Module – 1

APPLICATION LAYER

Principles of Network Applications

Network application development is writing programs that run on different end systems and communicate with each other over the network.

For example, in the Web application there are two distinct programs that communicate with each other: the browser program running in the user's host and the Web server program running in the Web server host.

Network Application Architectures.

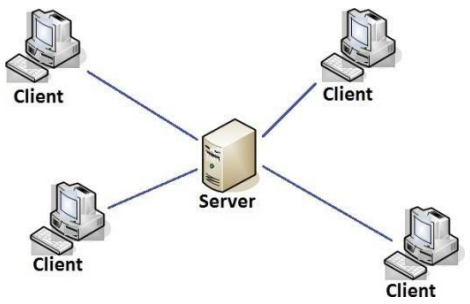
There are two different network application architecture, they are

1) Client Server Architecture
2) P2P Architecture

Client Server Architecture:

- In client-server architecture, there is an always-on host, called the server, which provides services when it receives requests from many other hosts, called clients.

  Example: In Web application Web server services requests from browsers running on client hosts. When a Web server receives a request for an object from a client host, it responds by sending the requested object to the client host.
- In client-server architecture, clients do not directly communicate with each other.
- The server has a fixed, well-known address, called an IP address. Because the server has a fixed, well-known address, and because the server is always on, a client can always contact the server by sending a packet to the server's IP address.
- Some of the better-known applications with a client-server architecture include the Web, FTP, Telnet, and e-mail.



Client Server Architecture

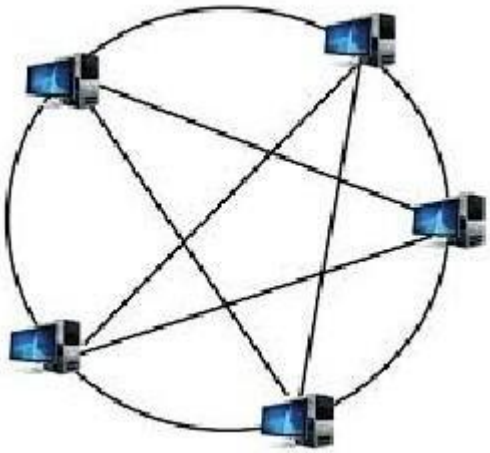
- In a client-server application, a single-server host is incapable of keeping up with all the requests from clients. For this reason, a data center, housing a large number of hosts, is often used to create a powerful virtual server.
- The most popular Internet services—such as search engines (e.g., Google and Bing), Internet commerce (e.g., Amazon and e-Bay), Web-based email (e.g., Gmail and Yahoo Mail), social networking (e.g., Facebook and Twitter)— employ one or more data centers.

Peer-to-peer (P2P) Architecture:

- In a P2P architecture, there is minimal dependence on dedicated servers in data centers.
- The application employs 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.
- Many of today's most popular and traffic-intensive applications are based on P2P architectures. These applications include file sharing (e.g., BitTorrent), Internet Telephony (e.g., Skype), and IPTV (e.g., Kankan and PPstream).
- **Features:**
  - **Self-scalability:**

    For example, in a P2P file-sharing application, although each peer generates workload by requesting files, each peer also adds service capacity to the system by distributing files to other peers.
  - **Cost effective:**

P2P architectures are also cost effective, since they normally don't require significant server infrastructure and server bandwidth



P2P Architecture

Future P2P applications face three major challenges:

1. **ISP Friendly.** Most residential ISPs have been dimensioned for "asymmetrical" bandwidth usage, that is, for much more downstream than upstream traffic. But P2P video streaming and file distribution applications shift upstream traffic from servers to residential ISPs, thereby putting significant stress on the ISPs. Future P2P applications need to be designed so that they are friendly to ISPs
2. **Security.** Because of their highly distributed and open nature, P2P applications can be a challenge to secure
3. **Incentives.** The success of future P2P applications also depends on convincing users to volunteer bandwidth, storage, and computation resources to the applications, which is the challenge of incentive design.

Processes Communicating

- A Process is a program or application under execution.
- When processes are running on the same or different 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.

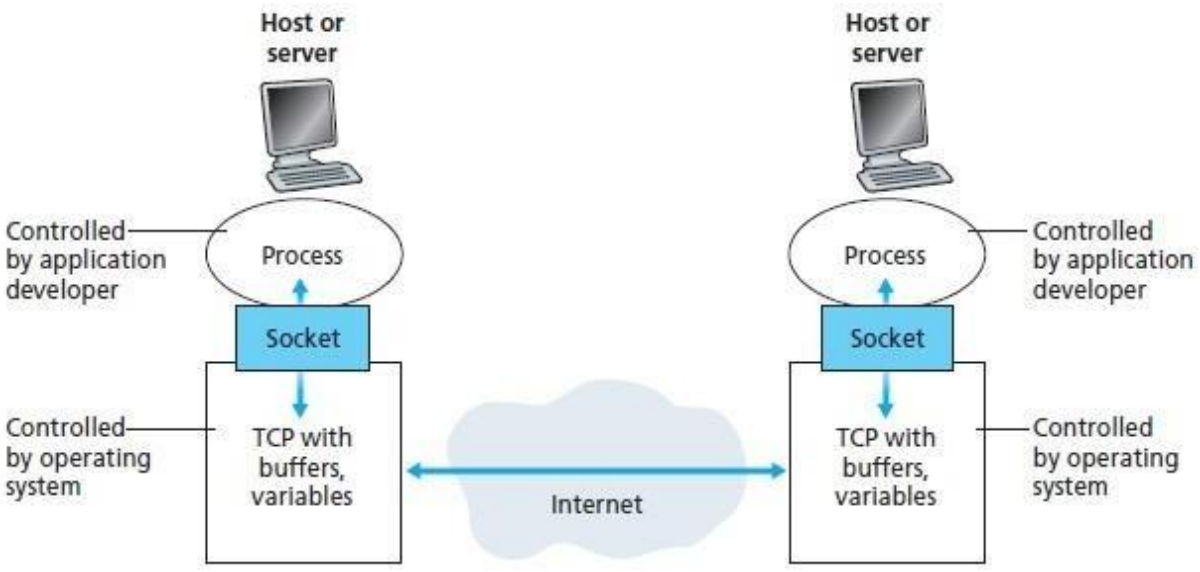
Client and Server Processes

- A network application consists of pairs of processes that send messages to each other over a network.

  For example, in the Web application a client browser process exchanges messages with a Web server process.
- In the context of a communication session between a pair of processes, the process that initiates the communication is labeled as the client. The process that waits to be contacted to begin the session is the server.

The Interface between the Process and the Computer Network

- A process sends messages into, and receives messages from, the network through a software interface called a socket.
- It is also referred to as the Application Programming Interface (API) between the application and the network, since the socket is the programming interface with which network applications are built.
- The application at the sending side pushes messages through the socket. At the other side of the socket, the transport-layer protocol has the responsibility of getting the messages to the socket of the receiving process.



Application processes, sockets, and underlying transport protocol

Addressing Processes

- 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.
- To identify the receiving process, two pieces of information need to be specified:
  (1) The address of the host
  (2) An identifier that specifies the receiving process in the destination host.
- In the Internet, the host is identified by its IP address.
- In addition to knowing the address of the host to which a message is destined, the sending process must also identify the receiving process running in the host. A destination port number serves this purpose. Popular applications have been assigned specific port numbers. 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.

Transport Services Available to Applications

1) Reliable Data Transfer

- Packets can get lost within a computer network. For example, a packet can overflow a buffer in a router, or can be discarded by a host or router after having some of its bits corrupted.

- For many applications—such as electronic mail, file transfer, remote host access, Web document transfers, and financial applications—data loss can have devastating consequences.
- Thus, to support these applications, something has to be done to guarantee that the data sent by one end of the application is delivered correctly and completely to the other end of the application.
- If a protocol provides such a guaranteed data delivery service, it is said to provide reliable data transfer. One important service that a transport-layer protocol can potentially provide to an application is process-to-process reliable data transfer.
- When a transport protocol provides this service, the sending process can just pass its data into the socket and know with complete confidence that the data will arrive without errors at the receiving process.
- When a transport-layer protocol doesn't provide reliable data transfer, some of the data sent by the sending process may never arrive at the receiving process. This may be acceptable for loss-tolerant applications, most notably multimedia applications such as conversational audio/video that can tolerate some amount of data loss.

2) Throughput

- Transport-layer protocol could provide guaranteed available throughput at some specified rate.
- With such a service, the application could request a guaranteed throughput of r bits/sec, and the transport protocol would then ensure that the available throughput is always at least r bits/sec. Such a guaranteed throughput service would appeal to many applications.
  For example, if an Internet telephony application encodes voice at 32 kbps, it needs to send data into the network and have data delivered to the receiving application at this rate.
- If the transport protocol cannot provide this throughput, the application would need to encode at a lower rate or may have to give up.
- Applications that have throughput requirements are said to be bandwidth-sensitive applications. Many current multimedia applications are bandwidth sensitive
- Elastic applications can make use of as much, or as little, throughput as happens to be available. Electronic mail, file transfer, and Web transfers are all elastic applications.

