COMPUTER NETWORKS

Module1: Application Layer

1. Principles of Network Applications

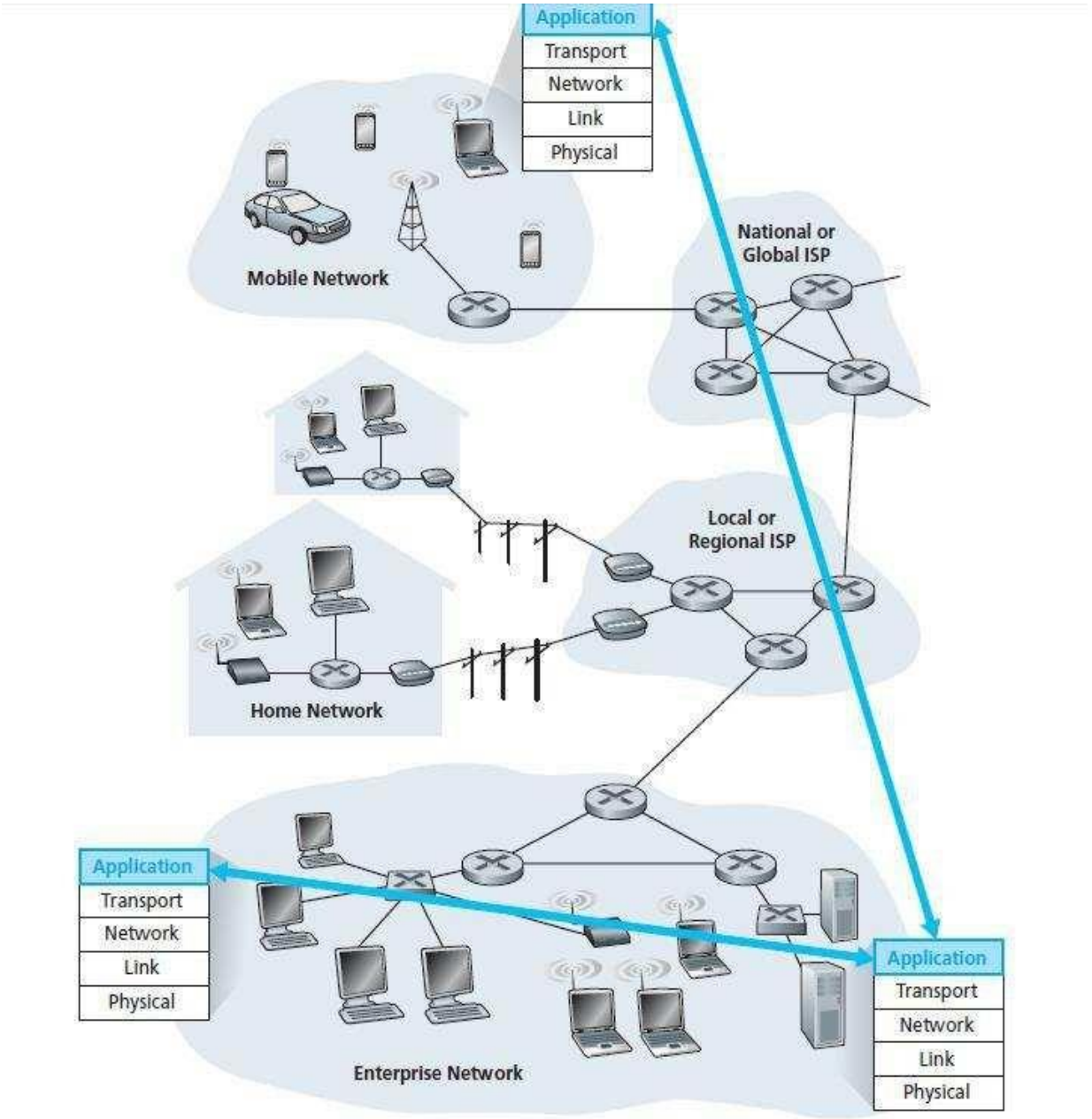

Figure 2.1 ♦ Communication for a network application takes place between end systems at the application layer

Network Application Architectures

The **application architecture** is designed by the application developer and dictates how the application is structured over the various end systems.
Two types:

i) the client-server architecture ,
ii) ii) the peer-to-peer (P2P) architecture

In a **client-server architecture**, there is an always-on host, called the *server*, which services requests from many other hosts, called *clients*. Example: Web browser and Web server.,(Web, FTP, Telnet, and e-mail.)
In a **P2P architecture**, there is minimal (or no) reliance on dedicated servers in data centers. Instead the application exploits direct communication between pairs of intermittently connected hosts, called *peers*. The peers are not owned by the service provider, but are instead desktops and laptops controlled by users, with most of the peers residing in homes, universities, and offices. Because the peers communicate without passing through a dedicated server, the architecture is called peer-to-peer. Example: file sharing (e.g., BitTorrent), peer-assisted download acceleration (e.g., Xunlei), Internet Telephony (e.g., Skype), and IPTV (e.g., Kankan and PPstream).

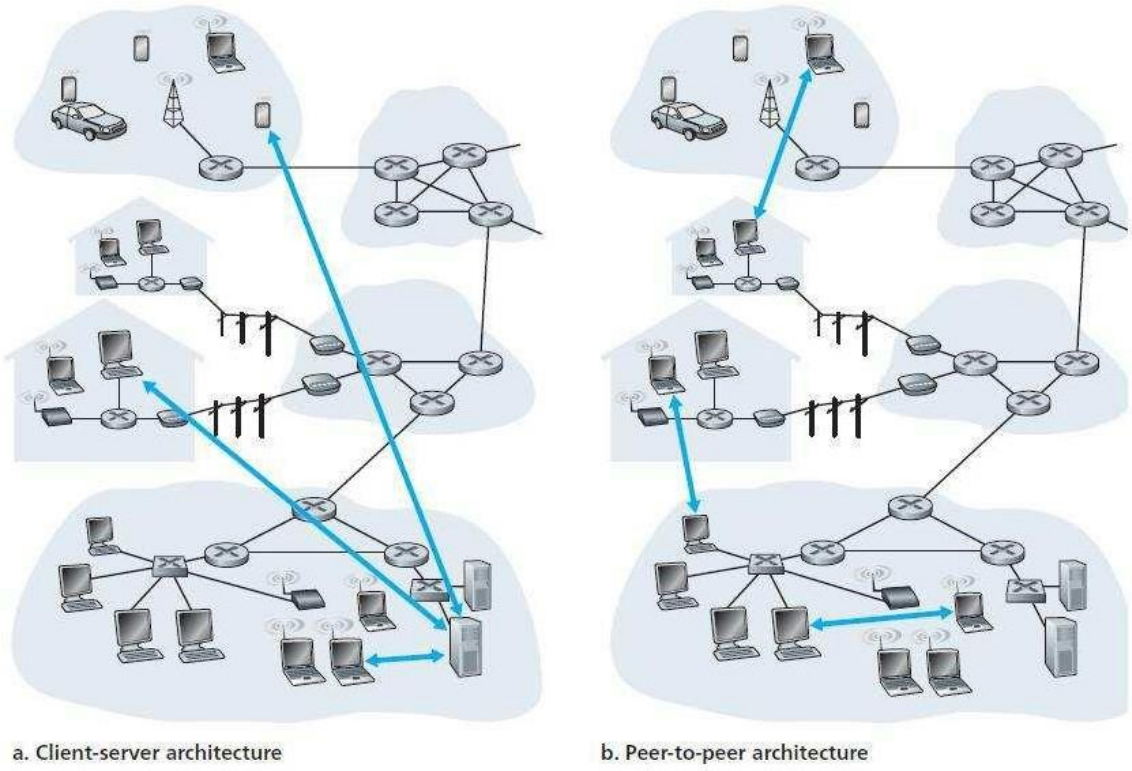

Figure 2.2 ♦ (a) Client-server architecture; (b) P2P architecture

Some applications have hybrid architectures, combining both client-server and P2P elements. For example, for many instant messaging applications, servers are used to track the IP addresses of users, but user-to-user messages are sent directly between user hosts (without passing through intermediate servers).

