

network layer provides a datagram service (in which case different packets between a given host-destination pair may take different routes) or a virtual circuit service (in which case all packets between a given source and destination will take the same path), the network layer must nonetheless determine the path for a packet. This is the job of the network layer routing protocol.

At the heart of any routing protocol is the algorithm (the "routing algorithm") that determines the path for a packet. The purpose of a routing algorithm is simple: given a set of routers, with links connecting the routers, a routing algorithm finds a "good" path from source to destination. Typically, a "good" path is one which has "least cost," but we will see that in practice, "real-world" concerns such as policy issues (e.g., a rule such as "router X, belonging to organization Y should not forward any packets originating from the network owned by organization Z") also come into play to complicate the conceptually simple and elegant algorithms whose theory underlies the practice of routing in today's networks.

The graph abstraction used to formulate routing algorithms is shown in Figure-3.6. (To view some graphs representing real network maps. Here, nodes in the graph represent routers - the points at which packet routing decisions are made - and the lines ("edges" in graph theory terminology) connecting these nodes represent the physical links between these routers. A link also has a value representing the "cost" of sending a packet across the link. The cost may reflect the level of congestion on that link (e.g., the current average delay for a packet across that link) or the physical distance traversed by that link (e.g., a transoceanic link might have a higher cost than a terrestrial link). For our current purposes, we will simply take the link costs as a given and won't worry about how they are determined.

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

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

In Figure-3.6, for example, the least cost path between nodes A (source) and C (destination) is along the path ADEC. (We will find it notationally easier to refer to the path in terms of the nodes on the path, rather than the links on the path).

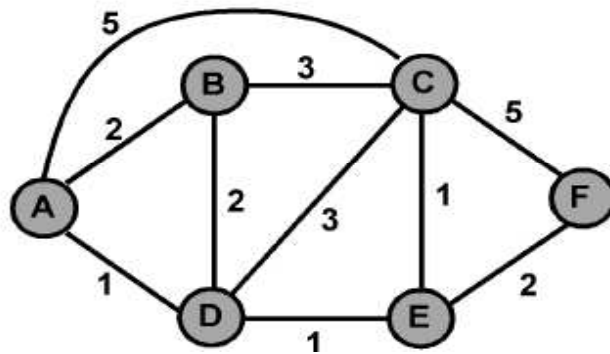


Figure 3.6: Abstract model of a network

Classification of Routing Algorithms

As a simple exercise, try finding the least cost path from nodes A to F, and reflect for a moment on how you calculated that path. If you are like most people, you found the path from A to F by examining Figure-3.6, tracing a few routes from A to F, and somehow convincing yourself that the path you had chosen was the least cost among all possible paths (Did you check all of the 12 possible paths between A and F? Probably not!). Such a calculation is an example of a centralized routing algorithm. Broadly, one way in which we can classify routing algorithms is according to whether they are centralized or decentralized:

- A global routing algorithm computes the least cost path between a source and destination using complete, global knowledge about the network. That is, the algorithm takes the connectivity between all nodes and all links costs as inputs. This then requires that the algorithm somehow obtain this information before actually performing the calculation. The calculation itself can be run at one site (a centralized global routing algorithm) or replicated at multiple sites. The key distinguishing feature here, however, is that a global algorithm has complete information about connectivity and link costs. In practice, algorithms with global state information are often referred to as link state algorithms, since the algorithm must be aware of the state (cost) of each link in the network.
- In a decentralized routing algorithm, the calculation of the least cost path is carried out in an iterative, distributed manner. No node has complete information about the costs of all network links. Instead, each node begins with only knowledge of the costs of its own directly attached links and then through an iterative process of calculation and exchange of information with its neighboring nodes (i. e., nodes which are at the "other end" of links to which

it itself is attached) gradually calculates the least cost path to a destination, or set of destinations. We will study a decentralized routing algorithm known as a distance vector algorithm in section 4.2.2. It is called a distance vector algorithm because a node never actually knows a complete path from source to destination. Instead, it only knows the direction (which neighbor) to which it should forward a packet in order to reach a given destination along the least cost path, and the cost of that path from itself to the destination.

A second broad way to classify routing algorithms is according to whether they are static or dynamic. In static routing algorithms, routes change very slowly over time, often as a result of human intervention (e.g., a human manually editing a router's forwarding table). Dynamic routing algorithms change the routing paths as the network traffic loads (and the resulting delays experienced by traffic) or topology change. A dynamic algorithm can be run either periodically or in direct response to topology or link cost changes. While dynamic algorithms are more responsive to network changes, they are also more susceptible to problems such as routing loops and oscillation in routes.

Only two types of routing algorithms are typically used in the Internet: a dynamic global link state algorithm, and a dynamic decentralized distance vector algorithm.

A Link State Routing Algorithm

Recall that in a link state algorithm, the network topology and all link costs are known, i.e., available as input to the link state algorithm. In practice this is accomplished by having each node broadcast the identities and costs of its attached links to all other routers in the network. This link state broadcast, can be accomplished without the nodes having to initially know the identities of all other nodes in the

network A node need only know the identities and costs to its directly-attached neighbors; it will then learn about the topology of the rest of the network by receiving link state broadcast from other nodes. (In Chapter 5, we will learn how a router learns the identities of its directly attached neighbors). The result of the nodes' link state broadcast is that all nodes have an identical and complete view of the network. Each node can then run the link state algorithm and compute the same set of least cost paths as every other node.