3) Timing

- A transport-layer protocol can also provide timing guarantees.
- Interactive real-time applications, such as Internet telephony, virtual environments, teleconferencing, and multiplayer games require tight timing constraints on data delivery in order to be effective.

4) Security

- Transport protocol can provide an application with one or more security services.

  For example, in the sending host, a transport protocol can encrypt all data transmitted by the sending process, and in the receiving host, the transport-layer protocol can decrypt the data before delivering the data to the receiving process.
- A transport protocol can provide security services like confidentiality, data integrity and end-point authentication.

Transport Services Provided by the Internet

The Internet makes two transport protocols available to applications, UDP and 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

Requirements of selected network applications

TCP Services

The TCP service model includes a connection-oriented service and a reliable data transfer service.

1) Connection-oriented service:

- In TCP the client and server exchange transport layer control information with each other before the application-level messages begin to flow.
- This handshaking procedure alerts the client and server, allowing them to prepare for an onslaught of packets.
- After the handshaking phase, a TCP connection is said to exist between the sockets of the two processes.
- The connection is a full-duplex connection in that the two processes can send messages to each other over the connection at the same time.
- When the application finishes sending messages, it must tear down the connection.

2) Reliable data transfer service:

- The communicating processes can rely on TCP to deliver all data sent without error and in the proper order.
- When one side of the application passes a stream of bytes into a socket, it can count on TCP to deliver the same stream of bytes to the receiving socket, with no missing or duplicate bytes.

TCP also includes a congestion-control mechanism.

UDP Services

- UDP is connectionless, so there is no handshaking before the two processes start to communicate.
- UDP provides an unreliable data transfer service—that is, when a process sends a message into a UDP socket, UDP provides no guarantee that the message will ever reach the receiving process.
- UDP does not include a congestion-control mechanism, so the sending side of UDP can

pump data into the layer below (the network layer) at any rate it pleases.

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

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

Application-Layer Protocols

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

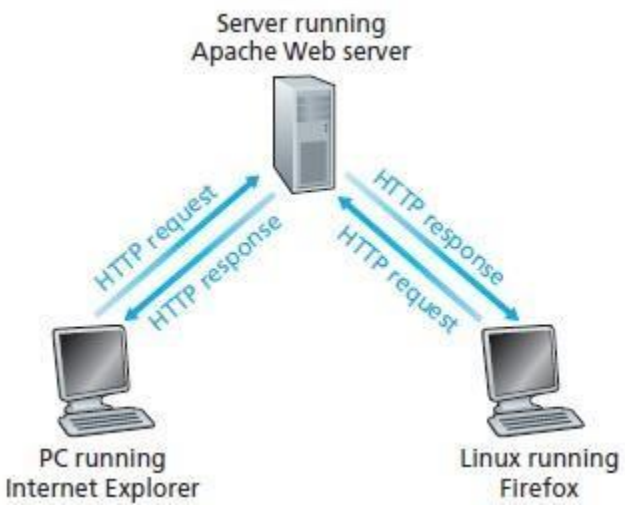
Overview of HTTP

- The Hyper Text 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 a server program. The client program and server program, 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.

- A Web page consists of objects. An object is simply a file like HTML file, a JPEG image, a Java applet, or a video clip—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 hostname of the server that houses the object and the object's path name.

  For example, the URL **http://abc.ac.in/admission/banner.jpg** has **www.abc.ac.in** for a hostname and **/admission/banner.jpg** for a path name.
- HTTP defines how Web clients request Web pages from Web servers and how servers transfer Web pages to clients.
- 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.
- HTTP uses TCP as its underlying transport protocol. 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.
- 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.



Non-Persistent and Persistent Connections

If Separate TCP connection is used for each request and response, then the connection is said to be non persistent. If same TCP connection is used for series of related request and response, then the connection is said to be persistent.

HTTP with Non-Persistent Connections

Let's suppose the page consists of a base HTML file and 10 JPEG images, and that all 11 of these objects reside on the same server.

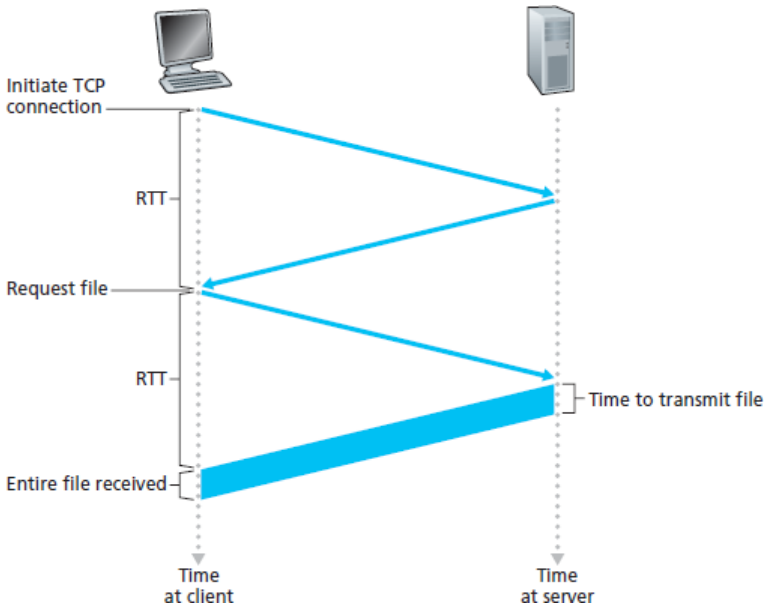
Further suppose the URL for the base HTML file is

http://www.someSchool.edu/someDepartment/home.index

Here is what happens:

1. The HTTP client process initiates a TCP connection to the server www.someSchool.edu on port number 80, which is the default port number for HTTP. Associated with the TCP connection, there will be a socket at the client and a socket at the server.
2. The HTTP client sends an HTTP request message to the server via its socket. The request message includes the path name /someDepartment/home.index.
3. The HTTP server process receives the request message via its socket, retrieves the object /someDepartment/home.index from its storage (RAM or disk), encapsulates the object in an HTTP response message, and sends the response message to the client via its socket.
4. The HTTP server process tells TCP to close the TCP connection.

5. The HTTP client receives the response message. The TCP connection terminates. The message indicates that the encapsulated object is an HTML file. The client extracts the file from the response message, examines the HTML file, and finds references to the 10 JPEG objects.
6. The first four steps are then repeated for each of the referenced JPEG objects.



- Round-trip time (RTT) is the time it takes for a small packet to travel from client to server and then back to the client.
- The RTT includes packet-propagation delays, packet queuing delays in intermediate routers and switches, and packet-processing delays.
- When a user clicks on a hyperlink, the browser 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.

HTTP with Persistent Connections

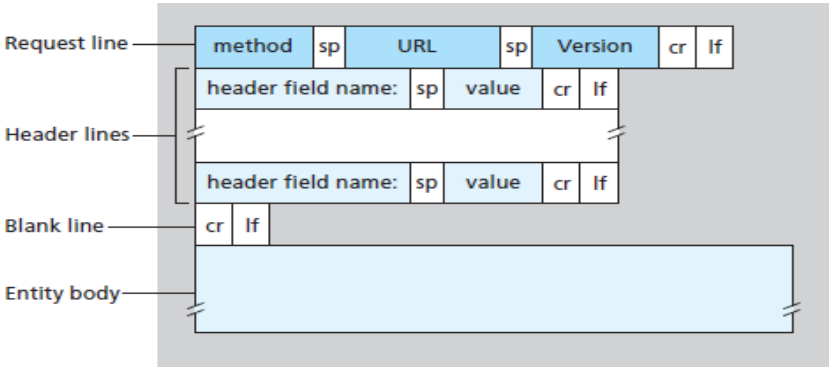
Non-persistent connections have some shortcomings.

1. A brand-new connection must be established and maintained for each requested object. For each of these connections, TCP buffers must be allocated and TCP variables must be kept in both the client and server. This can place a significant burden on the Web server, which may be serving requests from hundreds of different clients simultaneously.
2. Each object suffers a delivery delay of two RTTs— one RTT to establish the TCP connection and one RTT to request and receive an object.

With persistent connections, the server leaves the TCP connection open after sending a response. Subsequent requests and responses between the same client and server can be sent over the same connection. In particular, an entire Web page can be sent over a single persistent TCP connection. Moreover, multiple Web pages residing on the same server can be sent from the server to the same client over a single persistent TCP connection.

HTTP Message Format

HTTP Request Message:



Where sp – space, cr – carriage return and lf – line feed.

Method:

There are five HTTP methods:

- **GET:** The GET method is used when the browser requests an object, with the requested object identified in the URL field.
- **POST:** With a POST message, the user is still requesting a Web page from the server, but the specific contents of the Web page depend on what the user entered into the form fields. If the value of the method field is POST, then the entity body contains what the user entered into the form fields.
- **PUT:** The PUT method is also used by applications that need to upload objects to Web servers.
- **HEAD:** Used to retrieve header information. It is used for debugging purpose.
- **DELETE:** The DELETE method allows a user, or an application, to delete an object on a Web server.

