. Military vehicles on a battlefield with no existing infrastructure.
2. A fleet of ships at sea.

3. Emergency workers at an earthquake that destroyed the infrastructure.

4. A gathering of people with notebook computers in an area lacking 802.11.

In all these cases, and others, each node consists of a router and a host, usually on the same computer. Networks of nodes that just happen to be near each other are called ad hoc networks or MANETs (Mobile Ad hoc NETworks). Let us now examine them briefly. More information can be found in (Perkins, 2001).

What makes ad hoc networks different from wired networks is that all the usual rules about fixed topologies, fixed and known neighbors, fixed relationship between IP address and location, and more are suddenly tossed out the window. Routers can come and go or appear in new places at the drop of a bit. With a wired network, if a router has a valid path to some destination, that path continues to be valid indefinitely (barring a failure somewhere in the system). With an ad hoc network, the topology may be changing all the time, so desirability and even validity of paths can change spontaneously, without warning. Needless to say, these circumstances make routing in ad hoc networks quite different from routing in their fixed counterparts.

A variety of routing algorithms for ad hoc networks have been proposed. One of the more interesting ones is the AODV (Ad hoc On-demand Distance Vector) routing algorithm (Perkins and Royer, 1999). It is a distant relative of the Bellman-Ford distance vector algorithm but adapted to work in a mobile environment and takes into account the limited bandwidth and low battery life found in this environment. Another unusual characteristic is that it is an on-demand algorithm, that is, it determines a route to some destination only when somebody wants to send a packet to that destination. Let us now see what that means.

**Route Discovery**

At any instant of time, an ad hoc network can be described by a graph of the nodes (routers + hosts). Two nodes are connected (i.e., have an arc between them in the graph) if they can communicate directly using their radios. Since one of the two may have a more powerful transmitter than the other, it is possible that A is connected to B but B is not connected to A. However, for simplicity, we will assume all connections are symmetric. It should also be noted that the mere fact that two nodes are within radio range of each other does not mean that they are connected. There may be buildings, hills, or other obstacles that block their communication.

To describe the algorithm, consider the ad hoc network of **Fig. 5-20**, in which a process at node A wants to send a packet to node I. The AODV algorithm maintains a table at each node, key by destination, giving information about that destination, including which neighbor to send packets to in order to reach the destination. Suppose that A looks in its table and does not find an entry for I. It may now have to discover a route to I. This property of discovering routes only when they are needed is what makes this algorithm "on demand."

**Figure 5-20. (a) Range of A's broadcast. (b) After B and D have received A's broadcast. (c) After C, F, and G have received A's broadcast. (d) After E, H, and I have received A's broadcast. The shaded nodes are new recipients. The arrows show the possible reverse routes.**
To locate I, A constructs a special ROUTE REQUEST packet and broadcasts it. The packet reaches B and D, as illustrated in Fig. 5-20(a). In fact, the reason B and D are connected to A in the graph is that they can receive communication from A. F, for example, is not shown with an arc to A because it cannot receive A's radio signal. Thus, F is not connected to A.

The format of the ROUTE REQUEST packet is shown in Fig. 5-21. It contains the source and destination addresses, typically their IP addresses, which identify who is looking for whom. It also contains a Request ID, which is a local counter maintained separately by each node and incremented each time a ROUTE REQUEST is broadcast. Together, the Source address and Request ID fields uniquely identify the ROUTE REQUEST packet to allow nodes to discard any duplicates they may receive.

Figure 5-21. Format of a ROUTE REQUEST packet.

<table>
<thead>
<tr>
<th>Source address</th>
<th>Request ID</th>
<th>Destination address</th>
<th>Source sequence #</th>
<th>Dest. sequence #</th>
<th>Hop count</th>
</tr>
</thead>
</table>

In addition to the Request ID counter, each node also maintains a second sequence counter incremented whenever a ROUTE REQUEST is sent (or a reply to someone else's ROUTE REQUEST). It functions a little bit like a clock and is used to tell new routes from old routes. The fourth field of Fig. 5-21 is A's sequence counter; the fifth field is the most recent value of I's sequence number that A has seen (0 if it has never seen it). The use of these fields will become clear shortly. The final field, Hop count, will keep track of how many hops the packet has made. It is initialized to 0.

When a ROUTE REQUEST packet arrives at a node (B and D in this case), it is processed in the following steps.

1. The (Source address, Request ID) pair is looked up in a local history table to see if this request has already been seen and processed. If it is a duplicate, it is discarded and processing stops. If it is not a duplicate, the pair is entered into the history table so future duplicates can be rejected, and processing continues.

2. The receiver looks up the destination in its route table. If a fresh route to the destination is known, a ROUTE REPLY packet is sent back to the source telling it how to get to the destination (basically: Use me). Fresh means that the Destination sequence number stored in the routing table is greater than or equal to the Destination sequence number in the ROUTE REQUEST packet. If it is less, the stored route is older than the previous route the source had for the destination, so step 3 is executed.

3. Since the receiver does not know a fresh route to the destination, it increments the Hop
count field and rebroadcasts the ROUTE REQUEST packet. It also extracts the data from the packet and stores it as a new entry in its reverse route table. This information will be used to construct the reverse route so that the reply can get back to the source later. The arrows in Fig. 5-20 are used for building the reverse route. A timer is also started for the newly-made reverse route entry. If it expires, the entry is deleted.

Neither B nor D knows where I is, so each of them creates a reverse route entry pointing back to A, as shown by the arrows in Fig. 5-20, and broadcasts the packet with Hop count set to 1. The broadcast from B reaches C and D. C makes an entry for it in its reverse route table and rebroadcasts it. In contrast, D rejects it as a duplicate. Similarly, D's broadcast is rejected by B. However, D's broadcast is accepted by F and G and stored, as shown in Fig. 5-20(c). After E, H, and I receive the broadcast, the ROUTE REQUEST finally reaches a destination that knows where I is, namely, I itself, as illustrated in Fig. 5-20(d). Note that although we have shown the broadcasts in three discrete steps here, the broadcasts from different nodes are not coordinated in any way.

In response to the incoming request, I builds a ROUTE REPLY packet, as shown in Fig. 5-22. The Source address, Destination address, and Hop count are copied from the incoming request, but the Destination sequence number taken from its counter in memory. The Hop count field is set to 0. The Lifetime field controls how long the route is valid. This packet is unicast to the node that the ROUTE REQUEST packet came from, in this case, G. It then follows the reverse path to D and finally to A. At each node, Hop count is incremented so the node can see how far from the destination (I) it is.

**Figure 5-22. Format of a ROUTE REPLY packet.**

<table>
<thead>
<tr>
<th>Source address</th>
<th>Destination address</th>
<th>Destination sequence #</th>
<th>Hop count</th>
<th>Lifetime</th>
</tr>
</thead>
</table>

At each intermediate node on the way back, the packet is inspected. It is entered into the local routing table as a route to I if one or more of the following three conditions are met:

1. No route to I is known.
2. The sequence number for I in the ROUTE REPLY packet is greater than the value in the routing table.
3. The sequence numbers are equal but the new route is shorter.

In this way, all the nodes on the reverse route learn the route to I for free, as a byproduct of A's route discovery. Nodes that got the original REQUEST ROUTE packet but were not on the reverse path (B, C, E, F, and H in this example) discard the reverse route table entry when the associated timer expires.

In a large network, the algorithm generates many broadcasts, even for destinations that are close by. The number of broadcasts can be reduced as follows. The IP packet's Time to live is initialized by the sender to the expected diameter of the network and decremented on each hop. If it hits 0, the packet is discarded instead of being broadcast.

The discovery process is then modified as follows. To locate a destination, the sender broadcasts a ROUTE REQUEST packet with Time to live set to 1. If no response comes back within a reasonable time, another one is sent, this time with Time to live set to 2. Subsequent attempts use 3, 4, 5, etc. In this way, the search is first attempted locally, then in increasingly wider rings.
Route Maintenance

Because nodes can move or be switched off, the topology can change spontaneously. For example, in Fig. 5-20, if G is switched off, A will not realize that the route it was using to I (ADGI) is no longer valid. The algorithm needs to be able to deal with this. Periodically, each node broadcasts a Hello message. Each of its neighbors is expected to respond to it. If no response is forthcoming, the broadcaster knows that that neighbor has moved out of range and is no longer connected to it. Similarly, if it tries to send a packet to a neighbor that does not respond, it learns that the neighbor is no longer available.

This information is used to purge routes that no longer work. For each possible destination, each node, N, keeps track of its neighbors that have fed it a packet for that destination during the last ΔT seconds. These are called N's active neighbors for that destination. N does this by having a routing table keyed by destination and containing the outgoing node to use to reach the destination, the hop count to the destination, the most recent destination sequence number, and the list of active neighbors for that destination. A possible routing table for node D in our example topology is shown in Fig. 5-23(a).

Figure 5-23. (a) D's routing table before G goes down. (b) The graph after G has gone down.

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Next hop</th>
<th>Distance</th>
<th>Active neighbors</th>
<th>Other fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>1</td>
<td>F, G</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>1</td>
<td>F, G</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>B</td>
<td>2</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>1</td>
<td>A, B</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>1</td>
<td>A, B</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>F</td>
<td>2</td>
<td>A, B</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>G</td>
<td>2</td>
<td>A, B</td>
<td></td>
</tr>
</tbody>
</table>

When any of N's neighbors becomes unreachable, it checks its routing table to see which destinations have routes using the now-gone neighbor. For each of these routes, the active neighbors are informed that their route via N is now invalid and must be purged from their routing tables. The active neighbors then tell their active neighbors, and so on, recursively, until all routes depending on the now-gone node are purged from all routing tables.

As an example of route maintenance, consider our previous example, but now with G suddenly switched off. The changed topology is illustrated in Fig. 5-23(b). When D discovers that G is gone, it looks at its routing table and sees that G was used on routes to E, G, and I. The union of the active neighbors for these destinations is the set {A, B}. In other words, A and B depend on G for some of their routes, so they have to be informed that these routes no longer work. D tells them by sending them packets that cause them to update their own routing tables accordingly. D also purges the entries for E, G, and I from its routing table.

It may not have been obvious from our description, but a critical difference between AODV and Bellman-Ford is that nodes do not send out periodic broadcasts containing their entire routing table. This difference saves both bandwidth and battery life.

AODV is also capable of doing broadcast and multicast routing. For details, consult (Perkins and Royer, 2001). Ad hoc routing is a red-hot research area. A great deal has been published on the...
5.2.11 Node Lookup in Peer-to-Peer Networks

A relatively new phenomenon is peer-to-peer networks, in which a large number of people, usually with permanent wired connections to the Internet, are in contact to share resources. The first widespread application of peer-to-peer technology was for mass crime: 50 million Napster users were exchanging copyrighted songs without the copyright owners' permission until Napster was shut down by the courts amid great controversy. Nevertheless, peer-to-peer technology has many interesting and legal uses. It also has something similar to a routing problem, although it is not quite the same as the ones we have studied so far. Nevertheless, it is worth a quick look.

What makes peer-to-peer systems interesting is that they are totally distributed. All nodes are symmetric and there is no central control or hierarchy. In a typical peer-to-peer system the users each have some information that may be of interest to other users. This information may be free software, (public domain) music, photographs, and so on. If there are large numbers of users, they will not know each other and will not know where to find what they are looking for. One solution is a big central database, but this may not be feasible for some reason (e.g., nobody is willing to host and maintain it). Thus, the problem comes down to how a user finds a node that contains what he is looking for in the absence of a centralized database or even a centralized index.

Let us assume that each user has one or more data items such as songs, photographs, programs, files, and so on that other users might want to read. Each item has an ASCII string naming it. A potential user knows just the ASCII string and wants to find out if one or more people have copies and, if so, what their IP addresses are.

As an example, consider a distributed genealogical database. Each genealogist has some on-line records for his or her ancestors and relatives, possibly with photos, audio, or even video clips of the person. Multiple people may have the same great grandfather, so an ancestor may have records at multiple nodes. The name of the record is the person's name in some canonical form. At some point, a genealogist discovers his great grandfather's will in an archive, in which the great grandfather bequeaths his gold pocket watch to his nephew. The genealogist now knows the nephew's name and wants to find out if any other genealogist has a record for him. How, without a central database, do we find out who, if anyone, has records?

Various algorithms have been proposed to solve this problem. The one we will examine is Chord (Dabek et al., 2001a; and Stoica et al., 2001). A simplified explanation of how it works is as follows. The Chord system consists of \( n \) participating users, each of whom may have some stored records and each of whom is prepared to store bits and pieces of the index for use by other users. Each user node has an IP address that can be hashed to an \( m \)-bit number using a hash function, \( hash \). Chord uses SHA-1 for \( hash \). SHA-1 is used in cryptography; we will look at it in Chap. 8. For now, it is just a function that takes a variable-length byte string as argument and produces a highly-random 160-bit number. Thus, we can convert any IP address to a 160-bit number called the node identifier.

Conceptually, all the \( 2^{160} \) node identifiers are arranged in ascending order in a big circle. Some of them correspond to participating nodes, but most of them do not. In Fig. 5-24(a) we show the node identifier circle for \( m = 5 \) (just ignore the arcs in the middle for the moment). In this example, the nodes with identifiers 1, 4, 7, 12, 15, 20, and 27 correspond to actual nodes and are shaded in the figure; the rest do not exist.

**Figure 5-24. (a) A set of 32 node identifiers arranged in a circle. The**
shaded ones correspond to actual machines. The arcs show the fingers from nodes 1, 4, and 12. The labels on the arcs are the table indices. (b) Examples of the finger tables.

Let us now define the function $\text{successor}(k)$ as the node identifier of the first actual node following $k$ around the circle clockwise. For example, $\text{successor}(6) = 7$, $\text{successor}(8) = 12$, and $\text{successor}(22) = 27$.

The names of the records (song names, ancestors' names, and so on) are also hashed with $\text{hash}$ (i.e., SHA-1) to generate a 160-bit number, called the $\text{key}$. Thus, to convert $\text{name}$ (the ASCII name of the record) to its key, we use $\text{key} = \text{hash}(\text{name})$. This computation is just a local procedure call to $\text{hash}$. If a person holding a genealogical record for $\text{name}$ wants to make it available to everyone, he first builds a tuple consisting of ($\text{name}$, my-IP-address) and then asks $\text{successor(\text{hash(name)})}$ to store the tuple. If multiple records (at different nodes) exist for this name, their tuple will all be stored at the same node. In this way, the index is distributed over the nodes at random. For fault tolerance, $p$ different hash functions could be used to store each tuple at $p$ nodes, but we will not consider that further here.

If some user later wants to look up $\text{name}$, he hashes it to get $\text{key}$ and then uses $\text{successor(\text{key})}$ to find the IP address of the node storing its index tuples. The first step is easy; the second one is not. To make it possible to find the IP address of the node corresponding to a certain key, each node must maintain certain administrative data structures. One of these is the IP address of its successor node along the node identifier circle. For example, in Fig. 5-24, node 4's successor is 7 and node 7's successor is 12.

Lookup can now proceed as follows. The requesting node sends a packet to its successor containing its IP address and the key it is looking for. The packet is propagated around the ring until it locates
the successor to the node identifier being sought. That node checks to see if it has any information matching the key, and if so, returns it directly to the requesting node, whose IP address it has.

As a first optimization, each node could hold the IP addresses of both its successor and its predecessor, so that queries could be sent either clockwise or counterclockwise, depending on which path is thought to be shorter. For example, node 7 in Fig. 5-24 could go clockwise to find node identifier 10 but counterclockwise to find node identifier 3.

Even with two choices of direction, linearly searching all the nodes is very inefficient in a large peer-to-peer system since the mean number of nodes required per search is \( n/2 \). To greatly speed up the search, each node also maintains what Chord calls a **finger table**. The finger table has \( m \) entries, indexed by 0 through \( m - 1 \), each one pointing to a different actual node. Each of the entries has two fields: `start` and the IP address of `successor(start)`, as shown for three example nodes in Fig. 5-24(b). The values of the fields for entry \( i \) at node \( k \) are:

\[
\text{start} = k + 2^i \pmod{2^m} \\
\text{IP address of } \text{successor} \left( \text{start}[i] \right)
\]

Note that each node stores the IP addresses of a relatively small number of nodes and that most of these are fairly close by in terms of node identifier.

Using the finger table, the lookup of `key` at node \( k \) proceeds as follows. If `key` falls between \( k \) and `successor(k)`, then the node holding information about `key` is `successor(k)` and the search terminates. Otherwise, the finger table is searched to find the entry whose `start` field is the closest predecessor of `key`. A request is then sent directly to the IP address in that finger table entry to ask it to continue the search. Since it is closer to `key` but still below it, chances are good that it will be able to return the answer with only a small number of additional queries. In fact, since every lookup halves the remaining distance to the target, it can be shown that the average number of lookups is \( \log_2 n \).

As a first example, consider looking up `key = 3` at node 1. Since node 1 knows that 3 lies between it and its successor, 4, the desired node is 4 and the search terminates, returning node 4's IP address.

As a second example, consider looking up `key = 14` at node 1. Since 14 does not lie between 1 and 4, the finger table is consulted. The closest predecessor to 14 is 9, so the request is forwarded to the IP address of 9's entry, namely, that of node 12. Node 12 sees that 14 falls between it and its `successor` (15), so it returns the IP address of node 15.

As a third example, consider looking up `key = 16` at node 1. Again a query is sent to node 12, but this time node 12 does not know the answer itself. It looks for the node most closely preceding 16 and finds 14, which yields the IP address of node 15. A query is then sent there. Node 15 observes that 16 lies between it and its successor (20), so it returns the IP address of 20 to the caller, which works its way back to node 1.

Since nodes join and leave all the time, Chord needs a way to handle these operations. We assume that when the system began operation it was small enough that the nodes could just exchange information directly to build the first circle and finger tables. After that an automated procedure is needed, as follows. When a new node, \( r \), wants to join, it must contact some existing node and ask it to look up the IP address of `successor(r)` for it. The new node then asks `successor(r)` for its predecessor. The new node then asks both of these to insert \( r \) in between them in the circle. For example, if 24 in Fig. 5-24 wants to join, it asks any node to look up `successor(24)`, which is 27. Then it asks 27 for its predecessor (20). After it tells both of those about its existence, 20 uses 24 as its successor and 27 uses 24 as its predecessor. In addition, node 27 hands over those keys in
the range 21–24, which now belong to 24. At this point, 24 is fully inserted.

However, many finger tables are now wrong. To correct them, every node runs a background process that periodically recomputes each finger by calling successor. When one of these queries hits a new node, the corresponding finger entry is updated.

When a node leaves gracefully, it hands its keys over to its successor and informs its predecessor of its departure so the predecessor can link to the departing node's successor. When a node crashes, a problem arises because its predecessor no longer has a valid successor. To alleviate this problem, each node keeps track not only of its direct successor but also its \( s \) direct successors, to allow it to skip over up to \( s - 1 \) consecutive failed nodes and reconnect the circle.

Chord has been used to construct a distributed file system (Dabek et al., 2001b) and other applications, and research is ongoing. A different peer-to-peer system, Pastry, and its applications are described in (Rowstron and Druschel, 2001a; and Rowstron and Druschel, 2001b). A third peer-to-peer system, Freenet, is discussed in (Clarke et al., 2002). A fourth system of this type is described in (Ratnasamy et al., 2001).

### 5.3 Congestion Control Algorithms

When too many packets are present in (a part of) the subnet, performance degrades. This situation is called congestion. Figure 5-25 depicts the symptom. When the number of packets dumped into the subnet by the hosts is within its carrying capacity, they are all delivered (except for a few that are afflicted with transmission errors) and the number delivered is proportional to the number sent. However, as traffic increases too far, the routers are no longer able to cope and they begin losing packets. This tends to make matters worse. At very high traffic, performance collapses completely and almost no packets are delivered.

**Figure 5-25.** When too much traffic is offered, congestion sets in and performance degrades sharply.

Congestion can be brought on by several factors. If all of a sudden, streams of packets begin arriving on three or four input lines and all need the same output line, a queue will build up. If there is insufficient memory to hold all of them, packets will be lost. Adding more memory may help up to a point, but Nagle (1987) discovered that if routers have an infinite amount of memory, congestion gets worse, not better, because by the time packets get to the front of the queue, they
have already timed out (repeatedly) and duplicates have been sent. All these packets will be
dutifully forwarded to the next router, increasing the load all the way to the destination.

Slow processors can also cause congestion. If the routers' CPUs are slow at performing the
bookkeeping tasks required of them (queueing buffers, updating tables, etc.), queues can build up,
even though there is excess line capacity. Similarly, low-bandwidth lines can also cause congestion.
Upgrading the lines but not changing the processors, or vice versa, often helps a little, but
frequently just shifts the bottleneck. Also, upgrading part, but not all, of the system, often just
moves the bottleneck somewhere else. The real problem is frequently a mismatch between parts of
the system. This problem will persist until all the components are in balance.

It is worth explicitly pointing out the difference between congestion control and flow control, as the
relationship is subtle. Congestion control has to do with making sure the subnet is able to carry the
offered traffic. It is a global issue, involving the behavior of all the hosts, all the routers, the store-
and-forwarding processing within the routers, and all the other factors that tend to diminish the
carrying capacity of the subnet.

Flow control, in contrast, relates to the point-to-point traffic between a given sender and a given
receiver. Its job is to make sure that a fast sender cannot continually transmit data faster than the
receiver is able to absorb it. Flow control frequently involves some direct feedback from the
receiver to the sender to tell the sender how things are doing at the other end.

To see the difference between these two concepts, consider a fiber optic network with a capacity of
1000 gigabits/sec on which a supercomputer is trying to transfer a file to a personal computer at 1
Gbps. Although there is no congestion (the network itself is not in trouble), flow control is needed
to force the supercomputer to stop frequently to give the personal computer a chance to breathe.

At the other extreme, consider a store-and-forward network with 1-Mbps lines and 1000 large
computers, half of which are trying to transfer files at 100 kbps to the other half. Here the problem
is not that of fast senders overpowering slow receivers, but that the total offered traffic exceeds
what the network can handle.

The reason congestion control and flow control are often confused is that some congestion control
algorithms operate by sending messages back to the various sources telling them to slow down
when the network gets into trouble. Thus, a host can get a "slow down" message either because
the receiver cannot handle the load or because the network cannot handle it. We will come back to
this point later.

We will start our study of congestion control by looking at a general model for dealing with it. Then
we will look at broad approaches to preventing it in the first place. After that, we will look at
various dynamic algorithms for coping with it once it has set in.

5.3.1 General Principles of Congestion Control

Many problems in complex systems, such as computer networks, can be viewed from a control
theory point of view. This approach leads to dividing all solutions into two groups: open loop and
closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make
sure it does not occur in the first place. Once the system is up and running, midcourse corrections
are not made.

Tools for doing open-loop control include deciding when to accept new traffic, deciding when to
discard packets and which ones, and making scheduling decisions at various points in the network.
All of these have in common the fact that they make decisions without regard to the current state
of the network.
In contrast, closed loop solutions are based on the concept of a feedback loop. This approach has three parts when applied to congestion control:

1. Monitor the system to detect when and where congestion occurs.
2. Pass this information to places where action can be taken.
3. Adjust system operation to correct the problem.

A variety of metrics can be used to monitor the subnet for congestion. Chief among these are the percentage of all packets discarded for lack of buffer space, the average queue lengths, the number of packets that time out and are retransmitted, the average packet delay, and the standard deviation of packet delay. In all cases, rising numbers indicate growing congestion.

The second step in the feedback loop is to transfer the information about the congestion from the point where it is detected to the point where something can be done about it. The obvious way is for the router detecting the congestion to send a packet to the traffic source or sources, announcing the problem. Of course, these extra packets increase the load at precisely the moment that more load is not needed, namely, when the subnet is congested.

However, other possibilities also exist. For example, a bit or field can be reserved in every packet for routers to fill in whenever congestion gets above some threshold level. When a router detects this congested state, it fills in the field in all outgoing packets, to warn the neighbors.

Still another approach is to have hosts or routers periodically send probe packets out to explicitly ask about congestion. This information can then be used to route traffic around problem areas. Some radio stations have helicopters flying around their cities to report on road congestion to make it possible for their mobile listeners to route their packets (cars) around hot spots.

In all feedback schemes, the hope is that knowledge of congestion will cause the hosts to take appropriate action to reduce the congestion. For a scheme to work correctly, the time scale must be adjusted carefully. If every time two packets arrive in a row, a router yells STOP and every time a router is idle for 20 µsec, it yells GO, the system will oscillate wildly and never converge. On the other hand, if it waits 30 minutes to make sure before saying anything, the congestion control mechanism will react too sluggishly to be of any real use. To work well, some kind of averaging is needed, but getting the time constant right is a nontrivial matter.

Many congestion control algorithms are known. To provide a way to organize them in a sensible way, Yang and Reddy (1995) have developed a taxonomy for congestion control algorithms. They begin by dividing all algorithms into open loop or closed loop, as described above. They further divide the open loop algorithms into ones that act at the source versus ones that act at the destination. The closed loop algorithms are also divided into two subcategories: explicit feedback versus implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithms, the source deduces the existence of congestion by making local observations, such as the time needed for acknowledgements to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. Two solutions come to mind: increase the resources or decrease the load. For example, the subnet may start using dial-up telephone lines to temporarily increase the bandwidth between certain points. On satellite systems, increasing transmission power often gives higher bandwidth. Splitting traffic over multiple routes instead of always using the best one may also effectively increase the bandwidth. Finally, spare routers that are normally used only as backups (to make the system fault tolerant) can be put on-line to give more capacity when serious
congestion appears.

