2.1 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, consider web application, browser program running in the user's host communicates with server program running in the Web server host.
- In a P2P file-sharing system there is a program in each host that participates in the file-sharing community. Below figure depicts a communication network scenario.

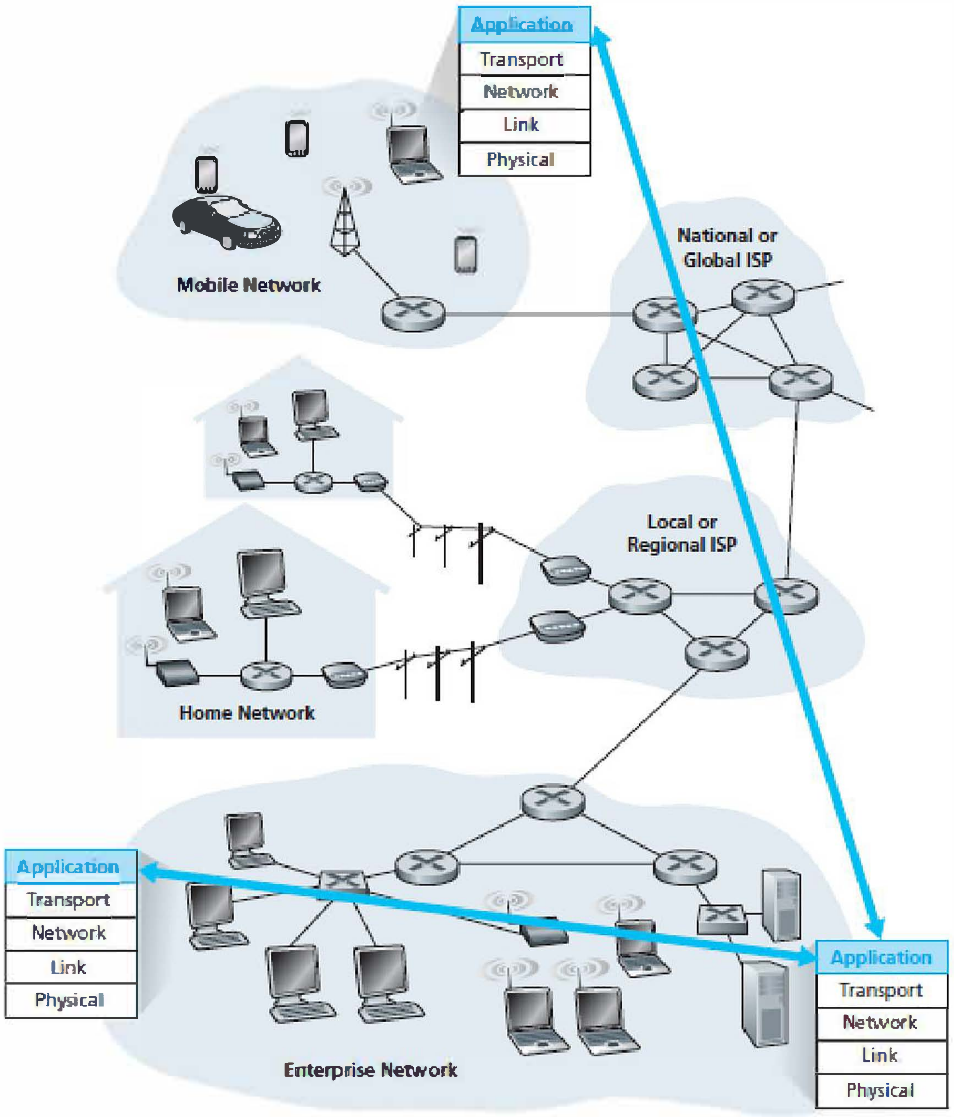


Figure 2.1 End Systems Communication

2.1.1 Network Application Architectures

Basically, one of the two predominant architectural paradigms used in modern network applications: the client-server architecture or the peer-to-peer (P2P) architecture.