URL: Specifies URL of the requested object

Version: This field represents HTTP version, usually HTTP/1.1

Header line:

Ex:

Host: www.someschool.edu

Connection: close

User-agent: Mozilla/5.0

Accept-language: fr

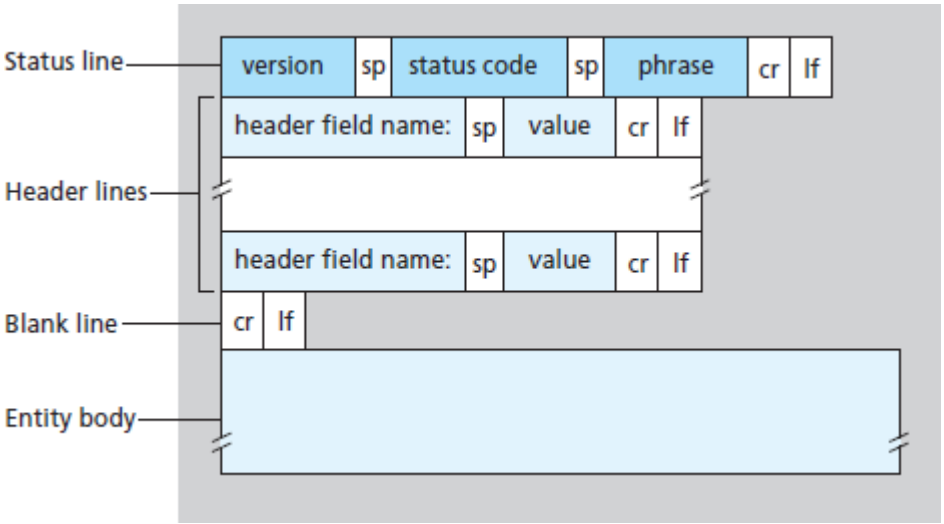
The header line **Host:www.someschool.edu** specifies the host on which the object resides.

By including the **Connection:close** header line, the browser is telling the server that it doesn't want to bother with persistent connections; 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.

The **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.

HTTP Response Message



Ex:

```
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 ...)
```

The **status line** has three fields: the protocol version field, a status code, and a corresponding status message.

Version is HTTP/1.1

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.

Header fields:

- The server uses the **Connection: close** header line to tell the client that it is going to close the TCP connection after sending the message.
- The **Date:** header line indicates the time and date when the HTTP response was created and sent by the server.
- The **Server:** header line indicates that the message was generated by an Apache Web server; it is analogous to the User-agent: header line in the HTTP request message.
- The **Last-Modified:** header line indicates the time and date when the object was created or last modified.
- The **Content-Length:** header line indicates the number of bytes in the object being sent.
- The **Content-Type:** header line indicates that the object in the entity body is HTML text.

User-Server Interaction: Cookies

It is often desirable for a Web site to identify users, either because the server wishes to restrict user access or because it wants to serve content as a function of the user identity. For these purposes, HTTP uses cookies.

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;

(4) A back-end database at the Web site.

Ex:

Suppose a user, who always accesses the Web using Internet Explorer from her home PC, contacts Amazon.com for the first time. Let us suppose that in the past he has already visited the eBay site. When the request comes into the Amazon Web server, the server creates a unique identification number and creates an entry in its back-end database that is indexed by the identification number. The Amazon Web server then responds to Susan's browser, including in the HTTP response a Set-cookie: header, which contains the identification number.

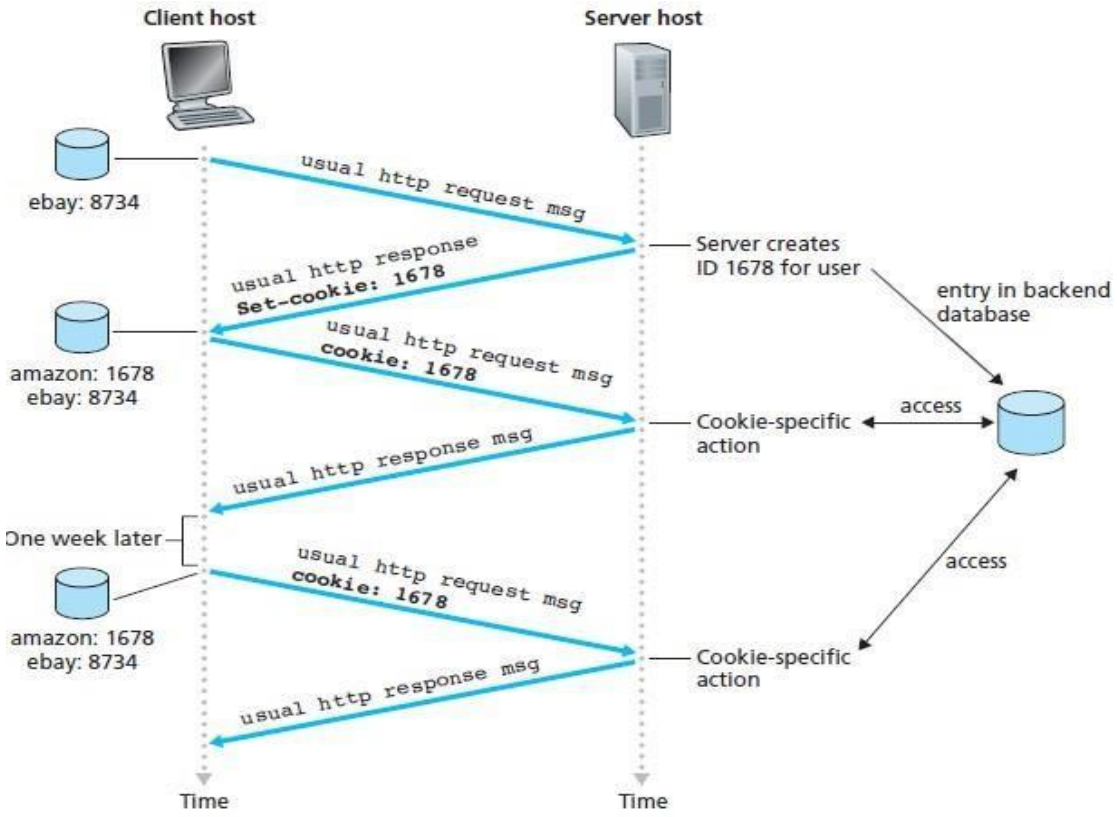
For example, the header line might be:

Set-cookie: 1678

When users browser receives the HTTP response message, it sees the Set-cookie: header. The browser then appends a line to the special cookie file that it manages. This line includes the hostname of the server and the identification number in the Set-cookie: header.

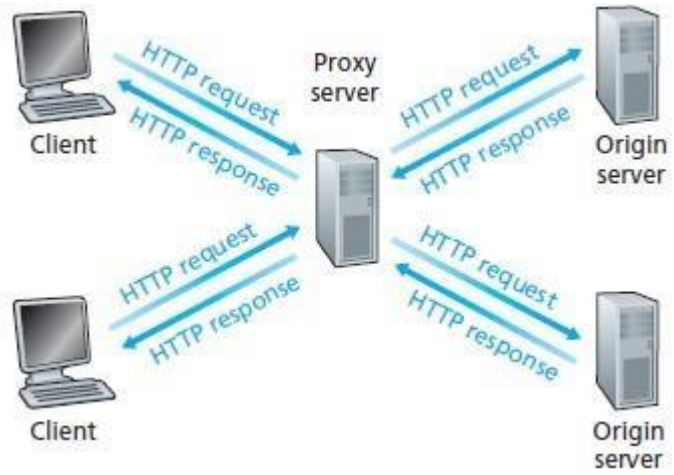
As user continues to browse the Amazon site, each time he requests a Web page, his browser consults his cookie file, extracts his identification number for this site, and puts a cookie header line that includes the identification number in the HTTP request. Specifically, each of his HTTP requests to the Amazon server includes the header line:

Cookie: 1678



Web Caching

- A Web cache—also called a proxy server—is a network entity that satisfies HTTP requests on the behalf of an origin Webserver.
- The Web cache has its own disk storage and keeps copies of recently requested objects in this storage.
- A user's browser can be configured so that all of the user's HTTP requests are first directed to the Web cache.



Ex: Suppose a browser is requesting the object http://www.someschool.edu/campus.gif. Here is what happens:

1. The browser establishes a TCP connection to the Web cache and sends an HTTP request for the object to the Web cache.
2. The Web cache checks to see if it has a copy of the object stored locally. If it does, the Web cache returns the object within an HTTP response message to the client browser.
3. If the Web cache does not have the object, the Web cache opens a TCP connection to the origin server, that is, to www.someschool.edu. The Web cache then sends an HTTP request for the object into the cache-to-server TCP connection.
4. After receiving this request, the origin server sends the object within an HTTP response to the Web cache.
5. When the Web cache receives the object, it stores a copy in its local storage and sends a copy, within an HTTP response message, to the client browser (over the existing TCP connection between the client browser and the Web cache).