The link state algorithm we present below is known as **Dijkstra's algorithm**, named after its inventor. It computes the least cost path from one node (the source, which we will refer to as A) to all other nodes in the network. Dijkstra's algorithm is iterative and has the property that after the k th iteration of the algorithm, the least cost paths are known to k destination nodes, and among the least cost paths to all destination nodes, these k path will have the k smallest costs. Let us define the following notation:

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

The link state algorithm consists of an initialization step followed by a loop. The number of times the loop is executed is equal to the number of nodes in the network. Upon termination, the algorithm will have calculated the shortest paths from the

source node to every other node in the network.

Link State (LS) Algorithm:

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

As an example, let us consider the network in Figure-3.6 and compute the shortest path from A to all possible destinations. A tabular summary of the algorithm's computation is shown in Table-3.2, where each line in the table gives the values of the algorithm's variables at the end of the iteration. Let us consider the few first steps in detail:

- In the initialization step, the currently known least path costs from A to its directly attached neighbors, B, C and D are initialized to 2, 5 and 1 respectively. Note in particular that the cost to C is set to 5 (even though we will soon see that a lesser cost path does indeed exist) since this is cost of the

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

Table 3.2: Steps in running the link state algorithm on network in Figure-3.6

direct (one hop) link from A to C. The costs to E and F are set to infinity since they are not directly connected to A.

- In the first iteration, we look among those nodes not yet added to the set N and find that node with the least cost as of the end of the previous iteration. That node is D, with a cost of 1, and thus D is added to the set N. Line 12 of the LS algorithm is then performed to update $D(v)$ for all nodes v , yielding the results shown in the second line (step 1) in Table 4.2-1. The cost of the path to B is unchanged. The cost of the path to C (which was 5 at the end of the initialization) through node D is found to have a cost of 4. Hence this lower cost path is selected and C's predecessor along the shortest path from A is set to D. Similarly, the cost to E (through D) is computed to be 2, and the table is updated accordingly.
- In the second iteration, nodes B and E are found to have the shortest path costs (2), and we break the tie arbitrarily and add E to the set N so that N now contains A, D, and E. The cost to the remaining nodes not yet in N, i.e., nodes B, C and F, are updated via line 12 of the LS algorithm, yielding the results shown in the third row in the above table.
- and so on ...

When the LS algorithm terminates, we have for each node, its predecessor along the least cost path from the source node. For each predecessor, we also have its predecessor and so in this manner we can construct the entire path from the source to all destinations.

3.2 Multicast IP

Traditional IP communication allows a host to send packets to a single host (unicast transmission) or to all hosts (broadcast transmission). IP multicast provides a third possibility: allowing a host to send packets to a subset of all hosts as a group transmission. This overview provides a brief, summary overview of IP Multicast. First, general topics such as multicast group concept, IP multicast addresses, and Layer 2 multicast addresses are discussed. Then intradomain multicast protocols are reviewed, such as Internet Group Management Protocol (IGMP), Cisco Group Management Protocol (CGMP), Protocol Independent Multicast (PIM) and Pragmatic General Multicast (PGM). Finally, interdomain protocols are covered, such as Multiprotocol Border Gateway Protocol (MBGP), Multicast Source Directory Protocol (MSDP), and Source Specific Multicast (SSM).

3.2.1 IP Multicast Basics

IP multicast is a bandwidth-conserving technology that reduces traffic by simultaneously delivering a single stream of information to potentially thousands of corporate recipients and homes. Applications that take advantage of multicast include video conferencing, corporate communications, distance learning, and distribution of software, stock quotes, and news.

IP multicast delivers application source traffic to multiple receivers without burdening the source or the receivers while using a minimum of network bandwidth. Multicast packets are replicated in the network at the point where paths diverge by Cisco routers enabled with Protocol Independent Multicast (PIM) and other supporting multicast protocols, resulting in the most efficient delivery of data to multiple receivers.

Many alternatives to IP multicast require the source to send more than one copy of the data. Some, such as application-level multicast, require the source to send an individual copy to each receiver. Even low-bandwidth applications can benefit from using Cisco IP multicast when there are thousands of receivers. High-bandwidth applications, such as MPEG video, may require a large portion of the available network bandwidth for a single stream. In these applications, IP multicast is the only way to send to more than one receiver simultaneously. Figure-3.7 shows how IP multicast is used to deliver data from one source to many interested recipients.

Many alternatives to IP multicast require the source to send more than one copy of the data. Some, such as application-level multicast, require the source to send an individual copy to each receiver. Even low-bandwidth applications can benefit from using Cisco IP multicast when there are thousands of receivers. High-bandwidth applications, such as MPEG video, may require a large portion of the available network bandwidth for a single stream. In these applications, IP multicast is the only way to send to more than one receiver simultaneously. Figure-3.7 shows how IP multicast is used to deliver data from one source to many interested recipients.

In the example shown in Figure-3.7, the receivers (the designated multicast group) are interested in receiving the video data stream from the source. The