Client – Server Architecture: In this, there is an always-on host, called the server, which services requests from many other hosts, called clients. Ex: Clients requesting pages from Web Server like amazon, ebay, vtu, etc.,

- In fact, clients do not directly communicate with each other, i.e., two browsers do not directly communicate.
- Both the client and server are processes, the server has a fixed address called an IP address and is always-on.
- Clients can always contact the server by sending a request packet to the server's IP address. Examples for Client-Server architecture are Web, FTP, Telnet, and e-mail.
- Since internet is wide, several clients would generate requests which leads to bottleneck for a single server. For this reason, a data center is used, housing a large number of hosts. Ex: Google, Amazon, Facebook, etc., are well known such centers
- In fact, Google has 30 to 50 data centers distributed around the world, which collectively handle search.

Peer – to – Peer Architecture: 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.

- Peers communicate directly via desktops, laptops or any hand held devices residing in homes, universities, and offices. Because the peers communicate without passing through a dedicated server, the architecture is called peer-to-peer.
- Examples include file sharing (e.g., BitTorrent), peer-assisted download acceleration (e.g., Xunlei), Internet Telephony (e.g., Skype), and IPTV (e.g.,Kankan and PPstream).

- One of the most compelling features of P2P architectures is their self-scalability. Like in file sharing, every peer assists each other in providing services

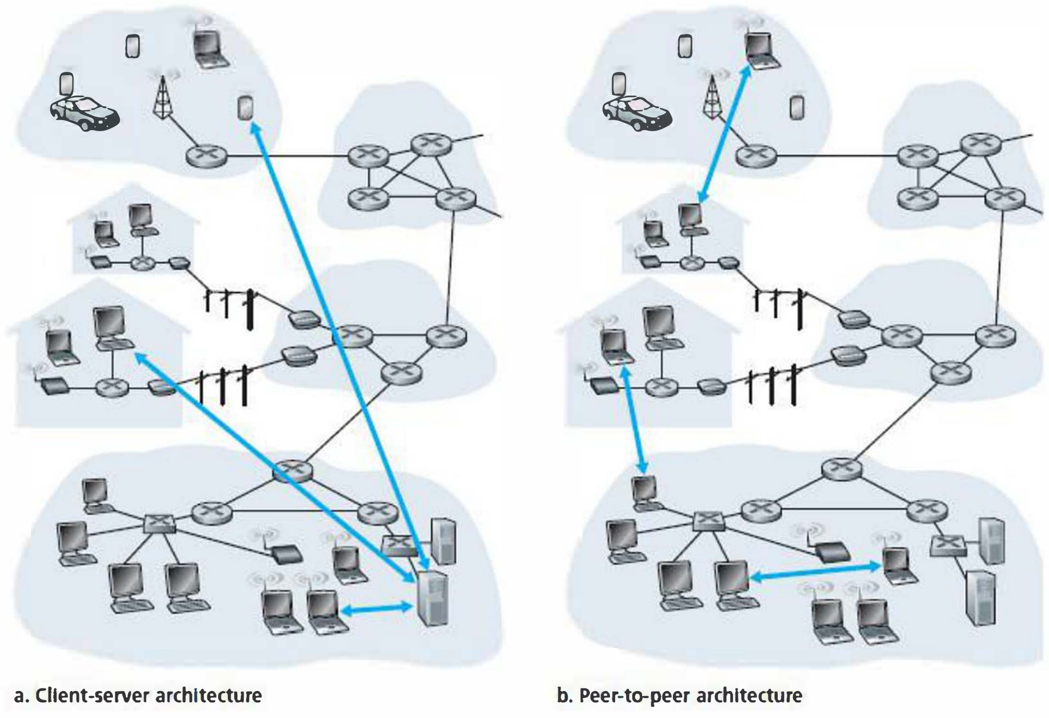


Figure 2.2 a) Client Server Architecture. b) Peer to Peer Architecture

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

Challenges faced by future P2P applications:

1. *ISP Friendly*. Most residential ISPs (Internet Service Providers) have been dimensioned for "asymmetrical" bandwidth usage, that is more bandwidth reserved for downstream than upstream traffic. But P2P video streaming and file distribution stresses the ISPs as reverse is the case.

