UNIT-III

THE NETWORK LAYER   THE NETWORK LAYER IN THE INTERNET

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 network (i.e., the set of all routers and links) and choose appropriate paths through it, even for large networks. It must also take care when choosing 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.

NETWORK LAYER DESIGN ISSUES:

STORE-AND-FORWARD PACKET SWITCHING:

Before starting to explain the details of the network layer, it is worth restating the context in which the network layer protocols operate. This context can be seen in Fig. 3.1. The major components of the network are the ISP's equipment (routers connected by transmission lines), shown inside the shaded oval, and the customers' equipment, shown outside the oval.

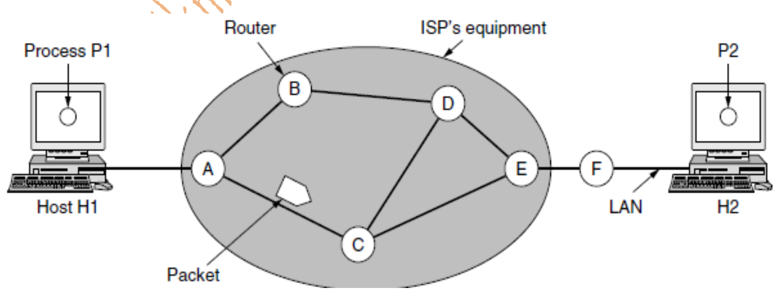


FIGURE 3.1: THE ENVIRONMENT OF THE NETWORK LAYER PROTOCOLS.

Host *H1* is directly connected to one of the ISP's routers, *A*, perhaps as a home computer that is plugged into a DSL modem. In contrast, *H2* is on a LAN, which might be an office Ethernet, with a router, *F*, owned and operated by the customer.

This router has a leased line to the ISP's equipment. We have shown *F* as being outside the oval because it does not belong to the ISP. For the purposes of this chapter, however, routers on customer premises are considered part of the ISP network because they run the same algorithms as the ISP's routers (and our main concern here is algorithms).

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 ISP. The packet is stored there until it has fully arrived and the link has finished its processing by verifying the checksum. Then it is forwarded to the next router along the path until it reaches the destination host, where it is delivered. This mechanism is called store-and-forward packet switching.

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 precisely what kind of services the network layer provides to the transport layer. The services need to be carefully 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.

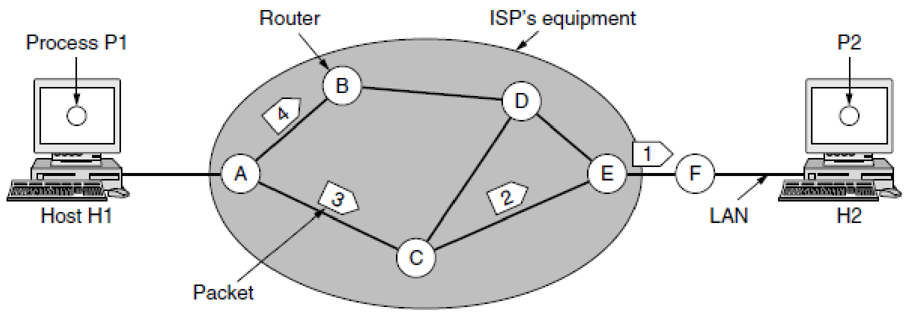
IMPLEMENTATION OF CONNECTIONLESS SERVICE:

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 network 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 network is called a **datagram network**.

If connection-oriented service is used, a path from the source router all the way 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 network is called a **virtual-circuit network**.

Let us now see how a datagram network works. Suppose that the process *P1* in Fig. 3.2 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.



A's table (initially) Dest.	Line	A's table (later) Dest.	Line	C's table Dest.	Line	E's table Dest.	Line
A	–	A	–	A	A	A	C
B	B	B	B	B	A	B	D
C	C	C	C	C	–	C	C
D	B	D	B	D	E	D	D
E	C	E	B	E	E	E	–
F	C	F	B	F	E	F	F

FIGURE 3.2: ROUTING WITHIN A DATAGRAM NETWORK

IMPLEMENTATION OF CONNECTION-ORIENTED SERVICE

For connection-oriented service, we need a virtual-circuit network. 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. 3.2. 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 shown in Fig. 3.3. Here, host *H1* has established connection 1 with host *H2*. This connection 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.

COMPARISON OF VIRTUAL-CIRCUIT AND DATAGRAM NETWORKS:

Both virtual circuits and datagrams have their supporters and their detractors. We will now attempt to summarize both sets of arguments. The major issues are listed in Fig. 3.4. Inside the network, several trade-offs exist between virtual circuits and datagrams.

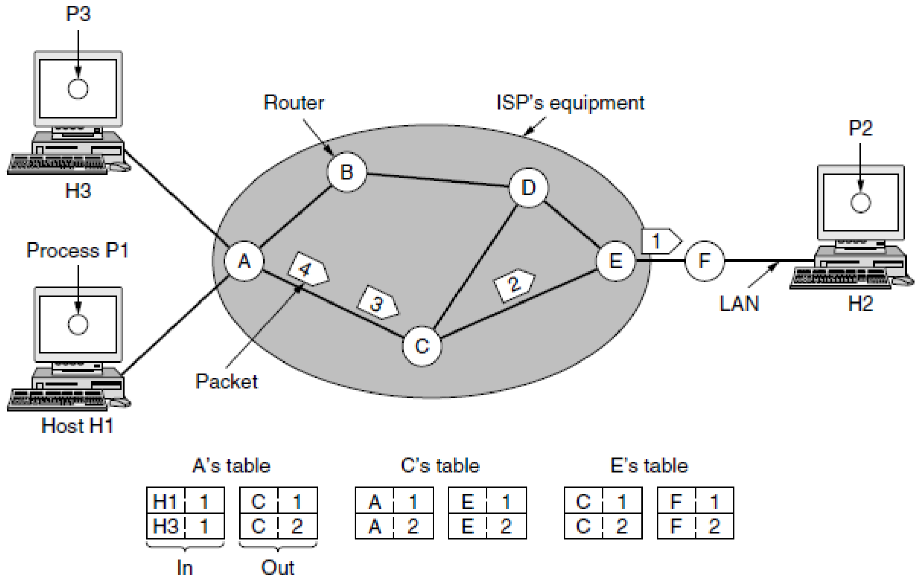


FIGURE 3.3: ROUTING WITHIN A VIRTUAL-CIRCUIT NETWORK

One trade-off is setup time versus address parsing time. Using virtual circuits requires a setup phase, which takes time and consumes resources. However, once this price is paid, figuring out what to do with a data packet in a virtual-circuit network is easy: the router just uses the circuit number to index into a table to find out where the packet goes. In a datagram network, no setup is needed but a more complicated lookup procedure is required to locate the entry for the destination.

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

FIGURE 3.4: COMPARISON OF DATAGRAM AND VIRTUAL-CIRCUIT NETWORKS

ROUTING ALGORITHMS:

The main function of the network layer is routing packets from the source machine to the destination machine. In most networks, 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 segment. 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 network 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 network uses virtual circuits internally, routing decisions are made only when a new virtual circuit is being set up. Thereafter, data packets just follow the already established route. The latter case is sometimes called **session routing** because a route remains in force for an entire session (e.g., while logged in over a VPN).

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 sent or only when new connections are established, certain properties are desirable in a routing algorithm: ***correctness***, ***simplicity***, ***robustness***, ***stability***, ***fairness***, and ***efficiency***.

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 system-wide 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. Imagine the havoc if the network needed to be rebooted every time some router crashed!

Stability is also an important goal for the routing algorithm. There exist routing algorithms that never converge to a fixed set of paths, no matter how long they run. A stable algorithm reaches equilibrium and stays there. It should converge quickly too, since communication may be disrupted until the routing algorithm has reached equilibrium.

Fairness and efficiency 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. 3.5. Suppose that there is enough traffic between *A* and *A′*, between *B* and *B′*, and between *C* and *C′* to saturate the horizontal links.

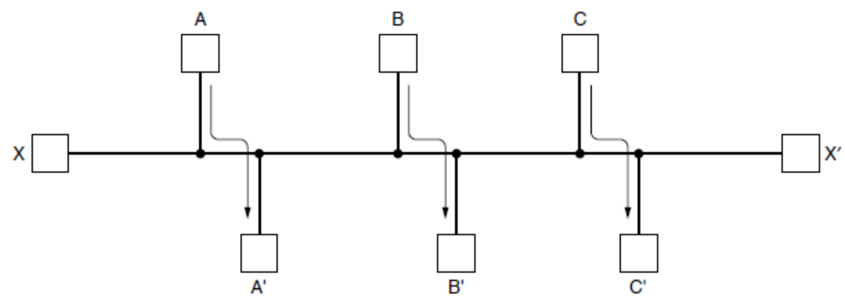


FIGURE 3.5: NETWORK WITH A CONFLICT BETWEEN FAIRNESS AND EFFICIENCY

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.

Routing algorithms can be grouped into two major classes: nonadaptive and adaptive. **Nonadaptive algorithms** do not base their routing decisions on any measurements or estimates of the current topology and traffic.

Instead, the choice of the route to use to get from *I* to *J* (for all *I* and *J*) is computed in advance, offline, and downloaded to the routers when the network is booted. This procedure is sometimes called **static routing**. Because it does not respond to failures, static routing is mostly useful for situations in which the routing choice is clear.

Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and sometimes changes in the traffic as well. These **dynamic routing** algorithms differ in where they get their information (e.g., locally, from adjacent routers, or from all routers), when they change the routes and what metric is used for optimization.

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** (Bellman, 1957).

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.

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. 3.6(b), where the distance metric is the number of hops. The goal of all routing algorithms is to discover and use the sink trees for all routers.

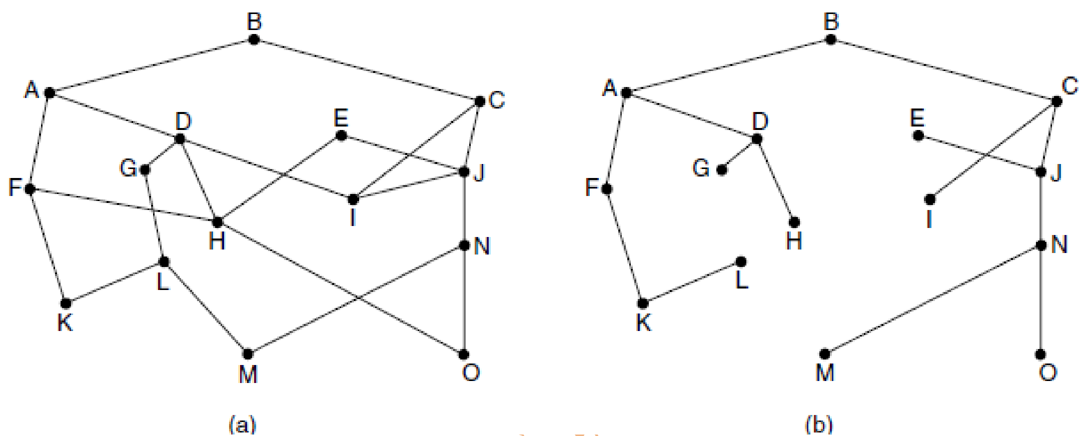


FIGURE 3.6: (A) A NETWORK. (B) A SINK TREE FOR ROUTER B

Note that a sink tree is not necessarily unique; other trees with the same path lengths may exist. If we allow all of the possible paths to be chosen, the tree becomes a more general structure called a **DAG** (**Directed Acyclic Graph**).