However, sometimes it is not possible to increase the capacity, or it has already been increased to the limit. The only way then to beat back the congestion is to decrease the load. Several ways exist to reduce the load, including denying service to some users, degrading service to some or all users, and having users schedule their demands in a more predictable way.

Some of these methods, which we will study shortly, can best be applied to virtual circuits. For subnets that use virtual circuits internally, these methods can be used at the network layer. For datagram subnets, they can nevertheless sometimes be used on transport layer connections. In this chapter, we will focus on their use in the network layer. In the next one, we will see what can be done at the transport layer to manage congestion.

### 5.3.2 Congestion Prevention Policies

Let us begin our study of methods to control congestion by looking at open loop systems. These systems are designed to minimize congestion in the first place, rather than letting it happen and reacting after the fact. They try to achieve their goal by using appropriate policies at various levels. In Fig. 5-26 we see different data link, network, and transport policies that can affect congestion (Jain, 1990).

**Figure 5-26. Policies that affect congestion.**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Policies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport</td>
<td>• Retransmission policy</td>
</tr>
<tr>
<td></td>
<td>• Out-of-order caching policy</td>
</tr>
<tr>
<td></td>
<td>• Acknowledgement policy</td>
</tr>
<tr>
<td></td>
<td>• Flow control policy</td>
</tr>
<tr>
<td></td>
<td>• Timeout determination</td>
</tr>
<tr>
<td>Network</td>
<td>• Virtual circuits versus datagram inside the subnet</td>
</tr>
<tr>
<td></td>
<td>• Packet queueing and service policy</td>
</tr>
<tr>
<td></td>
<td>• Packet discard policy</td>
</tr>
<tr>
<td></td>
<td>• Routing algorithm</td>
</tr>
<tr>
<td></td>
<td>• Packet lifetime management</td>
</tr>
<tr>
<td>Data link</td>
<td>• Retransmission policy</td>
</tr>
<tr>
<td></td>
<td>• Out-of-order caching policy</td>
</tr>
<tr>
<td></td>
<td>• Acknowledgement policy</td>
</tr>
<tr>
<td></td>
<td>• Flow control policy</td>
</tr>
</tbody>
</table>

Let us start at the data link layer and work our way upward. The retransmission policy is concerned with how fast a sender times out and what it transmits upon timeout. A jumpy sender that times out quickly and retransmits all outstanding packets using go back n will put a heavier load on the system than will a leisurely sender that uses selective repeat. Closely related to this is the buffering policy. If receivers routinely discard all out-of-order packets, these packets will have to be transmitted again later, creating extra load. With respect to congestion control, selective repeat is clearly better than go back n.

Acknowledgement policy also affects congestion. If each packet is acknowledged immediately, the acknowledgement packets generate extra traffic. However, if acknowledgements are saved up to piggyback onto reverse traffic, extra timeouts and retransmissions may result. A tight flow control scheme (e.g., a small window) reduces the data rate and thus helps fight congestion.

At the network layer, the choice between using virtual circuits and using datagrams affects
congestion since many congestion control algorithms work only with virtual-circuit subnets. Packet queueing and service policy relates to whether routers have one queue per input line, one queue per output line, or both. It also relates to the order in which packets are processed (e.g., round robin or priority based). Discard policy is the rule telling which packet is dropped when there is no space. A good policy can help alleviate congestion and a bad one can make it worse.

A good routing algorithm can help avoid congestion by spreading the traffic over all the lines, whereas a bad one can send too much traffic over already congested lines. Finally, packet lifetime management deals with how long a packet may live before being discarded. If it is too long, lost packets may clog up the works for a long time, but if it is too short, packets may sometimes time out before reaching their destination, thus inducing retransmissions.

In the transport layer, the same issues occur as in the data link layer, but in addition, determining the timeout interval is harder because the transit time across the network is less predictable than the transit time over a wire between two routers. If the timeout interval is too short, extra packets will be sent unnecessarily. If it is too long, congestion will be reduced but the response time will suffer whenever a packet is lost.

5.3.3 Congestion Control in Virtual-Circuit Subnets

The congestion control methods described above are basically open loop: they try to prevent congestion from occurring in the first place, rather than dealing with it after the fact. In this section we will describe some approaches to dynamically controlling congestion in virtual-circuit subnets. In the next two, we will look at techniques that can be used in any subnet.

One technique that is widely used to keep congestion that has already started from getting worse is admission control. The idea is simple: once congestion has been signaled, no more virtual circuits are set up until the problem has gone away. Thus, attempts to set up new transport layer connections fail. Letting more people in just makes matters worse. While this approach is crude, it is simple and easy to carry out. In the telephone system, when a switch gets overloaded, it also practices admission control by not giving dial tones.

An alternative approach is to allow new virtual circuits but carefully route all new virtual circuits around problem areas. For example, consider the subnet of Fig. 5-27(a), in which two routers are congested, as indicated.

**Figure 5-27. (a) A congested subnet. (b) A redrawn subnet that eliminates the congestion. A virtual circuit from A to B is also shown.**

Suppose that a host attached to router A wants to set up a connection to a host attached to router B. Normally, this connection would pass through one of the congested routers. To avoid this...
situation, we can redraw the subnet as shown in Fig. 5-27(b), omitting the congested routers and all of their lines. The dashed line shows a possible route for the virtual circuit that avoids the congested routers.

Another strategy relating to virtual circuits is to negotiate an agreement between the host and subnet when a virtual circuit is set up. This agreement normally specifies the volume and shape of the traffic, quality of service required, and other parameters. To keep its part of the agreement, the subnet will typically reserve resources along the path when the circuit is set up. These resources can include table and buffer space in the routers and bandwidth on the lines. In this way, congestion is unlikely to occur on the new virtual circuits because all the necessary resources are guaranteed to be available.

This kind of reservation can be done all the time as standard operating procedure or only when the subnet is congested. A disadvantage of doing it all the time is that it tends to waste resources. If six virtual circuits that might use 1 Mbps all pass through the same physical 6-Mbps line, the line has to be marked as full, even though it may rarely happen that all six virtual circuits are transmitting full blast at the same time. Consequently, the price of the congestion control is unused (i.e., wasted) bandwidth in the normal case.

5.3.4 Congestion Control in Datagram Subnets

Let us now turn to some approaches that can be used in datagram subnets (and also in virtual-circuit subnets). Each router can easily monitor the utilization of its output lines and other resources. For example, it can associate with each line a real variable, \( u \), whose value, between 0.0 and 1.0, reflects the recent utilization of that line. To maintain a good estimate of \( u \), a sample of the instantaneous line utilization, \( f \) (either 0 or 1), can be made periodically and \( u \) updated according to

\[
    u_{\text{new}} = au_{\text{old}} + (1 - a)f
\]

where the constant \( a \) determines how fast the router forgets recent history.

Whenever \( u \) moves above the threshold, the output line enters a "warning" state. Each newly-arriving packet is checked to see if its output line is in warning state. If it is, some action is taken. The action taken can be one of several alternatives, which we will now discuss.

The Warning Bit

The old DECNET architecture signaled the warning state by setting a special bit in the packet's header. So does frame relay. When the packet arrived at its destination, the transport entity copied the bit into the next acknowledgement sent back to the source. The source then cut back on traffic.

As long as the router was in the warning state, it continued to set the warning bit, which meant that the source continued to get acknowledgements with it set. The source monitored the fraction of acknowledgements with the bit set and adjusted its transmission rate accordingly. As long as the warning bits continued to flow in, the source continued to decrease its transmission rate. When they slowed to a trickle, it increased its transmission rate. Note that since every router along the path could set the warning bit, traffic increased only when no router was in trouble.

Choke Packets

The previous congestion control algorithm is fairly subtle. It uses a roundabout means to tell the
source to slow down. Why not just tell it directly? In this approach, the router sends a **choke packet** back to the source host, giving it the destination found in the packet. The original packet is tagged (a header bit is turned on) so that it will not generate any more choke packets farther along the path and is then forwarded in the usual way.

When the source host gets the choke packet, it is required to reduce the traffic sent to the specified destination by $X$ percent. Since other packets aimed at the same destination are probably already under way and will generate yet more choke packets, the host should ignore choke packets referring to that destination for a fixed time interval. After that period has expired, the host listens for more choke packets for another interval. If one arrives, the line is still congested, so the host reduces the flow still more and begins ignoring choke packets again. If no choke packets arrive during the listening period, the host may increase the flow again. The feedback implicit in this protocol can help prevent congestion yet not throttle any flow unless trouble occurs.

Hosts can reduce traffic by adjusting their policy parameters, for example, their window size. Typically, the first choke packet causes the data rate to be reduced to 0.50 of its previous rate, the next one causes a reduction to 0.25, and so on. Increases are done in smaller increments to prevent congestion from reoccurring quickly.

Several variations on this congestion control algorithm have been proposed. For one, the routers can maintain several thresholds. Depending on which threshold has been crossed, the choke packet can contain a mild warning, a stern warning, or an ultimatum.

Another variation is to use queue lengths or buffer utilization instead of line utilization as the trigger signal. The same exponential weighting can be used with this metric as with $u$, of course.

**Hop-by-Hop Choke Packets**

At high speeds or over long distances, sending a choke packet to the source hosts does not work well because the reaction is so slow. Consider, for example, a host in San Francisco (router A in Fig. 5-28) that is sending traffic to a host in New York (router D in Fig. 5-28) at 155 Mbps. If the New York host begins to run out of buffers, it will take about 30 msec for a choke packet to get back to San Francisco to tell it to slow down. The choke packet propagation is shown as the second, third, and fourth steps in Fig. 5-28(a). In those 30 msec, another 4.6 megabits will have been sent. Even if the host in San Francisco completely shuts down immediately, the 4.6 megabits in the pipe will continue to pour in and have to be dealt with. Only in the seventh diagram in Fig. 5-28(a) will the New York router notice a slower flow.

**Figure 5-28.** (a) A choke packet that affects only the source. (b) A choke packet that affects each hop it passes through.
An alternative approach is to have the choke packet take effect at every hop it passes through, as shown in the sequence of Fig. 5-28(b). Here, as soon as the choke packet reaches $F$, $F$ is required to reduce the flow to $D$. Doing so will require $F$ to devote more buffers to the flow, since the source is still sending away at full blast, but it gives $D$ immediate relief, like a headache remedy in a television commercial. In the next step, the choke packet reaches $E$, which tells $E$ to reduce the flow to $F$. This action puts a greater demand on $E$’s buffers but gives $F$ immediate relief. Finally, the choke packet reaches $A$ and the flow genuinely slows down.

The net effect of this hop-by-hop scheme is to provide quick relief at the point of congestion at the price of using up more buffers upstream. In this way, congestion can be nipped in the bud without losing any packets. The idea is discussed in detail and simulation results are given in (Mishra and Kanakia, 1992).

---

(a) (b)
5.3.5 Load Shedding

When none of the above methods make the congestion disappear, routers can bring out the heavy artillery: load shedding. **Load shedding** is a fancy way of saying that when routers are being inundated by packets that they cannot handle, they just throw them away. The term comes from the world of electrical power generation, where it refers to the practice of utilities intentionally blacking out certain areas to save the entire grid from collapsing on hot summer days when the demand for electricity greatly exceeds the supply.

A router drowning in packets can just pick packets at random to drop, but usually it can do better than that. Which packet to discard may depend on the applications running. For file transfer, an old packet is worth more than a new one because dropping packet 6 and keeping packets 7 through 10 will cause a gap at the receiver that may force packets 6 through 10 to be retransmitted (if the receiver routinely discards out-of-order packets). In a 12-packet file, dropping 6 may require 7 through 12 to be retransmitted, whereas dropping 10 may require only 10 through 12 to be retransmitted. In contrast, for multimedia, a new packet is more important than an old one. The former policy (old is better than new) is often called **wine** and the latter (new is better than old) is often called **milk**.

A step above this in intelligence requires cooperation from the senders. For many applications, some packets are more important than others. For example, certain algorithms for compressing video periodically transmit an entire frame and then send subsequent frames as differences from the last full frame. In this case, dropping a packet that is part of a difference is preferable to dropping one that is part of a full frame. As another example, consider transmitting a document containing ASCII text and pictures. Losing a line of pixels in some image is far less damaging than losing a line of readable text.

To implement an intelligent discard policy, applications must mark their packets in priority classes to indicate how important they are. If they do this, then when packets have to be discarded, routers can first drop packets from the lowest class, then the next lowest class, and so on. Of course, unless there is some significant incentive to mark packets as anything other than VERY IMPORTANT— NEVER, EVER DISCARD, nobody will do it.

The incentive might be in the form of money, with the low-priority packets being cheaper to send than the high-priority ones. Alternatively, senders might be allowed to send high-priority packets under conditions of light load, but as the load increased they would be discarded, thus encouraging the users to stop sending them.

Another option is to allow hosts to exceed the limits specified in the agreement negotiated when the virtual circuit was set up (e.g., use a higher bandwidth than allowed), but subject to the condition that all excess traffic be marked as low priority. Such a strategy is actually not a bad idea, because it makes more efficient use of idle resources, allowing hosts to use them as long as nobody else is interested, but without establishing a right to them when times get tough.

**Random Early Detection**

It is well known that dealing with congestion after it is first detected is more effective than letting it gum up the works and then trying to deal with it. This observation leads to the idea of discarding packets before all the buffer space is really exhausted. A popular algorithm for doing this is called **RED (Random Early Detection)** (Floyd and Jacobson, 1993). In some transport protocols (including TCP), the response to lost packets is for the source to slow down. The reasoning behind this logic is that TCP was designed for wired networks and wired networks are very reliable, so lost packets are mostly due to buffer overruns rather than transmission errors. This fact can be exploited to help reduce congestion.
By having routers drop packets before the situation has become hopeless (hence the "early" in the name), the idea is that there is time for action to be taken before it is too late. To determine when to start discarding, routers maintain a running average of their queue lengths. When the average queue length on some line exceeds a threshold, the line is said to be congested and action is taken.

Since the router probably cannot tell which source is causing most of the trouble, picking a packet at random from the queue that triggered the action is probably as good as it can do.

How should the router tell the source about the problem? One way is to send it a choke packet, as we have described. A problem with that approach is that it puts even more load on the already congested network. A different strategy is to just discard the selected packet and not report it. The source will eventually notice the lack of acknowledgement and take action. Since it knows that lost packets are generally caused by congestion and discards, it will respond by slowing down instead of trying harder. This implicit form of feedback only works when sources respond to lost packets by slowing down their transmission rate. In wireless networks, where most losses are due to noise on the air link, this approach cannot be used.

5.3.6 Jitter Control

For applications such as audio and video streaming, it does not matter much if the packets take 20 msec or 30 msec to be delivered, as long as the transit time is constant. The variation (i.e., standard deviation) in the packet arrival times is called jitter. High jitter, for example, having some packets taking 20 msec and others taking 30 msec to arrive will give an uneven quality to the sound or movie. Jitter is illustrated in Fig. 5-29. In contrast, an agreement that 99 percent of the packets be delivered with a delay in the range of 24.5 msec to 25.5 msec might be acceptable.

**Figure 5-29. (a) High jitter. (b) Low jitter.**

The range chosen must be feasible, of course. It must take into account the speed-of-light transit time and the minimum delay through the routers and perhaps leave a little slack for some inevitable delays.

The jitter can be bounded by computing the expected transit time for each hop along the path. When a packet arrives at a router, the router checks to see how much the packet is behind or ahead of its schedule. This information is stored in the packet and updated at each hop. If the packet is ahead of schedule, it is held just long enough to get it back on schedule. If it is behind schedule, the router tries to get it out the door quickly.
In fact, the algorithm for determining which of several packets competing for an output line should go next can always choose the packet furthest behind in its schedule. In this way, packets that are ahead of schedule get slowed down and packets that are behind schedule get speeded up, in both cases reducing the amount of jitter.

In some applications, such as video on demand, jitter can be eliminated by buffering at the receiver and then fetching data for display from the buffer instead of from the network in real time. However, for other applications, especially those that require real-time interaction between people such as Internet telephony and videoconferencing, the delay inherent in buffering is not acceptable.

Congestion control is an active area of research. The state-of-the-art is summarized in (Gevros et al., 2001).

5.4 Quality of Service

The techniques we looked at in the previous sections are designed to reduce congestion and improve network performance. However, with the growth of multimedia networking, often these ad hoc measures are not enough. Serious attempts at guaranteeing quality of service through network and protocol design are needed. In the following sections we will continue our study of network performance, but now with a sharper focus on ways to provide a quality of service matched to application needs. It should be stated at the start, however, that many of these ideas are in flux and are subject to change.

5.4.1 Requirements

A stream of packets from a source to a destination is called a flow. In a connection-oriented network, all the packets belonging to a flow follow the same route; in a connectionless network, they may follow different routes. The needs of each flow can be characterized by four primary parameters: reliability, delay, jitter, and bandwidth. Together these determine the QoS (Quality of Service) the flow requires. Several common applications and the stringency of their requirements are listed in Fig. 5-30.

Figure 5-30. How stringent the quality-of-service requirements are.

<table>
<thead>
<tr>
<th>Application</th>
<th>Reliability</th>
<th>Delay</th>
<th>Jitter</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-mail</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>File transfer</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>Web access</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>Remote login</td>
<td>High</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Audio on demand</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Video on demand</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Telephony</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Videoconferencing</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

The first four applications have stringent requirements on reliability. No bits may be delivered incorrectly. This goal is usually achieved by checksumming each packet and verifying the checksum at the destination. If a packet is damaged in transit, it is not acknowledged and will be retransmitted eventually. This strategy gives high reliability. The four final (audio/video)
applications can tolerate errors, so no checksums are computed or verified.

File transfer applications, including e-mail and video, are not delay sensitive. If all packets are delayed uniformly by a few seconds, no harm is done. Interactive applications, such as Web surfing and remote login, are more delay sensitive. Real-time applications, such as telephony and videoconferencing have strict delay requirements. If all the words in a telephone call are each delayed by exactly 2.000 seconds, the users will find the connection unacceptable. On the other hand, playing audio or video files from a server does not require low delay.

The first three applications are not sensitive to the packets arriving with irregular time intervals between them. Remote login is somewhat sensitive to that, since characters on the screen will appear in little bursts if the connection suffers much jitter. Video and especially audio are extremely sensitive to jitter. If a user is watching a video over the network and the frames are all delayed by exactly 2.000 seconds, no harm is done. But if the transmission time varies randomly between 1 and 2 seconds, the result will be terrible. For audio, a jitter of even a few milliseconds is clearly audible.

Finally, the applications differ in their bandwidth needs, with e-mail and remote login not needing much, but video in all forms needing a great deal.

ATM networks classify flows in four broad categories with respect to their QoS demands as follows:

1. Constant bit rate (e.g., telephony).
2. Real-time variable bit rate (e.g., compressed videoconferencing).
3. Non-real-time variable bit rate (e.g., watching a movie over the Internet).
4. Available bit rate (e.g., file transfer).

These categories are also useful for other purposes and other networks. Constant bit rate is an attempt to simulate a wire by providing a uniform bandwidth and a uniform delay. Variable bit rate occurs when video is compressed, some frames compressing more than others. Thus, sending a frame with a lot of detail in it may require sending many bits whereas sending a shot of a white wall may compress extremely well. Available bit rate is for applications, such as e-mail, that are not sensitive to delay or jitter.

5.4.2 Techniques for Achieving Good Quality of Service

Now that we know something about QoS requirements, how do we achieve them? Well, to start with, there is no magic bullet. No single technique provides efficient, dependable QoS in an optimum way. Instead, a variety of techniques have been developed, with practical solutions often combining multiple techniques. We will now examine some of the techniques system designers use to achieve QoS.

Overprovisioning

An easy solution is to provide so much router capacity, buffer space, and bandwidth that the packets just fly through easily. The trouble with this solution is that it is expensive. As time goes on and designers have a better idea of how much is enough, this technique may even become practical. To some extent, the telephone system is overprovisioned. It is rare to pick up a telephone and not get a dial tone instantly. There is simply so much capacity available there that demand can always be met.
**Buffering**

Flows can be buffered on the receiving side before being delivered. Buffering them does not affect the reliability or bandwidth, and increases the delay, but it smooths out the jitter. For audio and video on demand, jitter is the main problem, so this technique helps a lot.

We saw the difference between high jitter and low jitter in Fig. 5-29. In Fig. 5-31 we see a stream of packets being delivered with substantial jitter. Packet 1 is sent from the server at $t = 0$ sec and arrives at the client at $t = 1$ sec. Packet 2 undergoes more delay and takes 2 sec to arrive. As the packets arrive, they are buffered on the client machine.

**Figure 5-31. Smoothing the output stream by buffering packets.**

![Packet arrivals and buffer](image)

At $t = 10$ sec, playback begins. At this time, packets 1 through 6 have been buffered so that they can be removed from the buffer at uniform intervals for smooth play. Unfortunately, packet 8 has been delayed so much that it is not available when its play slot comes up, so playback must stop until it arrives, creating an annoying gap in the music or movie. This problem can be alleviated by delaying the starting time even more, although doing so also requires a larger buffer. Commercial Web sites that contain streaming audio or video all use players that buffer for about 10 seconds before starting to play.

**Traffic Shaping**

In the above example, the source outputs the packets with a uniform spacing between them, but in other cases, they may be emitted irregularly, which may cause congestion to occur in the network. Nonuniform output is common if the server is handling many streams at once, and it also allows other actions, such as fast forward and rewind, user authentication, and so on. Also, the approach we used here (buffering) is not always possible, for example, with videoconferencing. However, if something could be done to make the server (and hosts in general) transmit at a uniform rate, quality of service would be better. We will now examine a technique, **traffic shaping**, which smooths out the traffic on the server side, rather than on the client side.

Traffic shaping is about regulating the average rate (and burstiness) of data transmission. In contrast, the sliding window protocols we studied earlier limit the amount of data in transit at once, not the rate at which it is sent. When a connection is set up, the user and the subnet (i.e., the customer and the carrier) agree on a certain traffic pattern (i.e., shape) for that circuit. Sometimes this is called a **service level agreement**. As long as the customer fulfills her part of the bargain and only sends packets according to the agreed-on contract, the carrier promises to deliver them all in a timely fashion. Traffic shaping reduces congestion and thus helps the carrier live up to its promise. Such agreements are not so important for file transfers but are of great importance for real-time data, such as audio and video connections, which have stringent quality-of-service requirements.
In effect, with traffic shaping the customer says to the carrier: My transmission pattern will look like this; can you handle it? If the carrier agrees, the issue arises of how the carrier can tell if the customer is following the agreement and what to do if the customer is not. Monitoring a traffic flow is called **traffic policing**. Agreeing to a traffic shape and policing it afterward are easier with virtual-circuit subnets than with datagram subnets. However, even with datagram subnets, the same ideas can be applied to transport layer connections.

**The Leaky Bucket Algorithm**

Imagine a bucket with a small hole in the bottom, as illustrated in Fig. 5-32(a). No matter the rate at which water enters the bucket, the outflow is at a constant rate, $\rho$, when there is any water in the bucket and zero when the bucket is empty. Also, once the bucket is full, any additional water entering it spills over the sides and is lost (i.e., does not appear in the output stream under the hole).

**Figure 5-32. (a) A leaky bucket with water. (b) A leaky bucket with packets.**

The same idea can be applied to packets, as shown in Fig. 5-32(b). Conceptually, each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more processes within the host try to send a packet when the maximum number is already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulated by the host operating system. It was first proposed by Turner (1986) and is called the **leaky bucket algorithm**. In fact, it is nothing other than a single-server queueing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. Again, this can be enforced by the interface card or by the operating system. This mechanism turns an uneven flow of packets from the user processes inside the host into an even flow of packets onto the network, smoothing out bursts and greatly reducing the chances of congestion.

When the packets are all the same size (e.g., ATM cells), this algorithm can be used as described. However, when variable-sized packets are being used, it is often better to allow a fixed number of...
bytes per tick, rather than just one packet. Thus, if the rule is 1024 bytes per tick, a single 1024-byte packet can be admitted on a tick, two 512-byte packets, four 256-byte packets, and so on. If the residual byte count is too low, the next packet must wait until the next tick.

Implementing the original leaky bucket algorithm is easy. The leaky bucket consists of a finite queue. When a packet arrives, if there is room on the queue it is appended to the queue; otherwise, it is discarded. At every clock tick, one packet is transmitted (unless the queue is empty).

The byte-counting leaky bucket is implemented almost the same way. At each tick, a counter is initialized to $n$. If the first packet on the queue has fewer bytes than the current value of the counter, it is transmitted, and the counter is decremented by that number of bytes. Additional packets may also be sent, as long as the counter is high enough. When the counter drops below the length of the next packet on the queue, transmission stops until the next tick, at which time the residual byte count is reset and the flow can continue.