2. Processes Communicating

When processes are running on the same end system, they can communicate with each other with inter process communication, using rules that are governed by the end system's operating system. Processes on two different end systems communicate with each other by exchanging messages across the computer network. A sending process creates and sends messages into the network; a receiving process receives these messages and possibly responds by sending messages back.

Client and Server Processes:

A network application consists of pairs of processes that send messages to each other over a network. we typically label one of the two processes as the client and the other process as the server. With the Web, a browser is a client process and a Web server is a server process. With P2P file sharing, the peer that is downloading the file is labeled as the client, and the peer that is uploading the file is labeled as the server.

Definition: In the context of a communication session between a pair of processes, the process that initiates the communication (that is, initially contacts the other process at the beginning of the session) is labeled as the client. The process that waits to be contacted to begin the session is the server.

A process sends messages into, and receives messages from, the network through a software interface called a socket.

Illustration: A process is analogous to a house and its socket is analogous to its door. When a process wants to send a message to another process on another host, it shoves the message out its door (socket). This sending process assumes that there is a transportation infrastructure on the other side of its door that will transport the message to the door of the destination process. Once the message arrives at the destination host, the message passes through the receiving process's door (socket), and the receiving process then acts on the message.

As shown in this figure 2.3, a socket is the interface between the application layer and the transport layer within a host. It is also referred to as the Application Programming Interface (API) between the application and the network.

Addressing Processes:

In order for a process running on one host to send packets to a process running on another host, the receiving process needs to have an address. In the Internet, the host is identified by its IP address. To identify the receiving process, two pieces of information need to be specified:
(1) the address of the host (IP Address) and
(2) an identifier that specifies the receiving process in the destination host (Port No.).
For example, a Web server is identified by port number 80. A mail server process (using the SMTP protocol) is identified by port number 25. IP address is a 32-bit quantity that we can think of as uniquely identifying the host.

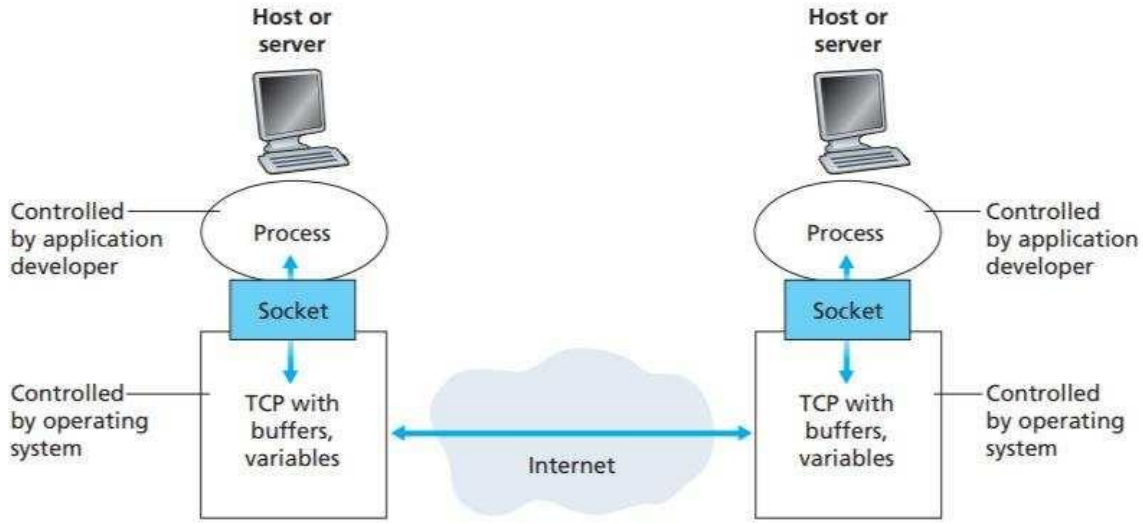

Figure 2.3 ♦ Application processes, sockets, and underlying transport protocol

3. Transport Services Available to Applications

Reliable Data Transfer : guaranteed data delivery service

Throughput: guaranteed throughput of *r* bits/sec, and the transport protocol would then ensure that the available throughput is always at least *r* bits/sec.

Timing: interactive real-time applications, such as Internet telephony, virtual environments, teleconferencing, and multiplayer games, all of which require tight timing constraints on data delivery in order to be effective.

Security: A transport protocol can also provide security services such as confidentiality, data integrity and end-point authentication.

4 Transport Services Provided by the Internet

When an application developer create a new network application for the Internet, one of the first decisions you have to make is whether to use UDP or TCP.

Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps–1Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Instant messaging	No loss	Elastic	Yes and no

Figure 2.4 ♦ Requirements of selected network applications

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube)	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

Figure 2.5 ♦ Popular Internet applications, their application-layer protocols, and their underlying transport protocols

5 Application-Layer Protocols

An **application-layer protocol** defines how an application's processes, running on different end systems, pass messages to each other. In particular, an application-layer protocol defines:

• The types of messages exchanged, for example, request messages and response messages
• The syntax of the various message types, such as the fields in the message and how the fields are delineated.
• The semantics of the fields, that is, the meaning of the information in the fields
• Rules for determining when and how a process sends messages and responds to messages

The Web and HTTP

HyperText Transfer Protocol (HTTP) is implemented in two programs: a client program and a server program. A **Web page** (also called a document) consists of objects. An **object** is simply a file—such as an HTML file, a JPEG image, a Java applet, or a video clip—that is addressable by a single URL. HTTP defines how Web clients request Web pages from Web servers and how servers transfer Web pages to clients. Figure 2.6. When a user requests a Web page (for example, 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.

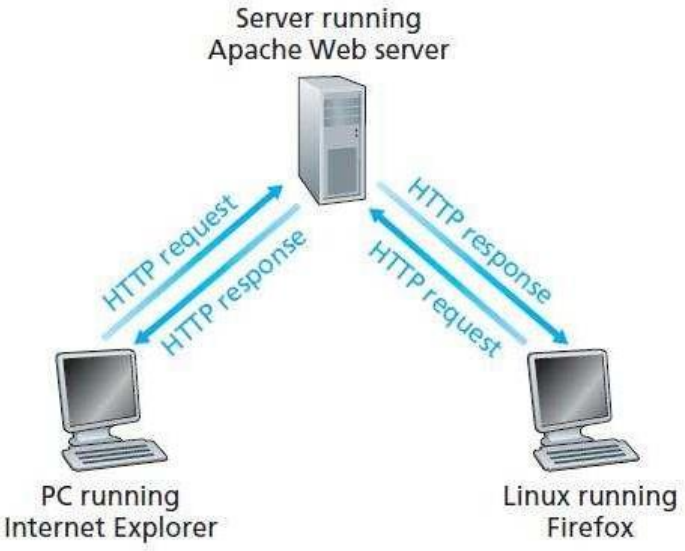

Figure 2.6 ♦ HTTP request-response behavior

HTTP uses TCP as its underlying transport protocol

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 its socket interface.

Non-Persistent and Persistent Connections

When this client-server interaction is taking place over TCP, the application developer needs to make an important decision—should each request/response pair be sent over a *separate* TCP connection(**non-persistent connections**), or should all of the requests and their corresponding responses be sent over the *same* TCP connection (**persistent connections**)?

HTTP with Non-Persistent Connections

What happens when a user clicks on a hyperlink. As shown in Figure 2.7, this causes the browser to initiate a TCP connection between the browser and the Web server; this involves a "three-way handshake"—the client sends a small TCP segment to the server, the server acknowledges and responds with a small TCP segment, and, finally, the client acknowledges back to the server. The first two parts of the three way handshake take one RTT. After completing the first two parts of the handshake, the client sends the HTTP request message combined with the third part of the three-way handshake (the acknowledgment) into the TCP connection. Once the request message arrives at the server, the server sends the HTML file into the

TCP connection. This HTTP request/response eats up another RTT. Thus, roughly, the total response time is two RTTs plus the transmission time at the server of the HTML file.