DAGs have no loops. We will use sink trees as convenient shorthand for both cases. Both cases also depend on the technical assumption that the paths do not interfere with each other so, for example, a traffic jam on one path will not cause another path to divert. 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.

SHORTEST PATH ALGORITHM:

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. 3.7 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).

However, many other metrics besides hops and physical distance are also possible. For example, each edge could be labeled with the mean delay of a standard test packet, as measured by hourly runs. With this graph labeling, the shortest path is the fastest path rather than the path with the fewest edges or kilometers.

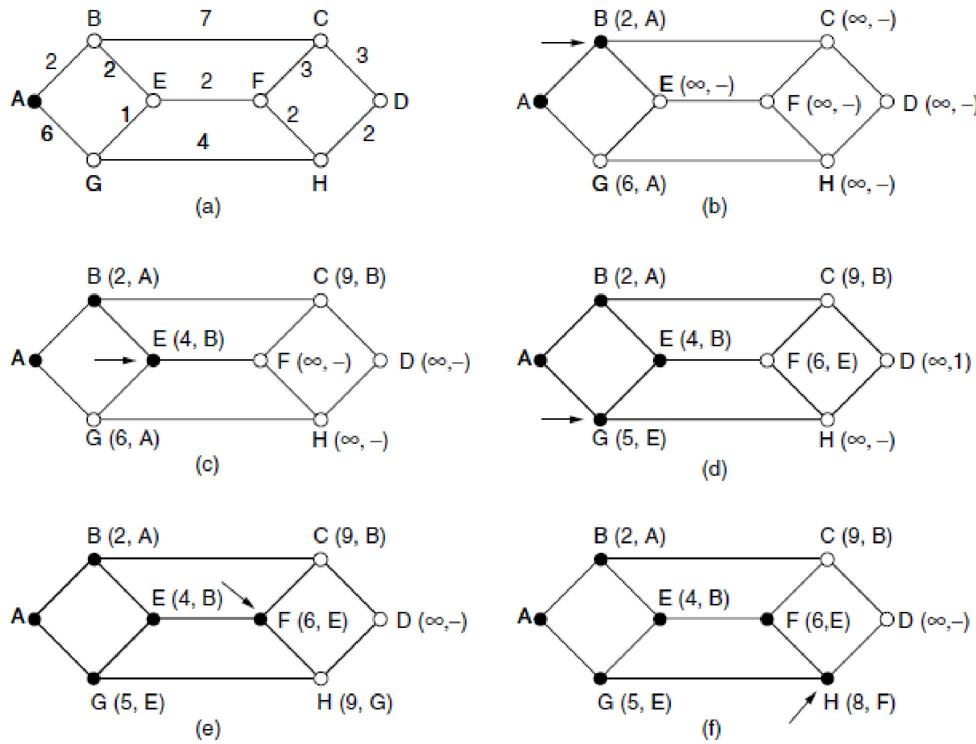


Figure 3.7: The first six steps used in computing the shortest path from *A* to *D*. The arrows indicate the working node.

Several algorithms for computing the shortest path between two nodes of a graph are known. This one is due to Dijkstra (1959) and finds the shortest paths between a source and all destinations in the network. Each node is labeled (in parentheses) with its distance from the source node along the best known path.

The distances must be non-negative, as they will be if they are based on real quantities like bandwidth and delay. 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. 3.7(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.

If the network had more than one shortest path from *A* to *D* and we wanted to find all of them, we would need to remember all of the probe nodes that could reach a node with the same distance.

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. 3.7(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 3.7 shows the first six steps of the algorithm.

To see why the algorithm works, look at Fig. 3.7(c). At this point we have just made *E* permanent. Suppose that there were a shorter path than *ABE*, say *AXYZE* (for some *X* and *Y*).

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. If the label at *Z* is greater than or equal to that at *E*, then *AXYZE* cannot be a shorter path than *ABE*. If the label is less than that of *E*, then *Z* and not *E* will become permanent first, allowing *E* to be probed from *Z*.

FLOODING:

When a routing algorithm is implemented, each router must make decisions based on local knowledge, not the complete picture of the network. A simple local technique 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 that 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 network.

Flooding with a hop count can produce an exponential number of duplicate packets as the hop count grows and routers duplicate packets they have seen before. A better technique for damming the flood is to have routers 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.

Flooding is not practical for sending most packets, but it does have some important uses. First, it ensures that a packet is delivered to every node in the network. This may be wasteful if there is a single destination that needs the packet, but it is effective for broadcasting information. 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.

Second, flooding is tremendously robust. Even if large numbers of routers are blown to bits (e.g., in a military network located in a war zone), flooding will find a path if one exists, to get a packet to its destination. Flooding also requires little in the way of setup. The routers only need to know their neighbors.

This means that flooding can be used as a building block for other routing algorithms that are more efficient but need more in the way of setup. Flooding can also be used 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).

DISTANCE VECTOR ROUTING:

A **distance vector routing** algorithm operates by having each router maintain a table (i.e., a vector) giving the best known distance to each destination and which link to use to get there. These tables are updated by exchanging information with the neighbors. Eventually, every router knows the best link to reach each destination.

The distance vector routing algorithm is sometimes called by other names, most commonly the distributed **Bellman-Ford** routing 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 network. This entry has two parts: the preferred outgoing line to use for that destination and an estimate of the distance to that destination. The distance might be measured as the number of hops or using another metric, as we discussed for computing shortest paths.

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 propagation delay, the router can measure it directly with special ECHO packets that the receiver just timestamps and sends back as fast as it can.

LINK STATE ROUTING:

Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing. The primary problem that caused its demise was that the algorithm often took too long to converge after the network topology changed (due to the count-to-infinity problem). Consequently, it was replaced by an entirely new algorithm, now called **link state routing**.

Variants of link state routing called IS-IS and OSPF are the routing algorithms that are most widely used inside large networks and the Internet today. The idea behind link state routing is fairly simple and can be stated as five parts. Each router must do the following things to make it work:

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

In effect, the complete topology is distributed to every router. Then Dijkstra’s algorithm can be run at each router to find the shortest path to every other router.

Link state routing is widely used in actual networks, so a few words about some example protocols are in order. Many ISPs use the **IS-IS** (**Intermediate System-Intermediate System**) link state protocol (Oran, 1990). It was designed for an early network called DECnet, later adopted by ISO for use with the OSI protocols and then modified to handle other protocols as well, most notably, IP.

OSPF (**Open Shortest Path First**) is the other main link state protocol. It was designed by IETF several years after IS-IS and adopted many of the innovations designed for IS-IS. These innovations 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 can carry information about multiple network layer protocols at the same time (e.g., IP, IPX, and AppleTalk). OSPF does not have this feature, and it is an advantage in large multiprotocol environments.

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**. Each router knows all the details about how to route packets to destinations within its own region but knows nothing about the internal structure of other regions. When different networks are interconnected, it is natural to regard each one as a separate region 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.

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 sending 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 network is for the source to simply send a distinct packet to each destination.

Not only is the method wasteful of bandwidth and slow, but it also requires the source to have a complete list of all destinations. This method is not desirable in practice, even though it is widely applicable.

An improvement is **multidestination routing**, in which 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 like a normal packet. Multidestination routing is like using separately addressed packets, except that when several packets must follow the same route, one of them pays full fare and the rest ride free.

The network bandwidth is therefore used more efficiently. However, this scheme still requires the source to know all the destinations, plus it is as much work for a router to determine where to send one multidestination packet as it is for multiple distinct packets.

MULTICAST ROUTING:

Sending a message to such a group is called **multicasting**, and the routing algorithm used is called **multicast routing**. All multicasting schemes require some way to create and destroy groups and to identify which routers are members of a group. How these tasks are accomplished is not of concern to the routing algorithm.

For now, we will assume that each group is identified by a multicast address and that routers know the groups to which they belong.

Multicast routing schemes build on the broadcast routing schemes we have already studied, sending packets along spanning trees to deliver the packets to the members of the group while making efficient use of bandwidth. However, the best spanning tree to use depends on whether the group is dense, with receivers scattered over most of the network, or sparse, with much of the network not belonging to the group.

If the group is dense, broadcast is a good start because it efficiently gets the packet to all parts of the network. But broadcast will reach some routers that are not members of the group, which is wasteful.

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.

Each router can then construct its own pruned spanning tree for each sender to the group in question by constructing a sink tree for the sender as usual and then removing all links that do not connect group members to the sink node. **MOSPF** (**Multicast OSPF**) is an example of a link state protocol that works in this way.

ANYCAST ROUTING:

So far, we have covered delivery models in which a source sends to a single destination (called **unicast**), to all destinations (called broadcast), and to a group of destinations (called multicast). Another delivery model, called **anycast** is sometimes also useful. In anycast, a packet is delivered to the nearest member of a group. Schemes that find these paths are called **anycast routing**.

ROUTING FOR MOBILE HOSTS:

Millions of people use computers while on the go, from truly mobile situations with wireless devices in moving cars, to nomadic situations in which laptop computers are used in a series of different locations. We will use the term **mobile hosts** to mean either category, as distinct from stationary hosts that never move.

Increasingly, people want to stay connected wherever in the world they may be, as easily as if they were at home. These mobile hosts introduce a new complication: to route a packet to a mobile host, the network first has to find it.

The model of the world that we will consider is one in which all hosts are assumed to have a permanent **home location** that never changes. Each hosts also has a permanent home address that can be used to determine its home location, 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 fixed home addresses and have the packets efficiently reach them wherever they may be. The trick, of course, is to find them.

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 emergency workers at an earthquake site, military vehicles on a battlefield, a fleet of ships at sea, or a gathering of people with laptop computers in an area lacking 802.11.

In all these cases, and others, each node communicates wirelessly and acts as both a host and a router. Networks of nodes that just happen to be near each other are called **ad hoc networks** or **MANETs** (**Mobile Ad hoc NETworks**).

What makes ad hoc networks different from wired networks is that the topology is suddenly tossed out the window. Nodes 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 barring failures, which are hopefully rare. With an ad hoc network, the topology may be changing all the time, so the desirability and even the validity of paths can change spontaneously without warning. Needless to say, these circumstances make routing in ad hoc networks more challenging than routing in their fixed counterparts.

CONGESTION CONTROL ALGORITHMS:

Too many packets present in (a part of) the network causes packet delay and loss that degrades performance. This situation is called **congestion**. The network and transport layers share the responsibility for handling congestion.

Since congestion occurs within the network, it is the network layer that directly experiences it and must ultimately determine what to do with the excess packets. However, the most effective way to control congestion is to reduce the load that the transport layer is placing on the network. This requires the network and transport layers to work together.

Figure 3.8 depicts the onset of congestion. When the number of packets hosts send into the network is well within its carrying capacity, the number delivered is proportional to the number sent. If twice as many are sent, twice as many are delivered.