As an example of a leaky bucket, imagine that a computer can produce data at 25 million bytes/sec (200 Mbps) and that the network also runs at this speed. However, the routers can accept this data rate only for short intervals (basically, until their buffers fill up). For long intervals, they work best at rates not exceeding 2 million bytes/sec. Now suppose data comes in 1-million-byte bursts, one 40-msec burst every second. To reduce the average rate to 2 MB/sec, we could use a leaky bucket with $\rho=2$ MB/sec and a capacity, $C$, of 1 MB. This means that bursts of up to 1 MB can be handled without data loss and that such bursts are spread out over 500 msec, no matter how fast they come in.

In Fig. 5-33(a) we see the input to the leaky bucket running at 25 MB/sec for 40 msec. In Fig. 5-33(b) we see the output draining out at a uniform rate of 2 MB/sec for 500 msec.

**Figure 5-33.** (a) Input to a leaky bucket. (b) Output from a leaky bucket. Output from a token bucket with capacities of (c) 250 KB, (d) 500 KB, and (e) 750 KB. (f) Output from a 500KB token bucket feeding a 10-MB/sec leaky bucket.
The Token Bucket Algorithm

The leaky bucket algorithm enforces a rigid output pattern at the average rate, no matter how bursty the traffic is. For many applications, it is better to allow the output to speed up somewhat when large bursts arrive, so a more flexible algorithm is needed, preferably one that never loses data. One such algorithm is the token bucket algorithm. In this algorithm, the leaky bucket holds tokens, generated by a clock at the rate of one token every $\Delta T$ sec. In Fig. 5-34(a) we see a bucket holding three tokens, with five packets waiting to be transmitted. For a packet to be transmitted, it must capture and destroy one token. In Fig. 5-34(b) we see that three of the five packets have gotten through, but the other two are stuck waiting for two more tokens to be generated.

**Figure 5-34. The token bucket algorithm. (a) Before. (b) After.**
The token bucket algorithm provides a different kind of traffic shaping than that of the leaky bucket algorithm. The leaky bucket algorithm does not allow idle hosts to save up permission to send large bursts later. The token bucket algorithm does allow saving, up to the maximum size of the bucket, \( n \). This property means that bursts of up to \( n \) packets can be sent at once, allowing some burstiness in the output stream and giving faster response to sudden bursts of input.

Another difference between the two algorithms is that the token bucket algorithm throws away tokens (i.e., transmission capacity) when the bucket fills up but never discards packets. In contrast, the leaky bucket algorithm discards packets when the bucket fills up.

Here, too, a minor variant is possible, in which each token represents the right to send not one packet, but \( k \) bytes. A packet can only be transmitted if enough tokens are available to cover its length in bytes. Fractional tokens are kept for future use.

The leaky bucket and token bucket algorithms can also be used to smooth traffic between routers, as well as to regulate host output as in our examples. However, one clear difference is that a token bucket regulating a host can make the host stop sending when the rules say it must. Telling a router to stop sending while its input keeps pouring in may result in lost data.

The implementation of the basic token bucket algorithm is just a variable that counts tokens. The counter is incremented by one every \( \Delta T \) and decremented by one whenever a packet is sent. When the counter hits zero, no packets may be sent. In the byte-count variant, the counter is incremented by \( k \) bytes every \( \Delta T \) and decremented by the length of each packet sent.

Essentially what the token bucket does is allow bursts, but up to a regulated maximum length. Look at Fig. 5-33(c) for example. Here we have a token bucket with a capacity of 250 KB. Tokens arrive at a rate allowing output at 2 MB/sec. Assuming the token bucket is full when the 1-MB burst arrives, the bucket can drain at the full 25 MB/sec for about 11 msec. Then it has to cut back to 2 MB/sec until the entire input burst has been sent.

Calculating the length of the maximum rate burst is slightly tricky. It is not just 1 MB divided by 25 MB/sec because while the burst is being output, more tokens arrive. If we call the burst length \( S \) sec, the token bucket capacity \( C \) bytes, the token arrival rate \( \rho \) bytes/sec, and the maximum output rate \( M \) bytes/sec, we see that an output burst contains a maximum of \( C + \rho S \) bytes. We
also know that the number of bytes in a maximum-speed burst of length $S$ seconds is $MS$. Hence we have

$$C + \rho S = MS$$

We can solve this equation to get $S = C/(M - \rho)$. For our parameters of $C = 250$ KB, $M = 25$ MB/sec, and $\rho = 2$ MB/sec, we get a burst time of about 11 msec. Figure 5-33(d) and Fig. 5-33(e) show the token bucket for capacities of 500 KB and 750 KB, respectively.

A potential problem with the token bucket algorithm is that it allows large bursts again, even though the maximum burst interval can be regulated by careful selection of $\rho$ and $M$. It is frequently desirable to reduce the peak rate, but without going back to the low value of the original leaky bucket.

One way to get smoother traffic is to insert a leaky bucket after the token bucket. The rate of the leaky bucket should be higher than the token bucket's $\rho$ but lower than the maximum rate of the network. Figure 5-33(f) shows the output for a 500-KB token bucket followed by a 10-MB/sec leaky bucket.

Policing all these schemes can be a bit tricky. Essentially, the network has to simulate the algorithm and make sure that no more packets or bytes are being sent than are permitted. Nevertheless, these tools provide ways to shape the network traffic into more manageable forms to assist meeting quality-of-service requirements.

**Resource Reservation**

Being able to regulate the shape of the offered traffic is a good start to guaranteeing the quality of service. However, effectively using this information implicitly means requiring all the packets of a flow to follow the same route. Spraying them over routers at random makes it hard to guarantee anything. As a consequence, something similar to a virtual circuit has to be set up from the source to the destination, and all the packets that belong to the flow must follow this route.

Once we have a specific route for a flow, it becomes possible to reserve resources along that route to make sure the needed capacity is available. Three different kinds of resources can potentially be reserved:

1. Bandwidth.
2. Buffer space.
3. CPU cycles.

The first one, bandwidth, is the most obvious. If a flow requires 1 Mbps and the outgoing line has a capacity of 2 Mbps, trying to direct three flows through that line is not going to work. Thus, reserving bandwidth means not oversubscribing any output line.

A second resource that is often in short supply is buffer space. When a packet arrives, it is usually deposited on the network interface card by the hardware itself. The router software then has to copy it to a buffer in RAM and queue that buffer for transmission on the chosen outgoing line. If no buffer is available, the packet has to be discarded since there is no place to put it. For a good quality of service, some buffers can be reserved for a specific flow so that flow does not have to compete for buffers with other flows. There will always be a buffer available when the flow needs
one, up to some maximum.

Finally, CPU cycles are also a scarce resource. It takes router CPU time to process a packet, so a router can process only a certain number of packets per second. Making sure that the CPU is not overloaded is needed to ensure timely processing of each packet.

At first glance, it might appear that if it takes, say, 1 µsec to process a packet, a router can process 1 million packets/sec. This observation is not true because there will always be idle periods due to statistical fluctuations in the load. If the CPU needs every single cycle to get its work done, losing even a few cycles due to occasional idleness creates a backlog it can never get rid of.

However, even with a load slightly below the theoretical capacity, queues can build up and delays can occur. Consider a situation in which packets arrive at random with a mean arrival rate of λ packets/sec. The CPU time required by each one is also random, with a mean processing capacity of µ packets/sec. Under the assumption that both the arrival and service distributions are Poisson distributions, it can be proven using queueing theory that the mean delay experienced by a packet, T, is

\[ T = \frac{1}{\mu} \times \frac{1}{1 - \lambda/\mu} = \frac{1}{\mu} \times \frac{1}{1 - \rho} \]

where ρ = λ/µ is the CPU utilization. The first factor, 1/µ, is what the service time would be in the absence of competition. The second factor is the slowdown due to competition with other flows. For example, if λ = 950,000 packets/sec and µ = 1,000,000 packets/sec, then ρ = 0.95 and the mean delay experienced by each packet will be 20 µsec instead of 1 µsec. This time accounts for both the queueing time and the service time, as can be seen when the load is very low (λ/µ ≈ 0). If there are, say, 30 routers along the flow's route, queueing delay alone will account for 600 µsec of delay.

**Admission Control**

Now we are at the point where the incoming traffic from some flow is well shaped and can potentially follow a single route in which capacity can be reserved in advance on the routers along the path. When such a flow is offered to a router, it has to decide, based on its capacity and how many commitments it has already made for other flows, whether to admit or reject the flow.

The decision to accept or reject a flow is not a simple matter of comparing the (bandwidth, buffers, cycles) requested by the flow with the router's excess capacity in those three dimensions. It is a little more complicated than that. To start with, although some applications may know about their bandwidth requirements, few know about buffers or CPU cycles, so at the minimum, a different way is needed to describe flows. Next, some applications are far more tolerant of an occasional missed deadline than others. Finally, some applications may be willing to haggle about the flow parameters and others may not. For example, a movie viewer that normally runs at 30 frames/sec may be willing to drop back to 25 frames/sec if there is not enough free bandwidth to support 30 frames/sec. Similarly, the number of pixels per frame, audio bandwidth, and other properties may be adjustable.

Because many parties may be involved in the flow negotiation (the sender, the receiver, and all the routers along the path between them), flows must be described accurately in terms of specific parameters that can be negotiated. A set of such parameters is called a flow specification. Typically, the sender (e.g., the video server) produces a flow specification proposing the parameters it would like to use. As the specification propagates along the route, each router examines it and modifies the parameters as need be. The modifications can only reduce the flow,
not increase it (e.g., a lower data rate, not a higher one). When it gets to the other end, the
parameters can be established.

As an example of what can be in a flow specification, consider the example of Fig. 5-35, which is
based on RFCs 2210 and 2211. It has five parameters, the first of which, the *Token bucket rate*, is
the number of bytes per second that are put into the bucket. This is the maximum sustained rate
the sender may transmit, averaged over a long time interval.

**Figure 5-35. An example flow specification.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token bucket rate</td>
<td>Bytes/sec</td>
</tr>
<tr>
<td>Token bucket size</td>
<td>Bytes</td>
</tr>
<tr>
<td>Peak data rate</td>
<td>Bytes/sec</td>
</tr>
<tr>
<td>Minimum packet size</td>
<td>Bytes</td>
</tr>
<tr>
<td>Maximum packet size</td>
<td>Bytes</td>
</tr>
</tbody>
</table>

The second parameter is the size of the bucket in bytes. If, for example, the *Token bucket rate* is 1
Mbps and the *Token bucket size* is 500 KB, the bucket can fill continuously for 4 sec before it fills
up (in the absence of any transmissions). Any tokens sent after that are lost.

The third parameter, the *Peak data rate*, is the maximum tolerated transmission rate, even for
brief time intervals. The sender must never exceed this rate.

The last two parameters specify the minimum and maximum packet sizes, including the transport
and network layer headers (e.g., TCP and IP). The minimum size is important because processing
each packet takes some fixed time, no matter how short. A router may be prepared to handle
10,000 packets/sec of 1 KB each, but not be prepared to handle 100,000 packets/sec of 50 bytes
each, even though this represents a lower data rate. The maximum packet size is important due to
internal network limitations that may not be exceeded. For example, if part of the path goes over
an Ethernet, the maximum packet size will be restricted to no more than 1500 bytes no matter
what the rest of the network can handle.

An interesting question is how a router turns a flow specification into a set of specific resource
reservations. That mapping is implementation specific and is not standardized. Suppose that a
router can process 100,000 packets/sec. If it is offered a flow of 1 MB/sec with minimum and
maximum packet sizes of 512 bytes, the router can calculate that it might get 2048 packets/sec
from that flow. In that case, it must reserve 2% of its CPU for that flow, preferably more to avoid
long queueing delays. If a router's policy is never to allocate more than 50% of its CPU (which
implies a factor of two delay, and it is already 49% full, then this flow must be rejected. Similar
calculations are needed for the other resources.

The tighter the flow specification, the more useful it is to the routers. If a flow specification states
that it needs a *Token bucket rate* of 5 MB/sec but packets can vary from 50 bytes to 1500 bytes,
then the packet rate will vary from about 3500 packets/sec to 105,000 packets/sec. The router
may panic at the latter number and reject the flow, whereas with a minimum packet size of 1000
bytes, the 5 MB/sec flow might have been accepted.

**Proportional Routing**

Most routing algorithms try to find the best path for each destination and send all traffic to that
destination over the best path. A different approach that has been proposed to provide a higher
quality of service is to split the traffic for each destination over multiple paths. Since routers generally do not have a complete overview of network-wide traffic, the only feasible way to split traffic over multiple routes is to use locally-available information. A simple method is to divide the traffic equally or in proportion to the capacity of the outgoing links. However, more sophisticated algorithms are also available (Nelakuditi and Zhang, 2002).

**Packet Scheduling**

If a router is handling multiple flows, there is a danger that one flow will hog too much of its capacity and starve all the other flows. Processing packets in the order of their arrival means that an aggressive sender can capture most of the capacity of the routers its packets traverse, reducing the quality of service for others. To thwart such attempts, various packet scheduling algorithms have been devised (Bhatti and Crowcroft, 2000).

One of the first ones was the **fair queueing** algorithm (Nagle, 1987). The essence of the algorithm is that routers have separate queues for each output line, one for each flow. When a line becomes idle, the router scans the queues round robin, taking the first packet on the next queue. In this way, with \( n \) hosts competing for a given output line, each host gets to send one out of every \( n \) packets. Sending more packets will not improve this fraction.

Although a start, the algorithm has a problem: it gives more bandwidth to hosts that use large packets than to hosts that use small packets. Demers et al. (1990) suggested an improvement in which the round robin is done in such a way as to simulate a byte-by-byte round robin, instead of a packet-by-packet round robin. In effect, it scans the queues repeatedly, byte-for-byte, until it finds the tick on which each packet will be finished. The packets are then sorted in order of their finishing and sent in that order. The algorithm is illustrated in Fig. 5-36.

**Figure 5-36. (a) A router with five packets queued for line O. (b) Finishing times for the five packets.**

In **Fig. 5-36(a)** we see packets of length 2 to 6 bytes. At (virtual) clock tick 1, the first byte of the packet on line \( A \) is sent. Then goes the first byte of the packet on line \( B \), and so on. The first packet to finish is \( C \), after eight ticks. The sorted order is given in **Fig. 5-36(b)**. In the absence of new arrivals, the packets will be sent in the order listed, from \( C \) to \( A \).

One problem with this algorithm is that it gives all hosts the same priority. In many situations, it is desirable to give video servers more bandwidth than regular file servers so that they can be given two or more bytes per tick. This modified algorithm is called **weighted fair queueing** and is widely used. Sometimes the weight is equal to the number of flows coming out of a machine, so each process gets equal bandwidth. An efficient implementation of the algorithm is discussed in (Shreedhar and Varghese, 1995). Increasingly, the actual forwarding of packets through a router or switch is being done in hardware (Elhanany et al., 2001).
5.4.3 Integrated Services

Between 1995 and 1997, IETF put a lot of effort into devising an architecture for streaming multimedia. This work resulted in over two dozen RFCs, starting with RFCs 2205–2210. The generic name for this work is flow-based algorithms or integrated services. It was aimed at both unicast and multicast applications. An example of the former is a single user streaming a video clip from a news site. An example of the latter is a collection of digital television stations broadcasting their programs as streams of IP packets to many receivers at various locations. Below we will concentrate on multicast, since unicast is a special case of multicast.

In many multicast applications, groups can change membership dynamically, for example, as people enter a video conference and then get bored and switch to a soap opera or the croquet channel. Under these conditions, the approach of having the senders reserve bandwidth in advance does not work well, since it would require each sender to track all entries and exits of its audience. For a system designed to transmit television with millions of subscribers, it would not work at all.

RSVP—The Resource reSerVation Protocol

The main IETF protocol for the integrated services architecture is RSVP. It is described in RFC 2205 and others. This protocol is used for making the reservations; other protocols are used for sending the data. RSVP allows multiple senders to transmit to multiple groups of receivers, permits individual receivers to switch channels freely, and optimizes bandwidth use while at the same time eliminating congestion.

In its simplest form, the protocol uses multicast routing using spanning trees, as discussed earlier. Each group is assigned a group address. To send to a group, a sender puts the group's address in its packets. The standard multicast routing algorithm then builds a spanning tree covering all group members. The routing algorithm is not part of RSVP. The only difference from normal multicasting is a little extra information that is multicast to the group periodically to tell the routers along the tree to maintain certain data structures in their memories.

As an example, consider the network of Fig. 5-37(a). Hosts 1 and 2 are multicast senders, and hosts 3, 4, and 5 are multicast receivers. In this example, the senders and receivers are disjoint, but in general, the two sets may overlap. The multicast trees for hosts 1 and 2 are shown in Fig. 5-37(b) and Fig. 5-37(c), respectively.

Figure 5-37. (a) A network. (b) The multicast spanning tree for host 1. (c) The multicast spanning tree for host 2.
To get better reception and eliminate congestion, any of the receivers in a group can send a reservation message up the tree to the sender. The message is propagated using the reverse path forwarding algorithm discussed earlier. At each hop, the router notes the reservation and reserves the necessary bandwidth. If insufficient bandwidth is available, it reports back failure. By the time the message gets back to the source, bandwidth has been reserved all the way from the sender to the receiver making the reservation request along the spanning tree.

An example of such a reservation is shown in Fig. 5-38(a). Here host 3 has requested a channel to host 1. Once it has been established, packets can flow from 1 to 3 without congestion. Now consider what happens if host 3 next reserves a channel to the other sender, host 2, so the user can watch two television programs at once. A second path is reserved, as illustrated in Fig. 5-38(b). Note that two separate channels are needed from host 3 to router E because two independent streams are being transmitted.

**Figure 5-38.** (a) Host 3 requests a channel to host 1. (b) Host 3 then requests a second channel, to host 2. (c) Host 5 requests a channel to host 1.
Finally, in Fig. 5-38(c), host 5 decides to watch the program being transmitted by host 1 and also makes a reservation. First, dedicated bandwidth is reserved as far as router H. However, this router sees that it already has a feed from host 1, so if the necessary bandwidth has already been reserved, it does not have to reserve any more. Note that hosts 3 and 5 might have asked for different amounts of bandwidth (e.g., 3 has a black-and-white television set, so it does not want the color information), so the capacity reserved must be large enough to satisfy the greediest receiver.

When making a reservation, a receiver can (optionally) specify one or more sources that it wants to receive from. It can also specify whether these choices are fixed for the duration of the reservation or whether the receiver wants to keep open the option of changing sources later. The routers use this information to optimize bandwidth planning. In particular, two receivers are only set up to share a path if they both agree not to change sources later on.

The reason for this strategy in the fully dynamic case is that reserved bandwidth is decoupled from the choice of source. Once a receiver has reserved bandwidth, it can switch to another source and keep that portion of the existing path that is valid for the new source. If host 2 is transmitting several video streams, for example, host 3 may switch between them at will without changing its reservation: the routers do not care what program the receiver is watching.

5.4.4 Differentiated Services

Flow-based algorithms have the potential to offer good quality of service to one or more flows because they reserve whatever resources are needed along the route. However, they also have a downside. They require an advance setup to establish each flow, something that does not scale well when there are thousands or millions of flows. Also, they maintain internal per-flow state in the routers, making them vulnerable to router crashes. Finally, the changes required to the router code are substantial and involve complex router-to-router exchanges for setting up the flows. As a consequence, few implementations of RSVP or anything like it exist yet.

For these reasons, IETF has also devised a simpler approach to quality of service, one that can be largely implemented locally in each router without advance setup and without having the whole path involved. This approach is known as class-based (as opposed to flow-based) quality of service. IETF has standardized an architecture for it, called differentiated services, which is described in RFCs 2474, 2475, and numerous others. We will now describe it.

Differentiated services (DS) can be offered by a set of routers forming an administrative domain (e.g., an ISP or a telco). The administration defines a set of service classes with corresponding forwarding rules. If a customer signs up for DS, customer packets entering the domain may carry a Type of Service field in them, with better service provided to some classes (e.g., premium service) than to others. Traffic within a class may be required to conform to some specific shape, such as a leaky bucket with some specified drain rate. An operator with a nose for business might charge extra for each premium packet transported or might allow up to $N$ premium packets per month for a fixed additional monthly fee. Note that this scheme requires no advance setup, no resource reservation, and no time-consuming end-to-end negotiation for each flow, as with integrated services. This makes DS relatively easy to implement.

Class-based service also occurs in other industries. For example, package delivery companies often offer overnight, two-day, and three-day service. Airlines offer first class, business class, and cattle class service. Long-distance trains often have multiple service classes. Even the Paris subway has two service classes. For packets, the classes may differ in terms of delay, jitter, and probability of being discarded in the event of congestion, among other possibilities (but probably not roomier Ethernet frames).
To make the difference between flow-based quality of service and class-based quality of service clearer, consider an example: Internet telephony. With a flow-based scheme, each telephone call gets its own resources and guarantees. With a class-based scheme, all the telephone calls together get the resources reserved for the class telephony. These resources cannot be taken away by packets from the file transfer class or other classes, but no telephone call gets any private resources reserved for it alone.

**Expedited Forwarding**

The choice of service classes is up to each operator, but since packets are often forwarded between subnets run by different operators, IETF is working on defining network-independent service classes. The simplest class is expedited forwarding, so let us start with that one. It is described in RFC 3246.

The idea behind expedited forwarding is very simple. Two classes of service are available: regular and expedited. The vast majority of the traffic is expected to be regular, but a small fraction of the packets are expedited. The expedited packets should be able to transit the subnet as though no other packets were present. A symbolic representation of this "two-tube" system is given in Fig. 5-39. Note that there is still just one physical line. The two logical pipes shown in the figure represent a way to reserve bandwidth, not a second physical line.

**Figure 5-39. Expedited packets experience a traffic-free network.**

One way to implement this strategy is to program the routers to have two output queues for each outgoing line, one for expedited packets and one for regular packets. When a packet arrives, it is queued accordingly. Packet scheduling should use something like weighted fair queueing. For example, if 10% of the traffic is expedited and 90% is regular, 20% of the bandwidth could be dedicated to expedited traffic and the rest to regular traffic. Doing so would give the expedited traffic twice as much bandwidth as it needs in order to provide low delay for it. This allocation can be achieved by transmitting one expedited packet for every four regular packets (assuming the size distribution for both classes is similar). In this way, it is hoped that expedited packets see an unloaded subnet, even when there is, in fact, a heavy load.

**Assured Forwarding**

A somewhat more elaborate scheme for managing the service classes is called assured forwarding. It is described in RFC 2597. It specifies that there shall be four priority classes, each class having its own resources. In addition, it defines three discard probabilities for packets that are undergoing congestion: low, medium, and high. Taken together, these two factors define 12 service classes.

**Figure 5-40** shows one way packets might be processed under assured forwarding. Step 1 is to classify the packets into one of the four priority classes. This step might be done on the sending
host (as shown in the figure) or in the ingress (first) router. The advantage of doing classification on the sending host is that more information is available about which packets belong to which flows there.

Figure 5-40. A possible implementation of the data flow for assured forwarding.

Step 2 is to mark the packets according to their class. A header field is needed for this purpose. Fortunately, an 8-bit *Type of service* field is available in the IP header, as we will see shortly. RFC 2597 specifies that six of these bits are to be used for the service class, leaving coding room for historical service classes and future ones.

Step 3 is to pass the packets through a shaper/dropper filter that may delay or drop some of them to shape the four streams into acceptable forms, for example, by using leaky or token buckets. If there are too many packets, some of them may be discarded here, by discard category. More elaborate schemes involving metering or feedback are also possible.

In this example, these three steps are performed on the sending host, so the output stream is now fed into the ingress router. It is worth noting that these steps may be performed by special networking software or even the operating system, to avoid having to change existing applications.

5.4.5 Label Switching and MPLS

While IETF was working out integrated services and differentiated services, several router vendors were working on better forwarding methods. This work focused on adding a label in front of each packet and doing the routing based on the label rather than on the destination address. Making the label an index into an internal table makes finding the correct output line becomes just a matter of table lookup. Using this technique, routing can be done very quickly and any necessary resources can be reserved along the path.

Of course, labeling flows this way comes perilously close to virtual circuits. X.25, ATM, frame relay, and all other networks with a virtual-circuit subnet also put a label (i.e., virtual-circuit identifier) in each packet, look it up in a table, and route based on the table entry. Despite the fact that many people in the Internet community have an intense dislike for connection-oriented networking, the idea seems to keep coming back, this time to provide fast routing and quality of service. However, there are essential differences between the way the Internet handles route construction and the way connection-oriented networks do it, so the technique is certainly not traditional circuit switching.

This "new" switching idea goes by various (proprietary) names, including *label switching* and *tag switching*. Eventually, IETF began to standardize the idea under the name **MPLS** (*MultiProtocol Label Switching*). We will call it MPLS below. It is described in RFC 3031 and many other RFCs.
As an aside, some people make a distinction between routing and switching. Routing is the process of looking up a destination address in a table to find where to send it. In contrast, switching uses a label taken from the packet as an index into a forwarding table. These definitions are far from universal, however.