2. *Security*. Because of their highly distributed and open nature, P2P applications can be a challenge to secure.

3. *Incentives*. More bandwidth, storage, and computation resources to the applications, are to be voluntarily contributed by the peers which is the challenge of incentive design.

Probable Question: ***Compare P2P and Client-Server architecture (CSA) with relevant points. Explain the challenges faced by future P2P Applications.***

2.1.2 Processes Communicating

- It is not actually programs but processes that communicate: A sending process and a receiving process. Among each pair of communicating processes, one is the client and the other process is 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.
- In order to establish communication between two processes irrespective of the underlying architecture, an interface is essential. Hence the concept of sockets appears before us.
- Sockets are basically APIs (Application Programming Interface) contributed by previous researches.

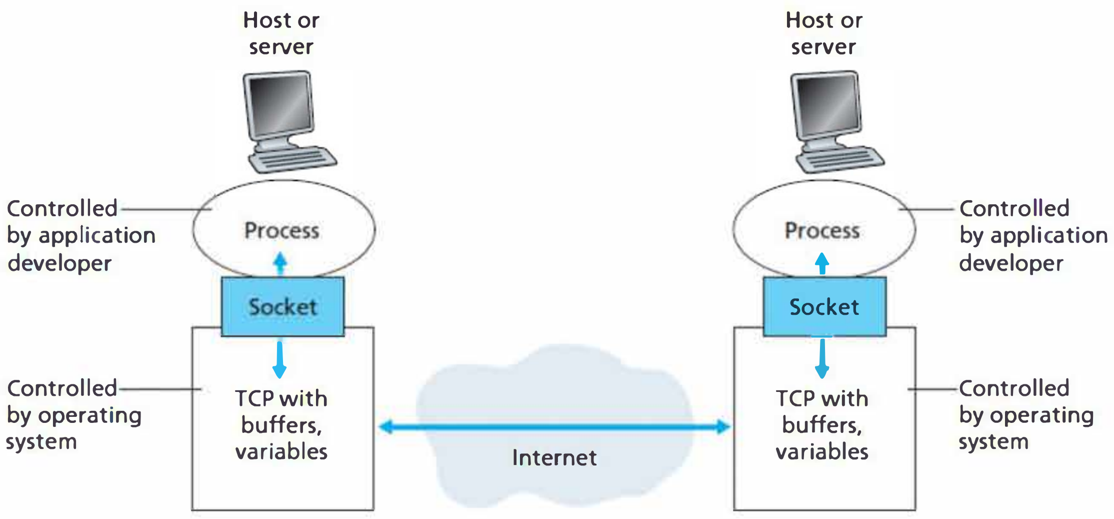

Host or server
Host or server
Controlled by application developer
Process
Socket
Controlled by operating system
TCP with buffers, variables
Internet
Process
Socket
TCP with buffers, variables
Controlled by application developer
Controlled by operating system

Figure 2.3: Processes communicating via internet using socket interface

- An address for the participating process is also of utmost important in order to identify it on hosts.

- To identify the receiving process, two pieces of information need to be specified:

  (1) the address of the host and

  (2) an identifier that specifies the receiving process in the destination host.

- This identity information is "IP Address" of size 32-bit. [ex: 192.168.45.12]
- Along with IP, port number is also needed to identify a process since a host runs several processes.

Probable Question: ***Explain process communication over Socket interface with a neat diagram***

2.1.3 Transport Services Available to Applications

- Transport layer protocols serve as mediator between the sockets at sending process and receiving process.
- Transport layer has two main protocols: TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).
- Choosing the right protocol is our responsibility. It is like choosing among train and flight for a travel.
- Transport layer protocol services can be portrayed along four dimensions: reliable data transfer, throughput, timing, and security.