However, as the offered load approaches the carrying capacity, bursts of traffic occasionally fill up the buffers inside routers and some packets are lost. These lost packets consume some of the capacity, so the number of delivered packets falls below the ideal curve. The network is now congested.

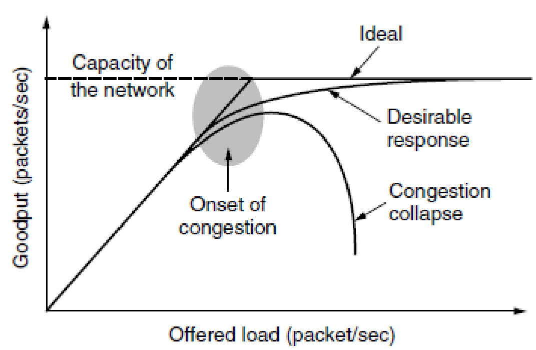


FIGURE 3.8: WITH TOO MUCH TRAFFIC, PERFORMANCE DROPS SHARPLY

Unless the network is well designed, it may experience a **congestion collapse**, in which performance plummets as the offered load increases beyond the capacity. This can happen because packets can be sufficiently delayed inside the network that they are no longer useful when they leave the network.

For example, in the early Internet, the time a packet spent waiting for a backlog of packets ahead of it to be sent over a slow 56-kbps link could reach the maximum time it was allowed to remain in the network. It then had to be thrown away.

A different failure mode occurs when senders retransmit packets that are greatly delayed, thinking that they have been lost. In this case, copies of the same packet will be delivered by the network, again wasting its capacity.

To capture these factors, the y-axis of Fig. 3.8 is given as **goodput**, which is the rate at which *useful* packets are delivered by the network. We would like to design networks that avoid congestion where possible and do not suffer from congestion collapse if they do become congested. Unfortunately, congestion cannot wholly be avoided.

APPROACHES TO CONGESTION CONTROL:

The presence of congestion means that the load is (temporarily) greater than the resources (in a part of the network) can handle. Two solutions come to mind: increase the resources or decrease the load. As shown in Fig. 3.9, these solutions are usually applied on different time scales to either prevent congestion or react to it once it has occurred.

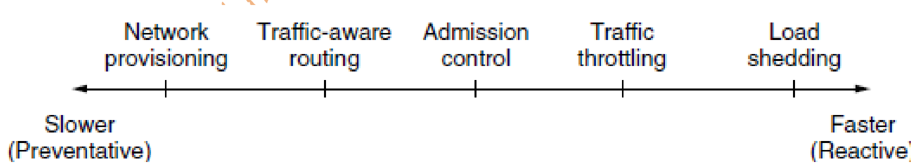


FIGURE 3.9: TIMESCALES OF APPROACHES TO CONGESTION CONTROL

The most basic way to avoid congestion is to build a network that is well matched to the traffic that it carries. If there is a low-bandwidth link on the path along which most traffic is directed, congestion is likely. Sometimes resources can be added dynamically when there is serious congestion.

For example, turning on spare routers or enabling lines that are normally used only as backups (to make the system fault tolerant) or purchasing bandwidth on the open market. More often, links and routers that are regularly heavily utilized are upgraded at the earliest opportunity. This is called **provisioning** and happens on a time scale of months, driven by long-term traffic trends.

To make the most of the existing network capacity, routes can be tailored to traffic patterns that change during the day as network user's wake and sleep in different time zones. For example, routes may be changed to shift traffic away from heavily used paths by changing the shortest path weights.

Some local 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 hotspots. This is called **traffic-aware routing**. Splitting traffic across multiple paths is also helpful.

However, sometimes it is not possible to increase capacity. The only way then to beat back the congestion is to decrease the load. In a virtual-circuit network, new connections can be refused if they would cause the network to become congested. This is called **admission control**.

TRAFFIC-AWARE ROUTING:

The first approach we will examine is traffic-aware routing. These schemes adapted to changes in topology, but not to changes in load; the goal in taking load into account when computing routes is to shift traffic away from hotspots that will be the first places in the network to experience congestion.

The most direct way to do this is to set the link weight to be a function of the (fixed) link bandwidth and propagation delay plus the (variable) measured load or average queuing delay. Least-weight paths will then favor paths that are more lightly loaded, all else being equal.

ADMISSION CONTROL:

One technique that is widely used in virtual-circuit networks to keep congestion at bay is **admission control**. The idea is simple: do not set up a new virtual circuit unless the network can carry the added traffic without becoming congested. Thus, attempts to set up a virtual circuit may fail. This is better than the alternative, as letting more people in when the network is busy just makes matters worse.

By analogy, in the telephone system, when a switch gets overloaded it practices admission control by not giving dial tones. The trick with this approach is working out when a new virtual circuit will lead to congestion. The task is straightforward in the telephone network because of the fixed bandwidth of calls (64 kbps for uncompressed audio).

However, virtual circuits in computer networks come in all shapes and sizes. Thus, the circuit must come with some characterization of its traffic if we are to apply admission control.

Traffic is often described in terms of its rate and shape. The problem of how to describe it in a simple yet meaningful way is difficult because traffic is typically bursty—the average rate is only half the story.

For example, traffic that varies while browsing the Web is more difficult to handle than a streaming movie with the same long-term throughput because the bursts of Web traffic are more likely to congest routers in the network.

A commonly used descriptor that captures this effect is the **leaky bucket** or **token bucket**. A leaky bucket has two parameters that bound the average rate and the instantaneous burst size of traffic. Leaky buckets are widely used for quality of service.

TRAFFIC THROTTLING:

In the Internet and many other computer networks, senders adjust their transmissions to send as much traffic as the network can readily deliver. In this setting, the network aims to operate just before the onset of congestion.

When congestion is imminent, it must tell the senders to throttle back their transmissions and slow down. This feedback is business as usual rather than an exceptional situation. The term **congestion avoidance** is sometimes used to contrast this operating point with the one in which the network has become (overly) congested.

Let us now look at some approaches to throttling traffic that can be used in both datagram networks and virtual-circuit networks. Each approach must solve two problems. First, routers must determine when congestion is approaching, ideally before it has arrived. To do so, each router can continuously monitor the resources it is using.

Three possibilities are the utilization of the output links, the buffering of queued packets inside the router, and the number of packets that are lost due to insufficient buffering. Of these possibilities, the second one is the most useful.

Averages of utilization do not directly account for the burstiness of most traffic—a utilization of 50% may be low for smooth traffic and too high for highly variable traffic. Counts of packet losses come too late. Congestion has already set in by the time that packets are lost.

Choke Packets:

The most direct way to notify a sender of congestion is to tell it directly. In this approach, the router selects a congested packet and sends a **choke packet** back to the source host, giving it the destination found in the packet.

The original packet may be tagged (a header bit is turned on) so that it will not generate any more choke packets farther along the path and then forwarded in the usual way. To avoid increasing load on the network during a time of congestion, the router may only send choke packets at a low rate.

When the source host gets the choke packet, it is required to reduce the traffic sent to the specified destination, for example, by 50%. In a datagram network, simply picking packets at random when there is congestion is likely to cause choke packets to be sent to fast senders, because they will have the most packets in the queue.

The feedback implicit in this protocol can help prevent congestion yet not throttle any sender unless it causes trouble. For the same reason, it is likely that multiple choke packets will be sent to a given host and destination.

The host should ignore these additional chokes for the fixed time interval until its reduction in traffic takes effect. After that period, further choke packets indicate that the network is still congested.

Explicit Congestion Notification:

Instead of generating additional packets to warn of congestion, a router can tag any packet it forwards (by setting a bit in the packet's header) to signal that it is experiencing congestion.

When the network delivers the packet, the destination can note that there is congestion and inform the sender when it sends a reply packet. The sender can then throttle its transmissions as before. This design is called **ECN** (**Explicit Congestion Notification shown in figure 3.10**) and is used in the Internet.

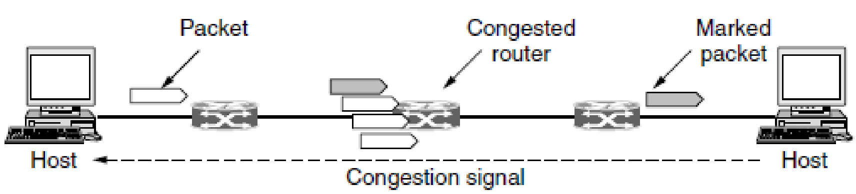


FIGURE 3.10: EXPLICIT CONGESTION NOTIFICATION

LOAD SHEDDING: It 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.

QUALITY OF SERVICE:

An easy solution to provide good quality of service is to build a network with enough capacity for whatever traffic will be thrown at it. The name for this solution is **over provisioning**. The resulting network will carry application traffic without significant loss and, assuming a decent routing scheme, will deliver packets with low latency. Performance doesn't get any better than this.

To some extent, the telephone system is over provisioned because it is rare to pick up a telephone and not get a dial tone instantly. There is simply so much capacity available that demand can almost always be met. The trouble with this solution is that it is expensive.

Four issues must be addressed to ensure quality of service:

1. What applications need from the network?
2. How to regulate the traffic that enters the network.
3. How to reserve resources at routers to guarantee performance.
4. Whether the network can safely accept more traffic.

No single technique deals efficiently with all these issues. Instead, a variety of techniques have been developed for use at the network (and transport) layer. Practical quality-of-service solutions combine multiple techniques. To this end, we will describe two versions of quality of service for the Internet called Integrated Services and Differentiated Services.

APPLICATION REQUIREMENTS:

A stream of packets from a source to a destination is called a **flow.** A flow might be all the packets of a connection in a connection-oriented network, or all the packets sent from one process to another process in a connectionless network. The needs of each flow can be characterized by four primary parameters: ***bandwidth***, ***delay***, ***jitter***, and ***loss***. Together, these determine the **QoS** (**Quality of Service**) the flow requires.

Several common applications and the stringency (meaning toughness/flexibility) of their network requirements are listed in Fig. 3.11. The applications differ in their bandwidth needs, with email, audio in all forms, and remote login not needing much, but file sharing and video in all forms needing a great deal.

More interesting are the delay requirements. File transfer applications, including email 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 too long, 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 variation (i.e., standard deviation) in the delay or packet arrival times is called **jitter**. The first three applications in Fig. 3.11 are not sensitive to the packets arriving with irregular time intervals between them. Remote login is somewhat sensitive to that, since updates 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 unless the application hides the jitter. For audio, a jitter of even a few milliseconds is clearly audible.

Application	Bandwidth	Delay	Jitter	Loss
Email	Low	Low	Low	Medium
File sharing	High	Low	Low	Medium
Web access	Medium	Medium	Low	Medium
Remote login	Low	Medium	Medium	Medium
Audio on demand	Low	Low	High	Low
Video on demand	High	Low	High	Low
Telephony	Low	High	High	Low
Videoconferencing	High	High	High	Low

FIGURE 3.11: STRINGENCY OF APPLICATIONS' QUALITY-OF-SERVICE REQUIREMENTS

To accommodate a variety of applications, networks may support different categories of QoS. An influential example comes from ATM networks. They support:

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 on demand).
4. Available bit rate (e.g., file transfer).

These categories are also useful for other purposes and other networks.