The first problem is where to put the label. Since IP packets were not designed for virtual circuits, there is no field available for virtual-circuit numbers within the IP header. For this reason, a new MPLS header had to be added in front of the IP header. On a router-to-router line using PPP as the framing protocol, the frame format, including the PPP, MPLS, IP, and TCP headers, is as shown in Fig. 5-41. In a sense, MPLS is thus layer 2.5.

**Figure 5-41. Transmitting a TCP segment using IP, MPLS, and PPP.**

![Figure 5-41.](image)

The generic MPLS header has four fields, the most important of which is the **Label** field, which holds the index. The **QoS** field indicates the class of service. The **S** field relates to stacking multiple labels in hierarchical networks (discussed below). If it hits 0, the packet is discarded. This feature prevents infinite looping in the case of routing instability.

Because the MPLS headers are not part of the network layer packet or the data link layer frame, MPLS is to a large extent independent of both layers. Among other things, this property means it is possible to build MPLS switches that can forward both IP packets and ATM cells, depending on what shows up. This feature is where the "multiprotocol" in the name MPLS came from.

When an MPLS-enhanced packet (or cell) arrives at an MPLS-capable router, the label is used as an index into a table to determine the outgoing line to use and also the new label to use. This label swapping is used in all virtual-circuit subnets because labels have only local significance and two different routers can feed unrelated packets with the same label into another router for transmission on the same outgoing line. To be distinguishable at the other end, labels have to be remapped at every hop. We saw this mechanism in action in Fig. 5-3. MPLS uses the same technique.

One difference from traditional virtual circuits is the level of aggregation. It is certainly possible for each flow to have its own set of labels through the subnet. However, it is more common for routers to group multiple flows that end at a particular router or LAN and use a single label for them. The flows that are grouped together under a single label are said to belong to the same **FEC** (Forwarding Equivalence Class). This class covers not only where the packets are going, but also their service class (in the differentiated services sense) because all their packets are treated the same way for forwarding purposes.

With traditional virtual-circuit routing, it is not possible to group several distinct paths with different end points onto the same virtual-circuit identifier because there would be no way to distinguish them at the final destination. With MPLS, the packets still contain their final destination address, in addition to the label, so that at the end of the labeled route the label header can be removed and forwarding can continue the usual way, using the network layer destination address.
One major difference between MPLS and conventional VC designs is how the forwarding table is constructed. In traditional virtual-circuit networks, when a user wants to establish a connection, a setup packet is launched into the network to create the path and make the forwarding table entries. MPLS does not work that way because there is no setup phase for each connection (because that would break too much existing Internet software).

Instead, there are two ways for the forwarding table entries to be created. In the **data-driven** approach, when a packet arrives, the first router it hits contacts the router downstream where the packet has to go and asks it to generate a label for the flow. This method is applied recursively. Effectively, this is on-demand virtual-circuit creation.

The protocols that do this spreading are very careful to avoid loops. They often use a technique called **colored threads**. The backward propagation of an FEC can be compared to pulling a uniquely colored thread back into the subnet. If a router ever sees a color it already has, it knows there is a loop and takes remedial action. The data-driven approach is primarily used on networks in which the underlying transport is ATM (such as much of the telephone system).

The other way, used on networks not based on ATM, is the **control-driven** approach. It has several variants. One of these works like this. When a router is booted, it checks to see for which routes it is the final destination (e.g., which hosts are on its LAN). It then creates one or more FECs for them, allocates a label for each one, and passes the labels to its neighbors. They, in turn, enter the labels in their forwarding tables and send new labels to their neighbors, until all the routers have acquired the path. Resources can also be reserved as the path is constructed to guarantee an appropriate quality of service.

MPLS can operate at multiple levels at once. At the highest level, each carrier can be regarded as a kind of metarouter, with there being a path through the metarouters from source to destination. This path can use MPLS. However, within each carrier's network, MPLS can also be used, leading to a second level of labeling. In fact, a packet may carry an entire stack of labels with it. The $S$ bit in Fig. 5-41 allows a router removing a label to know if there are any additional labels left. It is set to 1 for the bottom label and 0 for all the other labels. In practice, this facility is mostly used to implement virtual private networks and recursive tunnels.

Although the basic ideas behind MPLS are straightforward, the details are extremely complicated, with many variations and optimizations, so we will not pursue this topic further. For more information, see (Davie and Rekhter, 2000; Lin et al., 2002; Pepelnjak and Guichard, 2001; and Wang, 2001).

### 5.5 Internetworking

Until now, we have implicitly assumed that there is a single homogeneous network, with each machine using the same protocol in each layer. Unfortunately, this assumption is wildly optimistic. Many different networks exist, including LANs, MANs, and WANs. Numerous protocols are in widespread use in every layer. In the following sections we will take a careful look at the issues that arise when two or more networks are connected to form an **internet**.

Considerable controversy exists about the question of whether today's abundance of network types is a temporary condition that will go away as soon as everyone realizes how wonderful [fill in your favorite network] is or whether it is an inevitable, but permanent, feature of the world that is here to stay. Having different networks invariably means having different protocols.

We believe that a variety of different networks (and thus protocols) will always be around, for the
following reasons. First of all, the installed base of different networks is large. Nearly all personal computers run TCP/IP. Many large businesses have mainframes running IBM’s SNA. A substantial number of telephone companies operate ATM networks. Some personal computer LANs still use Novell NCP/IPX or AppleTalk. Finally, wireless is an up-and-coming area with a variety of protocols. This trend will continue for years due to legacy problems, new technology, and the fact that not all vendors perceive it in their interest for their customers to be able to easily migrate to another vendor’s system.

Second, as computers and networks get cheaper, the place where decisions get made moves downward in organizations. Many companies have a policy to the effect that purchases costing over a million dollars have to be approved by top management, purchases costing over 100,000 dollars have to be approved by middle management, but purchases under 100,000 dollars can be made by department heads without any higher approval. This can easily lead to the engineering department installing UNIX workstations running TCP/IP and the marketing department installing Macs with AppleTalk.

Third, different networks (e.g., ATM and wireless) have radically different technology, so it should not be surprising that as new hardware developments occur, new software will be created to fit the new hardware. For example, the average home now is like the average office ten years ago: it is full of computers that do not talk to one another. In the future, it may be commonplace for the telephone, the television set, and other appliances all to be networked together so that they can be controlled remotely. This new technology will undoubtedly bring new networks and new protocols.

As an example of how different networks might be connected, consider the example of Fig. 5-42. Here we see a corporate network with multiple locations tied together by a wide area ATM network. At one of the locations, an FDDI optical backbone is used to connect an Ethernet, an 802.11 wireless LAN, and the corporate data center’s SNA mainframe network.

**Figure 5-42. A collection of interconnected networks.**

![Diagram of interconnected networks](image)

The purpose of interconnecting all these networks is to allow users on any of them to communicate with users on all the other ones and also to allow users on any of them to access data on any of them. Accomplishing this goal means sending packets from one network to another. Since networks often differ in important ways, getting packets from one network to another is not always so easy, as we will now see.

### 5.5.1 How Networks Differ

Networks can differ in many ways. Some of the differences, such as different modulation techniques or frame formats, are in the physical and data link layers. These differences will not concern us here. Instead, in Fig. 5-43 we list some of the differences that can occur in the network
layer. It is papering over these differences that makes internetworking more difficult than operating within a single network.

**Figure 5-43. Some of the many ways networks can differ.**

<table>
<thead>
<tr>
<th>Item</th>
<th>Some Possibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service offered</td>
<td>Connection oriented versus connectionless</td>
</tr>
<tr>
<td>Protocols</td>
<td>IP, IPX, SNA, ATM, MPLS, AppleTalk, etc.</td>
</tr>
<tr>
<td>Addressing</td>
<td>Flat (802) versus hierarchical (IP)</td>
</tr>
<tr>
<td>Multicasting</td>
<td>Present or absent (also broadcasting)</td>
</tr>
<tr>
<td>Packet size</td>
<td>Every network has its own maximum</td>
</tr>
<tr>
<td>Quality of service</td>
<td>Present or absent; many different kinds</td>
</tr>
<tr>
<td>Error handling</td>
<td>Reliable, ordered, and unordered delivery</td>
</tr>
<tr>
<td>Flow control</td>
<td>Sliding window, rate control, other, or none</td>
</tr>
<tr>
<td>Congestion control</td>
<td>Leaky bucket, token bucket, RED, choke packets, etc.</td>
</tr>
<tr>
<td>Security</td>
<td>Privacy rules, encryption, etc.</td>
</tr>
<tr>
<td>Parameters</td>
<td>Different timeouts, flow specifications, etc.</td>
</tr>
<tr>
<td>Accounting</td>
<td>By connect time, by packet, by byte, or not at all</td>
</tr>
</tbody>
</table>

When packets sent by a source on one network must transit one or more foreign networks before reaching the destination network (which also may be different from the source network), many problems can occur at the interfaces between networks. To start with, when packets from a connection-oriented network must transit a connectionless one, they may be reordered, something the sender does not expect and the receiver is not prepared to deal with. Protocol conversions will often be needed, which can be difficult if the required functionality cannot be expressed. Address conversions will also be needed, which may require some kind of directory system. Passing multicast packets through a network that does not support multicasting requires generating separate packets for each destination.

The differing maximum packet sizes used by different networks can be a major nuisance. How do you pass an 8000-byte packet through a network whose maximum size is 1500 bytes? Differing qualities of service is an issue when a packet that has real-time delivery constraints passes through a network that does not offer any real-time guarantees.

Error, flow, and congestion control often differ among different networks. If the source and destination both expect all packets to be delivered in sequence without error but an intermediate network just discards packets whenever it smells congestion on the horizon, many applications will break. Also, if packets can wander around aimlessly for a while and then suddenly emerge and be delivered, trouble will occur if this behavior was not anticipated and dealt with. Different security mechanisms, parameter settings, and accounting rules, and even national privacy laws also can cause problems.

### 5.5.2 How Networks Can Be Connected

Networks can be interconnected by different devices, as we saw in Chap 4. Let us briefly review that material. In the physical layer, networks can be connected by repeaters or hubs, which just move the bits from one network to an identical network. These are mostly analog devices and do not understand anything about digital protocols (they just regenerate signals).

One layer up we find bridges and switches, which operate at the data link layer. They can accept frames, examine the MAC addresses, and forward the frames to a different network while doing
minor protocol translation in the process, for example, from Ethernet to FDDI or to 802.11.

In the network layer, we have routers that can connect two networks. If two networks have dissimilar network layers, the router may be able to translate between the packet formats, although packet translation is now increasingly rare. A router that can handle multiple protocols is called a **multiprotocol router**.

In the transport layer we find transport gateways, which can interface between two transport connections. For example, a transport gateway could allow packets to flow between a TCP network and an SNA network, which has a different transport protocol, by essentially gluing a TCP connection to an SNA connection.

Finally, in the application layer, application gateways translate message semantics. As an example, gateways between Internet e-mail (RFC 822) and X.400 e-mail must parse the e-mail messages and change various header fields.

In this chapter we will focus on internetworking in the network layer. To see how that differs from switching in the data link layer, examine **Fig. 5-44**. In **Fig. 5-44(a)**, the source machine, $S$, wants to send a packet to the destination machine, $D$. These machines are on different Ethernets, connected by a switch. $S$ encapsulates the packet in a frame and sends it on its way. The frame arrives at the switch, which then determines that the frame has to go to LAN 2 by looking at its MAC address. The switch just removes the frame from LAN 1 and deposits it on LAN 2.

**Figure 5-44. (a) Two Ethernets connected by a switch. (b) Two Ethernets connected by routers.**

Now let us consider the same situation but with the two Ethernets connected by a pair of routers instead of a switch. The routers are connected by a point-to-point line, possibly a leased line thousands of kilometers long. Now the frame is picked up by the router and the packet removed from the frame’s data field. The router examines the address in the packet (e.g., an IP address) and looks up this address in its routing table. Based on this address, it decides to send the packet to the remote router, potentially encapsulated in a different kind of frame, depending on the line protocol. At the far end, the packet is put into the data field of an Ethernet frame and deposited onto LAN 2.

An essential difference between the switched (or bridged) case and the routed case is this. With a switch (or bridge), the entire frame is transported on the basis of its MAC address. With a router, the packet is extracted from the frame and the address in the packet is used for deciding where to send it. Switches do not have to understand the network layer protocol being used to switch packets. Routers do.

### 5.5.3 Concatenated Virtual Circuits
Two styles of internetworking are possible: a connection-oriented concatenation of virtual-circuit subnets, and a datagram internet style. We will now examine these in turn, but first a word of caution. In the past, most (public) networks were connection oriented (and frame relay, SNA, 802.16, and ATM still are). Then with the rapid acceptance of the Internet, datagrams became fashionable. However, it would be a mistake to think that datagrams are forever. In this business, the only thing that is forever is change. With the growing importance of multimedia networking, it is likely that connection-orientation will make a come-back in one form or another since it is easier to guarantee quality of service with connections than without them. Therefore, we will devote some space to connection-oriented networking below.

In the concatenated virtual-circuit model, shown in Fig. 5-45, a connection to a host in a distant network is set up in a way similar to the way connections are normally established. The subnet sees that the destination is remote and builds a virtual circuit to the router nearest the destination network. Then it constructs a virtual circuit from that router to an external gateway (multiprotocol router). This gateway records the existence of the virtual circuit in its tables and proceeds to build another virtual circuit to a router in the next subnet. This process continues until the destination host has been reached.

**Figure 5-45. Internetworking using concatenated virtual circuits.**

Once data packets begin flowing along the path, each gateway relays incoming packets, converting between packet formats and virtual-circuit numbers as needed. Clearly, all data packets must traverse the same sequence of gateways. Consequently, packets in a flow are never reordered by the network.

The essential feature of this approach is that a sequence of virtual circuits is set up from the source through one or more gateways to the destination. Each gateway maintains tables telling which virtual circuits pass through it, where they are to be routed, and what the new virtual-circuit number is.

This scheme works best when all the networks have roughly the same properties. For example, if all of them guarantee reliable delivery of network layer packets, then barring a crash somewhere along the route, the flow from source to destination will also be reliable. Similarly, if none of them guarantee reliable delivery, then the concatenation of the virtual circuits is not reliable either. On the other hand, if the source machine is on a network that does guarantee reliable delivery but one of the intermediate networks can lose packets, the concatenation has fundamentally changed the nature of the service.

Concatenated virtual circuits are also common in the transport layer. In particular, it is possible to build a bit pipe using, say, SNA, which terminates in a gateway, and have a TCP connection go from the gateway to the next gateway. In this manner, an end-to-end virtual circuit can be built.
spanning different networks and protocols.

### 5.5.4 Connectionless Internetworking

The alternative internetwork model is the datagram model, shown in Fig. 5-46. In this model, the only service the network layer offers to the transport layer is the ability to inject datagrams into the subnet and hope for the best. There is no notion of a virtual circuit at all in the network layer, let alone a concatenation of them. This model does not require all packets belonging to one connection to traverse the same sequence of gateways. In Fig. 5-46 datagrams from host 1 to host 2 are shown taking different routes through the internetwork. A routing decision is made separately for each packet, possibly depending on the traffic at the moment the packet is sent. This strategy can use multiple routes and thus achieve a higher bandwidth than the concatenated virtual-circuit model. On the other hand, there is no guarantee that the packets arrive at the destination in order, assuming that they arrive at all.

**Figure 5-46. A connectionless internet.**

The model of Fig. 5-46 is not quite as simple as it looks. For one thing, if each network has its own network layer protocol, it is not possible for a packet from one network to transit another one. One could imagine the multiprotocol routers actually trying to translate from one format to another, but unless the two formats are close relatives with the same information fields, such conversions will always be incomplete and often doomed to failure. For this reason, conversion is rarely attempted.

A second, and more serious, problem is addressing. Imagine a simple case: a host on the Internet is trying to send an IP packet to a host on an adjoining SNA network. The IP and SNA addresses are different. One would need a mapping between IP and SNA addresses in both directions. Furthermore, the concept of what is addressable is different. In IP, hosts (actually, interface cards) have addresses. In SNA, entities other than hosts (e.g., hardware devices) can also have addresses. At best, someone would have to maintain a database mapping everything to everything to the extent possible, but it would constantly be a source of trouble.

Another idea is to design a universal "internet" packet and have all routers recognize it. This approach is, in fact, what IP is—a packet designed to be carried through many networks. Of course, it may turn out that IPv4 (the current Internet protocol) drives all other formats out of the market, IPv6 (the future Internet protocol) does not catch on, and nothing new is ever invented, but history suggests otherwise. Getting everybody to agree to a single format is difficult when companies perceive it to their commercial advantage to have a proprietary format that they control.

Let us now briefly recap the two ways internetworking can be approached. The concatenated virtual-circuit model has essentially the same advantages as using virtual circuits within a single
subnet: buffers can be reserved in advance, sequencing can be guaranteed, short headers can be used, and the troubles caused by delayed duplicate packets can be avoided.

It also has the same disadvantages: table space required in the routers for each open connection, no alternate routing to avoid congested areas, and vulnerability to router failures along the path. It also has the disadvantage of being difficult, if not impossible, to implement if one of the networks involved is an unreliable datagram network.

The properties of the datagram approach to internetworking are pretty much the same as those of datagram subnets: more potential for congestion, but also more potential for adapting to it, robustness in the face of router failures, and longer headers needed. Various adaptive routing algorithms are possible in an internet, just as they are within a single datagram network.

A major advantage of the datagram approach to internetworking is that it can be used over subnets that do not use virtual circuits inside. Many LANs, mobile networks (e.g., aircraft and naval fleets), and even some WANs fall into this category. When an internet includes one of these, serious problems occur if the internetworking strategy is based on virtual circuits.

5.5.5 Tunneling

Handling the general case of making two different networks interwork is exceedingly difficult. However, there is a common special case that is manageable. This case is where the source and destination hosts are on the same type of network, but there is a different network in between. As an example, think of an international bank with a TCP/IP-based Ethernet in Paris, a TCP/IP-based Ethernet in London, and a non-IP wide area network (e.g., ATM) in between, as shown in Fig. 5-47.

**Figure 5-47. Tunneling a packet from Paris to London.**

The solution to this problem is a technique called tunneling. To send an IP packet to host 2, host 1 constructs the packet containing the IP address of host 2, inserts it into an Ethernet frame addressed to the Paris multiprotocol router, and puts it on the Ethernet. When the multiprotocol router gets the frame, it removes the IP packet, inserts it in the payload field of the WAN network layer packet, and addresses the latter to the WAN address of the London multiprotocol router. When it gets there, the London router removes the IP packet and sends it to host 2 inside an Ethernet frame.

The WAN can be seen as a big tunnel extending from one multiprotocol router to the other. The IP packet just travels from one end of the tunnel to the other, snug in its nice box. It does not have to worry about dealing with the WAN at all. Neither do the hosts on either Ethernet. Only the multiprotocol router has to understand IP and WAN packets. In effect, the entire distance from the
middle of one multiprotocol router to the middle of the other acts like a serial line.

An analogy may make tunneling clearer. Consider a person driving her car from Paris to London. Within France, the car moves under its own power, but when it hits the English Channel, it is loaded into a high-speed train and transported to England through the Chunnel (cars are not permitted to drive through the Chunnel). Effectively, the car is being carried as freight, as depicted in Fig. 5-48. At the far end, the car is let loose on the English roads and once again continues to move under its own power. Tunneling of packets through a foreign network works the same way.

![Figure 5-48. Tunneling a car from France to England.](image)

5.5.6 Internetwork Routing

Routing through an internetwork is similar to routing within a single subnet, but with some added complications. Consider, for example, the internetwork of Fig. 5-49(a) in which five networks are connected by six (possibly multiprotocol) routers. Making a graph model of this situation is complicated by the fact that every router can directly access (i.e., send packets to) every other router connected to any network to which it is connected. For example, B in Fig. 5-49(a) can directly access A and C via network 2 and also D via network 3. This leads to the graph of Fig. 5-49(b).

![Figure 5-49. (a) An internetwork. (b) A graph of the internetwork.](image)

Once the graph has been constructed, known routing algorithms, such as the distance vector and link state algorithms, can be applied to the set of multiprotocol routers. This gives a two-level routing algorithm: within each network an interior gateway protocol is used, but between the networks, an exterior gateway protocol is used ("gateway" is an older term for "router"). In fact, since each network is independent, they may all use different algorithms. Because each network in an internetwork is independent of all the others, it is often referred to as an Autonomous System (AS).

A typical internet packet starts out on its LAN addressed to the local multiprotocol router (in the MAC layer header). After it gets there, the network layer code decides which multiprotocol router to forward the packet to, using its own routing tables. If that router can be reached using the
packet's native network protocol, the packet is forwarded there directly. Otherwise it is tunneled there, encapsulated in the protocol required by the intervening network. This process is repeated until the packet reaches the destination network.

One of the differences between internetwork routing and intranetwork routing is that internetwork routing may require crossing international boundaries. Various laws suddenly come into play, such as Sweden's strict privacy laws about exporting personal data about Swedish citizens from Sweden. Another example is the Canadian law saying that data traffic originating in Canada and ending in Canada may not leave the country. This law means that traffic from Windsor, Ontario to Vancouver may not be routed via nearby Detroit, even if that route is the fastest and cheapest.

Another difference between interior and exterior routing is the cost. Within a single network, a single charging algorithm normally applies. However, different networks may be under different managements, and one route may be less expensive than another. Similarly, the quality of service offered by different networks may be different, and this may be a reason to choose one route over another.

5.5.7 Fragmentation

Each network imposes some maximum size on its packets. These limits have various causes, among them:

1. Hardware (e.g., the size of an Ethernet frame).
2. Operating system (e.g., all buffers are 512 bytes).
3. Protocols (e.g., the number of bits in the packet length field).
4. Compliance with some (inter)national standard.
5. Desire to reduce error-induced retransmissions to some level.
6. Desire to prevent one packet from occupying the channel too long.

The result of all these factors is that the network designers are not free to choose any maximum packet size they wish. Maximum payloads range from 48 bytes (ATM cells) to 65,515 bytes (IP packets), although the payload size in higher layers is often larger.

An obvious problem appears when a large packet wants to travel through a network whose maximum packet size is too small. One solution is to make sure the problem does not occur in the first place. In other words, the internet should use a routing algorithm that avoids sending packets through networks that cannot handle them. However, this solution is no solution at all. What happens if the original source packet is too large to be handled by the destination network? The routing algorithm can hardly bypass the destination.

Basically, the only solution to the problem is to allow gateways to break up packets into fragments, sending each fragment as a separate internet packet. However, as every parent of a small child knows, converting a large object into small fragments is considerably easier than the reverse process. (Physicists have even given this effect a name: the second law of thermodynamics.) Packet-switching networks, too, have trouble putting the fragments back together again.

Two opposing strategies exist for recombining the fragments back into the original packet. The first
strategy is to make fragmentation caused by a "small-packet" network transparent to any subsequent networks through which the packet must pass on its way to the ultimate destination. This option is shown in Fig. 5-50(a). In this approach, the small-packet network has gateways (most likely, specialized routers) that interface to other networks. When an oversized packet arrives at a gateway, the gateway breaks it up into fragments. Each fragment is addressed to the same exit gateway, where the pieces are recombined. In this way passage through the small-packet network has been made transparent. Subsequent networks are not even aware that fragmentation has occurred. ATM networks, for example, have special hardware to provide transparent fragmentation of packets into cells and then reassembly of cells into packets. In the ATM world, fragmentation is called segmentation; the concept is the same, but some of the details are different.

Figure 5-50. (a) Transparent fragmentation. (b) Nontransparent fragmentation.

Transparent fragmentation is straightforward but has some problems. For one thing, the exit gateway must know when it has received all the pieces, so either a count field or an "end of packet" bit must be provided. For another thing, all packets must exit via the same gateway. By not allowing some fragments to follow one route to the ultimate destination and other fragments a disjoint route, some performance may be lost. A last problem is the overhead required to repeatedly reassemble and then refragment a large packet passing through a series of small-packet networks. ATM requires transparent fragmentation.

The other fragmentation strategy is to refrain from recombining fragments at any intermediate gateways. Once a packet has been fragmented, each fragment is treated as though it were an original packet. All fragments are passed through the exit gateway (or gateways), as shown in Fig. 5-50(b). Recombination occurs only at the destination host. IP works this way.

Nontransparent fragmentation also has some problems. For example, it requires every host to be able to do reassembly. Yet another problem is that when a large packet is fragmented, the total overhead increases because each fragment must have a header. Whereas in the first method this overhead disappears as soon as the small-packet network is exited, in this method the overhead remains for the rest of the journey. An advantage of nontransparent fragmentation, however, is that multiple exit gateways can now be used and higher performance can be achieved. Of course, if the concatenated virtual-circuit model is being used, this advantage is of no use.

When a packet is fragmented, the fragments must be numbered in such a way that the original data stream can be reconstructed. One way of numbering the fragments is to use a tree. If packet
0 must be split up, the pieces are called 0.0, 0.1, 0.2, etc. If these fragments themselves must be fragmented later on, the pieces are numbered 0.0.0, 0.0.1, 0.0.2, . . . , 0.1.0, 0.1.1, 0.1.2, etc. If enough fields have been reserved in the header for the worst case and no duplicates are generated anywhere, this scheme is sufficient to ensure that all the pieces can be correctly reassembled at the destination, no matter what order they arrive in.