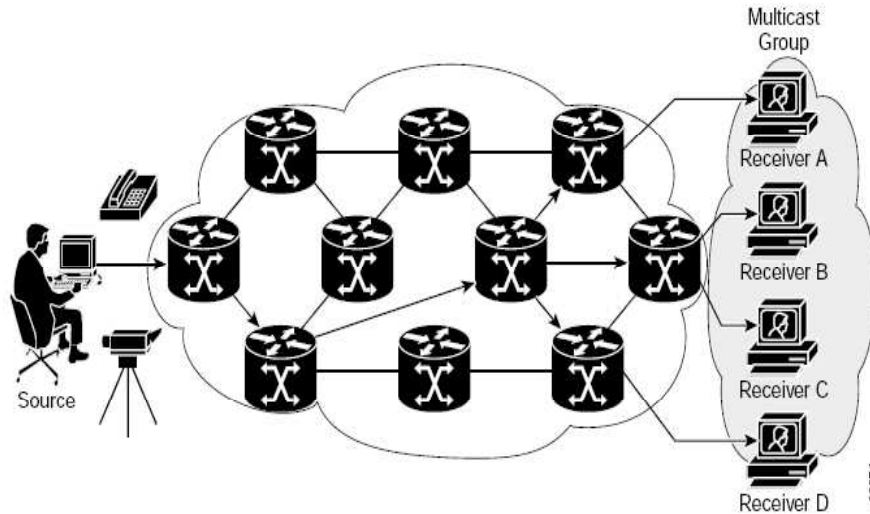


Figure 3.7: Multicast Transmission to Many Receivers

receivers indicate their interest by sending an Internet Group Management Protocol (IGMP) host report to the routers in the network. The routers are then responsible for delivering the data from the source to the receivers. The routers use Protocol Independent Multicast (PIM) to dynamically create a multicast distribution tree. The video data stream will then be delivered only to the network segments that are in the path between the source and the receivers.

3.3 Mobile IP

Mobile IP is an open standard, defined by the Internet Engineering Task Force (IETF) RFC 2002, that allows users to keep the same IP address, stay connected, and maintain ongoing applications while roaming between IP networks. Mobile IP is scalable for the Internet because it is based on IPany media that can support IP can support Mobile IP.

The number of wireless devices for voice or data is projected to surpass the number of fixed devices. Mobile data communication will likely emerge as the technology supporting most communication including voice and video. Mobile data communication will be pervasive in cellular systems such as 3G and in wireless LAN such as 802.11, and will extend into satellite communication. Though mobility may be enabled by link-layer technologies, data crossing networks or different link layers is still a problem. The solution to this problem is a standards-based protocol, Mobile IP.

3.3.1 Components of a Mobile IP Network

Mobile IP has the following three components, as shown in Figure-3.8:

- Mobile Node
- Home Agent
- Foreign Agent

The Mobile Node is a device such as a cell phone, personal digital assistant, or laptop whose software enables network roaming capabilities.

The Home Agent is a router on the home network serving as the anchor point for communication with the Mobile Node; it tunnels packets from a device on the Internet, called a Correspondent Node, to the roaming Mobile Node. (A tunnel is established between the Home Agent and a reachable point for the Mobile Node in the foreign network.)

The Foreign Agent is a router that may function as the point of attachment for the Mobile Node when it roams to a foreign network, delivering packets from the Home Agent to the Mobile Node.

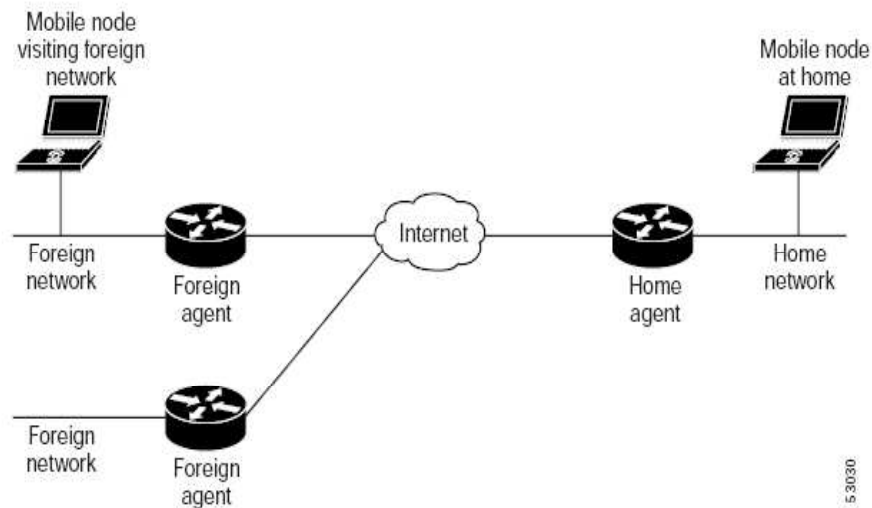


Figure 3.8: Mobile IP Components and Relationships

3.4 Internet Protocol version 6 (IPv6)

In the early 1990's the Internet Engineering Task force began an effort to develop a successor to the IPv4 protocol. A prime motivation for this effort was the realization that the 32-bit IP address space was beginning to be used up, with new networks and IP nodes being attached to the Internet (and being allocated unique IP addresses) at a breathtaking rate. To respond to this need of a large IP address space, a new IP protocol, IPv6, was developed. The designers of IPv6 also took this opportunity to tweak and augment other aspects of IPv4, based on the accumulated operational experience with IPv4. The point in time when IPv4 addresses would have been completely allocated (and hence no new networks could have attached to the Internet) was the subject of considerable debate. Based on current trends in address allocation, the estimates of the two leaders of the IETF's Address Lifetime Expectations working group were that addresses would become exhausted in 2008 and 2018 respectively. In 1996, the American Registry for Internet Number

(ARIN) reported that all of the IPv4 class A addresses have been assigned, 62 % of the class B addresses have been assigned, and 37 % of the class C addresses have been assigned. While these estimates and numbers suggested that a considerable amount of time might be left until the IPv4 address space became exhausted, it was realized that considerable time would be needed to deploy a new technology on such an extensive scale, and so the “Next Generation IP” (IPng) effort was begun.