TRAFFIC SHAPING: Before the network can make QoS guarantees, it must know what traffic is being guaranteed. In the telephone network, this characterization is simple. For example, a voice call (in uncompressed format) needs 64 kbps and consists of one 8-bit sample every 125 μsec.

However, traffic in data networks is **bursty**. It typically arrives at nonuniform rates as the traffic rate varies (e.g., videoconferencing with compression), users interact with applications (e.g., browsing a new Web page), and computers switch between tasks. Bursts of traffic are more difficult to handle than constant-rate traffic because they can fill buffers and cause packets to be lost.

Traffic shaping is a technique for regulating the average rate and burstiness of a flow of data that enters the network. The goal is to allow applications to transmit a wide variety of traffic that suits their needs, including some bursts, yet have a simple and useful way to describe the possible traffic patterns to the network.

When a flow is set up, the user and the network (i.e., the customer and the provider) agree on a certain traffic pattern (i.e., shape) for that flow. In effect, the customer says to the provider ''my transmission pattern will look like this; can you handle it?''

Sometimes this agreement is called an **SLA** (**Service Level Agreement**), especially when it is made over aggregate flows and long periods of time, such as all of the traffic for a given customer. As long as the customer fulfills her part of the bargain and only sends packets according to the agreed-on contract, the provider promises to deliver them all in a timely fashion.

Traffic shaping reduces congestion and thus helps the network live up to its promise. However, to make it work, there is also the issue of how the provider can tell if the customer is following the agreement and what to do if the customer is not. Packets in excess of the agreed pattern might be dropped by the network, or they might be marked as having lower priority. Monitoring a traffic flow is called **traffic policing**.

PACKET SCHEDULING:

Being able to regulate the shape of the offered traffic is a good start. However, to provide a performance guarantee, we must reserve sufficient resources along the route that the packets take through the network. To do this, we are assuming that the packets of a flow 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.

Algorithms that allocate router resources among the packets of a flow and between competing flows are called **packet scheduling algorithms**. Three different kinds of resources can potentially be reserved for different flows:

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 buffered inside the router until it can be transmitted on the chosen outgoing line. The purpose of the buffer is to absorb small bursts of traffic as the flows contend with each other.

If no buffer is available, the packet has to be discarded since there is no place to put it. For good quality of service, some buffers might be reserved for a specific flow so that flow does not have to compete for buffers with other flows. Up to some maximum value, there will always be a buffer available when the flow needs one.

Finally, CPU cycles may also be 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. While modern routers are able to process most packets quickly, some kinds of packets require greater CPU processing, such as the ICMP packets. Making sure that the CPU is not overloaded is needed to ensure timely processing of these packets.

INTERNETWORKING:

HOW NETWORKS DIFFER:

Networks can differ in many ways. Some of the differences, such as different modulation techniques or frame formats, are internal to the physical and data link layers. These differences will not concern us here. Instead, in Fig. 3.12 we list some of the differences that can be exposed to the network layer. It is papering over these differences that makes internetworking more difficult than operating within a single network.

When packets sent by a source on one network must transit one or more foreign networks before reaching the destination network, many problems can occur at the interfaces between networks. To start with, the source needs to be able to address the destination.

What do we do if the source is on an Ethernet network and the destination is on a WiMAX network? Assuming we can even specify a WiMAX destination from an Ethernet network, packets would cross from a connectionless network to a connection-oriented one.

This may require that a new connection be set up on short notice, which injects a delay, and much overhead if the connection is not used for many more packets. Many specific differences may have to be accommodated as well. How do we multicast a packet to a group with some members on a network that does not support multicast?

The differing max packet sizes used by different networks can be a major nuisance, too. How do you pass an 8000-byte packet through a network whose maximum size is 1500 bytes? If packets on a connection-oriented network transit a connectionless network, they may arrive in a different order than they were sent. That is something the sender likely did not expect, and it might come as an (unpleasant) surprise to the receiver as well.

Item	Some Possibilities
Service offered	Connectionless versus connection oriented
Addressing	Different sizes, flat or hierarchical
Broadcasting	Present or absent (also multicast)
Packet size	Every network has its own maximum
Ordering	Ordered and unordered delivery
Quality of service	Present or absent; many different kinds
Reliability	Different levels of loss
Security	Privacy rules, encryption, etc.
Parameters	Different timeouts, flow specifications, etc.
Accounting	By connect time, packet, byte, or not at all

FIGURE 3.12: SOME OF THE MANY WAYS NETWORKS CAN DIFFER.

How Networks Can Be Connected

There are two basic choices for connecting different networks: we can build devices that translate or convert packets from each kind of network into packets for each other network, or, like good computer scientists, we can try to solve the problem by adding a layer of indirection and building a common layer on top of the different networks. In either case, the devices are placed at the boundaries between networks.

Internetworking has been very successful at building large networks, but it only works when there is a common network layer. There have, in fact, been many network protocols over time. Getting everybody to agree on a single format is difficult when companies perceive it to their commercial advantage to have a proprietary format that they control.

A router that can handle multiple network protocols is called a **multiprotocol router**. It must either translate the protocols, or leave connection for a higher protocol layer. Neither approach is entirely satisfactory. Connection at a higher layer, say, by using TCP, requires that all the networks implement TCP (which may not be the case). Then, it limits usage across the networks to applications that use TCP (which does not include many real-time applications).

TUNNELING:

Handling the general case of making two different networks interwork is exceedingly difficult. However, there is a common special case that is manageable even for different network protocols. 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 an IPv6 network in Paris, an IPv6 network in London and connectivity between the offices via the IPv4 Internet. This situation is shown in Fig. 3.13.

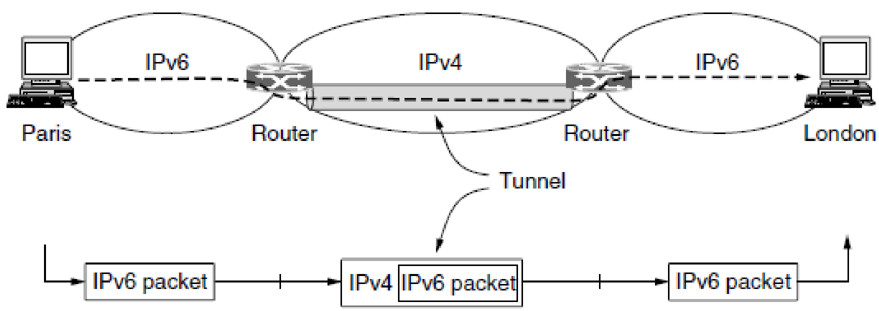


FIGURE 3.13: TUNNELING A PACKET FROM PARIS TO LONDON

The solution to this problem is a technique called **tunneling**. To send an IP packet to a host in the London office, a host in the Paris office constructs the packet containing an IPv6 address in London, and sends it to the multiprotocol router that connects the Paris IPv6 network to the IPv4 Internet.

When this router gets the IPv6 packet, it encapsulates the packet with an IPv4 header addressed to the IPv4 side of the multiprotocol router that connects to the London IPv6 network.

That is, the router puts a (IPv6) packet inside a (IPv4) packet. When this wrapped packet arrives, the London router removes the original IPv6 packet and sends it onward to the destination host. The path through the IPv4 Internet can be seen as a big tunnel extending from one multiprotocol router to the other.

The IPv6 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 IPv4 at all. Neither do the hosts in Paris or London. Only the multiprotocol routers have to understand both IPv4 and IPv6 packets.

In effect, the entire trip from one multiprotocol router to the other is like a hop over a single link. Tunneling is widely used to connect isolated hosts and networks using other networks.

INTERNETWORK ROUTING:

Routing through an internet poses the same basic problem as routing within a single network, but with some added complications. To start, the networks may internally use different routing algorithms. For example, one network may use link state routing and another distance vector routing. Since link state algorithms need to know the topology but distance vector algorithms do not, this difference alone would make it unclear how to find the shortest paths across the internet.

Networks run by different operators lead to bigger problems. First, the operators may have different ideas about what is a good path through the network. One operator may want the route with the least delay, while another may want the most inexpensive route. This will lead the operators to use different quantities to set the shortest-path costs.

Finally, the internet may be much larger than any of the networks that comprise it. It may therefore require routing algorithms that scale well by using a hierarchy, even if none of the individual networks need to use a hierarchy.

All of these considerations lead to a two-level routing algorithm. Within each network, an **intradomain** or **interior gateway protocol** is used for routing. (''Gateway'' is an older term for ''router.'') It might be a link state protocol of the Kind.

Across the networks that make up the internet, an **interdomain** or **exterior gateway protocol** is used. The networks may all use different intradomain protocols, but they must use the same interdomain protocol.

*In the Internet, the interdomain routing protocol is called **BGP** (**Border Gateway Protocol**).*

There is one more important term to introduce. Since each network is operated independently of all the others, it is often referred to as an **AS** (**Autonomous System**). A good mental model for an AS is an ISP network. In fact, an ISP network may be comprised of more than one AS, if it is managed, or, has been acquired, as multiple networks. But the difference is usually not significant.

PACKET FRAGMENTATION: Each network or link 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 old maximum packet size they wish. Maximum payloads for some common technologies are 1500 bytes for Ethernet and 2272 bytes for 802.11. IP is more generous, allows for packets as big as 65,515 bytes.

Hosts usually prefer to transmit large packets because this reduces packet overheads such as bandwidth wasted on header bytes. An obvious internetworking problem appears when a large packet wants to travel through a network whose maximum packet size is too small. This nuisance has been a persistent issue, and solutions to it have evolved along with much experience gained on the Internet.

One solution is to make sure the problem does not occur in the first place. However, this is easier said than done. A source does not usually know the path a packet will take through the network to a destination, so it certainly does not know how small packets must be to get there. This packet size is called the **Path MTU** (**Path Maximum Transmission Unit**).

The alternative solution to the problem is to allow routers to break up packets into **fragments**, sending each fragment as a separate network layer packet. However, as every parent of a small child knows, converting a large object into small fragments is considerably easier than the reverse process.

THE NETWORK LAYER IN THE INTERNET

THE IP VERSION 4 PROTOCOL:

An appropriate place to start our study of the network layer in the Internet is with the format of the IP datagrams themselves. An IPv4 datagram consists of a header part and a body or payload part. The header has a 20-byte fixed part and a variable-length optional part. The header format is shown in Fig. 3.14. The bits are transmitted from left to right and top to bottom, with the high-order bit of the *Version* field going first. (This is a ‘‘big-endian’’ network byte order.

On little-endian machines, such as Intel x86 computers, a software conversion is required on both transmission and reception.) In retrospect, little-endian would have been a better choice, but at the time IP was designed, no one knew it would come to dominate computing.