However, if even one network loses or discards packets, end-to-end retransmissions are needed, with unfortunate effects for the numbering system. Suppose that a 1024-bit packet is initially fragmented into four equal-sized fragments, 0.0, 0.1, 0.2, and 0.3. Fragment 0.1 is lost, but the other parts arrive at the destination. Eventually, the source times out and retransmits the original packet again. Only this time Murphy’s law strikes and the route taken passes through a network with a 512-bit limit, so two fragments are generated. When the new fragment 0.1 arrives at the destination, the receiver will think that all four pieces are now accounted for and reconstruct the packet incorrectly.

A completely different (and better) numbering system is for the internetwork protocol to define an elementary fragment size small enough that the elementary fragment can pass through every network. When a packet is fragmented, all the pieces are equal to the elementary fragment size except the last one, which may be shorter. An internet packet may contain several fragments, for efficiency reasons. The internet header must provide the original packet number and the number of the (first) elementary fragment contained in the packet. As usual, there must also be a bit indicating that the last elementary fragment contained within the internet packet is the last one of the original packet.

This approach requires two sequence fields in the internet header: the original packet number and the fragment number. There is clearly a trade-off between the size of the elementary fragment and the number of bits in the fragment number. Because the elementary fragment size is presumed to be acceptable to every network, subsequent fragmentation of an internet packet containing several fragments causes no problem. The ultimate limit here is to have the elementary fragment be a single bit or byte, with the fragment number then being the bit or byte offset within the original packet, as shown in Fig. 5-51.

**Figure 5-51.** Fragmentation when the elementary data size is 1 byte. (a) Original packet, containing 10 data bytes. (b) Fragments after passing through a network with maximum packet size of 8 payload bytes plus header. (c) Fragments after passing through a size 5 gateway.
Some internet protocols take this method even further and consider the entire transmission on a virtual circuit to be one giant packet, so that each fragment contains the absolute byte number of the first byte within the fragment.

---

### 5.6 The Network Layer in the Internet

Before getting into the specifics of the network layer in the Internet, it is worth taking at look at the principles that drove its design in the past and made it the success that it is today. All too often, nowadays, people seem to have forgotten them. These principles are enumerated and discussed in RFC 1958, which is well worth reading (and should be mandatory for all protocol designers—with a final exam at the end). This RFC draws heavily on ideas found in (Clark, 1988; and Saltzer et al., 1984). We will now summarize what we consider to be the top 10 principles (from most important to least important).

1. **Make sure it works.** Do not finalize the design or standard until multiple prototypes have successfully communicated with each other. All too often designers first write a 1000-page standard, get it approved, then discover it is deeply flawed and does not work. Then they write version 1.1 of the standard. This is not the way to go.

2. **Keep it simple.** When in doubt, use the simplest solution. William of Occam stated this principle (Occam's razor) in the 14th century. Put in modern terms: fight features. If a feature is not absolutely essential, leave it out, especially if the same effect can be achieved by combining other features.

3. **Make clear choices.** If there are several ways of doing the same thing, choose one. Having two or more ways to do the same thing is looking for trouble. Standards often have multiple options or modes or parameters because several powerful parties insist that their way is best. Designers should strongly resist this tendency. Just say no.

4. **Exploit modularity.** This principle leads directly to the idea of having protocol stacks, each of whose layers is independent of all the other ones. In this way, if circumstances that
require one module or layer to be changed, the other ones will not be affected.

5. **Expect heterogeneity.** Different types of hardware, transmission facilities, and applications will occur on any large network. To handle them, the network design must be simple, general, and flexible.

6. **Avoid static options and parameters.** If parameters are unavoidable (e.g., maximum packet size), it is best to have the sender and receiver negotiate a value than defining fixed choices.

7. **Look for a good design; it need not be perfect.** Often the designers have a good design but it cannot handle some weird special case. Rather than messing up the design, the designers should go with the good design and put the burden of working around it on the people with the strange requirements.

8. **Be strict when sending and tolerant when receiving.** In other words, only send packets that rigorously comply with the standards, but expect incoming packets that may not be fully conformant and try to deal with them.

9. **Think about scalability.** If the system is to handle millions of hosts and billions of users effectively, no centralized databases of any kind are tolerable and load must be spread as evenly as possible over the available resources.

10. **Consider performance and cost.** If a network has poor performance or outrageous costs, nobody will use it.

Let us now leave the general principles and start looking at the details of the Internet's network layer. At the network layer, the Internet can be viewed as a collection of subnetworks or **Autonomous Systems (ASes)** that are interconnected. There is no real structure, but several major backbones exist. These are constructed from high-bandwidth lines and fast routers. Attached to the backbones are regional (midlevel) networks, and attached to these regional networks are the LANs at many universities, companies, and Internet service providers. A sketch of this quasi-hierarchical organization is given in **Fig. 5-52**.

**Figure 5-52. The Internet is an interconnected collection of many networks.**
The glue that holds the whole Internet together is the network layer protocol, **IP (Internet Protocol)**. Unlike most older network layer protocols, it was designed from the beginning with internetworking in mind. A good way to think of the network layer is this. Its job is to provide a best-efforts (i.e., not guaranteed) way to transport datagrams from source to destination, without regard to whether these machines are on the same network or whether there are other networks in between them.

Communication in the Internet works as follows. The transport layer takes data streams and breaks them up into datagrams. In theory, datagrams can be up to 64 Kbytes each, but in practice they are usually not more than 1500 bytes (so they fit in one Ethernet frame). Each datagram is transmitted through the Internet, possibly being fragmented into smaller units as it goes. When all the pieces finally get to the destination machine, they are reassembled by the network layer into the original datagram. This datagram is then handed to the transport layer, which inserts it into the receiving process' input stream. As can be seen from Fig. 5-52, a packet originating at host 1 has to traverse six networks to get to host 2. In practice, it is often much more than six.

### 5.6.1 The IP Protocol

An appropriate place to start our study of the network layer in the Internet is the format of the IP datagrams themselves. An IP datagram consists of a header part and a text part. The header has a 20-byte fixed part and a variable length optional part. The header format is shown in Fig. 5-53. It is transmitted in big-endian order: from left to right, with the high-order bit of the Version field going first. (The SPARC is big endian; the Pentium is little-endian.) On little endian machines, software conversion is required on both transmission and reception.

**Figure 5-53. The IPv4 (Internet Protocol) header.**
The *Version* field keeps track of which version of the protocol the datagram belongs to. By including the version in each datagram, it becomes possible to have the transition between versions take years, with some machines running the old version and others running the new one. Currently a transition between IPv4 and IPv6 is going on, has already taken years, and is by no means close to being finished (Durand, 2001; Wiljakka, 2002; and Waddington and Chang, 2002). Some people even think it will never happen (Weiser, 2001). As an aside on numbering, IPv5 was an experimental real-time stream protocol that was never widely used.

Since the header length is not constant, a field in the header, *IHL*, is provided to tell how long the header is, in 32-bit words. The minimum value is 5, which applies when no options are present. The maximum value of this 4-bit field is 15, which limits the header to 60 bytes, and thus the *Options* field to 40 bytes. For some options, such as one that records the route a packet has taken, 40 bytes is far too small, making that option useless.

The *Type of service* field is one of the few fields that has changed its meaning (slightly) over the years. It was and is still intended to distinguish between different classes of service. Various combinations of reliability and speed are possible. For digitized voice, fast delivery beats accurate delivery. For file transfer, error-free transmission is more important than fast transmission.

Originally, the 6-bit field contained (from left to right), a three-bit *Precedence* field and three flags, *D*, *T*, and *R*. The *Precedence* field was a priority, from 0 (normal) to 7 (network control packet). The three flag bits allowed the host to specify what it cared most about from the set {Delay, Throughput, Reliability}. In theory, these fields allow routers to make choices between, for example, a satellite link with high throughput and high delay or a leased line with low throughput and low delay. In practice, current routers often ignore the *Type of service* field altogether.

Eventually, IETF threw in the towel and changed the field slightly to accommodate differentiated services. Six of the bits are used to indicate which of the service classes discussed earlier each packet belongs to. These classes include the four queueing priorities, three discard probabilities, and the historical classes.

The *Total length* includes everything in the datagram—both header and data. The maximum length is 65,535 bytes. At present, this upper limit is tolerable, but with future gigabit networks, larger datagrams may be needed.

The *Identification* field is needed to allow the destination host to determine which datagram a newly arrived fragment belongs to. All the fragments of a datagram contain the same *Identification* value.
Next comes an unused bit and then two 1-bit fields. \textit{DF} stands for Don't Fragment. It is an order to the routers not to fragment the datagram because the destination is incapable of putting the pieces back together again. For example, when a computer boots, its ROM might ask for a memory image to be sent to it as a single datagram. By marking the datagram with the \textit{DF} bit, the sender knows it will arrive in one piece, even if this means that the datagram must avoid a small-packet network on the best path and take a suboptimal route. All machines are required to accept fragments of 576 bytes or less.

\textit{MF} stands for More Fragments. All fragments except the last one have this bit set. It is needed to know when all fragments of a datagram have arrived.

The \textit{Fragment offset} tells where in the current datagram this fragment belongs. All fragments except the last one in a datagram must be a multiple of 8 bytes, the elementary fragment unit. Since 13 bits are provided, there is a maximum of 8192 fragments per datagram, giving a maximum datagram length of 65,536 bytes, one more than the \textit{Total length} field.

The \textit{Time to live} field is a counter used to limit packet lifetimes. It is supposed to count time in seconds, allowing a maximum lifetime of 255 sec. It must be decremented on each hop and is supposed to be decremented multiple times when queued for a long time in a router. In practice, it just counts hops. When it hits zero, the packet is discarded and a warning packet is sent back to the source host. This feature prevents datagrams from wandering around forever, something that otherwise might happen if the routing tables ever become corrupted.

When the network layer has assembled a complete datagram, it needs to know what to do with it. The \textit{Protocol} field tells it which transport process to give it to. TCP is one possibility, but so are UDP and some others. The numbering of protocols is global across the entire Internet. Protocols and other assigned numbers were formerly listed in RFC 1700, but nowadays they are contained in an on-line data base located at \url{www.iana.org}.

The \textit{Header checksum} verifies the header only. Such a checksum is useful for detecting errors generated by bad memory words inside a router. The algorithm is to add up all the 16-bit halfwords as they arrive, using one's complement arithmetic and then take the one's complement of the result. For purposes of this algorithm, the \textit{Header checksum} is assumed to be zero upon arrival. This algorithm is more robust than using a normal add. Note that the \textit{Header checksum} must be recomputed at each hop because at least one field always changes (the \textit{Time to live} field), but tricks can be used to speed up the computation.

The \textit{Source address} and \textit{Destination address} indicate the network number and host number. We will discuss Internet addresses in the next section. The \textit{Options} field was designed to provide an escape to allow subsequent versions of the protocol to include information not present in the original design, to permit experimenters to try out new ideas, and to avoid allocating header bits to information that is rarely needed. The options are variable length. Each begins with a 1-byte code identifying the option. Some options are followed by a 1-byte option length field, and then one or more data bytes. The \textit{Options} field is padded out to a multiple of four bytes. Originally, five options were defined, as listed in Fig. 5-54, but since then some new ones have been added. The current complete list is now maintained on-line at \url{www.iana.org/assignments/ip-parameters}.

\textbf{Figure 5-54. Some of the IP options.}
The Security option tells how secret the information is. In theory, a military router might use this
field to specify not to route through certain countries the military considers to be "bad guys." In
practice, all routers ignore it, so its only practical function is to help spies find the good stuff more
easily.

The Strict source routing option gives the complete path from source to destination as a sequence
of IP addresses. The datagram is required to follow that exact route. It is most useful for system
managers to send emergency packets when the routing tables are corrupted, or for making timing
measurements.

The Loose source routing option requires the packet to traverse the list of routers specified, and in
the order specified, but it is allowed to pass through other routers on the way. Normally, this
option would only provide a few routers, to force a particular path. For example, to force a packet
from London to Sydney to go west instead of east, this option might specify routers in New York,
Los Angeles, and Honolulu. This option is most useful when political or economic considerations
dictate passing through or avoiding certain countries.

The Record route option tells the routers along the path to append their IP address to the option
field. This allows system managers to track down bugs in the routing algorithms ("Why are packets
from Houston to Dallas visiting Tokyo first?"). When the ARPANET was first set up, no packet ever
passed through more than nine routers, so 40 bytes of option was ample. As mentioned above,
now it is too small.

Finally, the Timestamp option is like the Record route option, except that in addition to recording
its 32-bit IP address, each router also records a 32-bit timestamp. This option, too, is mostly for
debugging routing algorithms.

5.6.2 IP Addresses

Every host and router on the Internet has an IP address, which encodes its network number and
host number. The combination is unique: in principle, no two machines on the Internet have the
same IP address. All IP addresses are 32 bits long and are used in the Source address and
Destination address fields of IP packets. It is important to note that an IP address does not actually
refer to a host. It really refers to a network interface, so if a host is on two networks, it must have
two IP addresses. However, in practice, most hosts are on one network and thus have one IP
address.

For several decades, IP addresses were divided into the five categories listed in Fig. 5-55. This
allocation has come to be called classful addressing. It is no longer used, but references to it in the
literature are still common. We will discuss the replacement of classful addressing shortly.

**Figure 5-55. IP address formats.**
The class A, B, C, and D formats allow for up to 128 networks with 16 million hosts each, 16,384 networks with up to 64K hosts, and 2 million networks (e.g., LANs) with up to 256 hosts each (although a few of these are special). Also supported is multicast, in which a datagram is directed to multiple hosts. Addresses beginning with 1111 are reserved for future use. Over 500,000 networks are now connected to the Internet, and the number grows every year. Network numbers are managed by a nonprofit corporation called **ICANN (Internet Corporation for Assigned Names and Numbers)** to avoid conflicts. In turn, ICANN has delegated parts of the address space to various regional authorities, which then dole out IP addresses to ISPs and other companies.

Network addresses, which are 32-bit numbers, are usually written in **dotted decimal notation**. In this format, each of the 4 bytes is written in decimal, from 0 to 255. For example, the 32-bit hexadecimal address C0290614 is written as 192.41.6.20. The lowest IP address is 0.0.0.0 and the highest is 255.255.255.255.

The values 0 and -1 (all 1s) have special meanings, as shown in **Fig. 5-56**. The value 0 means this network or this host. The value of -1 is used as a broadcast address to mean all hosts on the indicated network.

**Figure 5-56. Special IP addresses.**

<table>
<thead>
<tr>
<th>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</th>
<th>This host</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 . . . 0 0</td>
<td>Host</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td>Broadcast on the local network</td>
</tr>
<tr>
<td>Network 1 1 1 . . . 1 1 1 1</td>
<td>Broadcast on a distant network</td>
</tr>
<tr>
<td>127</td>
<td>Loopback</td>
</tr>
</tbody>
</table>

The IP address 0.0.0.0 is used by hosts when they are being booted. IP addresses with 0 as network number refer to the current network. These addresses allow machines to refer to their own network without knowing its number (but they have to know its class to know how many 0s to include). The address consisting of all 1s allows broadcasting on the local network, typically a LAN. The addresses with a proper network number and all 1s in the host field allow machines to send broadcast packets to distant LANs anywhere in the Internet (although many network administrators disable this feature). Finally, all addresses of the form 127.xx.yy.zz are reserved for loopback testing. Packets sent to that address are not put out onto the wire; they are processed locally and treated as incoming packets. This allows packets to be sent to the local network without...
the sender knowing its number.

Subnets

As we have seen, all the hosts in a network must have the same network number. This property of IP addressing can cause problems as networks grow. For example, consider a university that started out with one class B network used by the Computer Science Dept. for the computers on its Ethernet. A year later, the Electrical Engineering Dept. wanted to get on the Internet, so they bought a repeater to extend the CS Ethernet to their building. As time went on, many other departments acquired computers and the limit of four repeaters per Ethernet was quickly reached. A different organization was required.

Getting a second network address would be hard to do since network addresses are scarce and the university already had enough addresses for over 60,000 hosts. The problem is the rule that a single class A, B, or C address refers to one network, not to a collection of LANs. As more and more organizations ran into this situation, a small change was made to the addressing system to deal with it.

The solution is to allow a network to be split into several parts for internal use but still act like a single network to the outside world. A typical campus network nowadays might look like that of Fig. 5-57, with a main router connected to an ISP or regional network and numerous Ethernets spread around campus in different departments. Each of the Ethernets has its own router connected to the main router (possibly via a backbone LAN, but the nature of the interrouter connection is not relevant here).

![Figure 5-57. A campus network consisting of LANs for various departments.](image)

In the Internet literature, the parts of the network (in this case, Ethernets) are called subnets. As we mentioned in Chap. 1, this usage conflicts with "subnet" to mean the set of all routers and communication lines in a network. Hopefully, it will be clear from the context which meaning is intended. In this section and the next one, the new definition will be the one used exclusively.

When a packet comes into the main router, how does it know which subnet (Ethernet) to give it to? One way would be to have a table with 65,536 entries in the main router telling which router to use for each host on campus. This idea would work, but it would require a very large table in the main router and a lot of manual maintenance as hosts were added, moved, or taken out of service.

Instead, a different scheme was invented. Basically, instead of having a single class B address with 14 bits for the network number and 16 bits for the host number, some bits are taken away from the host number to create a subnet number. For example, if the university has 35 departments, it
could use a 6-bit subnet number and a 10-bit host number, allowing for up to 64 Ethernets, each
with a maximum of 1022 hosts (0 and -1 are not available, as mentioned earlier). This split could
be changed later if it turns out to be the wrong one.

To implement subnetting, the main router needs a **subnet mask** that indicates the split between
network + subnet number and host, as shown in Fig. 5-58. Subnet masks are also written in
dotted decimal notation, with the addition of a slash followed by the number of bits in the network +
subnet part. For the example of Fig. 5-58, the subnet mask can be written as 255.255.252.0. An
alternative notation is /22 to indicate that the subnet mask is 22 bits long.

**Figure 5-58. A class B network subnetted into 64 subnets.**

Outside the network, the subnetting is not visible, so allocating a new subnet does not require
contacting ICANN or changing any external databases. In this example, the first subnet might use
IP addresses starting at 130.50.4.1; the second subnet might start at 130.50.8.1; the third subnet
might start at 130.50.12.1; and so on. To see why the subnets are counting by fours, note that the
corresponding binary addresses are as follows:

- Subnet 1: 10000010 00110010 000001|00 00000001
- Subnet 2: 10000010 00110010 000010|00 00000001
- Subnet 3: 10000010 00110010 000011|00 00000001

Here the vertical bar (|) shows the boundary between the subnet number and the host number. To
its left is the 6-bit subnet number; to its right is the 10-bit host number.

To see how subnets work, it is necessary to explain how IP packets are processed at a router. Each
router has a table listing some number of (network, 0) IP addresses and some number of (this-
network, host) IP addresses. The first kind tells how to get to distant networks. The second kind
tells how to get to local hosts. Associated with each table is the network interface to use to reach
the destination, and certain other information.

When an IP packet arrives, its destination address is looked up in the routing table. If the packet is
for a distant network, it is forwarded to the next router on the interface given in the table. If it is a
local host (e.g., on the router's LAN), it is sent directly to the destination. If the network is not
present, the packet is forwarded to a default router with more extensive tables. This algorithm
means that each router only has to keep track of other networks and local hosts, not (network,
host) pairs, greatly reducing the size of the routing table.

When subnetting is introduced, the routing tables are changed, adding entries of the form (this-
network, subnet, 0) and (this-network, this-subnet, host). Thus, a router on subnet k knows how
to get to all the other subnets and also how to get to all the hosts on subnet k. It does not have to
know the details about hosts on other subnets. In fact, all that needs to be changed is to have each
router do a Boolean AND with the network's subnet mask to get rid of the host number and look up
the resulting address in its tables (after determining which network class it is). For example, a
packet addressed to 130.50.15.6 and arriving at the main router is ANDed with the subnet mask
255.255.252.0/22 to give the address 130.50.12.0. This address is looked up in the routing tables
to find out which output line to use to get to the router for subnet 3. Subnetting thus reduces
router table space by creating a three-level hierarchy consisting of network, subnet, and host.

**CIDR—Classless InterDomain Routing**

IP has been in heavy use for decades. It has worked extremely well, as demonstrated by the exponential growth of the Internet. Unfortunately, IP is rapidly becoming a victim of its own popularity: it is running out of addresses. This looming disaster has sparked a great deal of discussion and controversy within the Internet community about what to do about it. In this section we will describe both the problem and several proposed solutions.

Back in 1987, a few visionaries predicted that some day the Internet might grow to 100,000 networks. Most experts pooh-poohed this as being decades in the future, if ever. The 100,000th network was connected in 1996. The problem, as mentioned above, is that the Internet is rapidly running out of IP addresses. In principle, over 2 billion addresses exist, but the practice of organizing the address space by classes (see Fig. 5-55) wastes millions of them. In particular, the real villain is the class B network. For most organizations, a class A network, with 16 million addresses is too big, and a class C network, with 256 addresses is too small. A class B network, with 65,536, is just right. In Internet folklore, this situation is known as the **three bears problem** (as in *Goldilocks and the Three Bears*).

In reality, a class B address is far too large for most organizations. Studies have shown that more than half of all class B networks have fewer than 50 hosts. A class C network would have done the job, but no doubt every organization that asked for a class B address thought that one day it would outgrow the 8-bit host field. In retrospect, it might have been better to have had class C networks use 10 bits instead of eight for the host number, allowing 1022 hosts per network. Had this been the case, most organizations would have probably settled for a class C network, and there would have been half a million of them (versus only 16,384 class B networks).

It is hard to fault the Internet designers for not having provided more (and smaller) class B addresses. At the time the decision was made to create the three classes, the Internet was a research network connecting the major research universities in the U.S. (plus a very small number of companies and military sites doing networking research). No one then perceived the Internet as becoming a mass market communication system rivaling the telephone network. At the time, someone no doubt said: "The U.S. has about 2000 colleges and universities. Even if all of them connect to the Internet and many universities in other countries join, too, we are never going to hit 16,000 since there are not that many universities in the whole world. Furthermore, having the host number be an integral number of bytes speeds up packet processing."

However, if the split had allocated 20 bits to the class B network number, another problem would have emerged: the routing table explosion. From the point of view of the routers, the IP address space is a two-level hierarchy, with network numbers and host numbers. Routers do not have to know about all the hosts, but they do have to know about all the networks. If half a million class C networks were in use, every router in the entire Internet would need a table with half a million entries, one per network, telling which line to use to get to that network, as well as providing other information.

The actual physical storage of half a million entry tables is probably doable, although expensive for critical routers that keep the tables in static RAM on I/O boards. A more serious problem is that the complexity of various algorithms relating to management of the tables grows faster than linear. Worse yet, much of the existing router software and firmware was designed at a time when the Internet had 1000 connected networks and 10,000 networks seemed decades away. Design choices made then often are far from optimal now.

In addition, various routing algorithms require each router to transmit its tables periodically (e.g.,
distance vector protocols). The larger the tables, the more likely it is that some parts will get lost underway, leading to incomplete data at the other end and possibly routing instabilities.

The routing table problem could have been solved by going to a deeper hierarchy. For example, having each IP address contain a country, state/province, city, network, and host field might work. Then each router would only need to know how to get to each country, the states or provinces in its own country, the cities in its state or province, and the networks in its city. Unfortunately, this solution would require considerably more than 32 bits for IP addresses and would use addresses inefficiently (Liechtenstein would have as many bits as the United States).

In short, some solutions solve one problem but create a new one. The solution that was implemented and that gave the Internet a bit of extra breathing room is **CIDR (Classless InterDomain Routing)**. The basic idea behind CIDR, which is described in RFC 1519, is to allocate the remaining IP addresses in variable-sized blocks, without regard to the classes. If a site needs, say, 2000 addresses, it is given a block of 2048 addresses on a 2048-byte boundary.

Dropping the classes makes forwarding more complicated. In the old classful system, forwarding worked like this. When a packet arrived at a router, a copy of the IP address was shifted right 28 bits to yield a 4-bit class number. A 16-way branch then sorted packets into A, B, C, and D (if supported), with eight of the cases for class A, four of the cases for class B, two of the cases for class C, and one each for D and E. The code for each class then masked off the 8-, 16-, or 24-bit network number and right aligned it in a 32-bit word. The network number was then looked up in the A, B, or C table, usually by indexing for A and B networks and hashing for C networks. Once the entry was found, the outgoing line could be looked up and the packet forwarded.

With CIDR, this simple algorithm no longer works. Instead, each routing table entry is extended by giving it a 32-bit mask. Thus, there is now a single routing table for all networks consisting of an array of (IP address, subnet mask, outgoing line) triples. When a packet comes in, its destination IP address is first extracted. Then (conceptually) the routing table is scanned entry by entry, masking the destination address and comparing it to the table entry looking for a match. It is possible that multiple entries (with different subnet mask lengths) match, in which case the longest mask is used. Thus, if there is a match for a /20 mask and a /24 mask, the /24 entry is used.

Complex algorithms have been devised to speed up the address matching process (Ruiz-Sanchez et al., 2001). Commercial routers use custom VLSI chips with these algorithms embedded in hardware.