- When web cache receives requests from and sends responses to a browser, it is a server. When it sends requests to and receives responses from an origin server, it is a client.
- Typically a Web cache is purchased and installed by an ISP. For example, a university might install a cache on its campus network and configure all of the campus browsers to point to the cache. Or a major residential ISP (such as AOL) might install one or more caches in its network and pre configure its shipped browsers to point to the installed caches.

- Web caching has seen deployment in the Internet for two reasons. First, a Web cache can substantially reduce the response time for a client request. Second, Web caches can substantially reduce traffic on an institution's access link to the Internet.

The Conditional GET

- Although caching can reduce user-perceived response times, it introduces a new problem—the copy of an object residing in the cache may be stale. In other words, the object housed in the Web server may have been modified since the copy was cached at the client.
- HTTP has a mechanism that allows a cache to verify that its objects are up to date. This mechanism is called the conditional GET.
- An HTTP request message is a so-called conditional GET message if (1) the request message uses the GET method and (2) the request message includes an If-Modified- Since: header line.

Ex: First, on the behalf of a requesting browser, a proxy cache sends a request message to a Web server:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

Second, the Web server sends a response message with the requested object to the cache:

```
HTTP/1.1 200 OK
Date: Sat, 8 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 7 Sep 2011 09:23:24
Content-Type: image/gif
(data data data data data ...)
```

The cache forwards the object to the requesting browser but also caches the object locally. Importantly, the cache also stores the last-modified date along with the object.

Third, one week later, another browser requests the same object via the cache, and the object is still in the cache. Since this object may have been modified at the Web server in the past week, the cache performs an up-to-date check by issuing a conditional GET. Specifically, the cache sends:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 7 Sep 2011 09:23:24
```

This conditional GET is telling the server to send the object only if the object has been modified since the specified date.

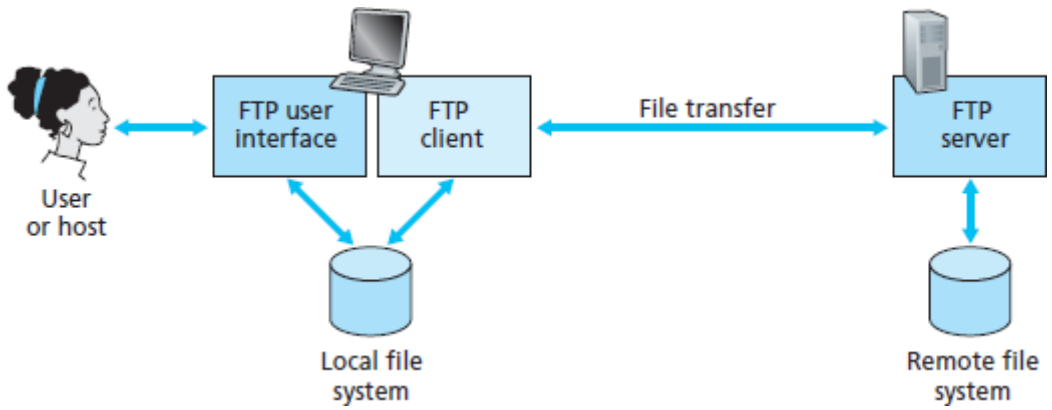
Suppose the object has not been modified since 7 Sep 2011 09:23:24. Then, fourth, the Web server sends a response message to the cache:

```
HTTP/1.1 304 Not Modified
Date: Sat, 15 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)
(empty entity body)
```

We see that in response to the conditional GET, the Web server still sends a response message but does not include the requested object in the response message.

File Transfer: FTP

- FTP is used for transferring file from one host to another host.
- In order for the user to access the remote account, the user must provide 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).



- 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).
- 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 directorytree.

FTP Commands and Replies

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.

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.

• 331 Username OK, password required

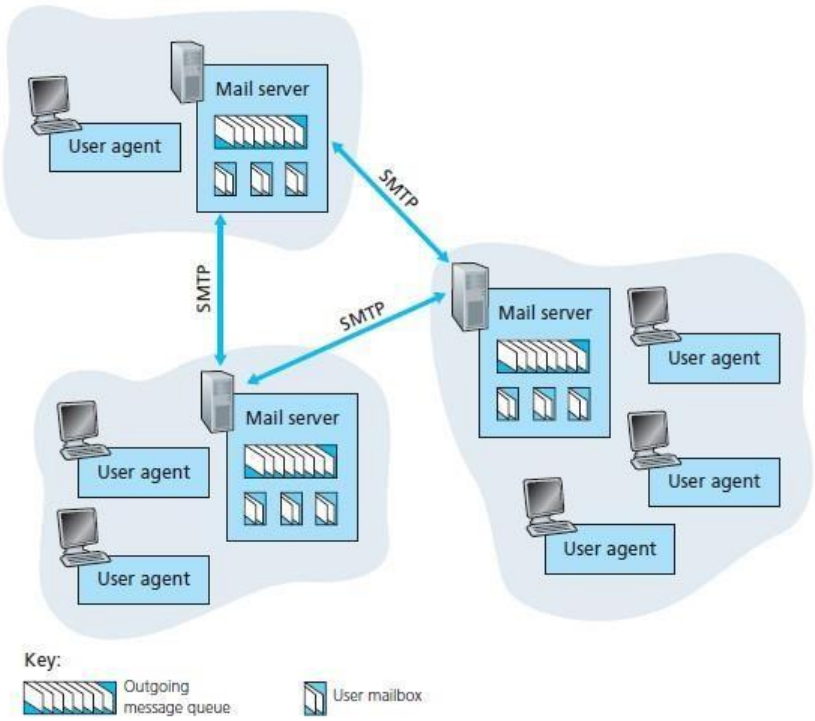
• 125 Data connection already open; transfer starting

• 425 Can't open data connection

• 452 Error writing file

Electronic Mail in the Internet

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



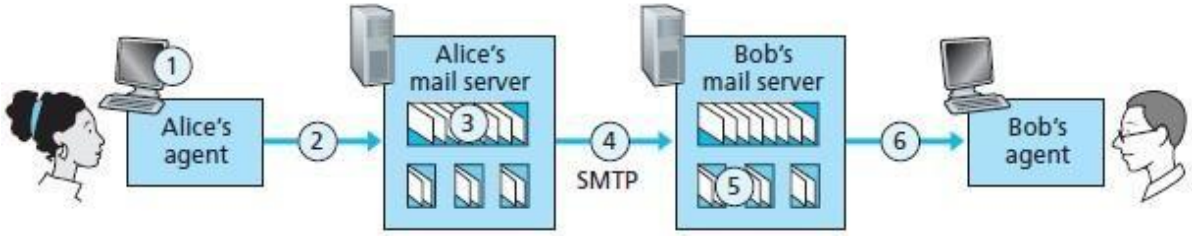
- **User agents** allow users to read, reply to, forward, save, and compose messages.
- **Mail servers** form the core of the e-mail infrastructure. Each recipient has a mailbox located in one of the mail servers. A typical message starts its journey in the sender's user agent, travels to the sender's mail server, and travels to the recipient's mail server, where it is deposited in the recipient's mailbox.
- **SMTP** is the principal 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 a server side, which executes on the recipient's mail server.

SMTP

SMTP transfers messages from senders' mail servers to the recipients' mail servers. It restricts the body (not just the headers) of all mail messages to simple 7-bit ASCII.

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.



An example transcript of messages exchanged between an SMTP client (C) and an SMTP server (S).

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

Comparison with HTTP

HTTP	SMTP
Pull Protocol- someone loads information on a Web server and users use HTTP to pull the information from the server at their convenience.	Push Protocol- the sending mail server pushes the file to the receiving mail server.
HTTP does not mandates data to be in 7-bit ASCII format.	SMTP requires each message, including the body of each message, to be in 7-bit ASCII format.
HTTP encapsulates each object in its own HTTP response message.	Internet mail places all of the message's objects into one message.

Mail Message Formats

When an e-mail message is sent from one person to another, a header containing peripheral information precedes the body of the message.

The header lines and the body of the message are separated by a blank line.

Every header must have a From: header line and a To: header line; a header may include a Subject: header line as well as other optional header lines.

A typical message header looks like this:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Searching for the meaning of life.
```

Mail Access Protocols

SMTP protocol delivers the mail to the mail server. To fetch the mail from mail server receiver used mail access protocols.

There are currently a number of popular mail access protocols, including Post Office Protocol—Version 3 (POP3), Internet Mail Access Protocol (IMAP), and HTTP.

POP3

- POP3 is an extremely simple mail access protocol.
- 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 **authorization phase**, the user agent sends a username and a password to authenticate the user.
- During the **transaction phase**, the user agent retrieves messages; also during this phase, the user agent can mark messages for deletion, remove deletion marks, and obtain mail statistics.
- The update phase occurs after the client has issued the quit command, ending the POP3 session; at this time, the mail server deletes the messages that were marked for deletion.