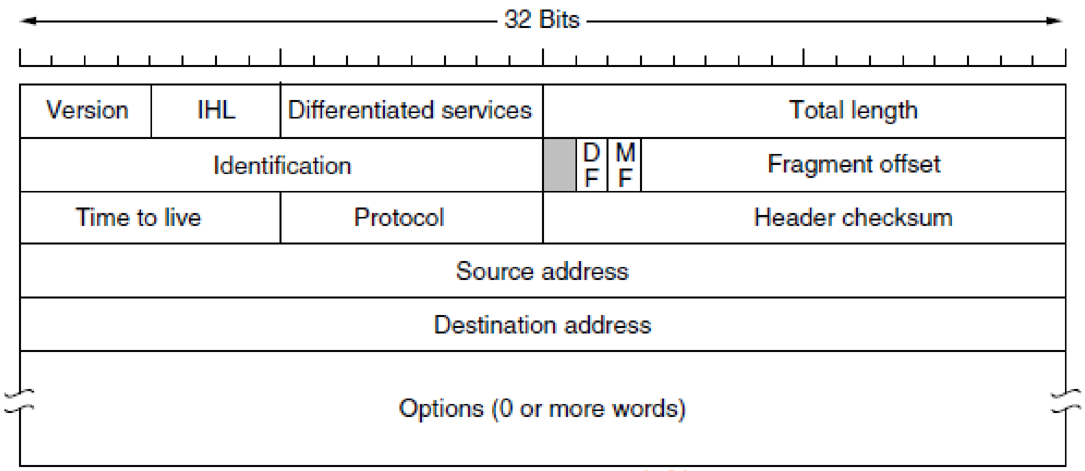


FIGURE 3.14: THE IPV4 (INTERNET PROTOCOL) HEADER

The *Version* field keeps track of which version of the protocol the datagram belongs to.

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.

The *Differentiated services* field is one of the few fields that have changed its meaning (slightly) over the years. Originally, it was called the *Type of service* field. 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. The *Type of service* field provided 3 bits to signal priority and 3 bits to signal whether a host cared more about delay, throughput, or reliability.

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 networks, larger datagrams may be needed.

The *Identification* field is needed to allow the destination host to determine which packet a newly arrived fragment belongs to. All the fragments of a packet contain the same *Identification* value.

DF stands for Don't Fragment. It is an order to the routers not to fragment the packet. Originally, it was intended to support hosts incapable of putting the pieces back together again.

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 *Fragment offset* tells where in the current packet 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, supporting a maximum packet length up to the limit of the *Total length* field. Working together, the *Identification*, *MF*, and *Fragment offset* fields are used to implement fragmentation.

The *TtL (Time to live)* field is a counter used to limit packet lifetimes. It was originally supposed to count time in seconds, allowing a maximum lifetime of 255 sec.

When the network layer has assembled a complete packet, it needs to know what to do with it. The *Protocol* field tells it which transport process to give the packet to. TCP is one possibility, but so are UDP and some others.

Since the header carries vital information such as addresses, it rates its own checksum for protection, the *Header checksum*. The algorithm is to add up all the 16-bit halfwords of the header as they arrive, using one's complement arithmetic, and then take the one's complement of the result. For purposes of this algorithm, the *Header checksum* is assumed to be zero upon arrival. Such a checksum is useful for detecting errors while the packet travels through the network.

The *Source address* and *Destination address* indicate the IP address of the source and destination network interfaces.

The *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 of variable length. The *Options* field is padded out to a multiple of 4 bytes. Originally, the five options listed in Fig. 3.15.

Option	Description
Security	Specifies how secret the datagram is
Strict source routing	Gives the complete path to be followed
Loose source routing	Gives a list of routers not to be missed
Record route	Makes each router append its IP address
Timestamp	Makes each router append its address and timestamp

FIGURE 3.15: SOME OF THE IP OPTIONS

IPV4 ADDRESSES:

The identifier used in the IP layer of the TCP/IP protocol suite to identify the connection of each device to the Internet is called the Internet address or IP address. An IPv4 address is a 32-bit address that uniquely and universally defines the connection of a host or a router to the Internet. The IP address is the address of the connection, not the host or the router, because if the device is moved to another network, the IP address may be changed.

IPv4 addresses are unique in the sense that each address defines one, and only one, connection to the Internet. If a device has two connections to the Internet, via two networks, it has two IPv4 addresses. IPv4 addresses are universal in the sense that the addressing system must be accepted by any host that wants to be connected to the Internet.

Address Space

A protocol like IPv4 that defines addresses has an address space. An **address space** is the total number of addresses used by the protocol. If a protocol uses *b* bits to define an address, the address space is 2^b because each bit can have two different values (0 or 1). IPv4 uses 32-bit addresses, which means that the address space is 232 or 4,294,967,296 (more than four billion). If there were no restrictions, more than 4 billion devices could be connected to the Internet.

Notation

There are three common notations to show an IPv4 address: binary notation (base 2), dotted-decimal notation (base 256), and hexadecimal notation (base 16). In *binary notation,* an IPv4 address is displayed as 32 bits. To make the address more readable, one or more spaces are usually inserted between each octet (8 bits). Each octet is often referred to as a byte. To make the IPv4 address more compact and easier to read, it is usually written in decimal form with a decimal point (dot) separating the bytes.

This format is referred to as *dotted-decimal notation*. Note that because each byte (octet) is only 8 bits, each number in the dotted-decimal notation is between 0 and 255. We sometimes see an IPv4 address in hexadecimal notation. Each hexadecimal digit is equivalent to four bits. This means that a 32-bit address has 8 hexadecimal digits. This notation is often used in network programming. Figure 3.16 shows an IP address in the three discussed notations.

HIERARCHY IN ADDRESSING: A 32-bit IPv4 address is also hierarchical, but divided only into two parts. The first part of the address, called the *prefix*, defines the network; the second part of the address, called the *suffix*, defines the node (connection of a device to the Internet).

Figure 3.17 shows the prefix and suffix of a 32-bit IPv4 address. The prefix length is *n* bits and the suffix length is (32 − *n*) bits.

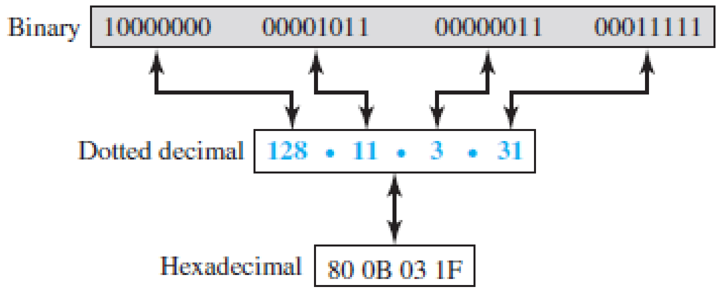


FIGURE 3.16: THREE DIFFERENT NOTATIONS IN IPV4 ADDRESSING

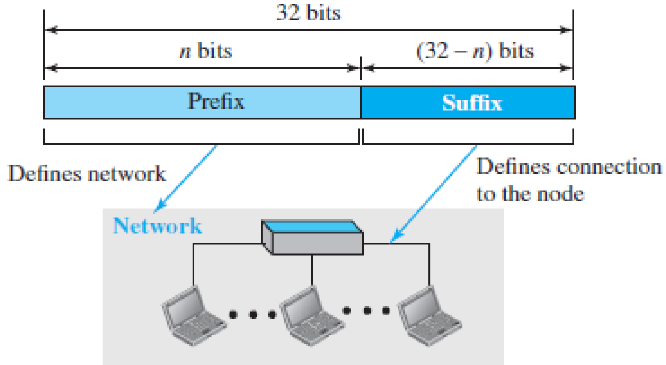


FIGURE 3.17: HIERARCHY IN ADDRESSING

A prefix can be fixed length or variable length. The network identifier in the IPv4 was first designed as a fixed-length prefix. This scheme, which is now obsolete, is referred to as classful addressing. The new scheme, which is referred to as classless addressing, uses a variable-length network prefix. First, we briefly discuss Classful addressing; then we concentrate on classless addressing.

Classful Addressing:

When the Internet started, an IPv4 address was designed with a fixed-length prefix, but to accommodate both small and large networks, three fixed-length prefixes were designed instead of one (*n* = 8, *n* = 16, and *n* = 24). The whole address space was divided into five classes (class A, B, C, D, and E), as shown in Figure 3.18. This scheme is referred to as **classful addressing.**

In class A, the network length is 8 bits, but since the first bit, which is 0, defines the class, we can have only seven bits as the network identifier. This means there are only 27 = 128 networks in the world that can have a class A address.

In class B, the network length is 16 bits, but since the first two bits, which are (10)2, define the class, we can have only 14 bits as the network identifier. This means there are only 214 = 16,384 networks in the world that can have a class B address.

All addresses that start with (110)2 belong to class C. In class C, the network length is 24 bits, but since three bits define the class, we can have only 21 bits as the network identifier. This means there are 221 = 2,097,152 networks in the world that can have a class C address.

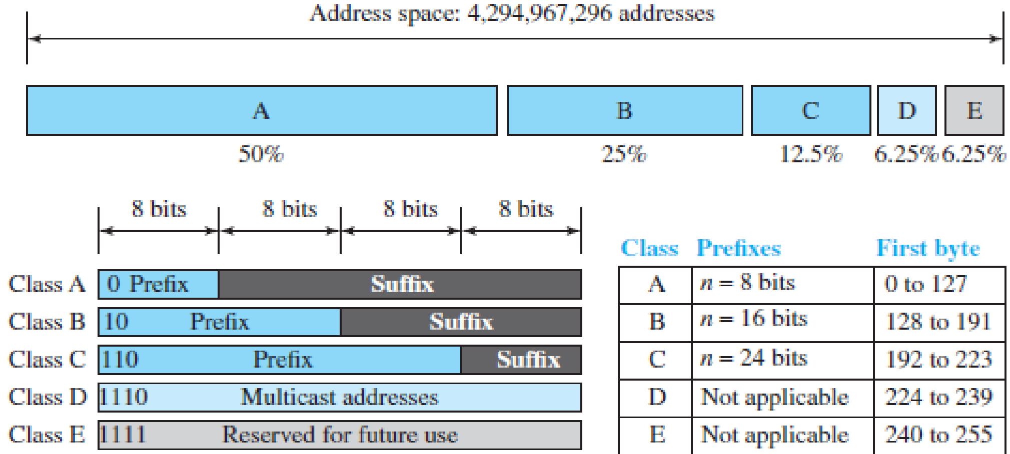


FIGURE 3.18: OCCUPATION OF THE ADDRESS SPACE IN CLASSFUL ADDRESSING

Class D is not divided into prefix and suffix. It is used for multicast addresses. All addresses that start with 1111 in binary belong to class E. As in Class D, Class E is not divided into prefix and suffix and is used as reserve.

Advantage of Classful Addressing:

Although classful addressing had several problems and became obsolete, it had one advantage: Given an address, we can easily find the class of the address and, since the prefix length for each class is fixed, we can find the prefix length immediately. In other words, the prefix length in classful addressing is inherent in the address; no extra information is needed to extract the prefix and the suffix.

Address Depletion: The reason that classful addressing has become obsolete is address depletion. Since the addresses were not distributed properly, the Internet was faced with the problem of the addresses being rapidly used up, resulting in no more addresses available for organizations and individuals that needed to be connected to the Internet.

Subnetting and Supernetting: To alleviate address depletion, two strategies were proposed and, to some extent, implemented: subnetting and Supernetting. In subnetting, a class A or class B block is divided into several subnets.

Each subnet has a larger prefix length than the original network. While subnetting was devised to divide a large block into smaller ones, Supernetting was devised to combine several class C blocks into a larger block to be attractive to organizations that need more than the 256 addresses available in a class C block. This idea did not work either because it makes the routing of packets more difficult.