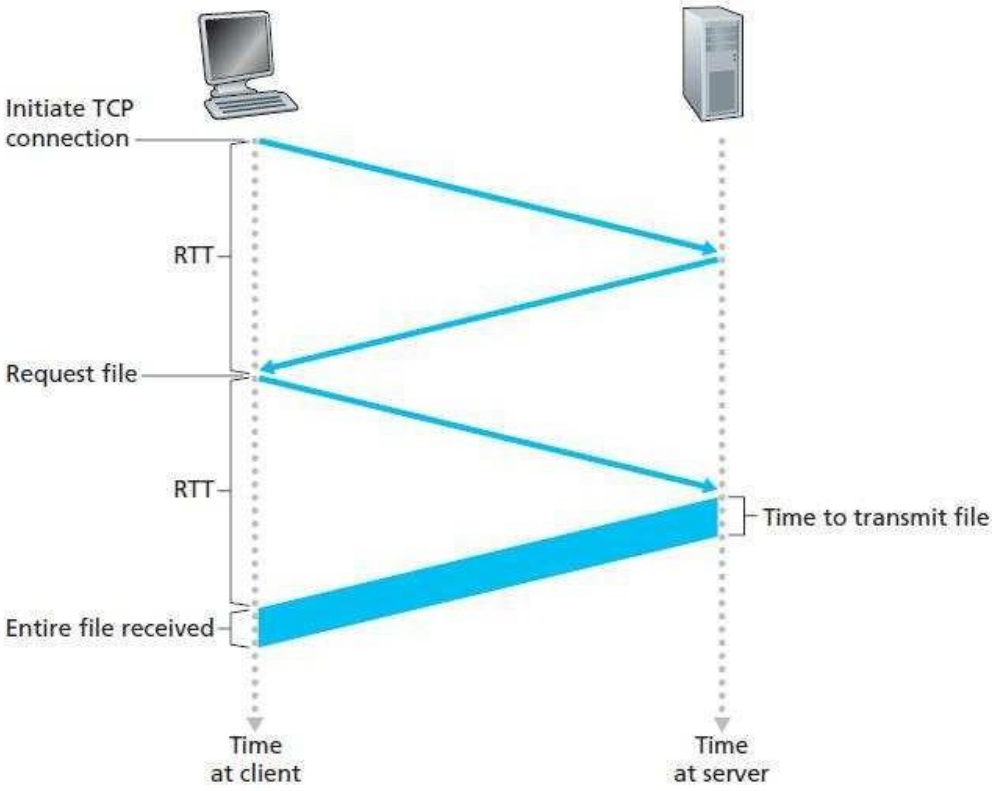

Figure 2.7 ♦ Back-of-the-envelope calculation for the time needed to request and receive an HTML file

HTTP with Persistent Connections

With persistent connections, the server leaves the TCP connection open after sending a response. Multiple Web pages residing on the same server can be sent from the server to the same client over a single persistent TCP connection. These requests for objects can be made back-to-back, without waiting for replies to pending requests (pipelining). Typically, the HTTP server closes a connection when it isn't used for a certain time. When the server receives the back-to-back requests, it sends the objects back-to-back.

HTTP Message Format

There are two types of HTTP messages: request messages and response messages.

HTTP Request Message

A typical HTTP request message:

GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
2.2 • THE WEB AND HTTP **103**
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr

The first line of an HTTP request message is called the **request line**; the subsequent lines are called the **header lines**. The request line has three fields: the method field, the URL field, and the HTTP version field. The method field can take on several different values, including GET, POST, HEAD, PUT, and DELETE. The great majority of HTTP request messages use the GET method.

The header line Host: www.someschool.edu specifies the host on which the object resides.
Connection: it wants the server to close the connection after sending the requested object.
The User-agent: header line specifies the user agent, that is, the browser type that is making the request to the server. Here the user agent is Mozilla/5.0, a Firefox browser.
Accept language: header indicates that the user prefers to receive a French version of the object, if such an object exists on the server; otherwise, the server should send its default version.

General format of a request message, as shown in Figure 2.8.

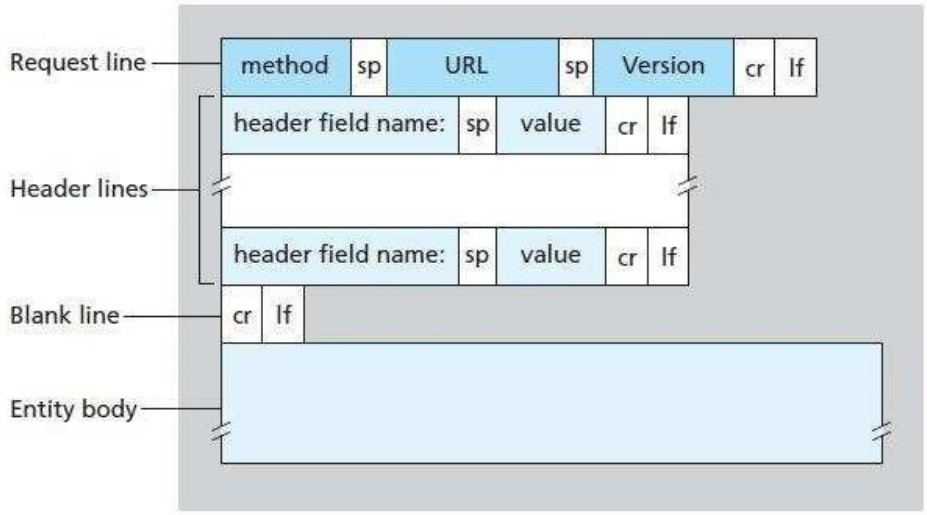

Figure 2.8 ♦ General format of an HTTP request message

HTTP Response Message

A typical HTTP response message:

HTTP/1.1 200 OK
Connection: close
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)

It has three sections: an initial **status line**, six **header lines**, and then the **entity body**.

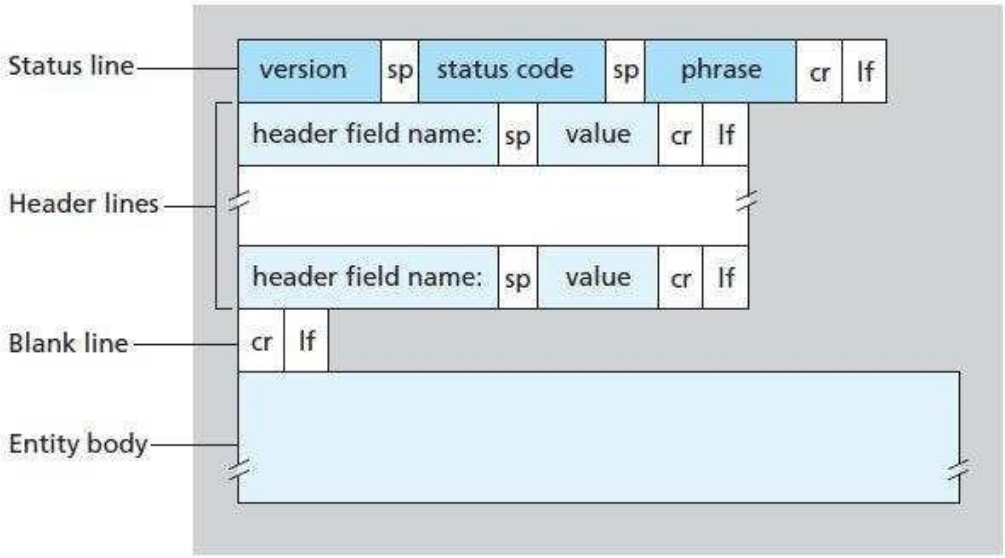

Figure 2.9 ♦ General format of an HTTP response message

The status line has three fields: the protocol version field, a status code, and a corresponding status message. The status code and associated phrase indicate the result of the request. Some common status codes and associated phrases include:

• 200 OK: Request succeeded and the information is returned in the response.
• 301 Moved Permanently: Requested object has been permanently moved; the new URL is specified in Location: header of the response message. The client software will automatically retrieve the new URL.
• 400 Bad Request: This is a generic error code indicating that the request could not be understood by the server.
• 404 Not Found: The requested document does not exist on this server.
• 505 HTTP Version Not Supported: The requested HTTP protocol version is not supported by the server.