3.4.1 IPv6 Packet Format

The format of the IPv6 packet is shown in Figure-3.9. The most important changes introduced in IPv6 are evident in the packet format:

- Expanded addressing capabilities. IPv6 increases the size of the IP address from 32 to 128 bits. This insures that the world won’t run out of IP addresses. Now, every grain of sand on the planet can be IP-addressable. In addition, the address space contains new hierarchical structure, allocating portions of the enlarged address space to geographical regions. In addition to unicast and multicast addresses, a new type of address, called an anycast address, has also been introduced, which allows a packet addressed to an anycase address to be delivered to any one of a group of hosts. This feature could be used, for example, to send an HTTP GET to the nearest of a number of mirror sites that contain a given document).
- A streamlined 40 byte header. As discussed below, a number of IPv4 fields have been dropped or made optional. The resulting 40-byte fixed-length header allows for faster processing of the IP packet. A new encoding of options allows for more flexible options processing.
- Flow labeling and priority. IPv6 has an elusive definition of a “flow.” [RFC

1752] and [RFC2460] state this allows “labeling of packets belonging to particular flows for which the sender requests special handling, such as a non-default quality of service or real-time service.” For example, audio and video transmission might likely be treated as a flow. On the other hand, the more traditional applications, such as file transfer and email might not be treated as flows. It is possible that the traffic carried by a high-priority user (e.g., someone paying for better service for their traffic) might also be treated as a flow. What is clear, however, is that the designers of IPv6 foresee the eventual need to be able to differentiate among the “flows,” even if the exact meaning of a flow has not yet been determined. The IPv6 header also has a 4-bit priority field. This field, as the TOS field in IPv4, can be used to give priority to certain packets within a flow, or it can be used to give priority to datagrams from certain applications (e.g., ICMP packets) over packets from other applications (e.g., network news).

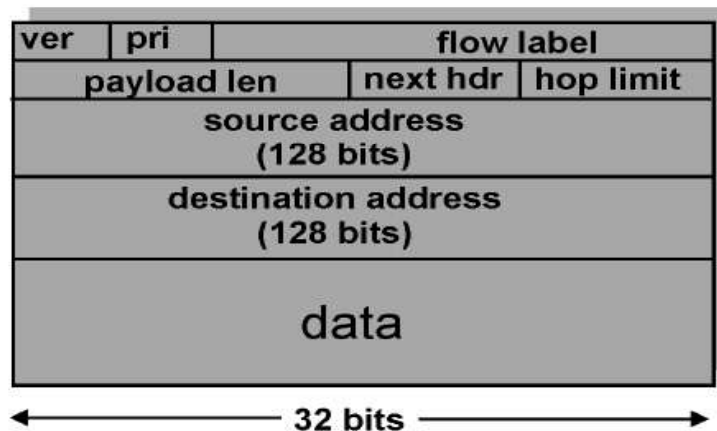


Figure 3.9: IP fragmentation

The IPv6 packet format is shown in Figure-3.9. As noted above, a comparison of Figure-3.9 with Figure-3.4 reveals the simpler, more streamlined structure of the

IPv6 packet. The following packet fields are defined in IPv6:

- **Version** This four bit field identifies the IP version number. Not surprisingly, IPv6 carries a value of “6” in this field. Note that putting a “4” in this field does not create a valid IPv4 packet (if it did, life would be a lot simpler – see the discussion below regarding the transition from IPv4 to IPv6).
- **Priority** This four bit field is similar in spirit to the ToS field we saw in IP version 4. [RFC 2460] states that values 0 through 7 are to be used for priority among traffic that is congestion-controlled (i.e., for which the source will back off on detection of congestion), while values 8 through 15 are used for non-congestion controlled traffic, such as constant bit rate real-time traffic.
- **Flow label** As discussed above, this field is used to identify a “flow” of packets.
- **Payload length** This 16-bit value is treated as an unsigned integer given the number of bytes in the IPv6 packet following the fixed length, 40 byte packet header.
- **Next header** This field identifies the protocol to which the contents (data field) of this packet will be delivered (e.g., to TCP or UDP). The field uses the same values as the Protocol field in the IPv4 header.
- **Hop limit** The contents of this field are decremented by one by each router that forward the packet. If the hop limit count reaches zero, the packet is discarded.
- **Source and Destination address** An IP v6 address has the following structure:

The discussion above identified the purpose of the fields that are included in the IPv6 packet. Comparing the IPv6 packet format in Figure-3.9 with the IPv4 packet



Figure 3.10: IPv6 address format

format that we saw earlier in Figure-3.4, we notice that several fields appearing in the IPv4 packet are no longer present in the IPv6 packet:

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

or UDP protocol headers can be the next header within an IP packet, so too can an options field. The removal of the options field results in a fixed length, 40 byte IP header.

3.5 Important Applications of Internet and TCP/IP

3.5.1 File Transfer Protocol (FTP)

FTP (File Transfer Protocol) is a protocol for transferring a file from one host to another host. The protocol dates back to 1971 (when the Internet was still an experiment), but remains enormously popular. FTP is described in [RFC 959]. Figure-3.11 provides an overview of the services provided by FTP.

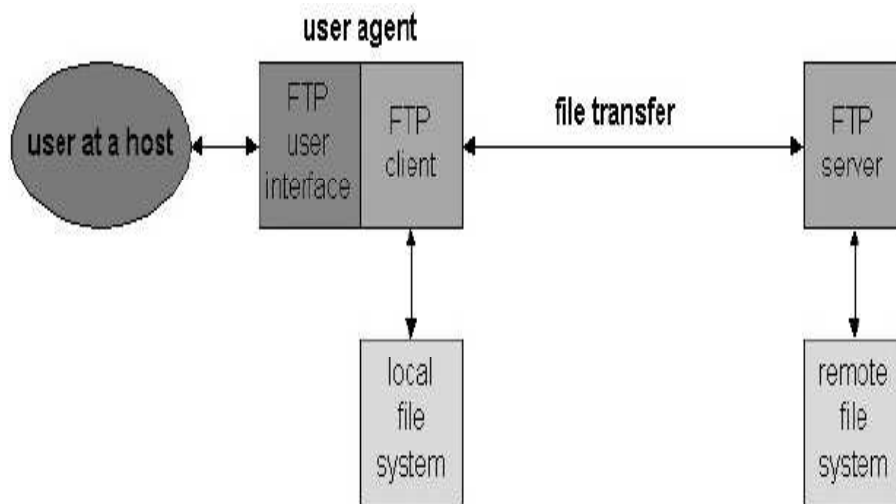


Figure 3.11: FTP moves files between local and remote file systems

In a typical FTP session, the user is sitting in front of one host (the local host) and wants to transfer files to or from a remote host. In order for the user to access