Reliable Data Transfer

- During communication via a network, data packets get lost due to buffer overflow at router's end, connectivity breakage, etc., leading to 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.

- Providing guaranteed data delivery is nothing but all the data packets transmitted at sender's end must reach the destination without any errors.
- Non-guaranteed delivery may acceptable for loss-tolerant applications, most notably multimedia applications such as video conferencing, chatting, etc.

Throughput

- Throughput here, is the rate at which the sending process can deliver bits to the receiving process.
- Transport layer protocol can offer guaranteed throughput service. 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 (minimum) r bits/sec.
- Applications that have throughput requirements are said to be bandwidth-sensitive applications and rest are termed elastic applications.
- Almost all interactive real time communications are bandwidth-sensitive whereas Electronic mail, file transfer, and Web transfers are all elastic applications.

Timing

- As transport-layer protocol provide throughput guarantee, they can also provide timing guarantees.
- For example: every bit that the sender pushes into the socket arrives at the receiver's socket no more than 100 msec later.
- All interactive real time applications often demand this guarantee.
- Applications like teleconferencing, real time games, video conferencing are applications to name a few.

Security

- Finally, a transport protocol can provide an application with one or more security services.

- Confidentiality is major factor in this regard and can be achieved through cryptographic measures.
- Sending process encrypts the data and receiver decrypts thereby safeguarding it from adversaries.
- A transport protocol can also provide other security services in addition to confidentiality, services like data integrity and end-point authentication.

Probable Question: ***List and explain the transport layer services available to applications***

2.1.4 Transport Services Provided by the Internet

- As seen earlier, the Internet makes two transport protocols available to applications, UDP and TCP.
- When we plan to create a new network application for the Internet, one of the first decisions you have to make is whether to use UDP or TCP.
- Each of these protocols offers a different set of services to the invoking applications.

TCP Services

The TCP service model includes a connection-oriented service and a reliable data transfer service. When an application invokes TCP as its transport protocol, the application receives both of these services from TCP. They are

- *Connection-oriented service*: TCP has procedure named 2-way/ 3-way handshaking, which provides the exchange of transport layer control information to client and server.

  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, connection is tore down.

- *Reliable data transfer service*: The communicating processes can rely on TCP to deliver all data sent without error and in the proper order.

  In this context, a predefine path will be chosen by the participating network elements based on few criteria. Hence all data gets transmitted in the identified path.

  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.

  Due to the drifts in transmission and reception rate, a kind of bottleneck takes place in the communication, so called congestion.

  TCP also includes a congestion-control mechanism, that throttles a sending process (client or server) when the network is congested between sender and receiver.

UDP Services

- UDP is a no-frills, lightweight transport protocol, providing minimal 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, no guarantee that the message will ever reach the receiving process.
- Since the data packets take different paths to reach destination, they do arrive at the receiving end out of order.
- UDP does not include a congestion-control mechanism, so the sending side of UDP can pump data into the socket at any rate it pleases.

Probable Question: ***Brief on the transport services provided by the internet***

2.1.5 Application-Layer Protocols

It is important that we clarify few doubts to have a better understanding on application layer protocols

- Message structure (Request & Response)?
- Meanings of the various fields in the messages?
- When do the processes send the messages?

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

- The types of messages exchanged (request 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
- Rules for determining when and how a process sends messages and responds to messages

Application layer protocols fall into either of two categories – RFC or Proprietary.

RFC adhering protocols are standard and if a browser developer follows the rules of the HTTP RFC, the browser will be able to retrieve Web pages from any Web server that has also followed the rules of the HTTP RFC.

Many other application-layer protocols are proprietary and intentionally not available in the public domain. Ex: Skype

Let us consider two applications and try to distinguish them via protocols.

- Web application (Client-Server): Protocol is HTTP, other components are standard document format (HTML), Web browsers (Firefox), Web server (VTU).