User-Server Interaction: Cookies

Cookies allow sites to keep track of users. Most major commercial Web sites use cookies today. cookie technology has four components: (1) a cookie header line in the HTTP response message; (2) a cookie header line in the HTTP request message; (3) a cookie file kept on the user's end system and managed by the user's browser; and (4) a back-end database at the Web site.

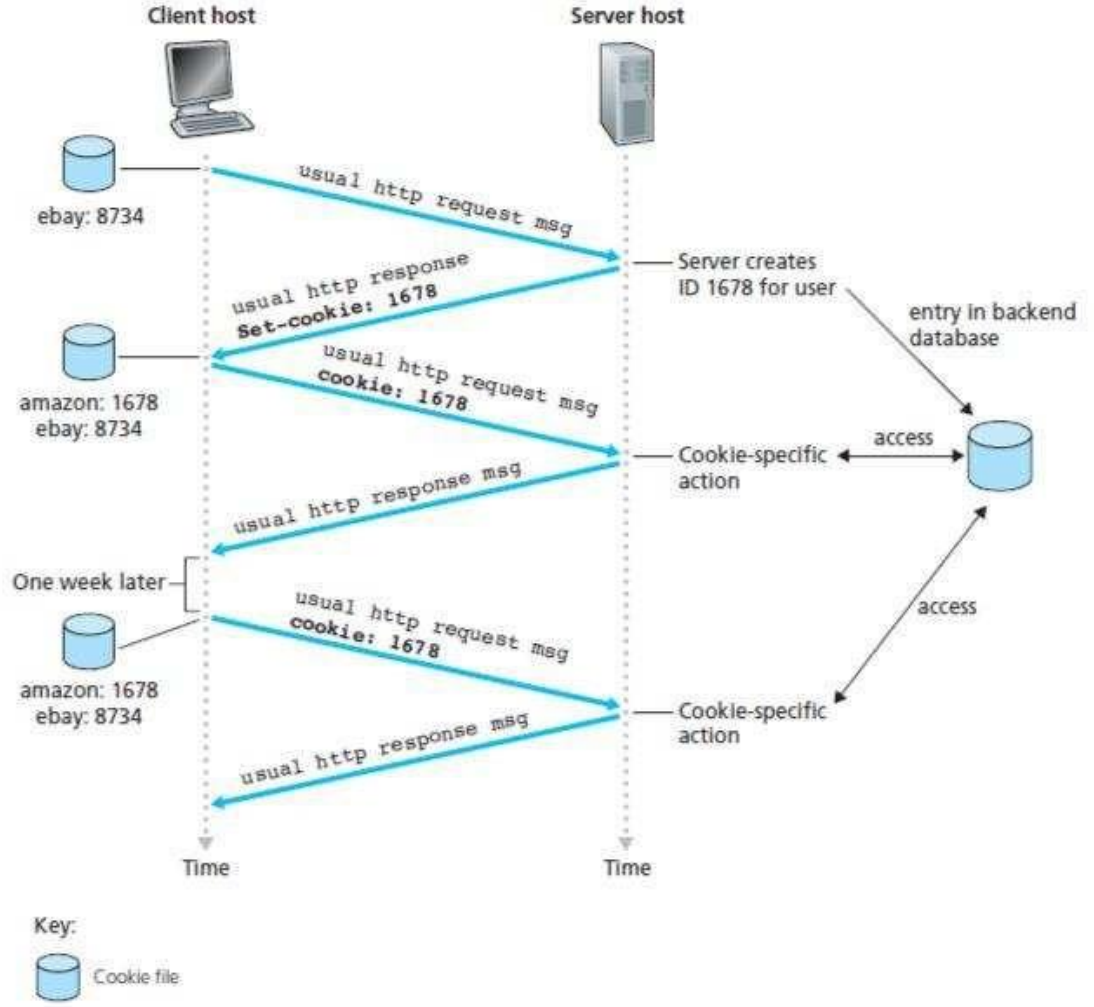

Figure 2.10 ♦ Keeping user state with cookies

Web Caching

A **Web cache**—also called a **proxy server**—is a network entity that satisfies HTTP requests on the behalf of an origin Web server. The Web cache has its own disk storage and keeps copies of recently requested objects in this storage.

As shown in Figure 2.11, a user's browser can be configured so that all of the user's HTTP requests are first directed to the Web cache. Once a browser is configured, each browser request for an object is first directed to the Web cache.

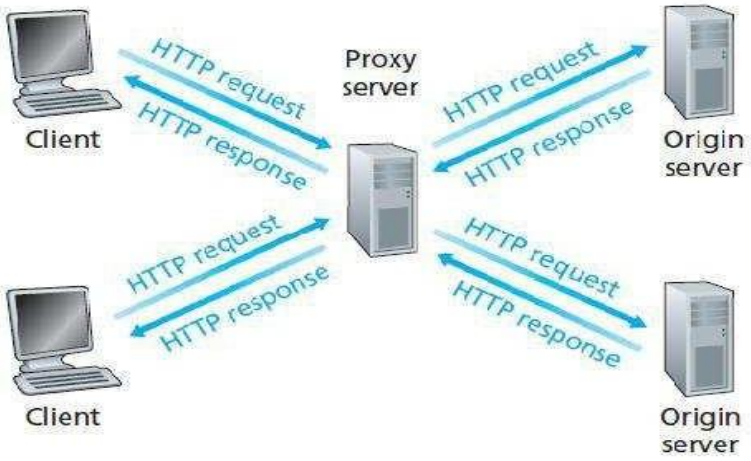

Figure 2.11 ♦ Clients requesting objects through a Web cache

3 **File Transfer: FTP**

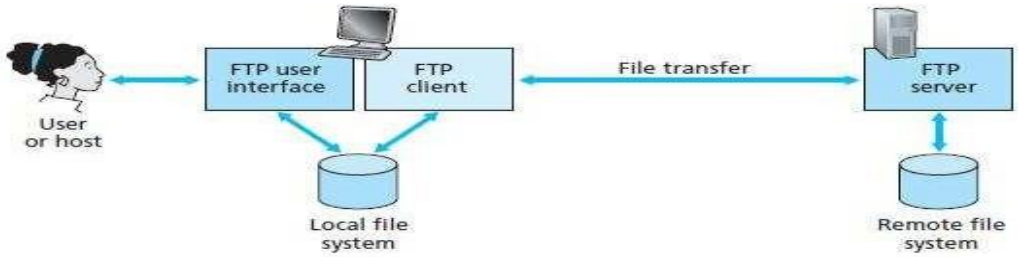

Figure 2.14 ♦ FTP moves files between local and remote file systems

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.

The user first provides the hostname of the remote host, causing 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 are 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).

Figure 2.15 ♦ Control and data connections

The FTP control and data connections are illustrated in Figure 2.15.

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.

When a user starts an FTP session with a remote host, the client side of FTP (user) first initiates a control TCP connection with the server side (remote host) 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 server side receives a command for a file transfer over the control connection (either to, or from, the remote host), the server side initiates a TCP data connection to the client side. 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 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 (that is, the data connections are non-persistent).