Classless Addressing:

Subnetting and Supernetting in classful addressing did not really solve the address depletion problem. With the growth of the Internet, it was clear that a larger address space was needed as a long-term solution. The larger address space, however, requires that the length of IP addresses also be increased, which means the format of the IP packets needs to be changed.

Although the long-range solution has already been devised and is called IPv6, a short-term solution was also devised to use the same address space but to change the distribution of addresses to provide a fair share to each organization. The short-term solution still uses IPv4 addresses, but it is called *classless addressing*. In other words, the class privilege was removed from the distribution to compensate for the address depletion.

In classless addressing, the whole address space is divided into variable length blocks. The prefix in an address defines the block (network); the suffix defines the node (device). Theoretically, we can have a block of 20, 21, 22, . . . , 232 addresses. One of the restrictions, as we discuss later, is that the number of addresses in a block needs to be a power of 2. An organization can be granted one block of addresses. Figure 3.19 shows the division of the whole address space into nonoverlapping blocks.

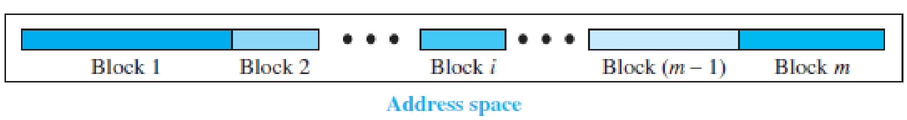


FIGURE 3.19: VARIABLE-LENGTH BLOCKS IN CLASSLESS ADDRESSING

Unlike classful addressing, the prefix length in classless addressing is variable. We can have a prefix length that ranges from 0 to 32. The size of the network is inversely proportional to the length of the prefix. A small prefix means a larger network; a large prefix means a smaller network.

We need to emphasize that the idea of classless addressing can be easily applied to classful addressing. An address in class A can be thought of as a classless address in which the prefix length is 8. An address in class B can be thought of as a classless address in which the prefix is 16, and so on. In other words, classful addressing is a special case of classless addressing.

Prefix Length: Slash Notation:

The first question that we need to answer in classless addressing is how to find the prefix length if an address is given. Since the prefix length is not inherent in the address, we need to separately give the length of the prefix. In this case, the prefix length, *n*, is added to the address, separated by a slash. The notation is informally referred to as *slash notation* and formally as ***classless interdomain routing*** or ***CIDR*** (pronounced cider) strategy. An address in classless addressing can then be represented as shown in Figure 3.20.

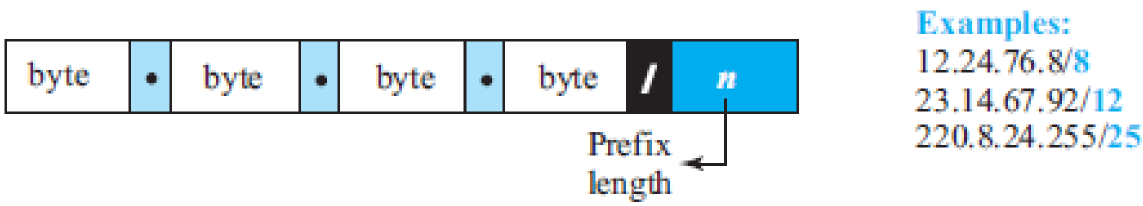


FIGURE 3.20: SLASH NOTATION (CIDR)

Extracting Information from an Address:

Given any address in the block, we normally like to know three pieces of information about the block to which the address belongs: the number of addresses, the first address in the block, and the last address. Since the value of prefix length, *n*, is given, we can easily find these three pieces of information, as shown in Figure 3.21.

1. The number of addresses in the block is found as $N = 2^{32-n}$.
2. To find the first address, we keep the *n* leftmost bits and set the $(32 - n)$ rightmost bits all to 0s.
3. To find the last address, we keep the *n* leftmost bits and set the $(32 - n)$ rightmost bits all to 1s.

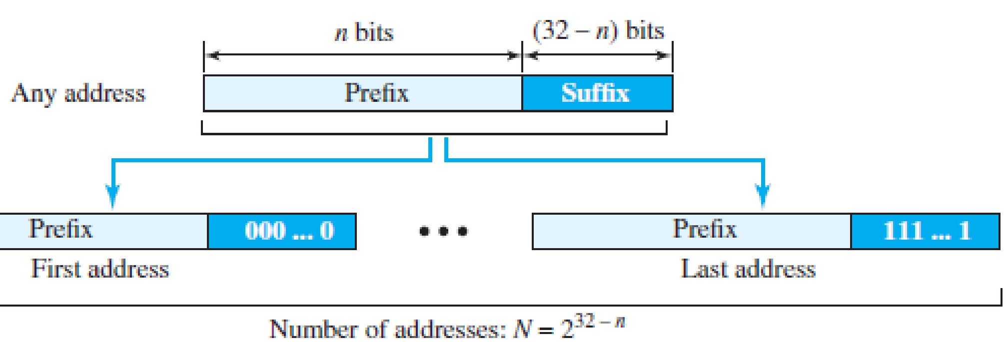


FIGURE 3.21: INFORMATION EXTRACTION IN CLASSLESS ADDRESSING

Example:

A classless address is given as 167.199.170.82/**27**. We can find the above three pieces of information as follows. The number of addresses in the network is 2^{32-n} = 25 = 32 addresses.

The first address can be found by keeping the first 27 bits and changing the rest of the bits to 0s.

Address: 167.199.170.82/**27** **10100111 11000111 10101010 01010010**

First address: 167.199.170.64/**27** **10100111 11000111 10101010 01000000**

The last address can be found by keeping the first 27 bits and changing the rest of the bits to 1s.

Address: 167.199.170.82/**27** **10100111 11000111 10101010 01011111**

Last address: 167.199.170.95/**27** **10100111 11000111 10101010 01011111**

IP VERSION 6:

IP has been in heavy use for decades. It has worked extremely well, as demonstrated by the exponential growth of the Internet. Unfortunately, IP has become a victim of its own popularity: it is close to running out of addresses. Even with CIDR and NAT using addresses more sparingly, the last IPv4 addresses are expected to be assigned by ICANN before the end of 2012.

IPv6 (**IP version 6**) is a replacement design that does just that. It uses 128-bit addresses; a shortage of these addresses is not likely any time in the foreseeable future. However, IPv6 has proved very difficult to deploy. It is a different network layer protocol that does not really interwork with IPv4, despite many similarities. Also, companies and users are not really sure why they should want IPv6 in any case.

In 1990 IETF started work on a new version of IP, one that 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 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).

5. Pay more attention to the 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.

The design of IPv6 presented a major opportunity to improve all of the features in IPv4 that fall short of what is now wanted. One proposal was to run TCP over CLNP, the network layer protocol designed for OSI. With its 160-bit addresses, CLNP would have provided enough address space forever.

IPv6 meets IETF's 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, with small modifications being required to deal with longer addresses.

The main features of IPv6 are discussed below.

- First and foremost, IPv6 has longer addresses than IPv4. They are 128 bits long, which solves the problem that IPv6 set out to solve: providing an effectively unlimited supply of Internet addresses.
- 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 improves throughput and delay.
- The third major improvement is better support for options. This change was essential with the new header because fields that previously were required are now optional (because they are not used so often).
    - 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.
- Finally, more attention has been paid to quality of service.

The Main IPv6 Header:

The IPv6 header is shown in Fig. 3.22. The *Version* field is always 6 for IPv6 (and 4 for IPv4). During the transition period from IPv4, which has already taken more than 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, given that the data link header usually indicates the network protocol for demultiplexing, so some routers may skip the check.

The *Differentiated services* field (originally called *Traffic class*) is used to distinguish the class of service for packets with different real-time delivery requirements.

The *Flow label* field provides a way for a source and destination to mark groups of packets that have the same requirements and should be treated in the same way by the network, forming a pseudo connection.

The *Payload length* field tells how many bytes follow the 40-byte header of Fig. 3.22. 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). This change means the payload can now be 65,535 bytes instead of a mere 65,515 bytes.

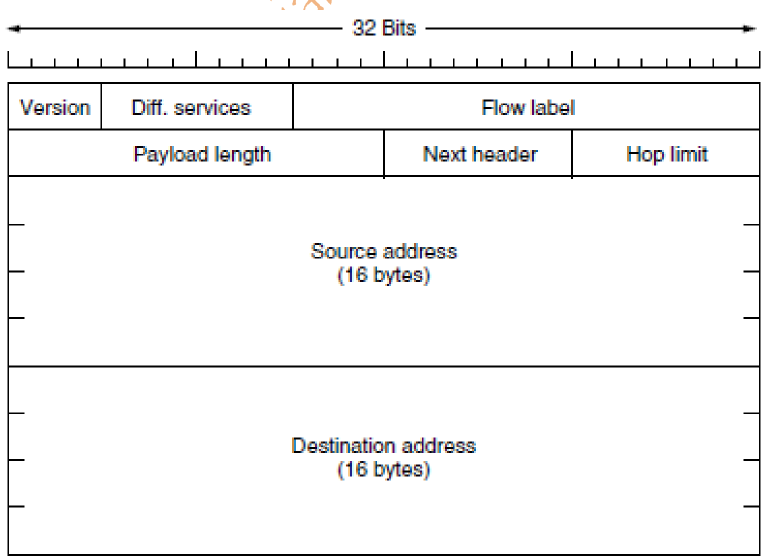


FIGURE 3.22: THE IPV6 FIXED HEADER (REQUIRED)

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

Next come the *Source address* and *Destination address* fields. 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:

8000:0000:0000:0000:0123:4567:89AB: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

INTERNET CONTROL PROTOCOLS:

In addition to IP, which is used for data transfer, the Internet has several companion control protocols that are used in the network layer. They include ICMP, ARP, and DHCP.

IMCP—The Internet Control Message Protocol:

The operation of the Internet is monitored closely by the routers. When something unexpected occurs during packet processing at a router, the event is reported to the sender by the **ICMP** (**Internet Control Message Protocol**). ICMP is also used to test the Internet. About a dozen types of ICMP messages are defined. Each ICMP message type is carried encapsulated in an IP packet. The most important ones are listed in Fig. 3.23.

Message type	Description
Destination unreachable	Packet could not be delivered
Time exceeded	Time to live field hit 0
Parameter problem	Invalid header field
Source quench	Choke packet
Redirect	Teach a router about geography
Echo and echo reply	Check if a machine is alive
Timestamp request/reply	Same as Echo, but with timestamp
Router advertisement/solicitation	Find a nearby router

FIGURE 3.23: THE PRINCIPAL ICMP MESSAGE TYPES

The DESTINATION UNREACHABLE message is used when the 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 *TtL (Time to live)* counter has reached zero. This event is a symptom that packets are looping, or that the counter 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 long ago used to throttle hosts that were sending too many packets. When a host received this message, it was expected to slow down.

The REDIRECT message is used when a router notices that a packet seems to be routed incorrectly. It is used by the router to tell the sending host to update to a better route.

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 can be used to measure network performance.