  Here the role of HTTP is to define the format and sequence of messages to be exchanged between browser and Web server.

- Internet e-mail application: Components include mail servers that contains user mailboxes, mail clients (such as Microsoft Outlook) that allow users to read and create messages, a standard for defining the structure of an e-mail message and application-layer protocol (SMTP) that defines how messages are passed between servers, how messages are passed between servers and mail clients, and how the contents of message headers are to be interpreted.

SMTP is a RFC adhering protocol.

I hope your perspective about application and application layer protocol is quiet rationale now.

2.2 The Web and HTTP

- The killer application WWW (World Wide Web) made internet reachable to everyone on this earth. Users receive what they want, when they want it not like traditional broadcast radio and television, which force users to tune in when the content provider makes the content available.
- WWW made easy for any individual to make information available over the Web—everyone could become a publisher at extremely low cost.
- Hyperlinks and search engines help us navigate through an ocean of Web sites.
- Graphics stimulate our senses. Forms, JavaScript, Java applets, and many other devices enable us to interact with pages and sites.
- And the Web served as a platform for many killer applications emerging after 2003, including YouTube, Gmail, and Facebook.
- Web uses client-server architecture and the server is always on, with a fixed IP address, and it services requests from potentially millions of different browsers.

2.2.1 Overview of HTTP

- HTTP is the heart of Web. It is implemented in two programs: a client program and a server program.
- Both these 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 (also called a document) consists of objects such as an 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 like images, videos, animations that are referenced by the base HTML file with the objects' URLs.
- Each URL has two components: the hostname of the server that houses the object and the object's path name. See the below shown URL:

http://www.someSchool.edu/someDepartment/picture.gif has www.someSchool.edu for a hostname and /someDepartment/ picture.gif for a path name.

- Web browsers (such as Internet Explorer and Firefox) implement the client side of HTTP, in the context of the Web. Web servers, which implement the server side of HTTP, house Web objects, each addressable by a URL.
- 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 (See Figure 2).

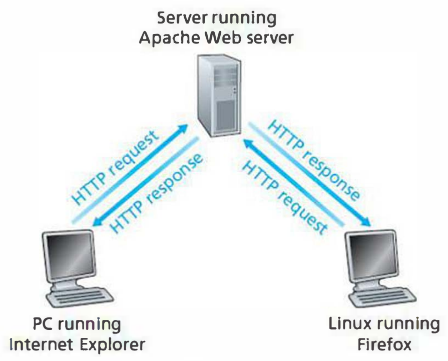


Figure 2.4: Browser – Server interaction via HTTP

- HTTP uses TCP as it is a connection oriented 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.

- On the client side the socket interface is the door between the client process and the TCP connection; on the server side it is the door between the server process and the TCP connection.
- The client sends HTTP request messages into its socket interface and receives HTTP response messages from its socket interface. Similarly, the HTTP server receives request messages from its socket interface and sends response messages into its socket interface.
- Once the client sends a message into its socket interface, the message is out of the client's hands and is "in the hands" of TCP.
- As TCP provides a reliable data transfer service to HTTP, the messages sent by the client as well as the server are assured of guaranteed delivery.
- Hence, HTTP is fully relaxed as the responsibility is more on underlying TCP.

Probable Question: ***Elaborate on the Web and the HTTP.***

2.2.2 Non-Persistent and Persistent Connections

- HTPP works on two kinds of TCP connections – Persistent & Non-Persistent.
- Non - Persistent connections is a way in which each request/response pair will be sent over a separate TCP connection whereas in Persistent, one common connection will be maintained.
- In many Internet applications, the client and server communicate for an extended period of time, with the client making a series of requests and the server responding to each of the requests.
- Depending on the application and on how the application is being used, the series of requests may be made back-to-back, periodically at regular intervals, or intermittently.
- Choosing persistent or not is up to the developer's decision and HTTP uses persistent connection by default.