FTP Commands and Replies

Each command consists of four uppercase ASCII characters, some with optional arguments. Some of the more common commands are given below:

- `USER` username: Used to send the user identification to the 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 connection rather than the control TCP connection.
- `RETR` filename: Used to retrieve (that is, get) a file from the current directory of the remote host. This command causes the remote host to initiate a data connection and to send the requested file over the data connection.
- `STOR` filename: Used to store (that is, 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. 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`

4. Electronic Mail in the Internet

It has three major components: **user agents**, **mail servers**, and the **Simple Mail Transfer Protocol (SMTP)**.

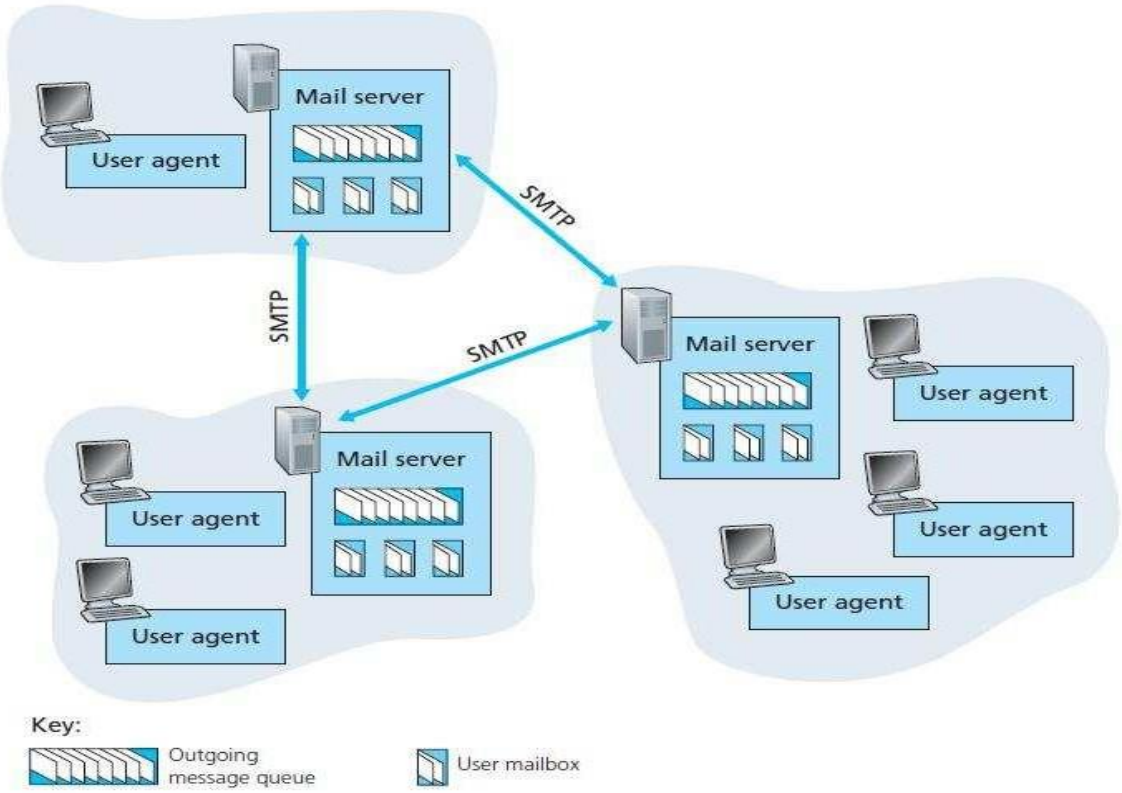

Figure 2.16 ♦ A high-level view of the Internet e-mail system

User agents allow users to read, reply to, forward, save, and compose messages. Microsoft Outlook and Apple Mail are examples of user agents for e-mail.

When Alice is finished composing her 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 retrieves the message from his mailbox in his mail server. A typical message starts its journey in the sender's user agent, travels to the sender's mail server, and travels to the recipient's mail server, where it is deposited in the recipient's mailbox.

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 e-mail message.

SMTP uses the reliable data transfer service of TCP to transfer mail from the sender's mail server to the recipient's mail server.

4.1 SMTP

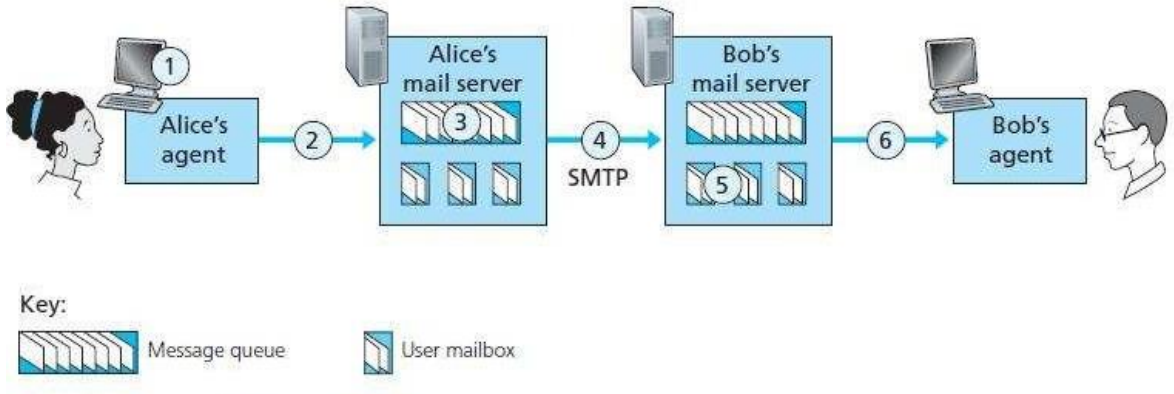

Figure 2.17 ♦ Alice sends a message to Bob

Suppose Alice wants to send Bob a simple ASCII message.

1. Alice invokes her user agent for e-mail, provides Bob's e-mail address (for example, bob@someschool.edu), composes a message, and instructs the user agent to send the message.

2. Alice's user agent sends the message to her mail server, where it is placed in a message queue.

3. The client side of SMTP, running on Alice's mail server, sees the message in the message queue. It opens a TCP connection to an SMTP server, running on Bob's mail server.

4. After some initial SMTP handshaking, the SMTP client sends Alice's message into the TCP connection.

5. At Bob's mail server, the server side of SMTP receives the message. Bob's mail server then places the message in Bob's mailbox.

6. Bob invokes his user agent to read the message at his convenience.

POP3: POP3 is an extremely simple mail access protocol. It is defined in [RFC 1939], which is short and quite readable. Because the protocol is so simple, its functionality is rather limited. POP3 begins when the user agent (the client) opens a TCP connection to the mail server (the server) on port 110. With the TCP connection established, POP3 progresses through three phases: authorization, transaction, and update.

During the first phase, authorization, the user agent sends a username and a password to authenticate the user. During the second phase, transaction, the user agent retrieves messages; also during this phase, the user agent can mark messages for deletion, remove deletion marks, and obtain mail statistics. Update: the mail server deletes the messages that were marked for deletion.

IMAP: An IMAP server will associate each message with a folder; when a message first arrives at the server, it is associated with the recipient's INBOX folder. The recipient can then move the message into a new, user-created folder, read the message, delete the message, and so on. The IMAP protocol provides commands to allow users to create folders and move messages from one folder to another. IMAP also provides commands that allow users to search remote folders for messages matching specific criteria.

5.DNS—The Internet's Directory Service

The main task of the Internet's **domain name system (DNS)** is to translate hostnames to IP addresses. In order to deal with the issue of scale, the DNS uses a large number of servers, organized in a hierarchical fashion and distributed around the world. No single DNS server has all of the mappings for all of the hosts in the Internet. Instead, the mappings are distributed across the DNS servers.

There are three classes of DNS servers—root DNS servers, top-level domain (TLD) DNS servers, and authoritative DNS servers

Root DNS servers. In the Internet there are 13 root DNS servers (labeled A through M), most of which are located in North America.

Top-level domain (TLD) servers. These servers are responsible for top-level domains such as com, org, net, edu, and gov, and all of the country top-level domains such as uk, fr, ca, and jp. The company Verisign Global Registry Services maintains the TLD servers for the com top-level domain, and the company Educause maintains the TLD servers for the edu top-level domain.