- In a POP3 transaction, the user agent issues commands, and the server responds to each command with a reply. There are two possible responses: +OK used by the server to indicate that the previous command was fine; and -ERR, used by the server to indicate that something was wrong with the previous command.
- The authorization phase has two principal commands: user <username> and pass <password>.

```
user bob
+OK
pass hungry
+OK user successfully logged on
```

- A user agent using POP3 can often be configured (by the user) to "**download and delete**" or to "**download and keep**."
- In the download-and-delete mode, the user agent will issue the list, retr, and dele commands.

Ex:

```
C: list
S: 1 498
S: 2 912
S: .
```

```
C: retr 1
S: (blah blah ...
S: .....
S .....blah)
S: .
C: dele 1
C: retr 2
S: (blah blah ...
S: .....
S .....blah)
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

- A problem with this download-and-delete mode is that the recipient cannot access mail messages from multiple machines.
- In the download-and keep mode, the user agent leaves the messages on the mail server after downloading them. In this case, user can reread messages from different machines.

IMAP

- With POP3 access, once user has downloaded his messages to the local machine, he can create mail folders and move the downloaded messages into the folders. User can then delete messages, move messages across folders, and search for messages (by sender name or subject).
- But this paradigm—namely, folders and messages in the local machine—poses a problem for the nomadic user, who would prefer to maintain a folder hierarchy on a remote server that can be accessed from any computer. This is not possible with POP3—the POP3 protocol does not provide any means for a user to create remote folders and assign messages to folders.
- To solve this and other problems, the IMAP protocol was invented. Like POP3, IMAP is a mail access protocol. It has many more features than POP3, but it is also significantly more complex.
- 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.
- Another important feature of IMAP is that it has commands that permit a user agent to obtain components of messages. For example, a user agent can obtain just the message header of a message or just one part of a multipart MIME message. This feature is useful when there is a

low-bandwidth connection (for example, a slow-speed modem link) between the user agent and its mail server. With a low bandwidth connection, the user may not want to download all of the messages in its mailbox, particularly avoiding long messages that might contain, for example, an audio or video clip.

Web-Based E-Mail

More and more users today are sending and accessing their e-mail through their Web browsers. In this case user communicates with its remote mailbox via HTTP.

DNS—The Internet's Directory Service

- All the hosts connected to network is identified by IP address. But it is difficult for human beings to remember these IP address to access a particular host. Hence hosts are identified by hostnames. Ex: google.com
- But the routers require IP address to forward the packet.
- In order to map hostname with the IP address DNS is used.

Services Provided by DNS

- The DNS is (1) a distributed database implemented in a hierarchy of DNS servers, and (2) an application-layer protocol that allows hosts to query the distributed database.
- DNS is commonly employed by other application-layer protocols—including HTTP, SMTP, and FTP—to translate user-supplied hostnames to IP addresses.

Example:
Consider what happens when a browser running on some user's host, requests the URL www.someschool.edu/index.html.

In order for the user's host to be able to send an HTTP request message to the Web server www.someschool.edu, the user's host must first obtain the IP address of www.someschool.edu. This is done as follows.

1. The same user machine runs the client side of the DNS application.

2. The browser extracts the hostname, www.someschool.edu, from the URL and passes the hostname to the client side of the DNS application.
3. The DNS client sends a query containing the hostname to a DNS server.
4. The DNS client eventually receives a reply, which includes the IP address for the hostname. Once the browser receives the IP address from DNS, it can initiate a TCP connection to the HTTP server process located at port 80 at that IP address.

DNS provides a few other important services in addition to translating hostnames to IP addresses:

- **Host aliasing:** A host with a complicated hostname can have one or more alias names. For example, a hostname such as relay1.west-coast.enterprise.com could have, say, two aliases such as enterprise.com and www.enterprise.com. In this case, the hostname relay1.westcoast. enterprise.com is said to be a **canonical hostname.** Alias hostnames, when present, are typically more mnemonic than canonical hostnames. DNS can be invoked by an application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host.
- **Mail server aliasing:** For obvious reasons, it is highly desirable that e-mail addresses be mnemonic. For example, if Bob has an account with Hotmail, Bob's e-mail address might be as simple as bob@hotmail.com. However, the hostname of the Hotmail mail server is more complicated and much less mnemonic than simply hotmail.com (for example, the canonical Host name might be something like relay1.west-coast.hotmail.com). DNS can be invoked by a mail application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host.
- **Load distribution:** DNS is also used to perform load distribution among replicated servers, such as replicated Web servers. Busy sites, such as cnn.com, are replicated over multiple servers, with each server running on a different end system and each having a different IP address. For replicated Web servers, a set of IP addresses is thus associated with one canonical hostname. The DNS database contains this set of IP addresses. When clients make a DNS query for a name mapped to a set of addresses, the server responds with the entire set

of IP addresses, but rotates the ordering of the addresses within each reply. Because a client typically sends its HTTP request message to the IP address that is listed first in the set, DNS rotation distributes the traffic among the replicated servers.

Overview of How DNS Works

- Suppose that some application running in a user's host needs to translate a hostname to an IP address. The application will invoke the client side of DNS, specifying the hostname that needs to be translated.
- DNS in the user's host then takes over, sending a query message into the network.
- All DNS query and reply messages are sent within UDP datagrams to port 53. After a delay, ranging from milliseconds to seconds, DNS in the user's host receives a DNS reply message that provides the desired mapping. This mapping is then passed to the invoking application.

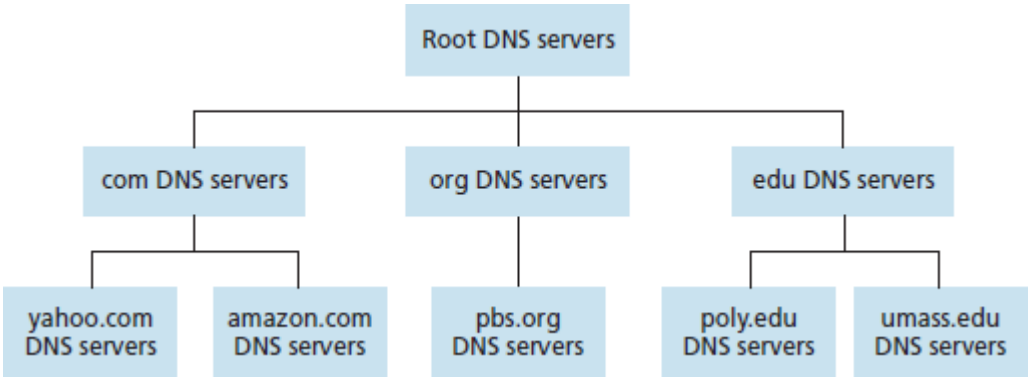
In this centralized design, clients simply direct all queries to the single DNS server, and the DNS server responds directly to the querying clients. Although the simplicity of this design is attractive, it is inappropriate for today's Internet, with its vast (and growing) number of hosts. The problems with a centralized design include:

- A single point of failure. If the DNS server crashes, so does the entire Internet!
- Traffic volume. A single DNS server would have to handle all DNS queries.
- Distant centralized database. A single DNS server cannot be "close to" all the querying clients. If we put the single DNS server in New York City, then all queries from Australia must travel to the other side of the globe, perhaps over slow and congested links. This can lead to significant delays.
- Maintenance. The single DNS server would have to keep records for all Internet hosts. Not only would this centralized database be huge, but it would have to be updated frequently to account for every new host.

A Distributed, Hierarchical Database

- 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.

- There are three classes of DNS servers—root DNS servers, top-level domain (TLD) DNS servers, and authoritative DNS servers—organized in a hierarchy.

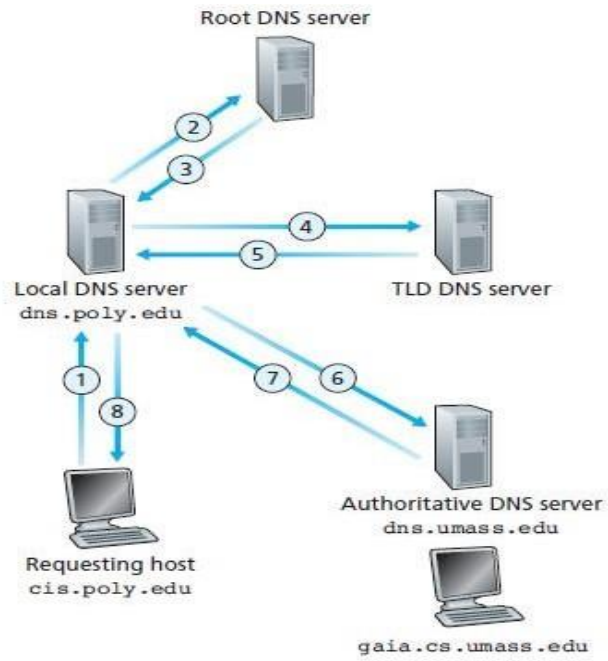


- **Root DNS servers.** In the Internet there are 13 root DNS servers (labeled A through M), most of which are located in North America.

  Although we have referred to each of the 13 root DNS servers as if it were a single server, each "server" is actually a network of replicated servers, for both security and reliability purposes. All together, there are 247 root servers.
- **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 in,uk, fr, ca.
- **Authoritative DNS servers:** Every organization with publicly accessible hosts 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.