HTTP with Non-Persistent Connections

- Consider a scenario of transferring a Web page from server to client for the case of non-persistent connections.

- Assume that the page consists of a base HTML file and 10 JPEG images altogether 11 objects reside on the server. Let the URL for the base HTML file be

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.

6. 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.

7. The first four steps are then repeated for each of the referenced JPEG objects.

This means 11 TCP connections are generated (10 for the objects and 1 for the base HTML file). Modern browsers do this job in parallel to reduce the number of connections.

- Let us estimate the amount of time that elapses from when a client requests the base HTML file until the entire file is received by the client.

  We need to understand Round-Trip Time (RTT) before we proceed further.

  RTT is the time it takes for a small packet to travel from client to server and then back to the client.

RTT includes packet-propagation delays, packet queuing delays in intermediate routers and switches, and packet-processing delays.

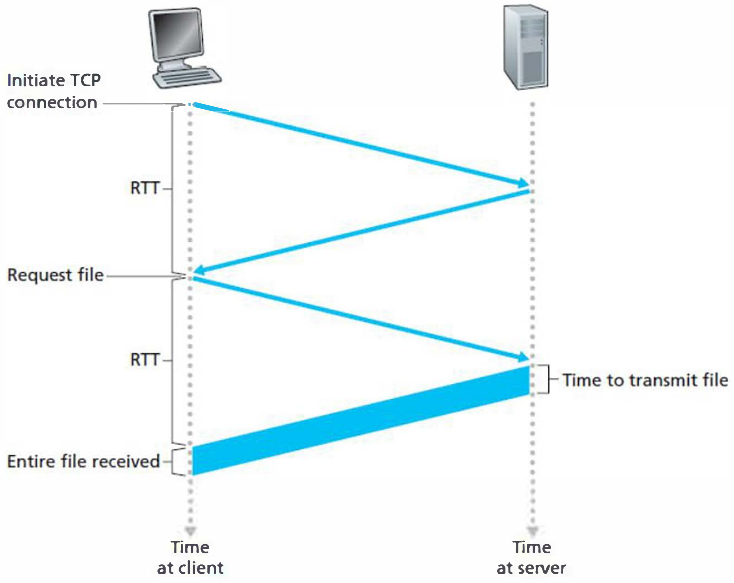
User hits the URL, this causes the browser to initiate a TCP connection between the browser and the Web server; this involves a "three-way handshake"—the client sends a small TCP segment to the server, the server acknowledges and responds with a small TCP segment, and, finally, the client acknowledges back to the server (First three arrows in below figure).

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 for base HTML file plus the acknowledgment into the TCP connection.

The server then sends the HTML file into the TCP connection. This HTTP request/response eats up another RTT.

Total time taken = $RTT_1 + RTT_2$ + the transmission time at the server for the HTML file.



HTTP with Persistent Connections

There are basically two drawbacks with non-persistent connections.

- Every time a client requests some object from web server, a brand-new connection must be established, 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 when there are requests from several clients simultaneously.

- 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 (in the example above, the base HTML file and the 10 images) can be sent over a single persistent TCP connection.
- Here, the HTTP server closes a connection when it is not used for a certain time (a configurable timeout interval).

Probable Question: ***Differentiate between Non - Persistent and Persistent connections with appropriate examples***

What are the steps involved in transferring a Web page from server to client for the case of non-persistent connections.

2.2.3 HTTP Message Format

As we already know, there are two types of HTTP messages, request messages and response messages.