the remote account, the user must provide a user identification and a password. After providing this authorization information, the user can transfer files from the local file system to the remote file system and vice versa. As shown in Figure-3.11, the user interacts with FTP through an FTP user agent. The user first provides the host name of the remote host, which causes the FTP client process in the local host to establish a TCP connection with the FTP server process in the remote host. The user then provides the user identification and password, which get sent over the TCP connection as part of FTP commands. Once the server has authorized the user, the user copies one or more files stored in the local file system into the remote file system (or vice versa).

HTTP and FTP are both file transfer protocols and have many common characteristics; for example, they both run on top of TCP, the Internet's connection-oriented, transport-layer, reliable data transfer protocol. However, the two application-layer protocols have some important differences. The most striking difference is that FTP uses two parallel TCP connections to transfer a file, a control connection and a data connection. The control connection is used for sending control information between the two hosts – information such as user identification, password, commands to change remote directory, and commands to “put” and “get” files. The data connection is used to actually send a file. Because FTP uses a separate control connection, FTP is said to send its control information out-of-band. In Chapter 6 we shall see that the RTSP protocol, which is used for controlling the transfer of continuous media such as audio and video, also sends its control information out-of-band. HTTP, as you recall, sends request and response header lines into the same TCP connection that carries the transferred file itself. For this reason, HTTP is said to send its control information in-band. In the next section we shall see that SMTP, the main protocol for electronic mail, also sends control information in-band.

The FTP control and data connections are illustrated in Figure-3.11. When a user starts an FTP session with a remote host, FTP first sets up a control TCP connection on server port number 21. The client side of FTP sends the user identification and password over this control connection. The client side of FTP also sends, over the control connection, commands to change the remote directory. When the user requests a file transfer (either to, or from, the remote host), FTP opens a TCP data connection on server port number 20. FTP sends exactly one file over the data connection and then closes the data connection. If, during the same session, the user wants to transfer another file, FTP opens another data TCP connection. Thus, with FTP, the control connection remains open throughout the duration of the user session, but a new data connection is created for each file transferred within a session (i.e., the data connections are non-persistent).

Throughout a session, the FTP server must maintain state about the user. In particular, the server must associate the control connection with a specific user account, and the server must keep track of the user's current directory as the user wanders about the remote directory tree. Keeping track of this state information for each ongoing user session significantly impedes the total number of sessions that FTP can maintain simultaneously. HTTP, on the other hand, is stateless – it does not have to keep track of any user state.

FTP Commands and Replies

We end this section with a brief discussion of some of the more common FTP commands. The commands, from client to server, and replies, from server to client, are sent across the control TCP connection in 7-bit ASCII format. Thus, like HTTP commands, FTP commands are readable by people. In order to delineate successive

commands, a carriage return and line feed end each command (and reply). Each command consists of four uppercase ASCII characters, some with optional arguments. Some of the more common commands are given below (with options in italics):

- **USER** *username* : Used to send the user identification to server.
- **PASS** *password* : Used to send the user password to the server.
- **LIST** : Used to ask the server to send back a list of all the files in the current remote directory. The list of files is sent over a (new and non-persistent) data TCP connection and not over the control TCP connection.
- **RETR** *filename* : Used to retrieve (i.e., get) a file from the current directory of the remote host.
- **STOR** *filename* : Used to store (i.e., put) a file into the current directory of the remote host.

There is typically a one-to-one correspondence between the command that the user issues and the FTP command sent across the control connection. Each command is followed by a reply, sent from server to client. The replies are three-digit numbers, with an optional message following the number. This is similar in structure to the status code and phrase in the status line of the HTTP response message; the inventors of HTTP intentionally included this similarity in the HTTP response messages. Some typical replies, along with their possible messages, are as follows:

- 331 Username OK, password required
- 125 Data connection already open; transfer starting
- 425 Can't open data connection

- 452 Error writing file

Readers who are interested in learning about the other FTP commands and replies are encouraged to read [RFC959]

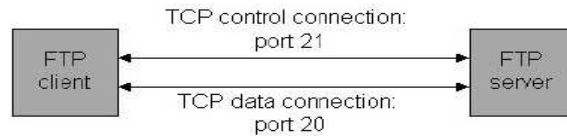


Figure 3.12: Control and data connections

3.5.2 Simple Mail Transfer Protocol (SMTP)

Along with the Web, electronic mail is one of the most popular Internet applications. Just like ordinary “snail mail,” email is asynchronous – people send and read messages when it is convenient for them, without having to coordinate with other peoples’ schedules. In contrast with snail mail, electronic mail is fast, easy to distribute, and inexpensive. Moreover, modern electronic mail messages can include hyperlinks, HTML formatted text, images, sound and even video. In this section we will examine the application-layer protocols that are at the heart of Internet electronic mail. But before we jump into an in-depth discussion of these protocols, let’s take a bird’s eye view of the Internet mail system and its key components.