OSPF—AN INTERIOR GATEWAY ROUTING PROTOCOL:

The Internet is made up of a large number of independent networks or **ASes** (**Autonomous Systems**) that are operated by different organizations, usually a company, university, or ISP. Inside of its own network, an organization can use its own algorithm for internal routing, or **intradomain routing**, as it is more commonly known. Nevertheless, there are only a handful of standard protocols that are popular.

An intradomain routing protocol is also called an **interior gateway protocol**. We will study the problem of routing between independently operated networks, or **interdomain routing**. For that case, all networks must use the same interdomain routing protocol or **exterior gateway protocol**. The protocol that is used in the Internet is BGP (Border Gateway Protocol).

Early intradomain routing protocols used a distance vector design, based on the distributed Bellman-Ford algorithm inherited from the ARPANET. It works well in small systems, but less well as networks get larger. It also suffers from the count-to-infinity problem and generally slow convergence.

The ARPANET switched over to a link state protocol in May 1979 because of these problems, and in 1988 IETF began work on a link state protocol for intradomain routing. That protocol, called **OSPF** (**Open Shortest Path First**), became a standard in 1990. It drew on a protocol called **IS-IS** (**Intermediate-System to Intermediate-System**), which became an ISO standard.

Given the long experience with other routing protocols, the group designing OSPF 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.

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. At the time, IP had 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, OSPF had to do load balancing, splitting the load over multiple lines. Most previous protocols sent all packets over a single best route, even if there were two routes that were equally good. The other route was not used at all. In many cases, splitting the load over multiple routes gives better performance.

Sixth, support for hierarchical systems was needed. By 1988, some networks had grown so large that no router could be expected to know the entire topology. OSPF had to be designed so that no router would have to.

OSPF supports both point-to-point links (e.g., SONET) and broadcast networks (e.g., most LANs). Actually, it is able to support networks with multiple routers, each of which can communicate directly with the others (called **multi-access networks**) even if they do not have broadcast capability. Earlier protocols did not handle this case well.

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.

In effect, it is acting as the single node that represents the LAN. 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. These messages gives its state and provide 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 link 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 link is brought up.

All these messages are sent directly in IP packets. The five kinds of messages are summarized in Fig. 3.24.

Message type	Description
Hello	Used to discover who the neighbors are
Link state update	Provides the sender's costs to its neighbors
Link state ack	Acknowledges link state update
Database description	Announces which updates the sender has
Link state request	Requests information from the partner

FIGURE 3.24: THE FIVE TYPES OF OSPF MESSAGES

BGP—THE EXTERIOR GATEWAY ROUTING PROTOCOL:

Within a single AS, OSPF and IS-IS are the protocols that are commonly used. Between ASes, a different protocol, called **BGP** (**Border Gateway Protocol**), is used. A different protocol is needed because the goals of an intradomain protocol and an interdomain protocol are not the same. All an intradomain protocol has to do is move packets as efficiently as possible from the source to the destination.

BGP is a form of distance vector protocol, but it is quite unlike intradomain distance vector protocols such as RIP. We have already seen that policy, instead of minimum distance, is used to pick which routes to use. Another large difference is that instead of maintaining just the cost of the route to each destination, each BGP router keeps track of the path used. This approach is called a **path vector protocol**.

The path consists of the next hop router (which may be on the other side of the ISP, not adjacent) and the sequence of ASes, or **AS path**, that the route has followed (given in reverse order). Finally, pairs of BGP routers communicate with each other by establishing TCP connections. Operating this way provides reliable communication and also hides all the details of the network being passed through.

An example of how BGP routes are advertised is shown in Fig. 3.25. There are three ASes and the middle one is providing transit to the left and right ISPs. A route advertisement to prefix *C* starts in *AS3*. When it is propagated across the link to *R2c* at the top of the figure, it has the AS path of simply *AS3* and the next hop router of *R3a*.

At the bottom, it has the same AS path but a different next hop because it came across a different link. This advertisement continues to propagate and crosses the boundary into *AS1*. At router *R1a*, at the top of the figure, the AS path is *AS2*, *AS3* and the next hop is *R2a*.

Carrying the complete path with the route makes it easy for the receiving router to detect and break routing loops. The rule is that each router that sends a route outside of the AS prepends its own AS number to the route. (This is why the list is in reverse order.)

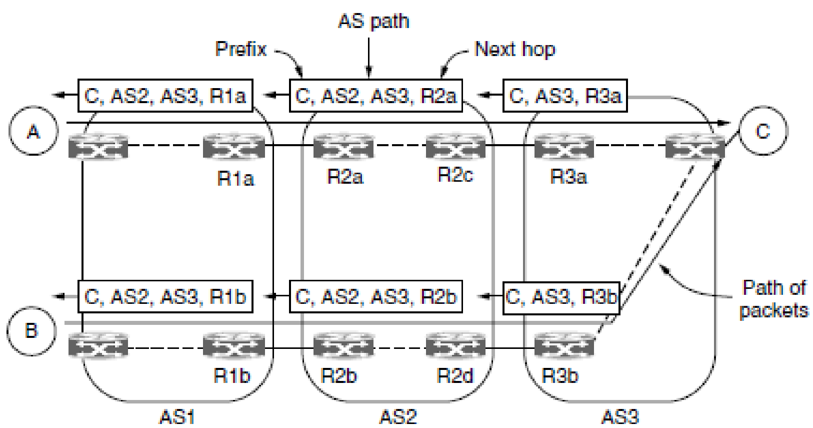


FIGURE 3.25: PROPAGATION OF BGP ROUTE ADVERTISEMENTS

When a router receives a route, it checks to see if its own AS number is already in the AS path. If it is, a loop has been detected and the advertisement is discarded.

INTERNET PROTOCOL (IP):

The network layer in version 4 can be thought of as one main protocol and three auxiliary ones. The main protocol, Internet Protocol version 4 (IPv4), is responsible for packetizing, forwarding, and delivery of a packet at the network layer.

The Internet Control Message Protocol version 4 (ICMPv4) helps IPv4 to handle some errors that may occur in the network-layer delivery. The Internet Group Management Protocol (IGMP) is used to help IPv4 in multicasting. The Address Resolution Protocol (ARP) is used to glue the network and data-link layers in mapping network-layer addresses to link-layer addresses. Figure 3.26 shows the positions of these four protocols in the TCP/IP protocol suite.

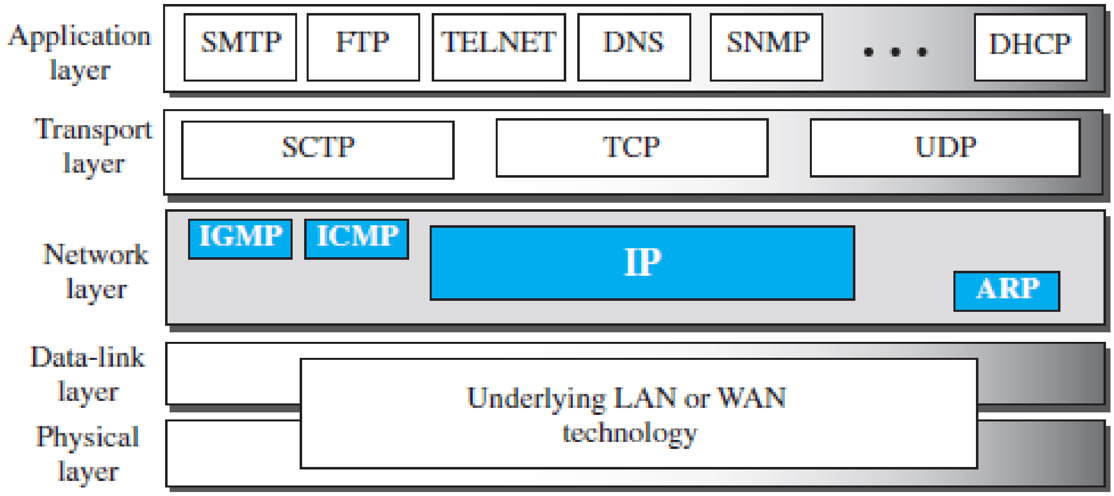


FIGURE 3.26: POSITION OF IP & OTHER NETWORK-LAYER PROTOCOLS IN TCP/IP PROTOCOL SUITE

IPv4 is an unreliable datagram protocol—a best-effort delivery service. The term *best-effort* means that IPv4 packets can be corrupted, be lost, arrive out of order, or be delayed, and may create congestion for the network. If reliability is important, IPv4 must be paired with a reliable transport-layer protocol such as TCP.

IPv4 is also a connectionless protocol that uses the datagram approach. This means that each datagram is handled independently, and each datagram can follow a different route to the destination. This implies that datagrams sent by the same source to the same destination could arrive out of order. Again, IPv4 relies on a higher-level protocol to take care of all these problems.

DATAGRAM FORMAT:

Packets used by the IP are called *datagrams*. Figure 3.27 shows the IPv4 datagram format. A datagram is a variable-length packet consisting of two parts: header and payload (data). The header is 20 to 60 bytes in length and contains information essential to routing and delivery. It is customary in TCP/IP to show the header in 4-byte sections.

- ***Version Number.*** The 4-bit version number (VER) field defines the version of the IPv4 protocol, which, obviously, has the value of 4.
- ***Header Length.*** The 4-bit header length (HLEN) field defines the total length of the datagram header in 4-byte words. The IPv4 datagram has a variable-length header.
- ***Service Type.*** In the original design of the IP header, this field was referred to as type of service (TOS), which defined how the datagram should be handled.

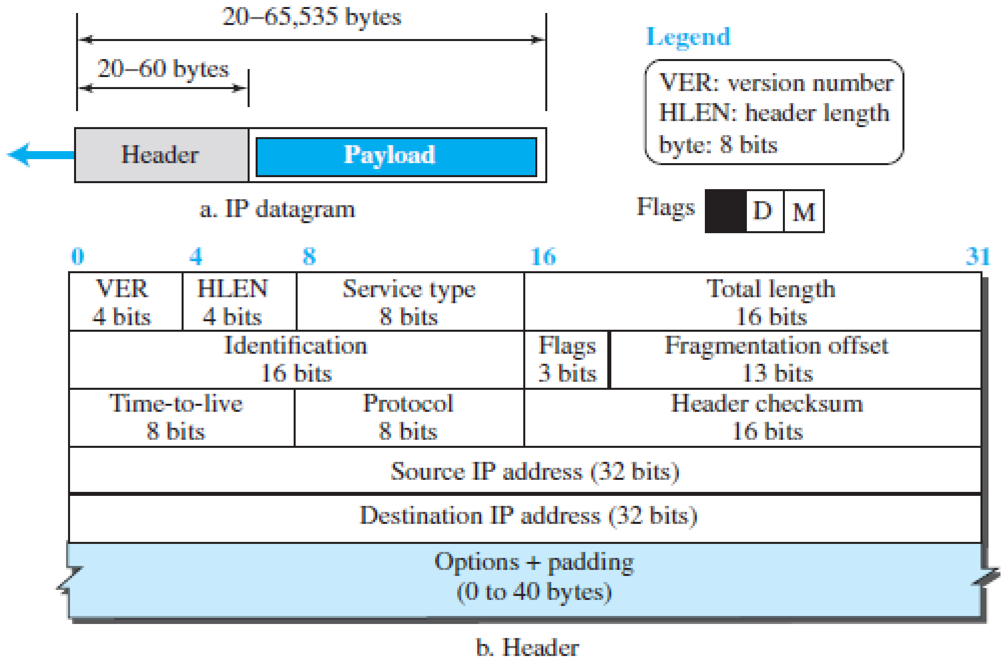


FIGURE 3.27: IP DATAGRAM

- ***Total Length*.** This 16-bit field defines the total length (header plus data) of the IP datagram in bytes. A 16-bit number can define a total length of up to 65,535 (when all bits are 1s).