Authoritative DNS servers. Every organization with publicly accessible hosts (such as Web servers and mail servers) on the Internet must provide publicly accessible DNS records that map the names of those hosts to IP addresses. An organization's authoritative DNS server houses these DNS records.

The example shown in Figure 2.21 makes use of both **recursive queries** and **iterative queries**. The query sent from cis.poly.edu to dns.poly.edu is a recursive query, since the query asks dns.poly.edu to obtain the mapping on its behalf. But the subsequent three queries are iterative since all of the replies are directly returned to dns.poly.edu.

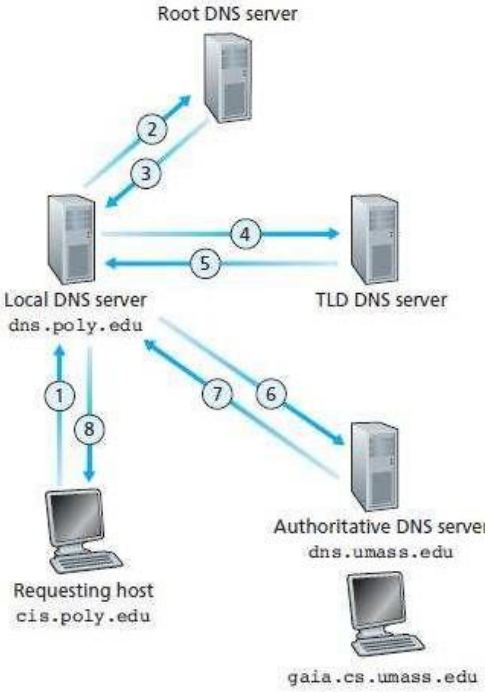

Figure 2.21 ♦ Interaction of the various DNS servers

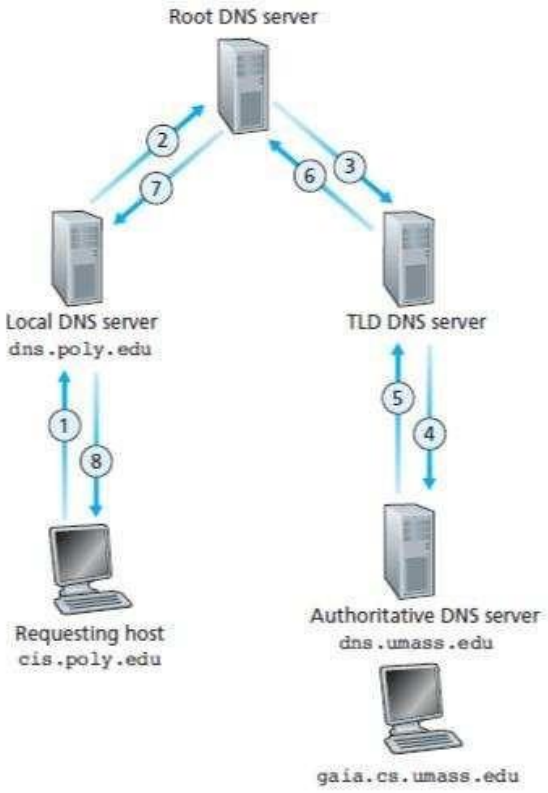

Figure 2.22 ♦ Recursive queries in DNS

DNS Records and Messages

The DNS servers that together implement the DNS distributed database store **resource records (RRs)**, including RRs that provide hostname-to-IP address mappings. Each DNS reply message carries one or more resource records. Resource record is a four-tuple that contains the following fields:

(Name, Value, Type, TTL)

TTL is the time to live of the resource record; it determines when a resource should be removed from a cache. In the example records given below, we ignore the TTL field. The meaning of Name and Value depend on Type:

• If Type=A, then Name is a hostname and Value is the IP address for the hostname. Thus, a Type A record provides the standard hostname-to-IP address mapping. As an example, (relay1.bar.foo.com, 145.37.93.126, A) is a Type A record.

• If Type=NS, then Name is a domain (such as foo.com) and Value is the hostname of an authoritative DNS server that knows how to obtain the IP addresses for hosts in the domain. This record is used to route DNS queries further along in the query chain. As an example, (foo.com, dns.foo.com, NS) is a Type NS record.

• If Type=CNAME, then Value is a canonical hostname for the alias hostname Name. This record can provide querying hosts the canonical name for a hostname. As an example, (foo.com, relay1.bar.foo.com, CNAME) is a CNAME record.

• If Type=MX, then Value is the canonical name of a mail server that has an alias hostname Name. As an example, (foo.com, mail.bar.foo.com, MX) is an MX record.

DNS Messages:

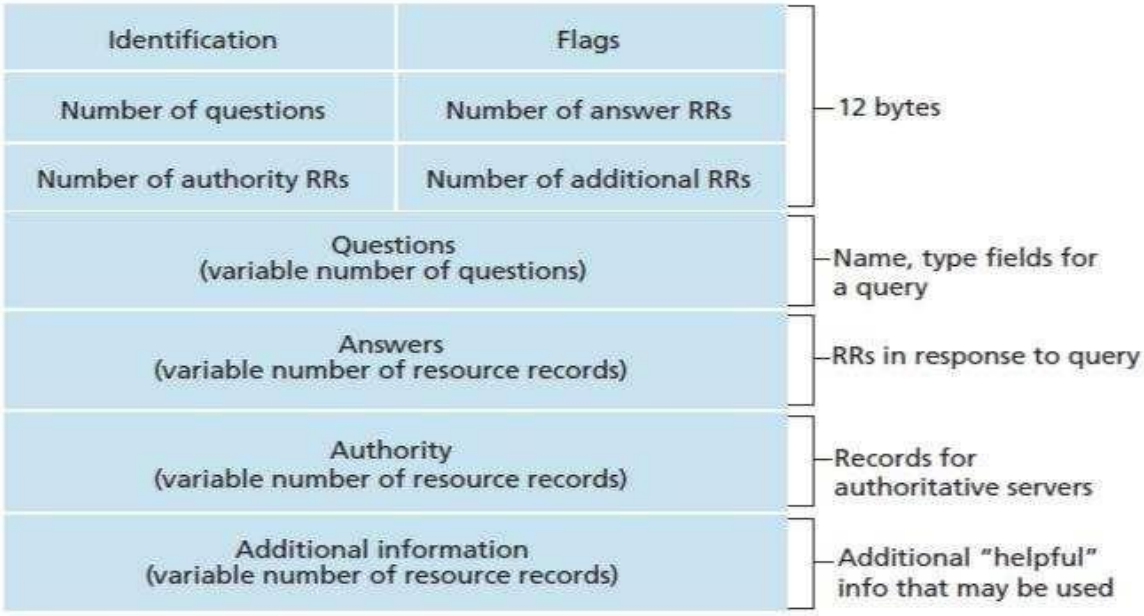

Figure 2.23 ♦ DNS message format

The first 12 bytes is the *header section*, which has a number of fields. The first field is a 16-bit number that identifies the query. This identifier is copied into the reply message to a query, allowing the client to match received replies with sent queries.

The *question section* contains information about the query that is being made.
In a reply from a DNS server, the *answer section* contains the resource records for the name that was originally queried.

The *authority section* contains records of other authoritative servers.

The *additional section* contains other helpful records.

6 Peer-to-Peer Applications

6.1 P2P File Distribution

Distributing a large file from a single server to a large number of hosts (called peers).
Calculating the distribution time for the P2P architecture is somewhat more complicated than for the client-server architecture, since the distribution time depends on how each peer distributes portions of the file to the other peers.