Figure-3.13 presents a high-level view of the Internet mail system. We see from this diagram that it has three major components: user agents, mail servers, and the Simple Mail Transfer Protocol (SMTP). We now describe each of these components in the context of a sender, Alice, sending an email message to a recipient, Bob. User agents allow users to read, reply to, forward, save, and compose messages. (User agents for electronic mail are sometimes called mail readers, although we will generally avoid this term in this book.) When Alice is finished composing her

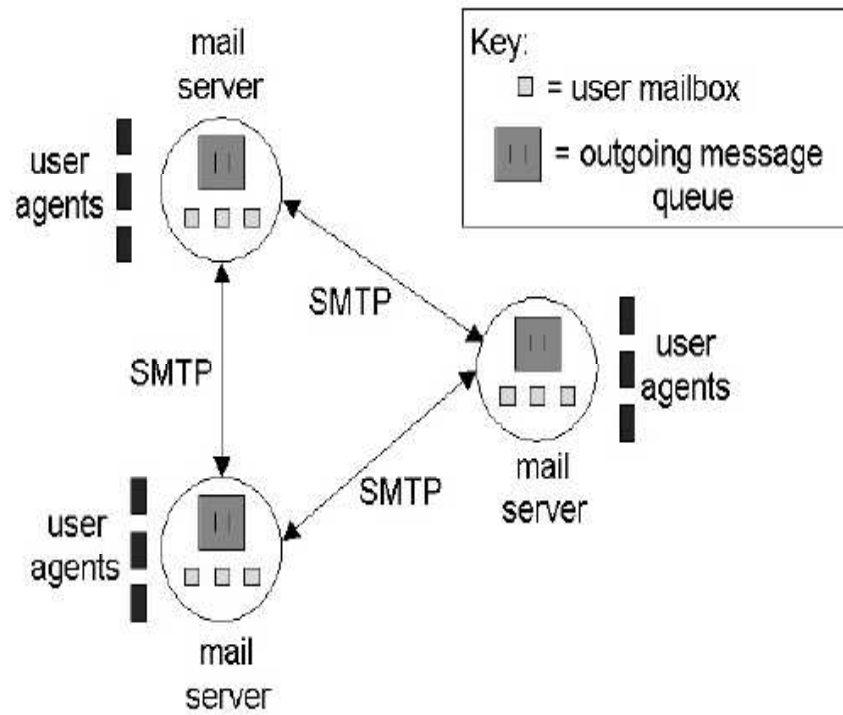


Figure 3.13: A bird's eye view of the Internet e-mail system.

message, her user agent sends the message to her mail server, where the message is placed in the mail server's outgoing message queue. When Bob wants to read a message, his user agent obtains the message from his mailbox in his mail server. In the late 1990s, GUI (graphical user interface) user agents became popular, allowing users to view and compose multimedia messages. Currently, Eudora, Microsoft's Outlook Express, and Netscape's Messenger are among the popular GUI user agents for email. There are also many text-based email user interfaces in the public domain, including mail, pine and elm.



Mail servers form the core of the e-mail infrastructure. Each recipient, such as Bob, has a mailbox located in one of the mail servers. Bob's mailbox manages and maintains the messages that have been sent to him. A typical message starts its journey in the sender's user agent, travels to the sender's mail server, and then travels to the recipient's mail server, where it is deposited in the recipient's mailbox. When Bob wants to access the messages in his mailbox, the mail server containing the mailbox authenticates Bob (with user names and passwords). Alice's mail server must also deal with failures in Bob's mail server. If Alice's server cannot deliver mail to Bob's server, Alice's server holds the message in a message queue and attempts to transfer the message later. Reattempts are often done every 30 minutes or so; if there is no success after several days, the server removes the message and notifies the sender (Alice) with an email message.

The Simple Mail Transfer Protocol (SMTP) is the principle application-layer protocol for Internet electronic mail. It uses the reliable data transfer service of TCP to transfer mail from the sender's mail server to the recipient's mail server. As

with most application-layer protocols, SMTP has two sides: a client side which executes on the sender's mail server, and server side which executes on the recipient's mail server. Both the client and server sides of SMTP run on every mail server. When a mail server sends mail (to other mail servers), it acts as an SMTP client. When a mail server receives mail (from other mail servers) it acts as an SMTP server.

SMTP, defined in [RFC 821], is at the heart of Internet electronic mail. As mentioned above, SMTP transfers messages from senders' mail servers to the recipients' mail servers. SMTP is much older than HTTP. (The SMTP RFC dates back to 1982, and SMTP was around long before that.) Although SMTP has numerous wonderful qualities, as evidenced by its ubiquity in the Internet, it is nevertheless a legacy technology that possesses certain "archaic" characteristics. For example, it restricts the body (not just the headers) of all mail messages to be in simple seven-bit ASCII. This restriction was not bothersome in the early 1980s when transmission capacity was scarce and no one was emailing large attachments or large image, audio or video files. But today, in the multimedia era, the seven-bit ASCII restriction is a bit of a pain – it requires binary multimedia data to be encoded to ASCII before being sent over SMTP; and it requires the corresponding ASCII message to be decoded back to binary after SMTP transport. Recall from Section 2.3 that HTTP does not require multimedia data to be ASCII encoded before transfer.

To illustrate the basic operation of SMTP, let's walk through a common scenario. Suppose Alice wants to send Bob a simple ASCII message:

- Alice invokes her user agent for email, provides Bob's email address (e.g., bob@someschool.edu), composes a message and instructs the user agent to send the message.
- Alice's user agent sends the message her mail server, where it is placed in a

message queue.