- There is another important type of DNS server called the **local DNS server**. A local DNS server does not strictly belong to the hierarchy of servers but is nevertheless central to the DNS architecture. Each ISP—such as a university, an academic department, an employee's company, or a residential ISP—has a local DNS server.

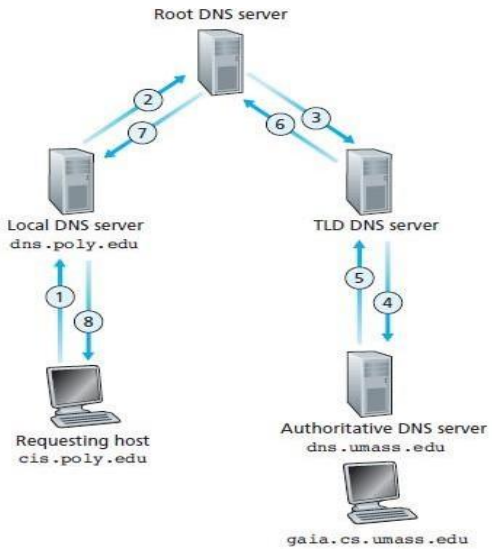
Two type of Interaction:

1) Recursive Queries:



Here DNS query is sent to local DNS server then to root server, then to TLD server and finally to authoritative DNS server. DNS response arrives in the reverse order.

2) Iterative Queries:



Here DNS query will be sent to Local DNS server, then to root server. Root server sends the IP address of TLD server. Now local DNS server sends query to TLD DNS server. TLD DNS server sends the IP address of authoritative DNS server to local DNS server. Now Local DNS server sends query to authoritative DNS server. Authoritative DNS server sends the IP address of host to local DNS server. Local DNS server sends it to the host.

DNS Caching

In a query chain, when a DNS server receives a DNS reply it can cache the mapping in its local memory.

If a hostname/IP address pair is cached in a DNS server and another query arrives to the DNS server for the same hostname, the DNS server can provide the desired IP address, even if it is not authoritative for the hostname. Because hosts and mappings between hostnames and IP addresses are by no means permanent, DNS servers discard cached information after a period of time (often set to two days).

DNS Records and Messages

The DNS servers that together implement the DNS distributed database store **resource records** (RRs).

A resource record is a four-tuple that contains the following fields:

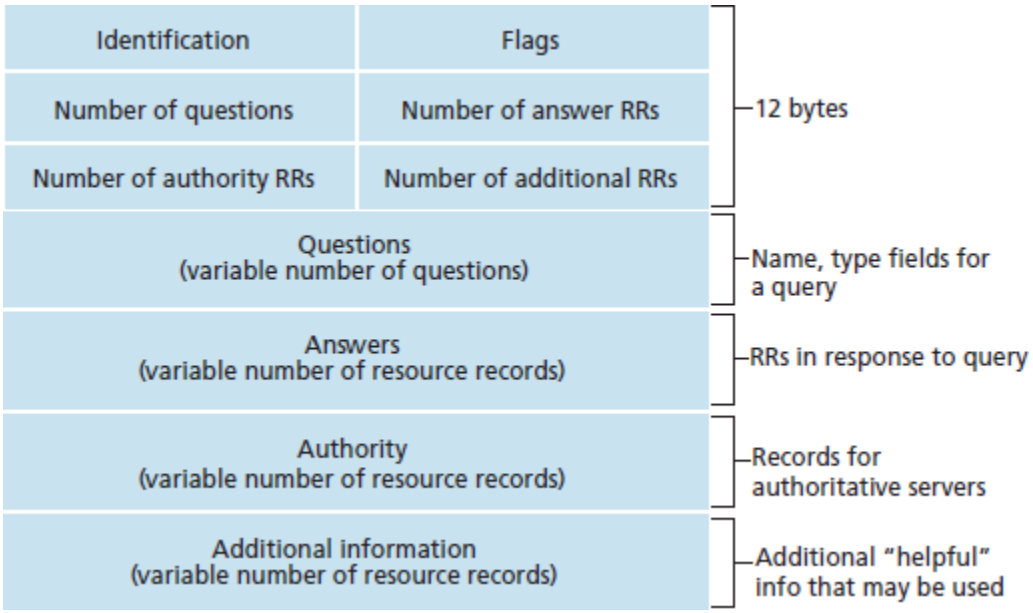
(Name, Value, Type, TTL)

TTL is the time to live of the resource record; it determines when a resource should be removed from a cache.

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.
- 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 thedomain.
- 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.
- If Type=MX, then Value is the canonical name of a mail server that has an aliashostname Name.

DNS Messages



- 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.
- There are a number of flags in the flag field.

  A 1-bit query/reply flag indicates whether the message is a query (0) or a reply (1). A1-bit authoritative flag is set in a reply message when a DNS server is an authoritative server for a queried name.

  A 1-bit recursion-desired flag is set when a client (host or DNS server) desires that the DNS server perform recursion when it doesn't have the record.

  A 1-bit recursion available field is set in a reply if the DNS server supports recursion.
- In the header, there are also four number-of fields. These fields indicate the number of occurrences of the four types of data sections that follow the header.
- The **question** section contains information about the query that is being made. This section includes (1) a name field that contains the name that is being queried, and (2) a type field that indicates the type of question being asked about the name

- 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.

Inserting Records into the DNS Database

Suppose you have just created an exciting new startup company called Network Utopia. The first thing you'll surely want to do is register the domain name networkutopia.com at a registrar. A registrar is a commercial entity that verifies the uniqueness of the domain name, enters the domain name into the DNS database (as discussed below), and collects a small fee from you for its services.

For the primary authoritative server for networkutopia.com, the registrar would insert the following two resource records into the DNS system:

(networkutopia.com, dns1.networkutopia.com, NS)

(dns1.networkutopia.com, 212.212.212.1, A)

Peer-to-Peer Applications

In P2P architecture, there is minimal (or no) reliance on always-on infrastructure servers. Instead, pairs of intermittently connected hosts, called peers, communicate directly with each other.

P2P File Distribution

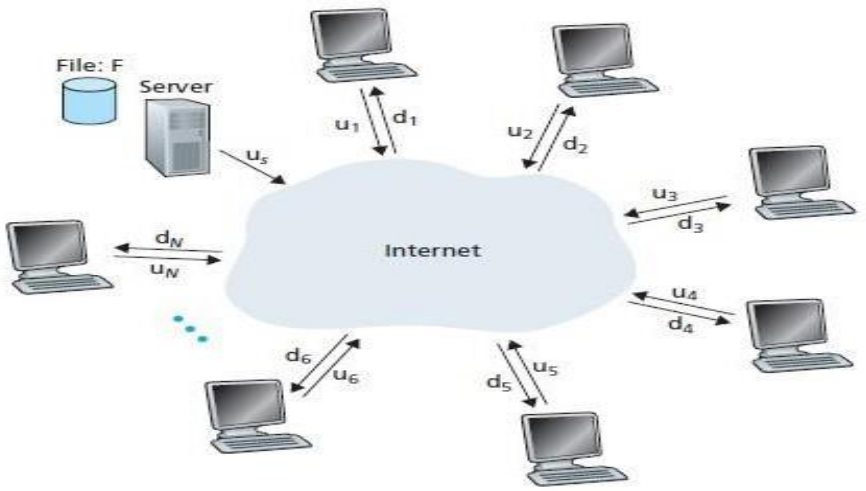
- In P2P file distribution, each peer can redistribute any portion of the file it has received to any other peers, thereby assisting the server in the distribution process.
- The most popular P2P file distribution protocol is BitTorrent.

Scalability of P2P Architectures

As shown in below Figure the server and the peers are connected to the Internet with access links. Denote the upload rate of the server's access link by u_s, the upload rate of the ith peer's

access link by u_i, and the download rate of the ith peer's access link by d_i. Also denote the size of the file to be distributed (in bits) by F and the number of peers that want to obtain a copy of the file by N.

The **distribution time** is the time it takes to get a copy of the file to all N peers.



In the client-server architecture, none of the peers aids in distributing the file. We make the following observations:

- The server must transmit one copy of the file to each of the N peers. Thus the server must transmit NF bits. Since the server's upload rate is us, the time to distribute the file must be at least NF/us.
- Let dmin denote the download rate of the peer with the lowest download rate, that is, dmin = min{d1,dp,...,dN}. The peer with the lowest download rate cannot obtain all F bits of the file in less than F/dmin seconds. Thus the minimum distribution time is at least F/dmin.

  Putting these two observations together, we obtain

$$D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\}.$$

In the P2P architecture we make the following observations:

- At the beginning of the distribution, only the server has the file. To get this file into the community of peers, the server must send each bit of the file at least once into its access link.

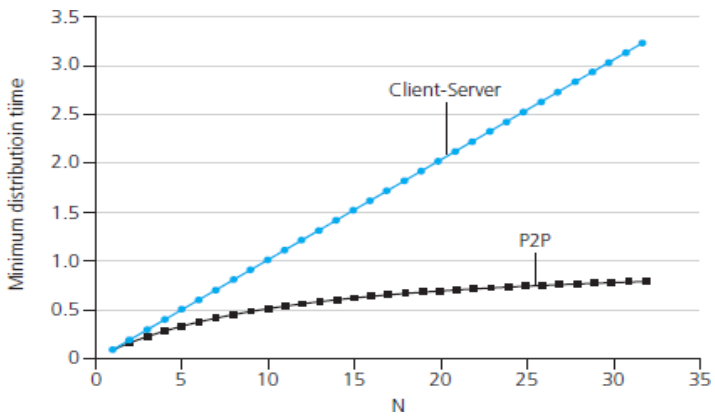
Thus, the minimum distribution time is at least F/us.

- As with the client-server architecture, the peer with the lowest download rate cannot obtain all F bits of the file in less than F/dmin seconds. Thus the minimum distribution time is at least F/dmin.
- Finally, observe that the total upload capacity of the system as a whole is equal to the upload rate of the server plus the upload rates of each of the individual peers, that is, utotal = us + u1 + … + uN. The system must deliver (upload) F bits to each of the N peers, thus delivering a total of NF bits. This cannot be done at a rate faster than utotal. Thus, the minimum distribution time is also at least NF/(us + u1 + … + uN).

Putting these three observations together, we obtain the minimum distribution time for P2P, denoted by D_{P2P}.

$$D_{P2P} \geq \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\}$$

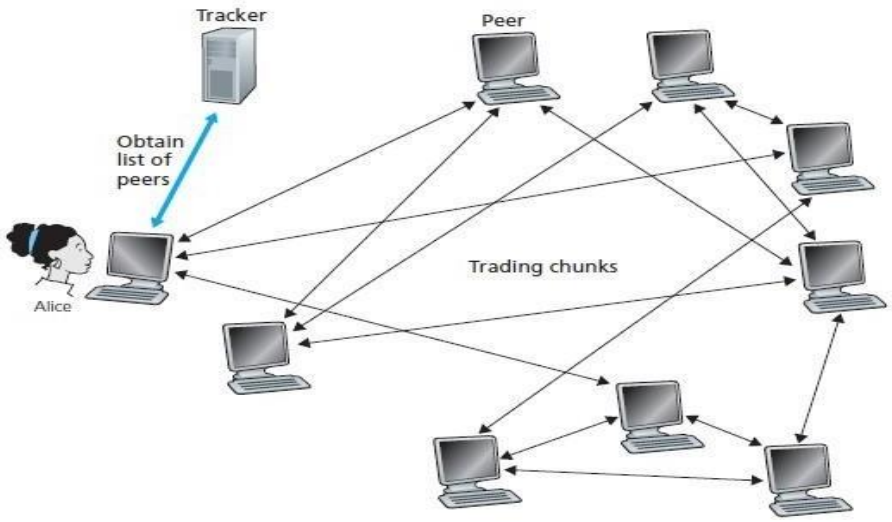
Below Figure compares the minimum distribution time for the client-server and P2P architectures assuming that all peers have the same upload rate u.



BitTorrent

- In BitTorrent, 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 leave the torrent, or 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.
- Each torrent has an infrastructure node called a tracker.
- When a peer joins a torrent, it registers itself with the tracker and periodically informs the tracker that it is still in the torrent. In this manner, the tracker keeps track of the peers that are participating in the torrent.
- When a new peer 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 new peer.
- Possessing this list of peers, new peer attempts to establish concurrent TCP connections with all the peers on this list. All the peers with which new peer succeeds in establishing a TCP connection will be called as "neighboring peers."
- As time evolves, some of these peers may leave and other peers (outside the initial 50) may attempt to establish TCP connections.
- Periodically, peer will ask each of its neighboring peers (over the TCP connections) for the list of the chunks they have. If peer has L different neighbors, it will obtain L lists of chunks. With this knowledge, peer will issue requests (again over the TCP connections) for chunks currently it does not have.
- In deciding which chunks to request, peer uses a technique called **rarest first**. The idea is to determine, from among the chunks peer does not have, the chunks that are the rarest among its neighbors and then request those rarest chunks first. In this manner, the rarest chunks get more quickly redistributed, aiming to equalize the numbers of copies of each chunk in the torrent.