To make the forwarding algorithm easier to understand, let us consider an example in which millions of addresses are available starting at 194.24.0.0. Suppose that Cambridge University needs 2048 addresses and is assigned the addresses 194.24.0.0 through 194.24.7.255, along with mask 255.255.248.0. Next, Oxford University asks for 4096 addresses. Since a block of 4096 addresses must lie on a 4096-byte boundary, they cannot be given addresses starting at 194.24.8.0. Instead, they get 194.24.16.0 through 194.24.31.255 along with subnet mask 255.255.252.0. Now the University of Edinburgh asks for 1024 addresses and is assigned addresses 194.24.8.0 through 194.24.11.255 and mask 255.255.252.0. These assignments are summarized in **Fig. 5-59**.

**Figure 5-59. A set of IP address assignments.**
The routing tables all over the world are now updated with the three assigned entries. Each entry contains a base address and a subnet mask. These entries (in binary) are:

<table>
<thead>
<tr>
<th>University</th>
<th>First address</th>
<th>Last address</th>
<th>How many</th>
<th>Written as</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cambridge</td>
<td>194.24.0.0</td>
<td>194.24.255</td>
<td>2048</td>
<td>194.24.0.0/21</td>
</tr>
<tr>
<td>Edinburgh</td>
<td>194.24.8.0</td>
<td>194.24.11.255</td>
<td>1024</td>
<td>194.24.8.0/22</td>
</tr>
<tr>
<td>Oxford</td>
<td>194.24.16.0</td>
<td>194.24.31.255</td>
<td>4096</td>
<td>194.24.16.0/20</td>
</tr>
</tbody>
</table>

Now consider what happens when a packet comes in addressed to 194.24.17.4, which in binary is represented as the following 32-bit string

```
11000010 00011000 00010001 00000100
```

First it is Boolean ANDed with the Cambridge mask to get

```
11000010 00011000 00010000 00000000
```

This value does not match the Cambridge base address, so the original address is next ANDed with the Edinburgh mask to get

```
11000010 00011000 00010000 00000000
```

This value does not match the Edinburgh base address, so Oxford is tried next, yielding

```
11000010 00011000 00010000 00000000
```

This value does match the Oxford base. If no longer matches are found farther down the table, the Oxford entry is used and the packet is sent along the line named in it.

Now let us look at these three universities from the point of view of a router in Omaha, Nebraska, that has only four outgoing lines: Minneapolis, New York, Dallas, and Denver. When the router software there gets the three new entries, it notices that it can combine all three entries into a single aggregate entry 194.24.0.0/19 with a binary address and submask as follows:

```
11000010 00000000 00000000 00000000 11111111 11111111 11100000 00000000
```

This entry sends all packets destined for any of the three universities to New York. By aggregating the three entries, the Omaha router has reduced its table size by two entries.

If New York has a single line to London for all U.K. traffic, it can use an aggregated entry as well. However, if it has separate lines for London and Edinburgh, then it has to have three separate entries. Aggregation is heavily used throughout the Internet to reduce the size of the router tables.

As a final note on this example, the aggregate route entry in Omaha also sends packets for the unassigned addresses to New York. As long as the addresses are truly unassigned, this does not matter because they are not supposed to occur. However, if they are later assigned to a company...
in California, an additional entry, 194.24.12.0/22, will be needed to deal with them.

**NAT—Network Address Translation**

IP addresses are scarce. An ISP might have a /16 (formerly class B) address, giving it 65,534 host numbers. If it has more customers than that, it has a problem. For home customers with dial-up connections, one way around the problem is to dynamically assign an IP address to a computer when it calls up and logs in and take the IP address back when the session ends. In this way, a single /16 address can handle up to 65,534 active users, which is probably good enough for an ISP with several hundred thousand customers. When the session is terminated, the IP address is reassigned to another caller. While this strategy works well for an ISP with a moderate number of home users, it fails for ISPs that primarily serve business customers.

The problem is that business customers expect to be on-line continuously during business hours. Both small businesses, such as three-person travel agencies, and large corporations have multiple computers connected by a LAN. Some computers are employee PCs; others may be Web servers. Generally, there is a router on the LAN that is connected to the ISP by a leased line to provide continuous connectivity. This arrangement means that each computer must have its own IP address all day long. In effect, the total number of computers owned by all its business customers combined cannot exceed the number of IP addresses the ISP has. For a /16 address, this limits the total number of computers to 65,534. For an ISP with tens of thousands of business customers, this limit will quickly be exceeded.

To make matters worse, more and more home users are subscribing to ADSL or Internet over cable. Two of the features of these services are (1) the user gets a permanent IP address and (2) there is no connect charge (just a monthly flat rate charge), so many ADSL and cable users just stay logged in permanently. This development just adds to the shortage of IP addresses. Assigning IP addresses on-the-fly as is done with dial-up users is of no use because the number of IP addresses in use at any one instant may be many times the number the ISP owns.

And just to make it a bit more complicated, many ADSL and cable users have two or more computers at home, often one for each family member, and they all want to be on-line all the time using the single IP address their ISP has given them. The solution here is to connect all the PCs via a LAN and put a router on it. From the ISP's point of view, the family is now the same as a small business with a handful of computers. Welcome to Jones, Inc.

The problem of running out of IP addresses is not a theoretical problem that might occur at some point in the distant future. It is happening right here and right now. The long-term solution is for the whole Internet to migrate to IPv6, which has 128-bit addresses. This transition is slowly occurring, but it will be years before the process is complete. As a consequence, some people felt that a quick fix was needed for the short term. This quick fix came in the form of **NAT (Network Address Translation)**, which is described in RFC 3022 and which we will summarize below. For additional information, see (Dutcher, 2001).

The basic idea behind NAT is to assign each company a single IP address (or at most, a small number of them) for Internet traffic. *Within* the company, every computer gets a unique IP address, which is used for routing intramural traffic. However, when a packet exits the company and goes to the ISP, an address translation takes place. To make this scheme possible, three ranges of IP addresses have been declared as private. Companies may use them internally as they wish. The only rule is that no packets containing these addresses may appear on the Internet itself. The three reserved ranges are:

```
10.0.0.0   -   10.255.255.255/8   (16,777,216 hosts)
172.16.0.0 -   172.31.255.255/12 (1,048,576 hosts)
```
The first range provides for 16,777,216 addresses (except for 0 and -1, as usual) and is the usual choice of most companies, even if they do not need so many addresses.

The operation of NAT is shown in Fig. 5-60. Within the company premises, every machine has a unique address of the form 10.x.y.z. However, when a packet leaves the company premises, it passes through a NAT box that converts the internal IP source address, 10.0.0.1 in the figure, to the company’s true IP address, 198.60.42.12 in this example. The NAT box is often combined in a single device with a firewall, which provides security by carefully controlling what goes into the company and what comes out. We will study firewalls in Chap. 8. It is also possible to integrate the NAT box into the company’s router.

**Figure 5-60. Placement and operation of a NAT box.**

So far we have glossed over one tiny little detail: when the reply comes back (e.g., from a Web server), it is naturally addressed to 198.60.42.12, so how does the NAT box know which address to replace it with? Herein lies the problem with NAT. If there were a spare field in the IP header, that field could be used to keep track of who the real sender was, but only 1 bit is still unused. In principle, a new option could be created to hold the true source address, but doing so would require changing the IP code on all the machines on the entire Internet to handle the new option. This is not a promising alternative for a quick fix.

What actually happened is as follows. The NAT designers observed that most IP packets carry either TCP or UDP payloads. When we study TCP and UDP in Chap. 6, we will see that both of these have headers containing a source port and a destination port. Below we will just discuss TCP ports, but exactly the same story holds for UDP ports. The ports are 16-bit integers that indicate where the TCP connection begins and ends. These ports provide the field needed to make NAT work.

When a process wants to establish a TCP connection with a remote process, it attaches itself to an unused TCP port on its own machine. This is called the source port and tells the TCP code where to send incoming packets belonging to this connection. The process also supplies a destination port to tell who to give the packets to on the remote side. Ports 0–1023 are reserved for well-known services. For example, port 80 is the port used by Web servers, so remote clients can locate them. Each outgoing TCP message contains both a source port and a destination port. Together, these ports serve to identify the processes using the connection on both ends.

An analogy may make the use of ports clearer. Imagine a company with a single main telephone number. When people call the main number, they reach an operator who asks which extension
they want and then puts them through to that extension. The main number is analogous to the company's IP address and the extensions on both ends are analogous to the ports. Ports are an extra 16-bits of addressing that identify which process gets which incoming packet.

Using the Source port field, we can solve our mapping problem. Whenever an outgoing packet enters the NAT box, the 10.x.y.z source address is replaced by the company's true IP address. In addition, the TCP Source port field is replaced by an index into the NAT box's 65,536-entry translation table. This table entry contains the original IP address and the original source port. Finally, both the IP and TCP header checksums are recomputed and inserted into the packet. It is necessary to replace the Source port because connections from machines 10.0.0.1 and 10.0.0.2 may both happen to use port 5000, for example, so the Source port alone is not enough to identify the sending process.

When a packet arrives at the NAT box from the ISP, the Source port in the TCP header is extracted and used as an index into the NAT box's mapping table. From the entry located, the internal IP address and original TCP Source port are extracted and inserted into the packet. Then both the IP and TCP checksums are recomputed and inserted into the packet. The packet is then passed to the company router for normal delivery using the 10.x.y.z address.

NAT can also be used to alleviate the IP shortage for ADSL and cable users. When the ISP assigns each user an address, it uses 10.x.y.z addresses. When packets from user machines exit the ISP and enter the main Internet, they pass through a NAT box that translates them to the ISP's true Internet address. On the way back, packets undergo the reverse mapping. In this respect, to the rest of the Internet, the ISP and its home ADSL/cable users just looks like a big company.

Although this scheme sort of solves the problem, many people in the IP community regard it as an abomination-on-the-face-of-the-earth. Briefly summarized, here are some of the objections. First, NAT violates the architectural model of IP, which states that every IP address uniquely identifies a single machine worldwide. The whole software structure of the Internet is built on this fact. With NAT, thousands of machines may (and do) use address 10.0.0.1.

Second, NAT changes the Internet from a connectionless network to a kind of connection-oriented network. The problem is that the NAT box must maintain information (the mapping) for each connection passing through it. Having the network maintain connection state is a property of connection-oriented networks, not connectionless ones. If the NAT box crashes and its mapping table is lost, all its TCP connections are destroyed. In the absence of NAT, router crashes have no effect on TCP. The sending process just times out within a few seconds and retransmits all unacknowledged packets. With NAT, the Internet becomes as vulnerable as a circuit-switched network.

Third, NAT violates the most fundamental rule of protocol layering: layer $k$ may not make any assumptions about what layer $k + 1$ has put into the payload field. This basic principle is there to keep the layers independent. If TCP is later upgraded to TCP-2, with a different header layout (e.g., 32-bit ports), NAT will fail. The whole idea of layered protocols is to ensure that changes in one layer do not require changes in other layers. NAT destroys this independence.

Fourth, processes on the Internet are not required to use TCP or UDP. If a user on machine $A$ decides to use some new transport protocol to talk to a user on machine $B$ (for example, for a multimedia application), introduction of a NAT box will cause the application to fail because the NAT box will not be able to locate the TCP Source port correctly.

Fifth, some applications insert IP addresses in the body of the text. The receiver then extracts these addresses and uses them. Since NAT knows nothing about these addresses, it cannot replace them, so any attempt to use them on the remote side will fail. FTP, the standard File Transfer Protocol...
Protocol works this way and can fail in the presence of NAT unless special precautions are taken. Similarly, the H.323 Internet telephony protocol (which we will study in Chap. 7) has this property and can fail in the presence of NAT. It may be possible to patch NAT to work with H.323, but having to patch the code in the NAT box every time a new application comes along is not a good idea.

Sixth, since the TCP Source port field is 16 bits, at most 65,536 machines can be mapped onto an IP address. Actually, the number is slightly less because the first 4096 ports are reserved for special uses. However, if multiple IP addresses are available, each one can handle up to 61,440 machines.

These and other problems with NAT are discussed in RFC 2993. In general, the opponents of NAT say that by fixing the problem of insufficient IP addresses with a temporary and ugly hack, the pressure to implement the real solution, that is, the transition to IPv6, is reduced, and this is a bad thing.

5.6.3 Internet Control Protocols

In addition to IP, which is used for data transfer, the Internet has several control protocols used in the network layer, including ICMP, ARP, RARP, BOOTP, and DHCP. In this section we will look at each of these in turn.

The Internet Control Message Protocol

The operation of the Internet is monitored closely by the routers. When something unexpected occurs, the event is reported by the ICMP (Internet Control Message Protocol), which is also used to test the Internet. About a dozen types of ICMP messages are defined. The most important ones are listed in Fig. 5-61. Each ICMP message type is encapsulated in an IP packet.

**Figure 5-61. The principal ICMP message types.**

<table>
<thead>
<tr>
<th>Message type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination unreachable</td>
<td>Packet could not be delivered</td>
</tr>
<tr>
<td>Time exceeded</td>
<td>Time to live field hit 0</td>
</tr>
<tr>
<td>Parameter problem</td>
<td>Invalid header field</td>
</tr>
<tr>
<td>Source quench</td>
<td>Choke packet</td>
</tr>
<tr>
<td>Redirect</td>
<td>Teach a router about geography</td>
</tr>
<tr>
<td>Echo</td>
<td>Ask a machine if it is alive</td>
</tr>
<tr>
<td>Echo reply</td>
<td>Yes, I am alive</td>
</tr>
<tr>
<td>Timestamp request</td>
<td>Same as Echo request, but with timestamp</td>
</tr>
<tr>
<td>Timestamp reply</td>
<td>Same as Echo reply, but with timestamp</td>
</tr>
</tbody>
</table>

The DESTINATION UNREACHABLE message is used when the subnet or a router cannot locate the destination or when a packet with the DF bit cannot be delivered because a "small-packet" network stands in the way.

The TIME EXCEEDED message is sent when a packet is dropped because its counter has reached zero. This event is a symptom that packets are looping, that there is enormous congestion, or that the timer values are being set too low.

The PARAMETER PROBLEM message indicates that an illegal value has been detected in a header
field. This problem indicates a bug in the sending host's IP software or possibly in the software of a router transited.

The SOURCE QUENCH message was formerly used to throttle hosts that were sending too many packets. When a host received this message, it was expected to slow down. It is rarely used any more because when congestion occurs, these packets tend to add more fuel to the fire. Congestion control in the Internet is now done largely in the transport layer; we will study it in detail in Chap. 6.

The REDIRECT message is used when a router notices that a packet seems to be routed wrong. It is used by the router to tell the sending host about the probable error.

The ECHO and ECHO REPLY messages are used to see if a given destination is reachable and alive. Upon receiving the ECHO message, the destination is expected to send an ECHO REPLY message back. The TIMESTAMP REQUEST and TIMESTAMP REPLY messages are similar, except that the arrival time of the message and the departure time of the reply are recorded in the reply. This facility is used to measure network performance.

In addition to these messages, others have been defined. The on-line list is now kept at www.iana.org/assignments/icmp-parameters.

ARP—The Address Resolution Protocol

Although every machine on the Internet has one (or more) IP addresses, these cannot actually be used for sending packets because the data link layer hardware does not understand Internet addresses. Nowadays, most hosts at companies and universities are attached to a LAN by an interface board that only understands LAN addresses. For example, every Ethernet board ever manufactured comes equipped with a 48-bit Ethernet address. Manufacturers of Ethernet boards request a block of addresses from a central authority to ensure that no two boards have the same address (to avoid conflicts should the two boards ever appear on the same LAN). The boards send and receive frames based on 48-bit Ethernet addresses. They know nothing at all about 32-bit IP addresses.

The question now arises: How do IP addresses get mapped onto data link layer addresses, such as Ethernet? To explain how this works, let us use the example of Fig. 5-62, in which a small university with several class C (now called /24) networks is illustrated. Here we have two Ethernets, one in the Computer Science Dept., with IP address 192.31.65.0 and one in Electrical Engineering, with IP address 192.31.63.0. These are connected by a campus backbone ring (e.g., FDDI) with IP address 192.31.60.0. Each machine on an Ethernet has a unique Ethernet address, labeled $E1$ through $E6$, and each machine on the FDDI ring has an FDDI address, labeled $F1$ through $F3$.

**Figure 5-62. Three interconnected /24 networks: two Ethernets and an FDDI ring.**
Let us start out by seeing how a user on host 1 sends a packet to a user on host 2. Let us assume the sender knows the name of the intended receiver, possibly something like mary@eagle.cs.uni.edu. The first step is to find the IP address for host 2, known as eagle.cs.uni.edu. This lookup is performed by the Domain Name System, which we will study in Chap. 7. For the moment, we will just assume that DNS returns the IP address for host 2 (192.31.65.5).

The upper layer software on host 1 now builds a packet with 192.31.65.5 in the Destination address field and gives it to the IP software to transmit. The IP software can look at the address and see that the destination is on its own network, but it needs some way to find the destination's Ethernet address. One solution is to have a configuration file somewhere in the system that maps IP addresses onto Ethernet addresses. While this solution is certainly possible, for organizations with thousands of machines, keeping all these files up to date is an error-prone, time-consuming job.

A better solution is for host 1 to output a broadcast packet onto the Ethernet asking: Who owns IP address 192.31.65.5? The broadcast will arrive at every machine on Ethernet 192.31.65.0, and each one will check its IP address. Host 2 alone will respond with its Ethernet address (E2). In this way host 1 learns that IP address 192.31.65.5 is on the host with Ethernet address E2. The protocol used for asking this question and getting the reply is called ARP (Address Resolution Protocol). Almost every machine on the Internet runs it. ARP is defined in RFC 826.

The advantage of using ARP over configuration files is the simplicity. The system manager does not have to do much except assign each machine an IP address and decide about subnet masks. ARP does the rest.

At this point, the IP software on host 1 builds an Ethernet frame addressed to E2, puts the IP packet (addressed to 192.31.65.5) in the payload field, and dumps it onto the Ethernet. The Ethernet board of host 2 detects this frame, recognizes it as a frame for itself, scoops it up, and causes an interrupt. The Ethernet driver extracts the IP packet from the payload and passes it to the IP software, which sees that it is correctly addressed and processes it.

Various optimizations are possible to make ARP work more efficiently. To start with, once a machine has run ARP, it caches the result in case it needs to contact the same machine shortly. Next time it will find the mapping in its own cache, thus eliminating the need for a second broadcast. In many cases host 2 will need to send back a reply, forcing it, too, to run ARP to determine the sender's Ethernet address. This ARP broadcast can be avoided by having host 1 include its IP-to-Ethernet mapping in the ARP packet. When the ARP broadcast arrives at host 2, the pair (192.31.65.7, E1) is entered into host 2's ARP cache for future use. In fact, all machines on the Ethernet can enter this mapping into their ARP caches.

Yet another optimization is to have every machine broadcast its mapping when it boots. This
broadcast is generally done in the form of an ARP looking for its own IP address. There should not be a response, but a side effect of the broadcast is to make an entry in everyone's ARP cache. If a response does (unexpectedly) arrive, two machines have been assigned the same IP address. The new one should inform the system manager and not boot.

To allow mappings to change, for example, when an Ethernet board breaks and is replaced with a new one (and thus a new Ethernet address), entries in the ARP cache should time out after a few minutes.

Now let us look at Fig. 5-62 again, only this time host 1 wants to send a packet to host 4 (192.31.63.8). Using ARP will fail because host 4 will not see the broadcast (routers do not forward Ethernet-level broadcasts). There are two solutions. First, the CS router could be configured to respond to ARP requests for network 192.31.63.0 (and possibly other local networks). In this case, host 1 will make an ARP cache entry of (192.31.63.8, E3) and happily send all traffic for host 4 to the local router. This solution is called proxy ARP. The second solution is to have host 1 immediately see that the destination is on a remote network and just send all such traffic to a default Ethernet address that handles all remote traffic, in this case E3. This solution does not require having the CS router know which remote networks it is serving.

Either way, what happens is that host 1 packs the IP packet into the payload field of an Ethernet frame addressed to E3. When the CS router gets the Ethernet frame, it removes the IP packet from the payload field and looks up the IP address in its routing tables. It discovers that packets for network 192.31.63.0 are supposed to go to router 192.31.60.7. If it does not already know the FDDI address of 192.31.60.7, it broadcasts an ARP packet onto the ring and learns that its ring address is F3. It then inserts the packet into the payload field of an FDDI frame addressed to F3 and puts it on the ring.

At the EE router, the FDDI driver removes the packet from the payload field and gives it to the IP software, which sees that it needs to send the packet to 192.31.63.8. If this IP address is not in its ARP cache, it broadcasts an ARP request on the EE Ethernet and learns that the destination address is E6, so it builds an Ethernet frame addressed to E6, puts the packet in the payload field, and sends it over the Ethernet. When the Ethernet frame arrives at host 4, the packet is extracted from the frame and passed to the IP software for processing.

Going from host 1 to a distant network over a WAN works essentially the same way, except that this time the CS router's tables tell it to use the WAN router whose FDDI address is F2.

**RARP, BOOTP, and DHCP**

ARP solves the problem of finding out which Ethernet address corresponds to a given IP address. Sometimes the reverse problem has to be solved: Given an Ethernet address, what is the corresponding IP address? In particular, this problem occurs when a diskless workstation is booted. Such a machine will normally get the binary image of its operating system from a remote file server. But how does it learn its IP address?

The first solution devised was to use RARP (Reverse Address Resolution Protocol) (defined in RFC 903). This protocol allows a newly-booted workstation to broadcast its Ethernet address and say: My 48-bit Ethernet address is 14.04.05.18.01.25. Does anyone out there know my IP address? The RARP server sees this request, looks up the Ethernet address in its configuration files, and sends back the corresponding IP address.

Using RARP is better than embedding an IP address in the memory image because it allows the same image to be used on all machines. If the IP address were buried inside the image, each workstation would need its own image.
A disadvantage of RARP is that it uses a destination address of all 1s (limited broadcasting) to reach the RARP server. However, such broadcasts are not forwarded by routers, so a RARP server is needed on each network. To get around this problem, an alternative bootstrap protocol called **BOOTP** was invented. Unlike RARP, BOOTP uses UDP messages, which are forwarded over routers. It also provides a diskless workstation with additional information, including the IP address of the file server holding the memory image, the IP address of the default router, and the subnet mask to use. BOOTP is described in RFCs 951, 1048, and 1084.

A serious problem with BOOTP is that it requires manual configuration of tables mapping IP address to Ethernet address. When a new host is added to a LAN, it cannot use BOOTP until an administrator has assigned it an IP address and entered its (Ethernet address, IP address) into the BOOTP configuration tables by hand. To eliminate this error-prone step, BOOTP was extended and given a new name: **DHCP (Dynamic Host Configuration Protocol)**. DHCP allows both manual IP address assignment and automatic assignment. It is described in RFCs 2131 and 2132. In most systems, it has largely replaced RARP and BOOTP.

Like RARP and BOOTP, DHCP is based on the idea of a special server that assigns IP addresses to hosts asking for one. This server need not be on the same LAN as the requesting host. Since the DHCP server may not be reachable by broadcasting, a **DHCP relay agent** is needed on each LAN, as shown in [Fig. 5-63](#).

**Figure 5-63. Operation of DHCP.**

To find its IP address, a newly-booted machine broadcasts a DHCP DISCOVER packet. The DHCP relay agent on its LAN intercepts all DHCP broadcasts. When it finds a DHCP DISCOVER packet, it sends the packet as a unicast packet to the DHCP server, possibly on a distant network. The only piece of information the relay agent needs is the IP address of the DHCP server.

An issue that arises with automatic assignment of IP addresses from a pool is how long an IP address should be allocated. If a host leaves the network and does not return its IP address to the DHCP server, that address will be permanently lost. After a period of time, many addresses may be lost. To prevent that from happening, IP address assignment may be for a fixed period of time, a technique called **leasing**. Just before the lease expires, the host must ask the DHCP for a renewal. If it fails to make a request or the request is denied, the host may no longer use the IP address it was given earlier.

### 5.6.4 OSPF—The Interior Gateway Routing Protocol

We have now finished our study of Internet control protocols. It is time to move on the next topic: routing in the Internet. As we mentioned earlier, the Internet is made up of a large number of autonomous systems. Each AS is operated by a different organization and can use its own routing algorithm inside. For example, the internal networks of companies X, Y, and Z are usually seen as three ASes if all three are on the Internet. All three may use different routing algorithms internally. Nevertheless, having standards, even for internal routing, simplifies the implementation at the
boundaries between ASes and allows reuse of code. In this section we will study routing within an AS. In the next one, we will look at routing between ASes. A routing algorithm within an AS is called an **interior gateway protocol**; an algorithm for routing between ASes is called an **exterior gateway protocol**.

The original Internet interior gateway protocol was a distance vector protocol (RIP) based on the Bellman-Ford algorithm inherited from the ARPANET. It worked well in small systems, but less well as ASes got larger. It also suffered from the count-to-infinity problem and generally slow convergence, so it was replaced in May 1979 by a link state protocol. In 1988, the Internet Engineering Task Force began work on a successor. That successor, called **OSPF (Open Shortest Path First)**, became a standard in 1990. Most router vendors now support it, and it has become the main interior gateway protocol. Below we will give a sketch of how OSPF works. For the complete story, see RFC 2328.