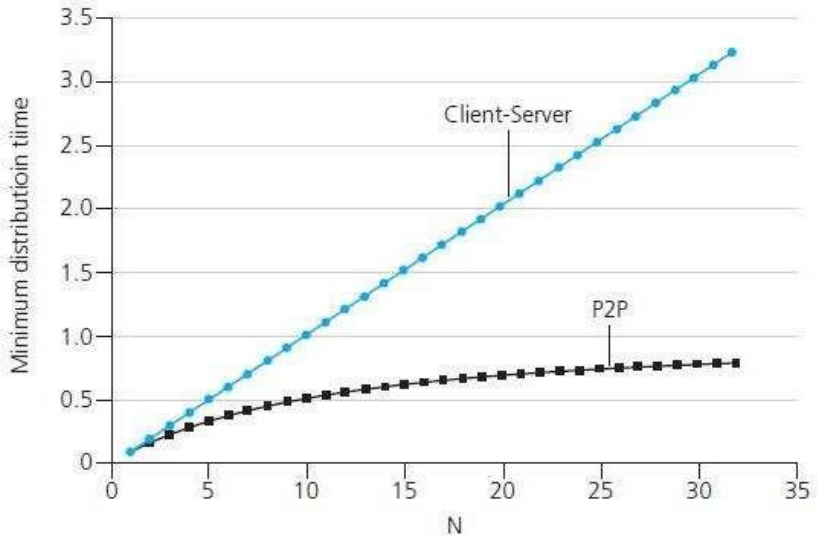

Figure 2.25 ♦ Distribution time for P2P and client-server architectures

N : Number of Peers

BitTorrent

BitTorrent is a popular P2P protocol for file distribution. In BitTorrent lingo, the collection of all peers participating in the distribution of a particular file is called a *torrent*. Peers in a torrent download equal-size *chunks* of the file from one another, with a typical chunk size of 256 KBytes. When a peer first joins a torrent, it has no chunks. Over time it accumulates more and more chunks. While it downloads chunks it also uploads chunks to other peers. Once a peer has acquired the entire file, it may (selfishly) leave the torrent, or (altruistically) remain in the torrent and continue to upload chunks to other peers. Also, any peer may leave the torrent at any time with only a subset of chunks, and later rejoin the torrent.

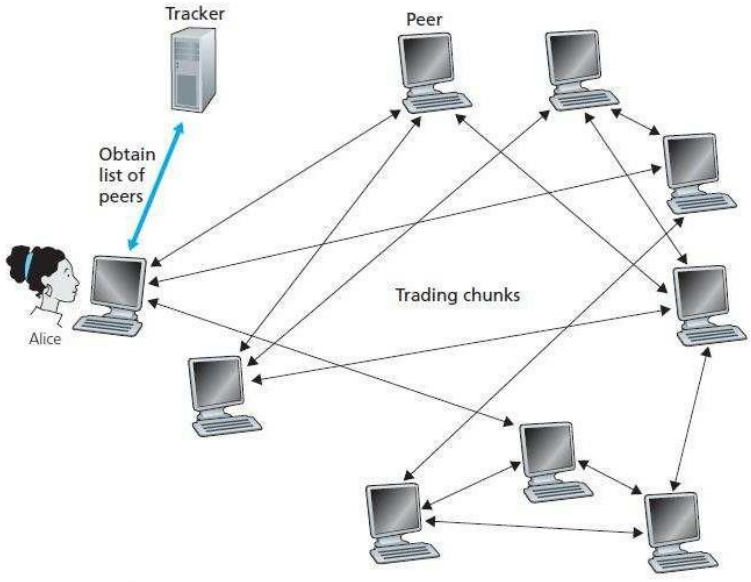

Figure 2.26 ♦ File distribution with BitTorrent

As shown in Figure 2.26, when a new peer, Alice, joins the torrent, the tracker randomly selects a subset of peers (for concreteness, say 50) from the set of participating peers, and sends the IP addresses of these 50 peers to Alice. Possessing this list of peers, Alice attempts to establish concurrent TCP connections with all

the peers on this list. Let's call all the peers with which Alice succeeds in establishing a TCP connection "neighboring peers." (In Figure 2.26, Alice is shown to have only three neighboring peers. Normally, she would have many more.) As time evolves, some of these peers may leave and other peers (outside the initial 50) may attempt to establish TCP connections with Alice. So a peer's neighboring peers will fluctuate over time.

6.2 Distributed Hash Tables (DHTs)

Let's consider here how to implement a simple database in a P2P network.

A centralized version of this simple database, which will simply contain (key, value) pairs. An example key-value pair is (Led Zeppelin IV, 128.17.123.38). We query the database with a key. If there are one or more key-value pairs in the database that match the query key, the database returns the corresponding values.

Consider how to build a distributed, P2P version of this database that will store the (key, value) pairs over millions of peers. In the P2P system, each peer will only hold a small subset of the totality of the (key, value) pairs. We'll allow any peer to query the distributed database with a particular key. The distributed database will then locate the peers that have the corresponding (key, value) pairs and return the key-value pairs to the querying peer. Any peer will also be allowed to insert new key-value pairs into the database. Such a distributed database is referred to as a **distributed hash table (DHT)**.

In DHT randomly scatter the (key, value) pairs across all the peers and have each peer maintain a list of the IP addresses of all participating peers. In this design, the querying peer sends its query to all other peers, and the peers containing the (key, value) pairs that match the key can respond with their matching pairs. Such an approach is completely unscalable, of course, as it would require each peer to not only know about all other peers (possibly millions of such peers!) but even worse, have each query sent to *all* peers.

Circular DHT

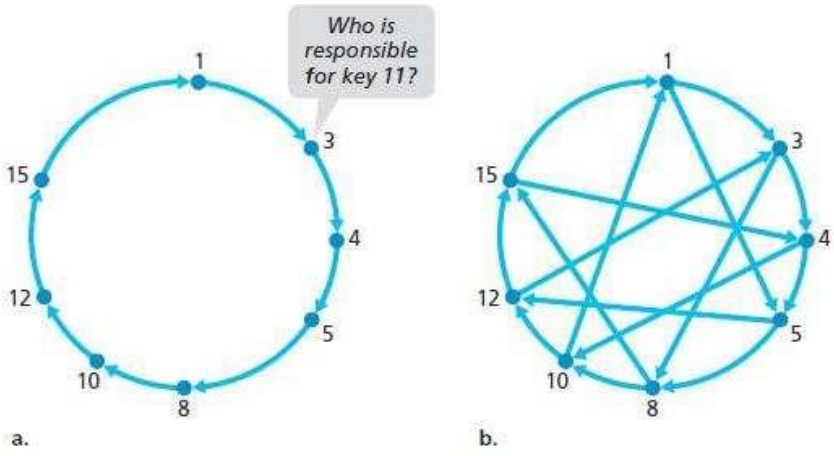

Figure 2.27 ♦ (a) A circular DHT. Peer 3 wants to determine who is responsible for key 11. (b) A circular DHT with shortcuts

Each peer is only aware of its immediate successor and predecessor; for example, peer 5 knows the IP address and identifier for peers 8 and 4 but does not necessarily know anything about any other peers that may be in the DHT. This circular arrangement of the peers is a special case of an **overlay network**. In an overlay network, the peers form an abstract logical network which resides above the "underlay" computer network consisting of physical links, routers, and hosts. The links in an overlay network are not physical links, but are simply virtual liaisons between pairs of peers. In the overlay in Figure 2.27(a), there are eight

peers and eight overlay links; in the overlay in Figure 2.27(b) there are eight peers and 16 overlay links. A single overlay link typically uses many physical links and physical routers in the underlay network.

7 Socket Programming: Creating Network Applications

7.1 Socket Programming with UDP

Demonstration of socket programming for both UDP and TCP:
1. The client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts the characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

We'll begin with the UDP client, which will send a simple application-level message to the server. In order for the server to be able to receive and reply to the client's message, it must be ready and running—that is, it must be running as a process before the client sends its message.