- The client side of SMTP, running on Alice's mail server, sees the message in the message queue. It opens a TCP connection to a SMTP server, running on Bob's mail server.
- After some initial SMTP handshaking, the SMTP client sends Alice's message into the TCP connection.
- At Bob's mail server host, the server side of SMTP receives the message. Bob's mail server then places the message in Bob's mailbox.
- Bob invokes his user agent to read the message at his convenience.

The scenario is summarized in the Figure-3.14.

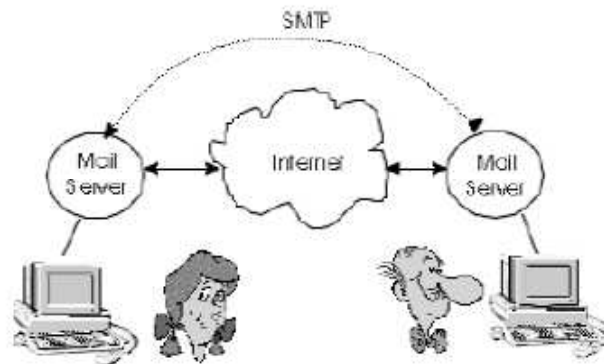


Figure 3.14: Alice's mail server transfers Alice's message to Bob's mail server.

It is important to observe that SMTP does not use intermediate mail servers for sending mail, even when the two mail servers are located at opposite ends of the world. If Alice's server is in Hong Kong and Bob's server is in Mobile, Alabama, the TCP "connection" is a direct connection between the Hong Kong and Mobile servers. In particular, if Bob's mail server is down, the message remains in Alice's

mail server and waits for a new attempt – the message does not get placed in some intermediate mail server.

Let's now take a closer look at how SMTP transfers a message from a sending mail server to a receiving mail server. We will see that the SMTP protocol has many similarities with protocols that are used for face-to-face human interaction. First, the client SMTP (running on the sending mail server host) has TCP establish a connection on port 25 to the server SMTP (running on the receiving mail server host). If the server is down, the client tries again later. Once this connection is established, the server and client perform some application-layer handshaking. Just as humans often introduce themselves before transferring information from one to another, SMTP clients and servers introduce themselves before transferring information. During this SMTP handshaking phase, the SMTP client indicates the email address of the sender (the person who generated the message) and the email address of the recipient. Once the SMTP client and server have introduced themselves to each other, the client sends the message. SMTP can count on the reliable data transfer service of TCP to get the message to the server without errors. The client then repeats this process over the same TCP connection if it has other messages to send to the server; otherwise, it instructs TCP to close the connection.

Let us take a look at an example transcript between client (C) and server (S). The host name of the client is `crepes.fr` and the host name of the server is `hamburger.edu`. The ASCII text prefaced with C: are exactly the lines the client sends into its TCP socket; and the ASCII text prefaced with S: are exactly the lines the server sends into its TCP socket. The following transcript begins as soon as the TCP connection is established:

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with ‘.’ on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

In the above example, the client sends a message (“Do you like ketchup? How about pickles?”) from mail server `crepes.fr` to mail server `hamburger.edu`. The client issued five commands: `HELO` (an abbreviation for `HELLO`), `MAIL FROM`, `RCPT TO`, `DATA`, and `QUIT`. These commands are self explanatory. The server issues replies to each command, with each reply having a reply code and some (optional) English-language explanation. We mention here that SMTP uses persistent connections: if the sending mail server has several messages to send to the same receiving mail server, it can send all of the messages over the same TCP connection. For each message, the client begins the process with a new `HELO crepes.fr` and only issues `QUIT` after all messages have been sent.

It is highly recommended that you use Telnet to carry out a direct dialogue with an SMTP server. To do this, issue `telnet serverName 25 .` When you do this, you are simply establishing a TCP connection between your local host and the mail server. After typing this line, you should immediately receive the 220 reply from the server. Then issue the SMTP commands `HELO`, `MAIL FROM`, `RCPT TO`, `DATA`, and `QUIT` at the appropriate times. If you Telnet into your friend's SMTP server, you should be able to send mail to your friend in this manner (i.e., without using your mail user agent).

3.5.3 Trivial File Transfer Protocol (TFTP)

Transfer data as blocks of 512 bytes. TFTP sends one block of 512 bytes and waits for an acknowledgment. TFTP retries after a timeout until it succeeds and then proceeds to the next block. TFTP numbers the blocks sequentially from 1. This robust protocol operates even when the transport layer is of low quality. However, it is not efficient and is useful only when a lightweight protocol is needed, for instance as a simple boot ROM application for loading kernels over the networks.

3.5.4 Hypertext Transfer Protocol (HTTP)

In the 1980s the Internet was used by researchers, academics and university students to login to remote hosts, to transfer files from local hosts to remote hosts and vice versa, to receive and send news, and to receive and send electronic mail. Although these applications were (and continue to be) extremely useful, the Internet was essentially unknown outside the academic and research communities. Then in early 1990s the Internet's killer application arrived on the scene – the World Wide Web. The Web is the Internet application that caught the general public's eye. It is dramatically changing how people interact inside and outside their work environments. It has spawned thousands of start up companies. It has elevated the Internet from

just one of many data networks (including online networks such as Prodigy, America On Line and CompuServe, national data networks such as Minitel/Transpac in France, and private X.25 and frame relay networks) to essentially the one and only data network.