- ***Identification, Flags, and Fragmentation Offset*.** These three fields are related to the fragmentation of the IP datagram when the size of the datagram is larger than the underlying network can carry.

- ***Time-to-live*.** The time-to-live (TTL) field is used to control the maximum number of hops (routers) visited by the datagram. When a source host sends the datagram, it stores a number in this field. This value is approximately two times the maximum number of routers between any two hosts. Each router that processes the datagram decrements this number by one. If this value, after being decremented, is zero, the router discards the datagram.

- ***Protocol*.** In TCP/IP, the data section of a packet, called the *payload,* carries the whole packet from another protocol. A datagram, for example, can carry a packet belonging to any transport-layer protocol such as UDP or TCP. A datagram can also carry a packet from other protocols that directly use the service of the IP, such as some routing protocols or some auxiliary protocols.

- ***Header checksum*.** IP is not a reliable protocol; it does not check whether the payload carried by a datagram is corrupted during the transmission. IP puts the burden of error checking of the payload on the protocol that owns the payload, such as UDP or TCP. The datagram header, however, is added by IP, and its error-checking is the responsibility of IP.

- ***Source and Destination Addresses*****.** These 32-bit source and destination address fields define the IP address of the source and destination respectively. The source host should know its IP address. The destination IP address is either known by the protocol that uses the service of IP or is provided by the DNS.

- ***Options.*** A datagram header can have up to 40 bytes of options. Options can be used for network testing and debugging. Although options are not a required part of the IP header, option processing is required of the IP software.

- ***Payload*****.** Payload, or data, is the main reason for creating a datagram. Payload is the packet coming from other protocols that use the service of IP. Comparing a datagram to a postal package, payload is the content of the package; the header is only the information written on the package.

ICMPv4:

The IPv4 has no error-reporting or error-correcting mechanism. The IP protocol also lacks a mechanism for host and management queries. A host sometimes needs to determine if a router or another host is alive. And sometimes a network manager needs information from another host or router.

The **Internet Control Message Protocol version 4 (ICMPv4)** has been designed to compensate for the above two deficiencies. It is a companion to the IP protocol. ICMP itself is a network-layer protocol.

However, its messages are not passed directly to the data-link layer as would be expected. Instead, the messages are first encapsulated inside IP datagrams before going to the lower layer. When an IP datagram encapsulates an ICMP message, the value of the protocol field in the IP datagram is set to 1 to indicate that the IP payroll is an ICMP message.

MESSAGES:

ICMP messages are divided into two broad categories: ***error-reporting messages*** and ***query messages***.

The ***error-reporting messages*** report problems that a router or a host (destination) may encounter when it processes an IP packet.

The ***query messages***, which occur in pairs, help a host or a network manager get specific information from a router or another host. For example, nodes can discover their neighbors. Also, hosts can discover and learn about routers on their network and routers can help a node redirect its messages.

An ICMP message has an 8-byte header and a variable-size data section. Although the general format of the header is different for each message type, the first 4 bytes are common to all.

As Figure 3.28 shows, the first field, ICMP type, defines the type of the message. The code field specifies the reason for the particular message type. The last common field is the checksum field (to be discussed later in the chapter). The rest of the header is specific for each message type.

The data section in error messages carries information for finding the original packet that had the error. In query messages, the data section carries extra information based on the type of query.

Error Reporting Messages: Since IP is an unreliable protocol, one of the main responsibilities of ICMP is to report some errors that may occur during the processing of the IP datagram. ICMP does not correct errors, it simply reports them.

Error correction is left to the higher-level protocols. Error messages are always sent to the original source because the only information available in the datagram about the route is the source and destination IP addresses.

ICMP uses the source IP address to send the error message to the source (originator) of the datagram. To make the error-reporting process simple, ICMP follows some rules in reporting messages:

- First, no error message will be generated for a datagram having a multicast address or special address (such as *this host* or *loopback*).
- Second, no ICMP error message will be generated in response to a datagram carrying an ICMP error message.
- Third, no ICMP error message will be generated for a fragmented datagram that is not the first fragment.

Destination Unreachable: The most widely used error message is the destination unreachable (type 3). This message uses different codes (0 to 15) to define the type of error message and the reason why a datagram has not reached its final destination.

Source Quench: Another error message is called the *source quench* (type 4) message, which informs the sender that the network has encountered congestion and the datagram has been dropped; the source needs to slow down sending more datagrams.

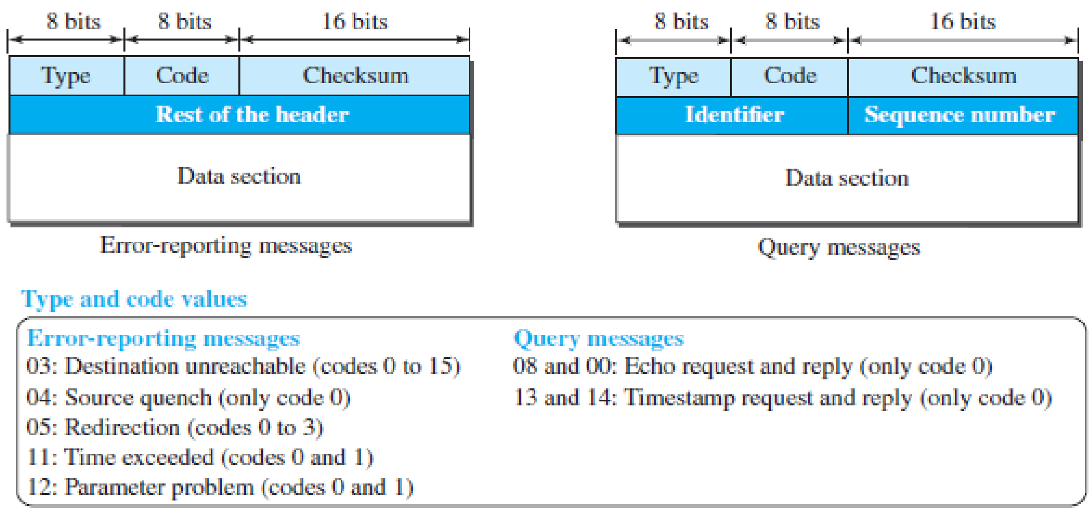


FIGURE 3.28: GENERAL FORMAT OF ICMP MESSAGES

Redirection Message: The *redirection message* (type 5) is used when the source uses a wrong router to send out its message. The router redirects the message to the appropriate router, but informs the source that it needs to change its default router in the future. The IP address of the default router is sent in the message.

Parameter Problem: A *parameter problem message* (type 12) can be sent when either there is a problem in the header of a datagram (code 0) or some options are missing or cannot be interpreted (code 1).

Query Messages: Query messages in ICMP can be used independently without relation to an IP datagram. Of course, a query message needs to be encapsulated in a datagram, as a carrier.

Query messages are used to probe or test the liveliness of hosts or routers in the Internet, find the one-way or the round-trip time for an IP datagram between two devices, or even find out whether the clocks in two devices are synchronized. Naturally, query messages come in pairs: request and reply.

IGMP:

The protocol that is used today for collecting information about group membership is the **Internet Group Management Protocol (IGMP).** IGMP is a protocol defined at the network layer; it is one of the auxiliary protocols, like ICMP, which is considered part of the IP. IGMP messages, like ICMP messages, are encapsulated in an IP datagram.

Messages:

There are only two types of messages in IGMP version 3, query and report messages, as shown in Figure 3.29. A query message is periodically sent by a router to all hosts attached to it to ask them to report their interests about membership in groups. A report message is sent by a host as a response to a query message.

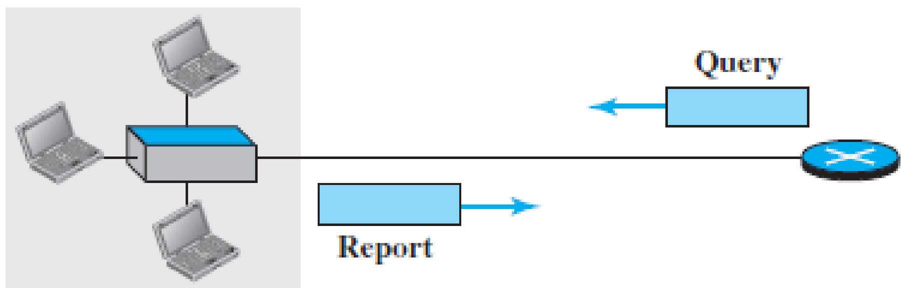


FIGURE 3.29: IGMP OPERATION

Query Message:

The query message is sent by a router to all hosts in each interface to collect information about their membership. There are three versions of query messages, as described below:

a. A ***general* query** message is sent about membership in any group. It is encapsulated in a datagram with the destination address 224.0.0.1 (all hosts and routers). Note that all routers attached to the same network receive this message to inform them that this message is already sent and that they should refrain from resending it.

b. A ***group-specific*** query message is sent from a router to ask about the membership related to a specific group. This is sent when a router does not receive a response about a specific group and wants to be sure that there is no active member of that group in the network. The group identifier (multicast address) is mentioned in the message. The message is encapsulated in a datagram with the destination address set to the corresponding multicast address. Although all hosts receive this message, those not interested drop it.

c. A ***source-and-group-specific*** query message is sent from a router to ask about the membership related to a specific group when the message comes from a specific source or sources. Again the message is sent when the router does not hear about a specific group related to a specific host or hosts. The message is encapsulated in a datagram with the destination address set to the corresponding multicast address. Although all hosts receive this message, those not interested drop it.

Report Message

A report message is sent by a host as a response to a query message. The message contains a list of records in which each record gives the identifier of the corresponding group (multicast address) and the addresses of all sources that the host is interested in receiving messages from (inclusion).

The record can also mention the source addresses from which the host does not desire to receive a group message (exclusion). The message is encapsulated in a datagram with the multicast address 224.0.0.22 (multicast address assigned to IGMPv3).

In IGMPv3, if a host needs to join a group, it waits until it receives a query message and then sends a report message. If a host needs to leave a group, it does not respond to a query message. If no other host responds to the corresponding message, the group is purged from the router database.

Propagation of Membership Information:

After a router has collected membership information from the hosts and other routers at its own level in the tree, it can propagate it to the router located in a higher level of the tree. Finally, the router at the tree root can get the membership information to build the multicast tree. The process, however, is more complex than what we can explain in one paragraph. Interested readers can check the book website for the complete description of this protocol.

Encapsulation: The IGMP message is encapsulated in an IP datagram with the value of the protocol field set to 2 and the TTL field set to 1. The destination IP address of the datagram, however, depends on the type of message, as shown in figure 3.30.

Message Type	*IP Address*
General Query	224.0.0.1
Other Queries	Group address
Report	224.0.0.22

FIGURE 3.30: DESTINATION IP ADDRESSES