HTTP Request Message

A typical HTTP request message is shown below:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
```

```
Accept-language: fr
```

- The message consists of five lines, each followed by a carriage return and a line feed.
- The last line is followed by an additional carriage return and line feed (to indicate end of header lines).
- 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 GET method is used when the browser requests an object, with the requested object identified in the URL field.
- In the above message, the browser is requesting the object /somedir/page.html.
- The version is self-explanatory; in the above message, the browser implements version HTTP/1.1.
- We can interpret the above message as, "a client has initiated a HTTP request to a sever at *www.someschool.edu* to receive objects and a base HTML file at an URL */somedir/page.html* via non-persistent TCP connection (Connection: close) through client side browser *Mozilla/5.0* with *HTTP version 1.1*in *French language*".

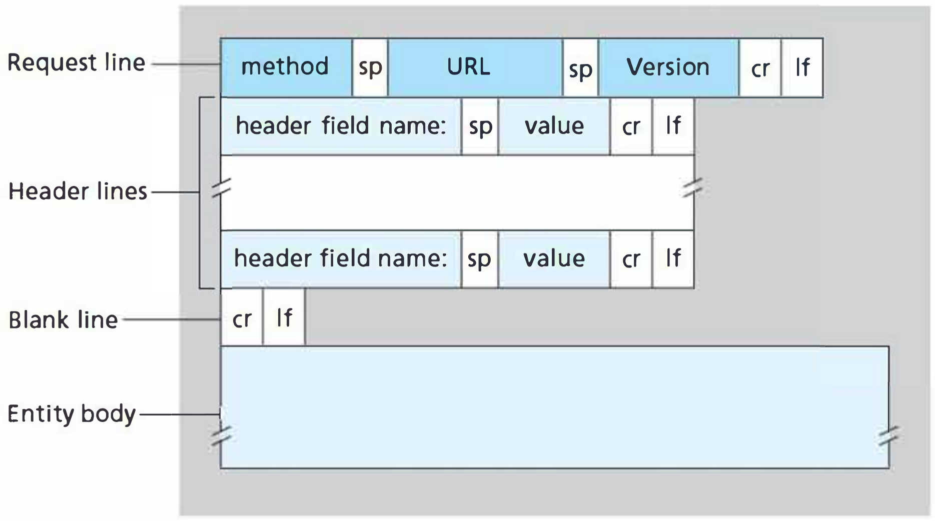


Figure 2.5: General format of HTTP request message

The above diagram depicts general format for HTTP request message. sp: space, cr: carriage return and lf: line feed

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.

The entity body is the meat of the message—it contains the requested object itself (represented by `data data data data data ...`).

Status line: has three fields: the protocol version field, a status code, and a corresponding status message. In the above message, the status line indicates that the server is using HTTP/1.1, status code is 200 and OK is the status message.

Header lines: The example message has 6 header lines.

- *Connection: close header line* to tell the client that it is going to close the TCP connection after sending the message.
- *Date: header line* indicates the time and date when the HTTP response was created and sent by the server. Note that this is not the time when the server retrieves the object from its file system, inserts the object into the response message, and sends the response message.