History is sprinkled with the arrival of electronic communication technologies that have had major societal impacts. The first such technology was the telephone, invented in the 1870s. The telephone allowed two persons to orally communicate in real-time without being in the same physical location. It had a major impact on society – both good and bad. The next electronic communication technology was broadcast radio/television, which arrived in the 1920s and 1930s. Broadcast radio/television allowed people to receive vast quantities of audio and video information. It also had a major impact on society – both good and bad. The third major communication technology that has changed the way people live and work is the Web. Perhaps what appeals the most to users about the Web is that it is on demand. Users receive what they want, when they want it. This is unlike broadcast radio and television, which force users to “tune in” when the content provider makes the content available. In addition to being on demand, the Web has many other wonderful features that people love and cherish. It is enormously easy for any individual to make any available available over the Web; everyone can become a publisher at extremely low cost. Hyperlinks and search engines help us navigate through an ocean of Web sites. Graphics and animated graphics stimulate our senses. Forms, Java applets, Active X components, as well as many other devices enable us to interact with pages and sites. And more and more, the Web provides a menu interface to vast quantities of audio and video material stored in the Internet, audio and video that can be accessed on demand.

Overview of HTTP

The Hypertext Transfer Protocol (HTTP), the Web's application-layer protocol, is at the heart of the Web. HTTP is implemented in two programs: a client program and server program. The client program and server programs, executing on different end systems, talk to each other by exchanging HTTP messages. HTTP defines the structure of these messages and how the client and server exchange the messages. Before explaining HTTP in detail, it is useful to review some Web terminology.

A Web page (also called a document) consists of objects. An object is a simply file – such as a HTML file, a JPEG image, a GIF image, a Java applet, an audio clip, etc. – that is addressable by a single URL. Most Web pages consist of a base HTML file and several referenced objects. For example, if a Web page contains HTML text and five JPEG images, then the Web page has six objects: the base HTML file plus the five images. The base HTML file references the other objects in the page with the objects' URLs. Each URL has two components: the host name of the server that houses the object and the object's path name. For example, the URL

`www.someSchool.edu/someDepartment/picture.gif`

has `www.someSchool.edu` for a host name and `/someDepartment/picture.gif` for a path name. A browser is a user agent for the Web; it displays to the user the requested Web page and provides numerous navigational and configuration features. Web browsers also implement the client side of HTTP. Thus, in the context of the Web, we will interchangeably use the words “browser” and “client”. Popular Web browsers include Netscape Communicator and Microsoft Explorer. A Web server houses Web objects, each addressable by a URL. Web servers also implement the server side of HTTP. Popular Web servers include Apache, Microsoft Internet Information Server, and the Netscape Enterprise Server. (Netcraft provides a nice survey

of Web server penetration.)

HTTP defines how Web clients (i.e., browsers) request Web pages from servers (i.e., Web servers) and how servers transfer Web pages to clients. We discuss the interaction between client and server in detail below, but the general idea is illustrated in Figure 2.2-1. When a user requests a Web page (e.g., clicks on a hyperlink), the browser sends HTTP request messages for the objects in the page to the server. The server receives the requests and responds with HTTP response messages that contain the objects. Through 1997 essentially all browsers and Web servers implement version HTTP/1.0, which is defined in [RFC 1945]. Beginning in 1998 Web servers and browsers began to implement version HTTP/1.1, which is defined in [RFC 2068]. HTTP/1.1 is backward compatible with HTTP/1.0; a Web server running 1.1 can “talk” with a browser running 1.0, and a browser running 1.1 can “talk” with a server running 1.0.

Both HTTP/1.0 and HTTP/1.1 use TCP as their underlying transport protocol (rather than running on top of UDP). The HTTP client first initiates a TCP connection with the server. Once the connection is established, the browser and the server processes access TCP through their socket interfaces. As described earlier, on the client side the socket interface is the “door” between the client process and the TCP connection; on the server side it is the “door” between the server process and the TCP connection. The client sends HTTP request messages into its socket interface and receives HTTP response messages from its socket interface. Similarly, the HTTP server receives request messages from its socket interface and sends response messages into the socket interface. Once the client sends a message into its socket interface, the message is “out of the client’s hands” and is “in the hands of TCP”. Recall that TCP provides a reliable data transfer service to HTTP. This implies that

each HTTP request message emitted by a client process eventually arrives in tact at the server; similarly, each HTTP response message emitted by the server process eventually arrives in tact at the client. Here we see one of the great advantages of a layered architecture - HTTP need not worry about lost data, or the details of how TCP recovers from loss or reordering of data within the network. That is the job of TCP and the protocols in the lower layers of the protocol stack. We only mention here that this mechanism forces each new TCP connection to initially transmit data at a relatively slow rate, but then allows each connection to ramp up to a relatively high rate when the network is uncongested. The initial slow-transmission phase is referred to as slow start. It is important to note that the server sends requested files to clients without storing any state information about the client. If a particular client asks for the same object twice in a period of a few seconds, the server does not respond by saying that it just served the object to the client; instead, the server resends the object, as it has completely forgotten what it did earlier. Because an HTTP server maintains no information about the clients, HTTP is said to be a stateless protocol.

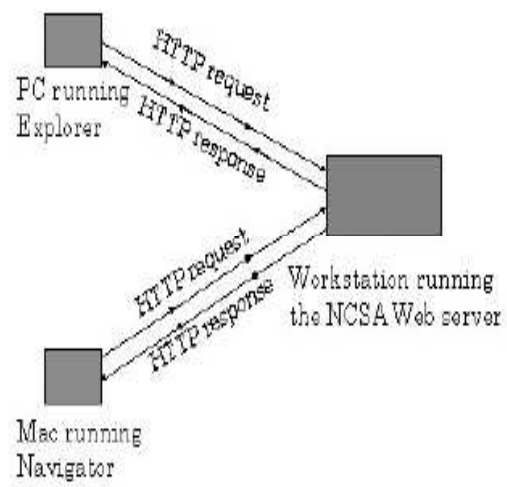


Figure 3.15: HTTP request-response behavior