- To determine which requests peer responds to, BitTorrent uses a clever trading algorithm. The basic idea is that peer gives priority to the neighbors that are currently supplying data to it at the highest rate. Specifically, for each of its neighbors, peer continually measures the rate at which it receives bits and determines the four peers that are feeding bits at the highest rate. Peer then reciprocates by sending chunks to these same four peers.
- Every 10 seconds, peer recalculates the rates and possibly modifies the set of four peers.
- In BitTorrent lingo, these four peers are said to be **unchoked**.
- Importantly, every 30 seconds, peer also picks one additional neighbor at random and sends it chunks. In BitTorrent lingo, this randomly selected peer is said to be **optimistically unchoked.**
- The random neighbor selection also allows new peers to get chunks, so that they can have something to trade.
- The incentive mechanism for trading just described is often referred to as tit-for-tat.

Distributed Hash Tables (DHTs)

- Centralized version of this simple database will simply contain (key, value) pairs. 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.
- Building such a database is straightforward with client-server architecture that stores all the

(key, value) pairs in one central server.

- P2P version of this database 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).
- One naïve approach to building a DHT is to 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 as it would require each peer to know about all other peers and have each query sent to all peers.
- An elegant approach to designing a DHT is to first assign an identifier to each peer, where each identifier is an integer in the range $[0, 2^n-1]$ for some fixed n.
- This also require each key to be an integer in the same range.
- To create integers out of such keys, we will use a hash function that maps each key (e.g., social security number) to an integer in the range $[0, 2^n-1]$.

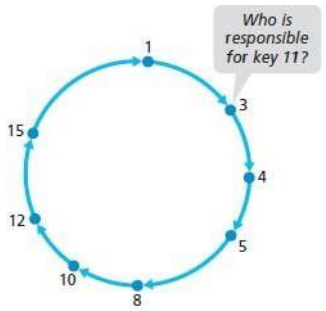
Problem of storing the (key, value) pairs in the DHT:

- The central issue here is defining a rule for assigning keys to peers. Given that each peer has an integer identifier and that each key is also an integer in the same range, a natural approach is to assign each (key, value) pair to the peer whose identifier is the closest to the key.
- To implement such a scheme, let's define the closest peer as the closest successor of the key.
- Now suppose a peer, Alice, wants to insert a (key, value) pair into the DHT. Conceptually, this is straightforward: She first determines the peer whose identifier is closest to the key; she then sends a message to that peer, instructing it to store the (key, value) pair.
- If Alice were to keep track of all the peers in the system (peer IDs and corresponding IP addresses), she could locally determine the closest peer. But such an approach requires each

peer to keep track of all other peers in the DHT—which is completely impractical for a large-scale system with millions of peers.

Circular DHT

To address this problem of scale, let's now consider organizing the peers into a circle. In this circular arrangement, each peer only keeps track of its immediate successor and immediate predecessor (modulo 2^n).



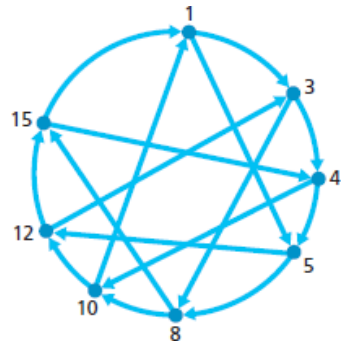
Each peer is only aware of its immediate successor and predecessor; for example, peer 5 knows the IP address and identifier for peers 8 and 4 but does not necessarily know anything about any other peers that may be in the DHT.

Now suppose that peer 3 wants to determine which peer in the DHT is responsible for key 11. Using the circular overlay, the origin peer (peer 3) creates a message saying "Who is responsible for key 11?" and sends this message clockwise around the circle. Whenever a peer receives such a message, because it knows the identifier of its successor and predecessor, it can determine whether it is responsible for (that is, closest to) the key in question. If a peer is not responsible for the key, it simply sends the message to its successor. So, for example, when peer 4 receives the message asking about key 11, it determines that it is not responsible for the key (because its successor is closer to the key), so it just passes the message along to peer 5. This process continues until the message arrives at peer 12, who determines that it is the closest peer to key 11. At this point, peer 12 can send a message back to the querying peer, peer 3, indicating that it is responsible for key 11.

Although each peer is only aware of two neighboring peers, to find the node responsible for a key (in the worst case), all N nodes in the DHT will have to forward a message around the circle; N/2 messages are sent on average.

Shortcuts are used to expedite the routing of query messages. Specifically, when a peer receives

a message that is querying for a key, it forwards the message to the neighbor (successor neighbor or one of the shortcut neighbors) which is the closet to the key.



When peer 4 receives the message asking about key 11, it determines that the closet peer to the key (among its neighbors) is its shortcut neighbor 10 and then forwards the message directly to peer 10. Clearly, shortcuts can significantly reduce the number of messages used to process a query.

Peer Churn

In P2P systems, a peer can come or go without warning. Thus, when designing a DHT, we also must be concerned about maintaining the DHT overlay in the presence of such peer churn.

To handle peer churn, we will now require each peer to track its first and second successors; for example, peer 4 now tracks both peer 5 and peer 8. We also require each peer to periodically verify that its two successors are alive

Let's now consider how the DHT is maintained when a peer abruptly leaves. For example, suppose peer 5 in above figure abruptly leaves. In this case, the two peers preceding the departed peer (4 and 3) learn that 5 has departed, since it no longer responds to ping messages. Peers 4 and 3 thus need to update their successor state information. Let's consider how peer 4 updates its state:

1. Peer 4 replaces its first successor (peer 5) with its second successor (peer 8).

2. Peer 4 then asks its new first successor (peer 8) for the identifier and IP address of its immediate successor (peer 10). Peer 4 then makes peer 10 its second successor.

Let's say a peer with identifier 13 wants to join the DHT, and at the time of joining, it only knows about peer 1's existence in the DHT. Peer 13 would first send peer 1 a message, saying "what will be 13's predecessor and successor?" This message gets forwarded through the DHT

until it reaches peer 12, who realizes that it will be 13's predecessor and that its current successor, peer 15, will become 13's successor. Next, peer 12 sends this predecessor and successor information to peer 13. Peer 13 can now join the DHT by making peer 15 its successor and by notifying peer 12 that it should change its immediate successor to 13.

Socket Programming: Creating Network Applications

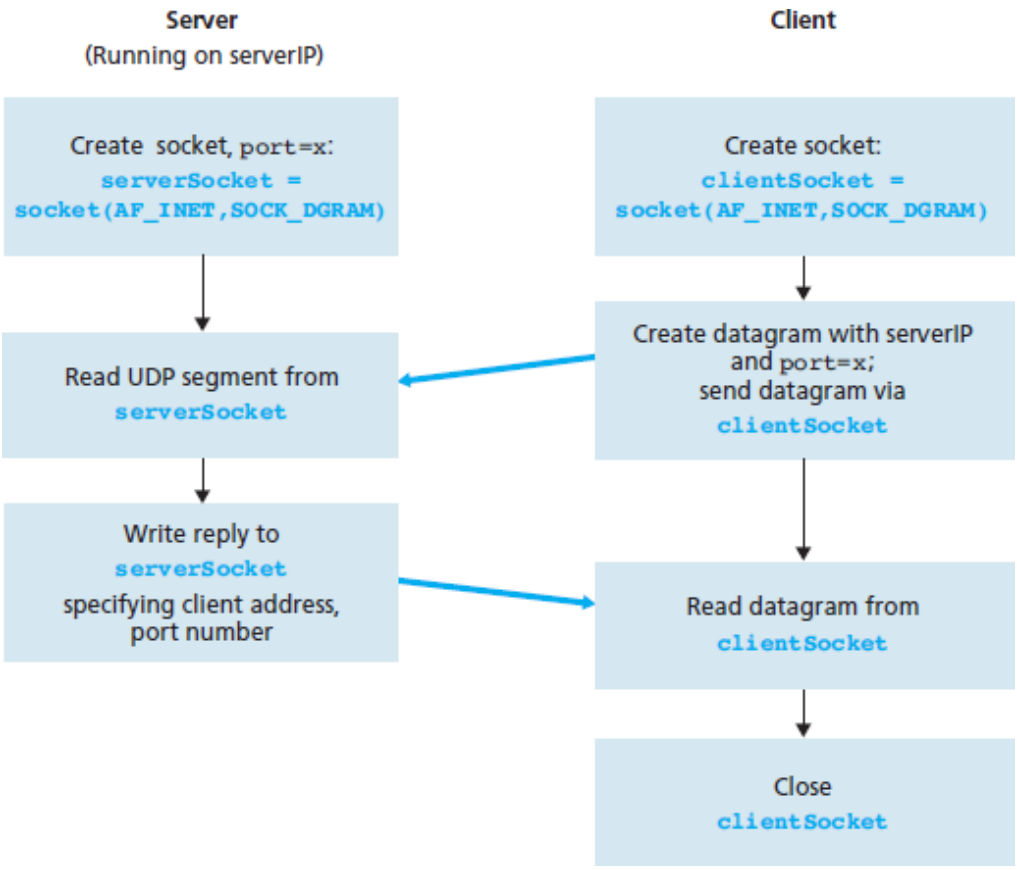
- A typical network application consists of a pair of programs—a client program and a server program—residing in two different end systems.
- When these two programs are executed, a client process and a server process are created, and these processes communicate with each other by reading from, and writing to, sockets.
- When creating a network application, the developer's main task is therefore to write the code for both the client and server programs.

Socket Programming with UDP

Before the sending process can push a packet of data out the socket door, when using UDP, it must first attach a destination address to the packet. After the packet passes through the sender's socket, the Internet will use this destination address to route the packet through the Internet to the socket in the receiving process. When the packet arrives at the receiving socket, the receiving process will retrieve the packet through the socket, and then inspect the packet's contents and take appropriate action.

Example application:

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.


Server
(Running on serverIP)
Client
Create socket, port=x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)
Create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)
Create datagram with serverIP
and port=x;
send datagram via
clientSocket
Read UDP segment from
serverSocket
Write reply to
serverSocket
specifying client address,
port number
Read datagram from
clientSocket
Close
clientSocket

UDPClient.py

Here is the code for the client side of the application:

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

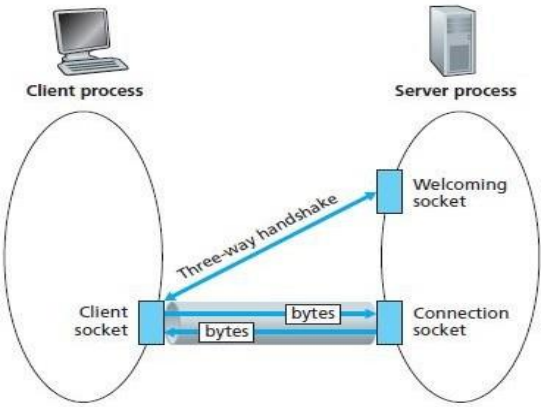
UDPServer.py

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind((’’, serverPort))
print ”The server is ready to receive”
while 1:
        message, clientAddress = serverSocket.recvfrom(2048)
        modifiedMessage = message.upper()
        serverSocket.sendto(modifiedMessage, clientAddress)
```