Given the long experience with other routing protocols, the group designing the new protocol had a long list of requirements that had to be met. First, the algorithm had to be published in the open literature, hence the "O" in OSPF. A proprietary solution owned by one company would not do. Second, the new protocol had to support a variety of distance metrics, including physical distance, delay, and so on. Third, it had to be a dynamic algorithm, one that adapted to changes in the topology automatically and quickly.

Fourth, and new for OSPF, it had to support routing based on type of service. The new protocol had to be able to route real-time traffic one way and other traffic a different way. The IP protocol has a **Type of Service** field, but no existing routing protocol used it. This field was included in OSPF but still nobody used it, and it was eventually removed.

Fifth, and related to the above, the new protocol had to do load balancing, splitting the load over multiple lines. Most previous protocols sent all packets over the best route. The second-best route was not used at all. In many cases, splitting the load over multiple lines gives better performance.

Sixth, support for hierarchical systems was needed. By 1988, the Internet had grown so large that no router could be expected to know the entire topology. The new routing protocol had to be designed so that no router would have to.

Seventh, some modicum of security was required to prevent fun-loving students from spoofing routers by sending them false routing information. Finally, provision was needed for dealing with routers that were connected to the Internet via a tunnel. Previous protocols did not handle this well.

OSPF supports three kinds of connections and networks:

1. Point-to-point lines between exactly two routers.
2. Multiaccess networks with broadcasting (e.g., most LANs).
3. Multiaccess networks without broadcasting (e.g., most packet-switched WANs).

A multiaccess network is one that can have multiple routers on it, each of which can directly communicate with all the others. All LANs and WANs have this property. Figure 5-64(a) shows an AS containing all three kinds of networks. Note that hosts do not generally play a role in OSPF.

**Figure 5-64. (a) An autonomous system. (b) A graph representation of (a).**
OSPF operates by abstracting the collection of actual networks, routers, and lines into a directed graph in which each arc is assigned a cost (distance, delay, etc.). It then computes the shortest path based on the weights on the arcs. A serial connection between two routers is represented by a pair of arcs, one in each direction. Their weights may be different. A multiaccess network is represented by a node for the network itself plus a node for each router. The arcs from the network node to the routers have weight 0 and are omitted from the graph.

Figure 5-64(b) shows the graph representation of the network of Fig. 5-64(a). Weights are symmetric, unless marked otherwise. What OSPF fundamentally does is represent the actual network as a graph like this and then compute the shortest path from every router to every other router.

Many of the ASes in the Internet are themselves large and nontrivial to manage. OSPF allows them to be divided into numbered areas, where an area is a network or a set of contiguous networks. Areas do not overlap but need not be exhaustive, that is, some routers may belong to no area. An area is a generalization of a subnet. Outside an area, its topology and details are not visible.

Every AS has a backbone area, called area 0. All areas are connected to the backbone, possibly by tunnels, so it is possible to go from any area in the AS to any other area in the AS via the backbone. A tunnel is represented in the graph as an arc and has a cost. Each router that is connected to two or more areas is part of the backbone. As with other areas, the topology of the backbone is not visible outside the backbone.

Within an area, each router has the same link state database and runs the same shortest path algorithm. Its main job is to calculate the shortest path from itself to every other router in the area, including the router that is connected to the backbone, of which there must be at least one. A router that connects to two areas needs the databases for both areas and must run the shortest path algorithm for each one separately.
During normal operation, three kinds of routes may be needed: intra-area, interarea, and inter-AS. Intra-area routes are the easiest, since the source router already knows the shortest path to the destination router. Interarea routing always proceeds in three steps: go from the source to the backbone; go across the backbone to the destination area; go to the destination. This algorithm forces a star configuration on OSPF with the backbone being the hub and the other areas being spokes. Packets are routed from source to destination "as is." They are not encapsulated or tunneled, unless going to an area whose only connection to the backbone is a tunnel. Figure 5-65 shows part of the Internet with ASes and areas.

**Figure 5-65. The relation between ASes, backbones, and areas in OSPF.**

OSPF distinguishes four classes of routers:

1. Internal routers are wholly within one area.
2. Area border routers connect two or more areas.
3. Backbone routers are on the backbone.
4. AS boundary routers talk to routers in other ASes.

These classes are allowed to overlap. For example, all the border routers are automatically part of the backbone. In addition, a router that is in the backbone but not part of any other area is also an internal router. Examples of all four classes of routers are illustrated in Fig. 5-65.

When a router boots, it sends HELLO messages on all of its point-to-point lines and multicasts them on LANs to the group consisting of all the other routers. On WANs, it needs some configuration information to know who to contact. From the responses, each router learns who its neighbors are. Routers on the same LAN are all neighbors.
OSPF works by exchanging information between adjacent routers, which is not the same as between neighboring routers. In particular, it is inefficient to have every router on a LAN talk to every other router on the LAN. To avoid this situation, one router is elected as the designated router. It is said to be adjacent to all the other routers on its LAN, and exchanges information with them. Neighboring routers that are not adjacent do not exchange information with each other. A backup designated router is always kept up to date to ease the transition should the primary designated router crash and need to be replaced immediately.

During normal operation, each router periodically floods LINK STATE UPDATE messages to each of its adjacent routers. This message gives its state and provides the costs used in the topological database. The flooding messages are acknowledged, to make them reliable. Each message has a sequence number, so a router can see whether an incoming LINK STATE UPDATE is older or newer than what it currently has. Routers also send these messages when a line goes up or down or its cost changes.

DATABASE DESCRIPTION messages give the sequence numbers of all the link state entries currently held by the sender. By comparing its own values with those of the sender, the receiver can determine who has the most recent values. These messages are used when a line is brought up.

Either partner can request link state information from the other one by using LINK STATE REQUEST messages. The result of this algorithm is that each pair of adjacent routers checks to see who has the most recent data, and new information is spread throughout the area this way. All these messages are sent as raw IP packets. The five kinds of messages are summarized in Fig. 5-66.

Finally, we can put all the pieces together. Using flooding, each router informs all the other routers in its area of its neighbors and costs. This information allows each router to construct the graph for its area(s) and compute the shortest path. The backbone area does this too. In addition, the backbone routers accept information from the area border routers in order to compute the best route from each backbone router to every other router. This information is propagated back to the area border routers, which advertise it within their areas. Using this information, a router about to send an interarea packet can select the best exit router to the backbone.

### 5.6.5 BGP—The Exterior Gateway Routing Protocol

Within a single AS, the recommended routing protocol is OSPF (although it is certainly not the only one in use). Between ASes, a different protocol, BGP (Border Gateway Protocol), is used. A different protocol is needed between ASes because the goals of an interior gateway protocol and an exterior gateway protocol are not the same. All an interior gateway protocol has to do is move packets as efficiently as possible from the source to the destination. It does not have to worry about politics.

Exterior gateway protocol routers have to worry about politics a great deal (Metz, 2001). For
example, a corporate AS might want the ability to send packets to any Internet site and receive packets from any Internet site. However, it might be unwilling to carry transit packets originating in a foreign AS and ending in a different foreign AS, even if its own AS was on the shortest path between the two foreign ASes ("That's their problem, not ours"). On the other hand, it might be willing to carry transit traffic for its neighbors or even for specific other ASes that paid it for this service. Telephone companies, for example, might be happy to act as a carrier for their customers, but not for others. Exterior gateway protocols in general, and BGP in particular, have been designed to allow many kinds of routing policies to be enforced in the interAS traffic.

Typical policies involve political, security, or economic considerations. A few examples of routing constraints are:

1. No transit traffic through certain ASes.
2. Never put Iraq on a route starting at the Pentagon.
3. Do not use the United States to get from British Columbia to Ontario.
4. Only transit Albania if there is no alternative to the destination.
5. Traffic starting or ending at IBM should not transit Microsoft.

Policies are typically manually configured into each BGP router (or included using some kind of script). They are not part of the protocol itself.

From the point of view of a BGP router, the world consists of ASes and the lines connecting them. Two ASes are considered connected if there is a line between a border router in each one. Given BGP's special interest in transit traffic, networks are grouped into one of three categories. The first category is the stub networks, which have only one connection to the BGP graph. These cannot be used for transit traffic because there is no one on the other side. Then come the multiconnected networks. These could be used for transit traffic, except that they refuse. Finally, there are the transit networks, such as backbones, which are willing to handle third-party packets, possibly with some restrictions, and usually for pay.

Pairs of BGP routers communicate with each other by establishing TCP connections. Operating this way provides reliable communication and hides all the details of the network being passed through.

BGP is fundamentally a distance vector protocol, but quite different from most others such as RIP. Instead of maintaining just the cost to each destination, each BGP router keeps track of the path used. Similarly, instead of periodically giving each neighbor its estimated cost to each possible destination, each BGP router tells its neighbors the exact path it is using.

As an example, consider the BGP routers shown in Fig. 5-67(a). In particular, consider F's routing table. Suppose that it uses the path FGCD to get to D. When the neighbors give it routing information, they provide their complete paths, as shown in Fig. 5-67(b) (for simplicity, only destination D is shown here).

**Figure 5-67. (a) A set of BGP routers. (b) Information sent to F.**
After all the paths come in from the neighbors, $F$ examines them to see which is the best. It quickly discards the paths from $I$ and $E$, since these paths pass through $F$ itself. The choice is then between using $B$ and $G$. Every BGP router contains a module that examines routes to a given destination and scores them, returning a number for the "distance" to that destination for each route. Any route violating a policy constraint automatically gets a score of infinity. The router then adopts the route with the shortest distance. The scoring function is not part of the BGP protocol and can be any function the system managers want.

BGP easily solves the count-to-infinity problem that plagues other distance vector routing algorithms. For example, suppose $G$ crashes or the line $FG$ goes down. $F$ then receives routes from its three remaining neighbors. These routes are $BCD$, $IFGCD$, and $EFGCD$. It can immediately see that the two latter routes are pointless, since they pass through $F$ itself, so it chooses $FBCD$ as its new route. Other distance vector algorithms often make the wrong choice because they cannot tell which of their neighbors have independent routes to the destination and which do not. The definition of BGP is in RFCs 1771 to 1774.

### 5.6.6 Internet Multicasting

Normal IP communication is between one sender and one receiver. However, for some applications it is useful for a process to be able to send to a large number of receivers simultaneously. Examples are updating replicated, distributed databases, transmitting stock quotes to multiple brokers, and handling digital conference (i.e., multiparty) telephone calls.

IP supports multicasting, using class D addresses. Each class D address identifies a group of hosts. Twenty-eight bits are available for identifying groups, so over 250 million groups can exist at the same time. When a process sends a packet to a class D address, a best-efforts attempt is made to deliver it to all the members of the group addressed, but no guarantees are given. Some members may not get the packet.

Two kinds of group addresses are supported: permanent addresses and temporary ones. A permanent group is always there and does not have to be set up. Each permanent group has a permanent group address. Some examples of permanent group addresses are:

- 224.0.0.1 All systems on a LAN
- 224.0.0.2 All routers on a LAN
- 224.0.0.5 All OSPF routers on a LAN
- 224.0.0.6 All designated OSPF routers on a LAN
Temporary groups must be created before they can be used. A process can ask its host to join a specific group. It can also ask its host to leave the group. When the last process on a host leaves a group, that group is no longer present on the host. Each host keeps track of which groups its processes currently belong to.

Multicasting is implemented by special multicast routers, which may or may not be colocated with the standard routers. About once a minute, each multicast router sends a hardware (i.e., data link layer) multicast to the hosts on its LAN (address 224.0.0.1) asking them to report back on the groups their processes currently belong to. Each host sends back responses for all the class D addresses it is interested in.

These query and response packets use a protocol called IGMP (Internet Group Management Protocol), which is vaguely analogous to ICMP. It has only two kinds of packets: query and response, each with a simple, fixed format containing some control information in the first word of the payload field and a class D address in the second word. It is described in RFC 1112.

Multicast routing is done using spanning trees. Each multicast router exchanges information with its neighbors, using a modified distance vector protocol in order for each one to construct a spanning tree per group covering all group members. Various optimizations are used to prune the tree to eliminate routers and networks not interested in particular groups. The protocol makes heavy use of tunneling to avoid bothering nodes not in a spanning tree.

5.6.7 Mobile IP

Many users of the Internet have portable computers and want to stay connected to the Internet when they visit a distant Internet site and even on the road in between. Unfortunately, the IP addressing system makes working far from home easier said than done. In this section we will examine the problem and the solution. A more detailed description is given in (Perkins, 1998a).

The real villain is the addressing scheme itself. Every IP address contains a network number and a host number. For example, consider the machine with IP address 160.80.40.20/16. The 160.80 gives the network number (8272 in decimal); the 40.20 is the host number (10260 in decimal). Routers all over the world have routing tables telling which line to use to get to network 160.80. Whenever a packet comes in with a destination IP address of the form 160.80.xxx.yyy, it goes out on that line.

If all of a sudden, the machine with that address is carted off to some distant site, the packets for it will continue to be routed to its home LAN (or router). The owner will no longer get e-mail, and so on. Giving the machine a new IP address corresponding to its new location is unattractive because large numbers of people, programs, and databases would have to be informed of the change.

Another approach is to have the routers use complete IP addresses for routing, instead of just the network. However, this strategy would require each router to have millions of table entries, at astronomical cost to the Internet.

When people began demanding the ability to connect their notebook computers to the Internet wherever they were, IETF set up a Working Group to find a solution. The Working Group quickly formulated a number of goals considered desirable in any solution. The major ones were:

1. Each mobile host must be able to use its home IP address anywhere.
2. Software changes to the fixed hosts were not permitted.
3. Changes to the router software and tables were not permitted.

4. Most packets for mobile hosts should not make detours on the way.

5. No overhead should be incurred when a mobile host is at home.

The solution chosen was the one described in Sec. 5.2.8. To review it briefly, every site that wants to allow its users to roam has to create a home agent. Every site that wants to allow visitors has to create a foreign agent. When a mobile host shows up at a foreign site, it contacts the foreign host there and registers. The foreign host then contacts the user's home agent and gives it a care-of address, normally the foreign agent's own IP address.

When a packet arrives at the user's home LAN, it comes in at some router attached to the LAN. The router then tries to locate the host in the usual way, by broadcasting an ARP packet asking, for example: What is the Ethernet address of 160.80.40.20? The home agent responds to this query by giving its own Ethernet address. The router then sends packets for 160.80.40.20 to the home agent. It, in turn, tunnels them to the care-of address by encapsulating them in the payload field of an IP packet addressed to the foreign agent. The foreign agent then decapsulates and delivers them to the data link address of the mobile host. In addition, the home agent gives the care-of address to the sender, so future packets can be tunneled directly to the foreign agent. This solution meets all the requirements stated above.

One small detail is probably worth mentioning. At the time the mobile host moves, the router probably has its (soon-to-be-invalid) Ethernet address cached. Replacing that Ethernet address with the home agent's is done by a trick called gratuitous ARP. This is a special, unsolicited message to the router that causes it to replace a specific cache entry, in this case, that of the mobile host about to leave. When the mobile host returns later, the same trick is used to update the router's cache again.

Nothing in the design prevents a mobile host from being its own foreign agent, but that approach only works if the mobile host (in its capacity as foreign agent) is logically connected to the Internet at its current site. Also, the mobile host must be able to acquire a (temporary) care-of IP address to use. That IP address must belong to the LAN to which it is currently attached.

The IETF solution for mobile hosts solves a number of other problems not mentioned so far. For example, how are agents located? The solution is for each agent to periodically broadcast its address and the type of services it is willing to provide (e.g., home, foreign, or both). When a mobile host arrives somewhere, it can just listen for these broadcasts, called advertisements. Alternatively, it can broadcast a packet announcing its arrival and hope that the local foreign agent responds to it.

Another problem that had to be solved is what to do about impolite mobile hosts that leave without saying goodbye. The solution is to make registration valid only for a fixed time interval. If it is not refreshed periodically, it times out, so the foreign host can clear its tables.

Yet another issue is security. When a home agent gets a message asking it to please forward all of Roberta's packets to some IP address, it had better not comply unless it is convinced that Roberta is the source of this request, and not somebody trying to impersonate her. Cryptographic authentication protocols are used for this purpose. We will study such protocols in Chap. 8.

A final point addressed by the Working Group relates to levels of mobility. Imagine an airplane with an on-board Ethernet used by the navigation and avionics computers. On this Ethernet is a standard router that talks to the wired Internet on the ground over a radio link. One fine day, some clever marketing executive gets the idea to install Ethernet connectors in all the arm rests so
passengers with mobile computers can also plug in.

Now we have two levels of mobility: the aircraft's own computers, which are stationary with respect to the Ethernet, and the passengers' computers, which are mobile with respect to it. In addition, the on-board router is mobile with respect to routers on the ground. Being mobile with respect to a system that is itself mobile can be handled using recursive tunneling.

### 5.6.8 IPv6

While CIDR and NAT may buy a few more years' time, everyone realizes that the days of IP in its current form (IPv4) are numbered. In addition to these technical problems, another issue looms in the background. In its early years, the Internet was largely used by universities, high-tech industry, and the U.S. Government (especially the Dept. of Defense). With the explosion of interest in the Internet starting in the mid-1990s, it began to be used by a different group of people, especially people with different requirements. For one thing, numerous people with wireless portables use it to keep in contact with their home bases. For another, with the impending convergence of the computer, communication, and entertainment industries, it may not be that long before every telephone and television set in the world is an Internet node, producing a billion machines being used audio and video on demand. Under these circumstances, it became apparent that IP had to evolve and become more flexible.

Seeing these problems on the horizon, in 1990, IETF started work on a new version of IP, one which would never run out of addresses, would solve a variety of other problems, and be more flexible and efficient as well. Its major goals were:

1. Support billions of hosts, even with inefficient address space allocation.
2. Reduce the size of the routing tables.
3. Simplify the protocol, to allow routers to process packets faster.
4. Provide better security (authentication and privacy) than current IP.
5. Pay more attention to type of service, particularly for real-time data.
6. Aid multicasting by allowing scopes to be specified.
7. Make it possible for a host to roam without changing its address.
8. Allow the protocol to evolve in the future.
9. Permit the old and new protocols to coexist for years.

To develop a protocol that met all these requirements, IETF issued a call for proposals and discussion in RFC 1550. Twenty-one responses were received, not all of them full proposals. By December 1992, seven serious proposals were on the table. They ranged from making minor patches to IP, to throwing it out altogether and replacing with a completely different protocol.

One proposal was to run TCP over CLNP, which, with its 160-bit addresses would have provided enough address space forever and would have unified two major network layer protocols. However, many people felt that this would have been an admission that something in the OSI world was actually done right, a statement considered Politically Incorrect in Internet circles. CLNP was patterned closely on IP, so the two are not really that different. In fact, the protocol ultimately
chosen differs from IP far more than CLNP does. Another strike against CLNP was its poor support for service types, something required to transmit multimedia efficiently.

Three of the better proposals were published in *IEEE Network* (Deering, 1993; Francis, 1993; and Katz and Ford, 1993). After much discussion, revision, and jockeying for position, a modified combined version of the Deering and Francis proposals, by now called **SIPP (Simple Internet Protocol Plus)** was selected and given the designation **IPv6**.

IPv6 meets the goals fairly well. It maintains the good features of IP, discards or deemphasizes the bad ones, and adds new ones where needed. In general, IPv6 is not compatible with IPv4, but it is compatible with the other auxiliary Internet protocols, including TCP, UDP, ICMP, IGMP, OSPF, BGP, and DNS, sometimes with small modifications being required (mostly to deal with longer addresses). The main features of IPv6 are discussed below. More information about it can be found in RFCs 2460 through 2466.

First and foremost, IPv6 has longer addresses than IPv4. They are 16 bytes long, which solves the problem that IPv6 set out to solve: provide an effectively unlimited supply of Internet addresses. We will have more to say about addresses shortly.

The second major improvement of IPv6 is the simplification of the header. It contains only seven fields (versus 13 in IPv4). This change allows routers to process packets faster and thus improve throughput and delay. We will discuss the header shortly, too.

The third major improvement was better support for options. This change was essential with the new header because fields that previously were required are now optional. In addition, the way options are represented is different, making it simple for routers to skip over options not intended for them. This feature speeds up packet processing time.

A fourth area in which IPv6 represents a big advance is in security. IETF had its fill of newspaper stories about precocious 12-year-olds using their personal computers to break into banks and military bases all over the Internet. There was a strong feeling that something had to be done to improve security. Authentication and privacy are key features of the new IP. These were later retrofitted to IPv4, however, so in the area of security the differences are not so great any more.

Finally, more attention has been paid to quality of service. Various half-hearted efforts have been made in the past, but now with the growth of multimedia on the Internet, the sense of urgency is greater.

**The Main IPv6 Header**

The IPv6 header is shown in **Fig. 5-68**. The *Version* field is always 6 for IPv6 (and 4 for IPv4). During the transition period from IPv4, which will probably take a decade, routers will be able to examine this field to tell what kind of packet they have. As an aside, making this test wastes a few instructions in the critical path, so many implementations are likely to try to avoid it by using some field in the data link header to distinguish IPv4 packets from IPv6 packets. In this way, packets can be passed to the correct network layer handler directly. However, having the data link layer be aware of network packet types completely violates the design principle that each layer should not be aware of the meaning of the bits given to it from the layer above. The discussions between the "Do it right" and "Make it fast" camps will no doubt be lengthy and vigorous.

**Figure 5-68. The IPv6 fixed header (required).**
The Traffic class field is used to distinguish between packets with different real-time delivery requirements. A field designed for this purpose has been in IP since the beginning, but it has been only sporadically implemented by routers. Experiments are now underway to determine how best it can be used for multimedia delivery.

The Flow label field is also still experimental but will be used to allow a source and destination to set up a pseudoconnection with particular properties and requirements. For example, a stream of packets from one process on a certain source host to a certain process on a certain destination host might have stringent delay requirements and thus need reserved bandwidth. The flow can be set up in advance and given an identifier. When a packet with a nonzero Flow label shows up, all the routers can look it up in internal tables to see what kind of special treatment it requires. In effect, flows are an attempt to have it both ways: the flexibility of a datagram subnet and the guarantees of a virtual-circuit subnet.

Each flow is designated by the source address, destination address, and flow number, so many flows may be active at the same time between a given pair of IP addresses. Also, in this way, even if two flows coming from different hosts but with the same flow label pass through the same router, the router will be able to tell them apart using the source and destination addresses. It is expected that flow labels will be chosen randomly, rather than assigned sequentially starting at 1, so routers as expected to hash them.

The Payload length field tells how many bytes follow the 40-byte header of Fig. 5-68. The name was changed from the IPv4 Total length field because the meaning was changed slightly: the 40 header bytes are no longer counted as part of the length (as they used to be).

The Next header field lets the cat out of the bag. The reason the header could be simplified is that there can be additional (optional) extension headers. This field tells which of the (currently) six extension headers, if any, follow this one. If this header is the last IP header, the Next header field tells which transport protocol handler (e.g., TCP, UDP) to pass the packet to.

The Hop limit field is used to keep packets from living forever. It is, in practice, the same as the Time to live field in IPv4, namely, a field that is decremented on each hop. In theory, in IPv4 it was a time in seconds, but no router used it that way, so the name was changed to reflect the way it is actually used.
Next come the *Source address* and *Destination address* fields. Deering's original proposal, SIP, used 8-byte addresses, but during the review process many people felt that with 8-byte addresses IPv6 would run out of addresses within a few decades, whereas with 16-byte addresses it would never run out. Other people argued that 16 bytes was overkill, whereas still others favored using 20-byte addresses to be compatible with the OSI datagram protocol. Still another faction wanted variable-sized addresses. After much debate, it was decided that fixed-length 16-byte addresses were the best compromise.

A new notation has been devised for writing 16-byte addresses. They are written as eight groups of four hexadecimal digits with colons between the groups, like this:

\[
8000:0000:0000:0123:4567:89AB:CDEF
\]

Since many addresses will have many zeros inside them, three optimizations have been authorized. First, leading zeros within a group can be omitted, so 0123 can be written as 123. Second, one or more groups of 16 zero bits can be replaced by a pair of colons. Thus, the above address now becomes

\[
8000::123:4567:89AB:CDEF
\]

Finally, IPv4 addresses can be written as a pair of colons and an old dotted decimal number, for example

\[
::192.31.20.46
\]

Perhaps it is unnecessary to be so explicit about it, but there are a lot of 16-byte addresses. Specifically, there are \(2^{128}\) of them, which is approximately \(3 \times 10^{38}\). If the entire earth, land and water, were covered with computers, IPv6 would allow \(7 \times 10^{23}\) IP addresses per square meter. Students of chemistry will notice that this number is larger than Avogadro’s number. While it was not the intention to give every molecule on the surface of the earth its own IP address, we are not that far off.

In practice, the address space will not be used efficiently, just as the telephone number address space is not (the area code for Manhattan, 212, is nearly full, but that for Wyoming, 307, is nearly empty). In RFC 3194, Durand and Huitema calculated that, using the allocation of telephone numbers as a guide, even in the most pessimistic scenario there will still be well over 1000 IP addresses per square meter of the entire earth’s surface (land and water). In any likely scenario, there will be trillions of them per square meter. In short, it seems unlikely that we will run out in the foreseeable future.