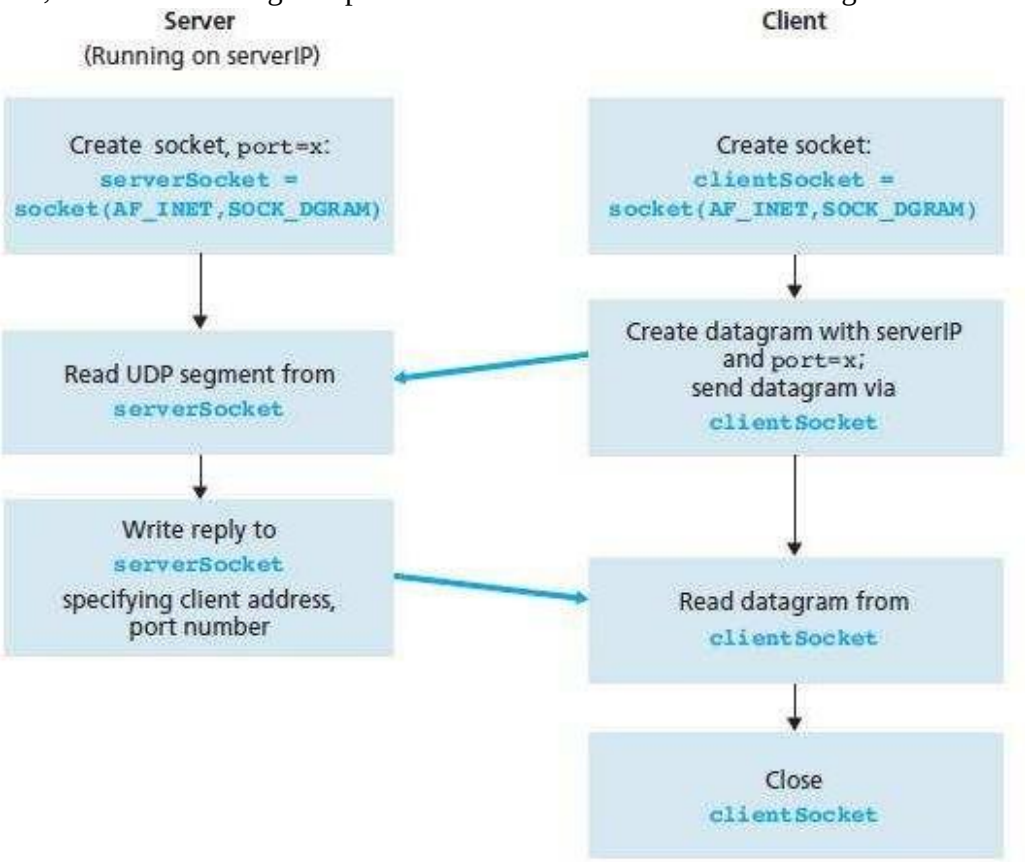

Figure 2.28 ♦ The client-server application using UDP

AF_INET indicates that the underlying network is using IPv4.
The second parameter indicates that the socket is of type SOCK_DGRAM, which means it is a UDP socket.

7.2 Socket Programming with TCP

With the server process running, the client process can initiate a TCP connection to the server. This is done in the client program by creating a TCP socket. When the client creates its TCP socket, it specifies the

address of the welcoming socket in the server, namely, the IP address of the server host and the port number of the socket. After creating its socket, the client initiates a three-way handshake and establishes a TCP connection with the server.

The three-way handshake, which takes place within the transport layer, is completely invisible to the client and server programs.

A special socket—that welcomes some initial contact from a client process running on an arbitrary host: TCP socket object that we call serverSocket; the newly created socket dedicated to the client making the connection is called connectionSocket.

From the application's perspective, the client's socket and the server's connection socket are directly connected by a pipe. As shown in Figure 2.29, the client process can send arbitrary bytes into its socket, and TCP guarantees that the server process will receive (through the connection socket) each byte in the order sent.

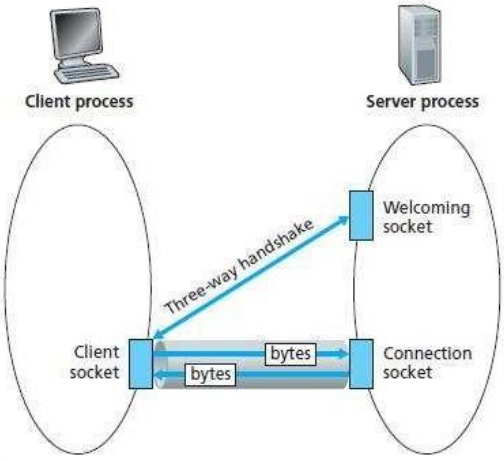

Figure 2.29 ♦ The TCPServer process has two sockets

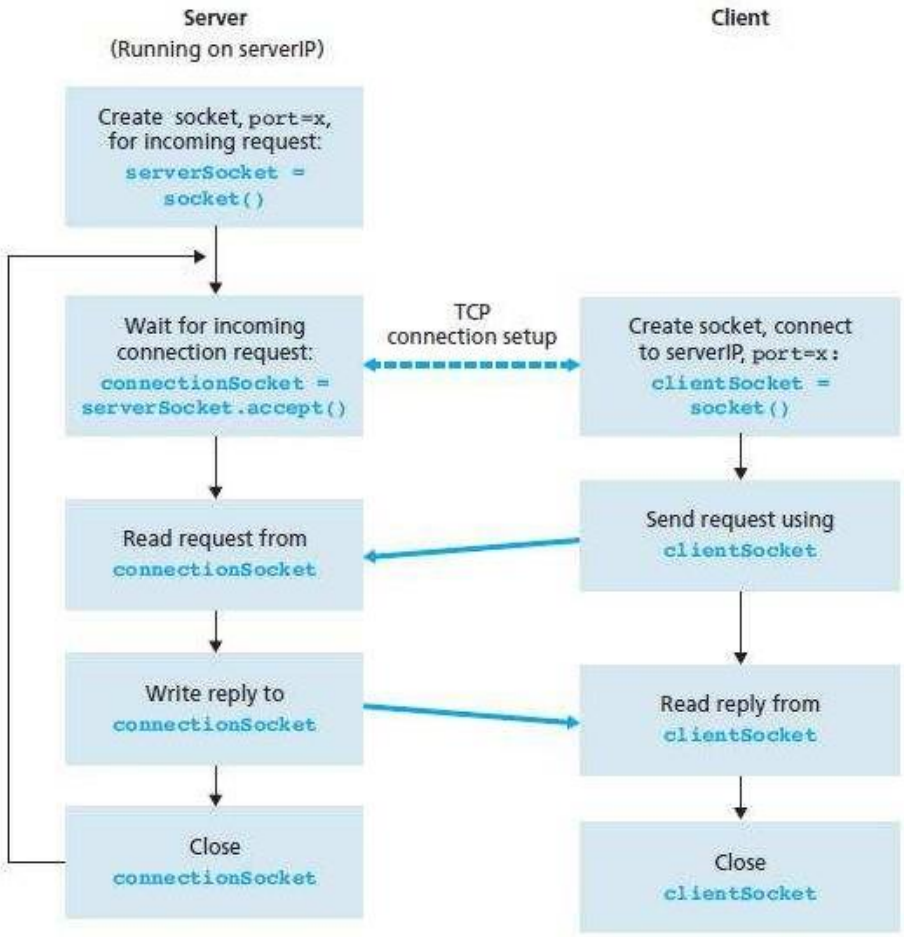

Figure 2.30 ♦ The client-server application using TCP

First line is the creation of the client socket.

clientSocket = socket(AF_INET, SOCK_STREAM)
The first parameter again indicates that the underlying network is using IPv4. The second parameter indicates that the socket is of type SOCK_STREAM, which means it is a TCP socket.

clientSocket.connect((serverName,serverPort))

Recall that before the client can send data to the server (or vice versa) using a TCP socket, a TCP connection must first be established between the client and server.
The above line initiates the TCP connection between the client and server. The parameter of the connect() method is the address of the server side of the connection.
After this line of code is executed, the three-way handshake is performed and a TCP connection is established between the client and server.