Socket Programming with TCP

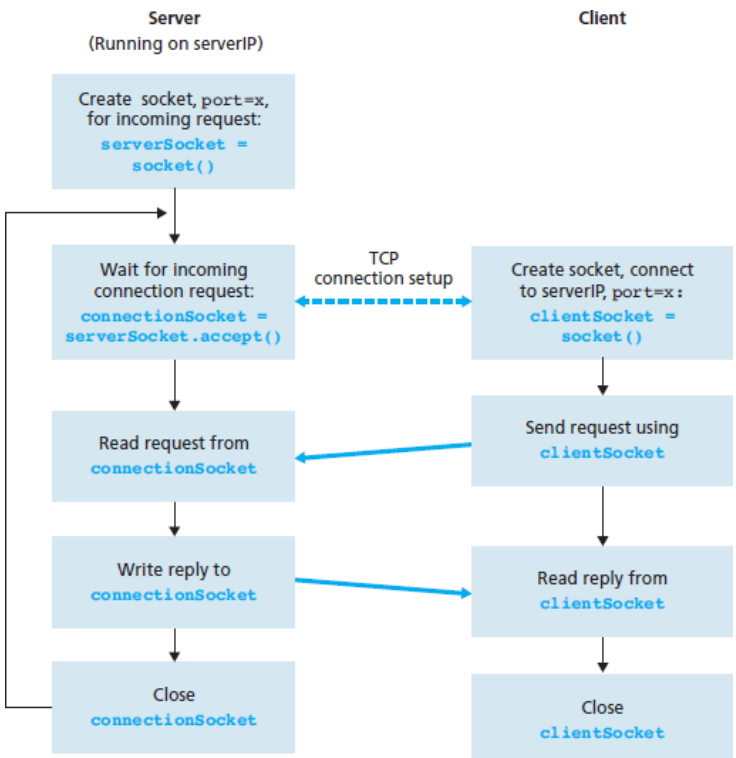
- Unlike UDP, TCP is a connection-oriented protocol. This means that before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection.

- One end of the TCP connection is attached to the client socket and the other end is attached to a server socket.
- When creating the TCP connection, we associate with it the client socket address (IP address and port number) and the server socket address (IP address and port number). With the TCP connection established, when one side wants to send data to the other side, it just drops the data into the TCP connection via its socket. This is different from UDP, for which the server must attach a destination address to the packet before dropping it into the socket.
- During the three-way handshake, the client process knocks on the welcoming door of the server process. When the server "hears" the knocking, it creates a new door— more precisely, a new socket that is dedicated to that particular client.



TCPClient.py

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```



TCPServer.py

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
print ‘The server is ready to receive’
while 1:
connectionSocket, addr = serverSocket.accept()
sentence = connectionSocket.recv(1024)
capitalizedSentence = sentence.upper()
connectionSocket.send(capitalizedSentence)
connectionSocket.close()
```