It is instructive to compare the IPv4 header (Fig. 5-53) with the IPv6 header (Fig. 5-68) to see what has been left out in IPv6. The *IHL* field is gone because the IPv6 header has a fixed length. The *Protocol* field was taken out because the *Next header* field tells what follows the last IP header (e.g., a UDP or TCP segment).

All the fields relating to fragmentation were removed because IPv6 takes a different approach to fragmentation. To start with, all IPv6-conformant hosts are expected to dynamically determine the datagram size to use. This rule makes fragmentation less likely to occur in the first place. Also, the minimum has been raised from 576 to 1280 to allow 1024 bytes of data and many headers. In addition, when a host sends an IPv6 packet that is too large, instead of fragmenting it, the router that is unable to forward it sends back an error message. This message tells the host to break up all future packets to that destination. Having the host send packets that are the right size in the first place is ultimately much more efficient than having the routers fragment them on the fly.
Finally, the Checksum field is gone because calculating it greatly reduces performance. With the reliable networks now used, combined with the fact that the data link layer and transport layers normally have their own checksums, the value of yet another checksum was not worth the performance price it extracted. Removing all these features has resulted in a lean and mean network layer protocol. Thus, the goal of IPv6—a fast, yet flexible, protocol with plenty of address space—has been met by this design.

Extension Headers

Some of the missing IPv4 fields are occasionally still needed, so IPv6 has introduced the concept of an (optional) extension header. These headers can be supplied to provide extra information, but encoded in an efficient way. Six kinds of extension headers are defined at present, as listed in Fig. 5-69. Each one is optional, but if more than one is present, they must appear directly after the fixed header, and preferably in the order listed.

**Figure 5-69. IPv6 extension headers.**

<table>
<thead>
<tr>
<th>Extension header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hop-by-hop options</td>
<td>Miscellaneou information for routers</td>
</tr>
<tr>
<td>Destination options</td>
<td>Additional information for the destination</td>
</tr>
<tr>
<td>Routing</td>
<td>Loose list of routers to visit</td>
</tr>
<tr>
<td>Fragmentation</td>
<td>Management of datagram fragments</td>
</tr>
<tr>
<td>Authentication</td>
<td>Verification of the sender's identity</td>
</tr>
<tr>
<td>Encrypted security payload</td>
<td>Information about the encrypted contents</td>
</tr>
</tbody>
</table>

Some of the headers have a fixed format; others contain a variable number of variable-length fields. For these, each item is encoded as a (Type, Length, Value) tuple. The Type is a 1-byte field telling which option this is. The Type values have been chosen so that the first 2 bits tell routers that do not know how to process the option what to do. The choices are: skip the option; discard the packet; discard the packet and send back an ICMP packet; and the same as the previous one, except do not send ICMP packets for multicast addresses (to prevent one bad multicast packet from generating millions of ICMP reports).

The Length is also a 1-byte field. It tells how long the value is (0 to 255 bytes). The Value is any information required, up to 255 bytes.

The hop-by-hop header is used for information that all routers along the path must examine. So far, one option has been defined: support of datagrams exceeding 64K. The format of this header is shown in Fig. 5-70. When it is used, the Payload length field in the fixed header is set to zero.

**Figure 5-70. The hop-by-hop extension header for large datagrams (jumbograms).**

```
<table>
<thead>
<tr>
<th>Next header</th>
<th>0</th>
<th>194</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jumbo payload length</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

As with all extension headers, this one starts out with a byte telling what kind of header comes next. This byte is followed by one telling how long the hop-by-hop header is in bytes, excluding the first 8 bytes, which are mandatory. All extensions begin this way.
The next 2 bytes indicate that this option defines the datagram size (code 194) and that the size is a 4-byte number. The last 4 bytes give the size of the datagram. Sizes less than 65,536 bytes are not permitted and will result in the first router discarding the packet and sending back an ICMP error message. Datagrams using this header extension are called **jumbograms**. The use of jumbograms is important for supercomputer applications that must transfer gigabytes of data efficiently across the Internet.

The destination options header is intended for fields that need only be interpreted at the destination host. In the initial version of IPv6, the only options defined are null options for padding this header out to a multiple of 8 bytes, so initially it will not be used. It was included to make sure that new routing and host software can handle it, in case someone thinks of a destination option some day.

The routing header lists one or more routers that must be visited on the way to the destination. It is very similar to the IPv4 loose source routing in that all addresses listed must be visited in order, but other routers not listed may be visited in between. The format of the routing header is shown in **Fig. 5-71**.

**Figure 5-71. The extension header for routing.**

The first 4 bytes of the routing extension header contain four 1-byte integers. The *Next header* and *Header extension length* fields were described above. The *Routing type* field gives the format of the rest of the header. Type 0 says that a reserved 32-bit word follows the first word, followed by some number of IPv6 addresses. Other types may be invented in the future as needed. Finally, the *Segments left* field keeps track of how many of the addresses in the list have not yet been visited. It is decremented every time one is visited. When it hits 0, the packet is on its own with no more guidance about what route to follow. Usually at this point it is so close to the destination that the best route is obvious.

The fragment header deals with fragmentation similarly to the way IPv4 does. The header holds the datagram identifier, fragment number, and a bit telling whether more fragments will follow. In IPv6, unlike in IPv4, only the source host can fragment a packet. Routers along the way may not do this. Although this change is a major philosophical break with the past, it simplifies the routers' work and makes routing go faster. As mentioned above, if a router is confronted with a packet that is too big, it discards the packet and sends an ICMP packet back to the source. This information allows the source host to fragment the packet into smaller pieces using this header and try again.

The authentication header provides a mechanism by which the receiver of a packet can be sure of who sent it. The encrypted security payload makes it possible to encrypt the contents of a packet so that only the intended recipient can read it. These headers use cryptographic techniques to accomplish their missions.

**Controversies**

Given the open design process and the strongly-held opinions of many of the people involved, it should come as no surprise that many choices made for IPv6 were highly controversial, to say the
least. We will summarize a few of these briefly below. For all the gory details, see the RFCs.

We have already mentioned the argument about the address length. The result was a compromise: 16-byte fixed-length addresses.

Another fight developed over the length of the Hop limit field. One camp felt strongly that limiting the maximum number of hops to 255 (implicit in using an 8-bit field) was a gross mistake. After all, paths of 32 hops are common now, and 10 years from now much longer paths may be common. These people argued that using a huge address size was farsighted but using a tiny hop count was short-sighted. In their view, the greatest sin a computer scientist can commit is to provide too few bits somewhere.

The response was that arguments could be made to increase every field, leading to a bloated header. Also, the function of the Hop limit field is to keep packets from wandering around for a long time and 65,535 hops is far too long. Finally, as the Internet grows, more and more long-distance links will be built, making it possible to get from any country to any other country in half a dozen hops at most. If it takes more than 125 hops to get from the source and destination to their respective international gateways, something is wrong with the national backbones. The 8-bitters won this one.

Another hot potato was the maximum packet size. The supercomputer community wanted packets in excess of 64 KB. When a supercomputer gets started transferring, it really means business and does not want to be interrupted every 64 KB. The argument against large packets is that if a 1-MB packet hits a 1.5-Mbps T1 line, that packet will tie the line up for over 5 seconds, producing a very noticeable delay for interactive users sharing the line. A compromise was reached here: normal packets are limited to 64 KB, but the hop-by-hop extension header can be used to permit jumbograms.

A third hot topic was removing the IPv4 checksum. Some people likened this move to removing the brakes from a car. Doing so makes the car lighter so it can go faster, but if an unexpected event happens, you have a problem.

The argument against checksums was that any application that really cares about data integrity has to have a transport layer checksum anyway, so having another one in IP (in addition to the data link layer checksum) is overkill. Furthermore, experience showed that computing the IP checksum was a major expense in IPv4. The antichecksum camp won this one, and IPv6 does not have a checksum.

Mobile hosts were also a point of contention. If a portable computer flies halfway around the world, can it continue operating at the destination with the same IPv6 address, or does it have to use a scheme with home agents and foreign agents? Mobile hosts also introduce asymmetries into the routing system. It may well be the case that a small mobile computer can easily hear the powerful signal put out by a large stationary router, but the stationary router cannot hear the feeble signal put out by the mobile host. Consequently, some people wanted to build explicit support for mobile hosts into IPv6. That effort failed when no consensus could be found for any specific proposal.

Probably the biggest battle was about security. Everyone agreed it was essential, The war was about where and how. First where. The argument for putting it in the network layer is that it then becomes a standard service that all applications can use without any advance planning. The argument against it is that really secure applications generally want nothing less than end-to-end encryption, where the source application does the encryption and the destination application undoes it. With anything less, the user is at the mercy of potentially buggy network layer implementations over which he has no control. The response to this argument is that these applications can just refrain from using the IP security features and do the job themselves. The
rejoinder to that is that the people who do not trust the network to do it right, do not want to pay the price of slow, bulky IP implementations that have this capability, even if it is disabled.

Another aspect of where to put security relates to the fact that many (but not all) countries have stringent export laws concerning cryptography. Some, notably France and Iraq, also restrict its use domestically, so that people cannot have secrets from the police. As a result, any IP implementation that used a cryptographic system strong enough to be of much value could not be exported from the United States (and many other countries) to customers worldwide. Having to maintain two sets of software, one for domestic use and one for export, is something most computer vendors vigorously oppose.

One point on which there was no controversy is that no one expects the IPv4 Internet to be turned off on a Sunday morning and come back up as an IPv6 Internet Monday morning. Instead, isolated "islands" of IPv6 will be converted, initially communicating via tunnels. As the IPv6 islands grow, they will merge into bigger islands. Eventually, all the islands will merge, and the Internet will be fully converted. Given the massive investment in IPv4 routers currently deployed, the conversion process will probably take a decade. For this reason, an enormous amount of effort has gone into making sure that this transition will be as painless as possible. For more information about IPv6, see (Loshin, 1999).

5.7 Summary

The network layer provides services to the transport layer. It can be based on either virtual circuits or datagrams. In both cases, its main job is routing packets from the source to the destination. In virtual-circuit subnets, a routing decision is made when the virtual circuit is set up. In datagram subnets, it is made on every packet.

Many routing algorithms are used in computer networks. Static algorithms include shortest path routing and flooding. Dynamic algorithms include distance vector routing and link state routing. Most actual networks use one of these. Other important routing topics are hierarchical routing, routing for mobile hosts, broadcast routing, multicast routing, and routing in peer-to-peer networks.

Subnets can easily become congested, increasing the delay and lowering the throughput for packets. Network designers attempt to avoid congestion by proper design. Techniques include retransmission policy, caching, flow control, and more. If congestion does occur, it must be dealt with. Choke packets can be sent back, load can be shed, and other methods applied.

The next step beyond just dealing with congestion is to actually try to achieve a promised quality of service. The methods that can be used for this include buffering at the client, traffic shaping, resource reservation, and admission control. Approaches that have been designed for good quality of service include integrated services (including RSVP), differentiated services, and MPLS.

Networks differ in various ways, so when multiple networks are interconnected problems can occur. Sometimes the problems can be finessed by tunneling a packet through a hostile network, but if the source and destination networks are different, this approach fails. When different networks have different maximum packet sizes, fragmentation may be called for.

The Internet has a rich variety of protocols related to the network layer. These include the data transport protocol, IP, but also the control protocols ICMP, ARP, and RARP, and the routing protocols OSPF and BGP. The Internet is rapidly running out of IP addresses, so a new version of IP, IPv6, has been developed.
Problems

1. Give two example computer applications for which connection-oriented service is appropriate. Now give two examples for which connectionless service is best.

2. Are there any circumstances when connection-oriented service will (or at least should) deliver packets out of order? Explain.

3. Datagram subnets route each packet as a separate unit, independent of all others. Virtual-circuit subnets do not have to do this, since each data packet follows a predetermined route. Does this observation mean that virtual-circuit subnets do not need the capability to route isolated packets from an arbitrary source to an arbitrary destination? Explain your answer.

4. Give three examples of protocol parameters that might be negotiated when a connection is set up.

5. Consider the following design problem concerning implementation of virtual-circuit service. If virtual circuits are used internal to the subnet, each data packet must have a 3-byte header and each router must tie up 8 bytes of storage for circuit identification. If datagrams are used internally, 15-byte headers are needed but no router table space is required. Transmission capacity costs 1 cent per $10^6$ bytes, per hop. Very fast router memory can be purchased for 1 cent per byte and is depreciated over two years, assuming a 40-hour business week. The statistically average session runs for 1000 sec, in which time 200 packets are transmitted. The mean packet requires four hops. Which implementation is cheaper, and by how much?

6. Assuming that all routers and hosts are working properly and that all software in both is free of all errors, is there any chance, however small, that a packet will be delivered to the wrong destination?

7. Consider the network of Fig. 5-7, but ignore the weights on the lines. Suppose that it uses flooding as the routing algorithm. If a packet sent by A to D has a maximum hop count of 3, list all the routes it will take. Also tell how many hops worth of bandwidth it consumes.

8. Give a simple heuristic for finding two paths through a network from a given source to a given destination that can survive the loss of any communication line (assuming two such paths exist). The routers are considered reliable enough, so it is not necessary to worry about the possibility of router crashes.

9. Consider the subnet of Fig. 5-13(a). Distance vector routing is used, and the following vectors have just come in to router C: from B: (5, 0, 8, 12, 6, 2); from D: (16, 12, 6, 0, 9, 10); and from E: (7, 6, 3, 9, 0, 4). The measured delays to B, D, and E, are 6, 3, and 5, respectively. What is C’s new routing table? Give both the outgoing line to use and the expected delay.

10. If delays are recorded as 8-bit numbers in a 50-router network, and delay vectors are exchanged twice a second, how much bandwidth per (full-duplex) line is chewed up by the distributed routing algorithm? Assume that each router has three lines to other routers.

11. In Fig. 5-14 the Boolean OR of the two sets of ACF bits are 111 in every row. Is this just an accident here, or does it hold for all subnets under all circumstances?

12. For hierarchical routing with 4800 routers, what region and cluster sizes should be chosen to
minimize the size of the routing table for a three-layer hierarchy? A good starting place is the hypothesis that a solution with \( k \) clusters of \( k \) regions of \( k \) routers is close to optimal, which means that \( k \) is about the cube root of 4800 (around 16). Use trial and error to check out combinations where all three parameters are in the general vicinity of 16.

13. In the text it was stated that when a mobile host is not at home, packets sent to its home LAN are intercepted by its home agent on that LAN. For an IP network on an 802.3 LAN, how does the home agent accomplish this interception?

14. Looking at the subnet of Fig. 5-6, how many packets are generated by a broadcast from \( B \), using

   a. (a) reverse path forwarding?

   b. (b) the sink tree?

15. Consider the network of Fig. 5-16(a). Imagine that one new line is added, between \( F \) and \( G \), but the sink tree of Fig. 5-16(b) remains unchanged. What changes occur to Fig. 5-16(c)?

16. Compute a multicast spanning tree for router \( C \) in the following subnet for a group with members at routers \( A, B, C, D, E, F, I, \) and \( K \).

17. In Fig. 5-20, do nodes \( H \) or \( I \) ever broadcast on the lookup shown starting at \( A \)?

18. Suppose that node \( B \) in Fig. 5-20 has just rebooted and has no routing information in its tables. It suddenly needs a route to \( H \). It sends out broadcasts with TTL set to 1, 2, 3, and so on. How many rounds does it take to find a route?

19. In the simplest version of the Chord algorithm for peer-to-peer lookup, searches do not use the finger table. Instead, they are linear around the circle, in either direction. Can a node accurately predict which direction it should search? Discuss your answer.

20. Consider the Chord circle of Fig. 5-24. Suppose that node 10 suddenly goes on line. Does this affect node 1's finger table, and if so, how?

21. As a possible congestion control mechanism in a subnet using virtual circuits internally, a router could refrain from acknowledging a received packet until (1) it knows its last transmission along the virtual circuit was received successfully and (2) it has a free buffer. For simplicity, assume that the routers use a stop-and-wait protocol and that each virtual circuit has one buffer dedicated to it for each direction of traffic. If it takes \( T \) sec to transmit a packet (data or acknowledgement) and there are \( n \) routers on the path, what is the rate at which packets are delivered to the destination host? Assume that transmission errors are
rare and that the host-router connection is infinitely fast.

22. A datagram subnet allows routers to drop packets whenever they need to. The probability of a router discarding a packet is $p$. Consider the case of a source host connected to the source router, which is connected to the destination router, and then to the destination host. If either of the routers discards a packet, the source host eventually times out and tries again. If both host-router and router-router lines are counted as hops, what is the mean number of

a. (a) hops a packet makes per transmission?

b. (b) transmissions a packet makes?

c. (c) hops required per received packet?

23. Describe two major differences between the warning bit method and the RED method.

24. Give an argument why the leaky bucket algorithm should allow just one packet per tick, independent of how large the packet is.

25. The byte-counting variant of the leaky bucket algorithm is used in a particular system. The rule is that one 1024-byte packet, or two 512-byte packets, etc., may be sent on each tick. Give a serious restriction of this system that was not mentioned in the text.

26. An ATM network uses a token bucket scheme for traffic shaping. A new token is put into the bucket every 5 usec. Each token is good for one cell, which contains 48 bytes of data. What is the maximum sustainable data rate?

27. A computer on a 6-Mbps network is regulated by a token bucket. The token bucket is filled at a rate of 1 Mbps. It is initially filled to capacity with 8 megabits. How long can the computer transmit at the full 6 Mbps?

28. Imagine a flow specification that has a maximum packet size of 1000 bytes, a token bucket rate of 10 million bytes/sec, a token bucket size of 1 million bytes, and a maximum transmission rate of 50 million bytes/sec. How long can a burst at maximum speed last?

29. The network of Fig. 5-37 uses RSVP with multicast trees for hosts 1 and 2 as shown. Suppose that host 3 requests a channel of bandwidth 2 MB/sec for a flow from host 1 and another channel of bandwidth 1 MB/sec for a flow from host 2. At the same time, host 4 requests a channel of bandwidth 2 MB/sec for a flow from host 1 and host 5 requests a channel of bandwidth 1 MB/sec for a flow from host 2. How much total bandwidth will be reserved for these requests at routers $A$, $B$, $C$, $E$, $H$, $J$, $K$, and $L$?

30. The CPU in a router can process 2 million packets/sec. The load offered to it is 1.5 million packets/sec. If a route from source to destination contains 10 routers, how much time is spent being queued and serviced by the CPUs?

31. Consider the user of differentiated services with expedited forwarding. Is there a guarantee that expedited packets experience a shorter delay than regular packets? Why or why not?

32. Is fragmentation needed in concatenated virtual-circuit internets or only in datagram systems?

33. Tunneling through a concatenated virtual-circuit subnet is straightforward: the multiprotocol
router at one end just sets up a virtual circuit to the other end and passes packets through it. Can tunneling also be used in datagram subnets? If so, how?

34. Suppose that host A is connected to a router R1, R1 is connected to another router, R2, and R2 is connected to host B. Suppose that a TCP message that contains 900 bytes of data and 20 bytes of TCP header is passed to the IP code at host A for delivery to B. Show the Total length, Identification, DF, MF, and Fragment offset fields of the IP header in each packet transmitted over the three links. Assume that link A-R1 can support a maximum frame size of 1024 bytes including a 14-byte frame header, link R1-R2 can support a maximum frame size of 512 bytes, including an 8-byte frame header, and link R2-B can support a maximum frame size of 512 bytes including a 12-byte frame header.

35. A router is blasting out IP packets whose total length (data plus header) is 1024 bytes. Assuming that packets live for 10 sec, what is the maximum line speed the router can operate at without danger of cycling through the IP datagram ID number space?

36. An IP datagram using the Strict source routing option has to be fragmented. Do you think the option is copied into each fragment, or is it sufficient to just put it in the first fragment? Explain your answer.

37. Suppose that instead of using 16 bits for the network part of a class B address originally, 20 bits had been used. How many class B networks would there have been?

38. Convert the IP address whose hexadecimal representation is C2F1582 to dotted decimal notation.

39. A network on the Internet has a subnet mask of 255.255.240.0. What is the maximum number of hosts it can handle?

40. A large number of consecutive IP address are available starting at 198.16.0.0. Suppose that four organizations, A, B, C, and D, request 4000, 2000, 4000, and 8000 addresses, respectively, and in that order. For each of these, give the first IP address assigned, the last IP address assigned, and the mask in the w.x.y.z/s notation.

41. A router has just received the following new IP addresses: 57.6.96.0/21, 57.6.104.0/21, 57.6.112.0/21, and 57.6.120.0/21. If all of them use the same outgoing line, can they be aggregated? If so, to what? If not, why not?

42. The set of IP addresses from 29.18.0.0 to 19.18.128.255 has been aggregated to 29.18.0.0/17. However, there is a gap of 1024 unassigned addresses from 29.18.60.0 to 29.18.63.255 that are now suddenly assigned to a host using a different outgoing line. Is it now necessary to split up the aggregate address into its constituent blocks, add the new block to the table, and then see if any reaggregation is possible? If not, what can be done instead?

43. A router has the following (CIDR) entries in its routing table:

<table>
<thead>
<tr>
<th>Address/mask</th>
<th>Next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>135.46.56.0/22</td>
<td>Interface 0</td>
</tr>
<tr>
<td>135.46.60.0/22</td>
<td>Interface 1</td>
</tr>
<tr>
<td>192.53.40.0/23</td>
<td>Router 1</td>
</tr>
</tbody>
</table>

file://C:\Documents and Settings\donnierzajones\Local Settings\Temp~hh9EF4.htm 9/3/2008
For each of the following IP addresses, what does the router do if a packet with that address arrives?

a. (a) 135.46.63.10
b. (b) 135.46.57.14
c. (c) 135.46.52.2
d. (d) 192.53.40.7
e. (e) 192.53.56.7

44. Many companies have a policy of having two (or more) routers connecting the company to the Internet to provide some redundancy in case one of them goes down. Is this policy still possible with NAT? Explain your answer.

45. You have just explained the ARP protocol to a friend. When you are all done, he says: "I've got it. ARP provides a service to the network layer, so it is part of the data link layer." What do you say to him?

46. ARP and RARP both map addresses from one space to another. In this respect, they are similar. However, their implementations are fundamentally different. In what major way do they differ?

47. Describe a way to reassemble IP fragments at the destination.

48. Most IP datagram reassembly algorithms have a timer to avoid having a lost fragment tie up reassembly buffers forever. Suppose that a datagram is fragmented into four fragments. The first three fragments arrive, but the last one is delayed. Eventually, the timer goes off and the three fragments in the receiver's memory are discarded. A little later, the last fragment stumble in. What should be done with it?

49. In both IP and ATM, the checksum covers only the header and not the data. Why do you suppose this design was chosen?

50. A person who lives in Boston travels to Minneapolis, taking her portable computer with her. To her surprise, the LAN at her destination in Minneapolis is a wireless IP LAN, so she does not have to plug in. Is it still necessary to go through the entire business with home agents and foreign agents to make e-mail and other traffic arrive correctly?

51. IPv6 uses 16-byte addresses. If a block of 1 million addresses is allocated every picosecond, how long will the addresses last?

52. The Protocol field used in the IPv4 header is not present in the fixed IPv6 header. Why not?

53. When the IPv6 protocol is introduced, does the ARP protocol have to be changed? If so, are the changes conceptual or technical?

54. Write a program to simulate routing using flooding. Each packet should contain a counter
that is decremented on each hop. When the counter gets to zero, the packet is discarded. Time is discrete, with each line handling one packet per time interval. Make three versions of the program: all lines are flooded, all lines except the input line are flooded, and only the (statically chosen) best k lines are flooded. Compare flooding with deterministic routing (k = 1) in terms of both delay and the bandwidth used.

55. Write a program that simulates a computer network using discrete time. The first packet on each router queue makes one hop per time interval. Each router has only a finite number of buffers. If a packet arrives and there is no room for it, it is discarded and not retransmitted. Instead, there is an end-to-end protocol, complete with timeouts and acknowledgement packets, that eventually regenerates the packet from the source router. Plot the throughput of the network as a function of the end-to-end timeout interval, parameterized by error rate.

56. Write a function to do forwarding in an IP router. The procedure has one parameter, an IP address. It also has access to a global table consisting of an array of triples. Each triple contains three integers: an IP address, a subnet mask, and the outline line to use. The function looks up the IP address in the table using CIDR and returns the line to use as its value.

57. Use the \texttt{traceroute} (UNIX) or \texttt{tracert} (Windows) programs to trace the route from your computer to various universities on other continents. Make a list of transoceanic links you have discovered. Some sites to try are

\begin{itemize}
\item \url{www.berkeley.edu} (California)
\item \url{www.mit.edu} (Massachusetts)
\item \url{www.vu.nl} (Amsterdam)
\item \url{www.ucl.ac.uk} (London)
\item \url{www.usyd.edu.au} (Sydney)
\item \url{www.u-tokyo.ac.jp} (Tokyo)
\item \url{www.uct.ac.za} (Cape Town)
\end{itemize}