- *The Server: header line* indicates that the message was generated by an Apache Web server.
- *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.

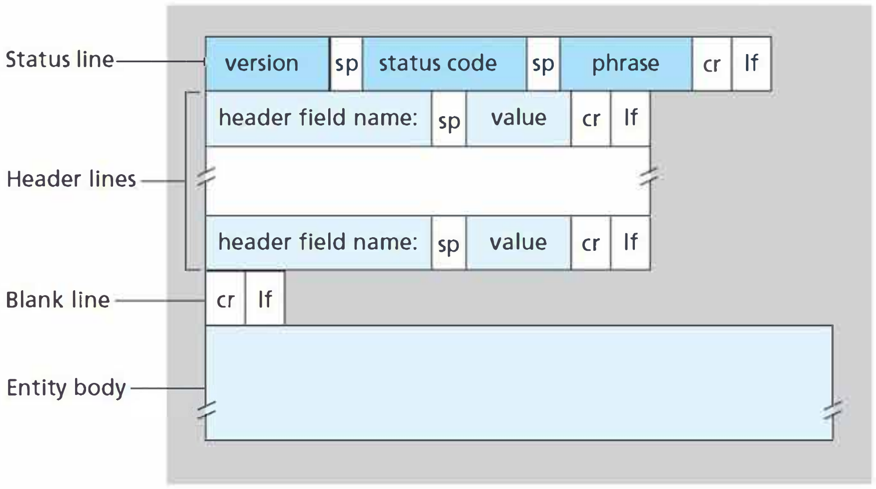


Figure 2.6: General format of a HTTP response message.

Other status code and message:

• 301 Object permanently moved to 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.

Probable Question: ***Explain the general format of HTTP request message***

Explain the general format of HTTP response message

2.2.4 User-Server Interaction: Cookies

- Stateless nature of HTTP has made the web server design easy, that means server maintains no information about clients.
- However, 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. Hence *cookies.*
- Cookies allow sites to keep track of users. Most major commercial Web sites like amazon, ebay, etc., uses cookies today.

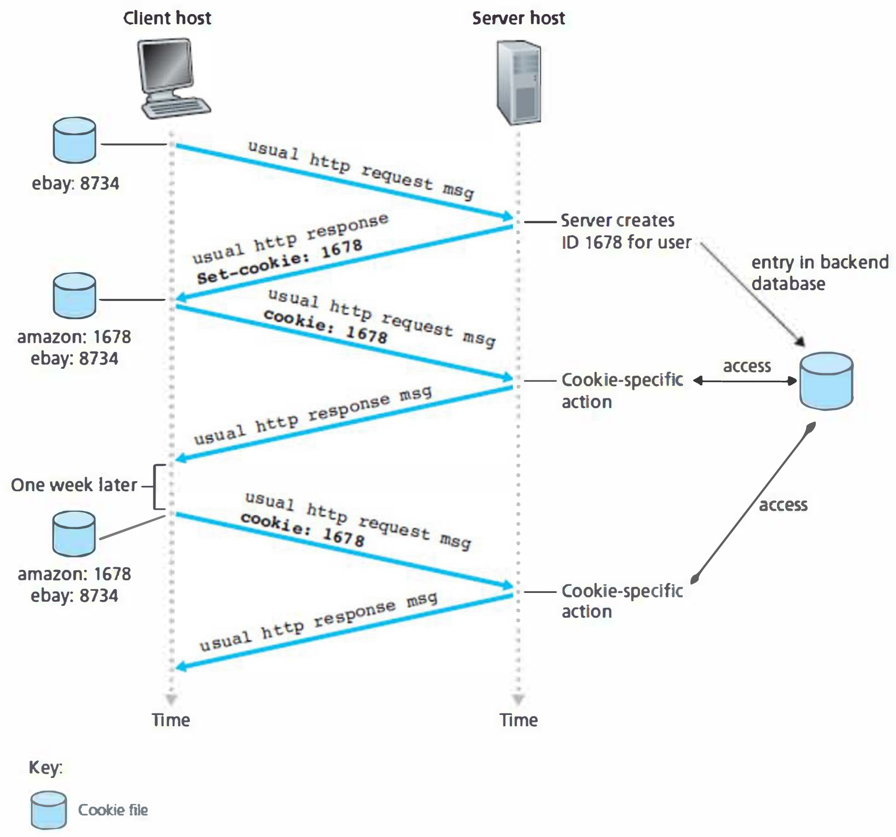


Figure 2.7: Keeping user state with cookies

- As shown in above figure, 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.

- Whenever a client sends request afresh, cookies at the server's end tags the request with an ID (see figure) and saves it in back-end database.
- In response, it inserts ID into header line of HTTP message which will be saved by the client in its cookie file for future perusal.
- Client notices this by seeing setcookie: header. For example, the header line might be:

- Set-cookie: 1678 [1678 is the ID generated for a client]
- Next onwards, server identifies the client via this ID and tracks & records user's browsing pattern.
- This helps server to build suggestion for each and every user.
- Client notices Setcookie: header upon receiving the HTTP response message.
- 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.
- During the subsequent sessions, the browser passes a cookie header to the server, thereby identifying the user to the server.

Probable Question: ***Explain the role of cookies in User – Server interactions***

2.2.5 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.

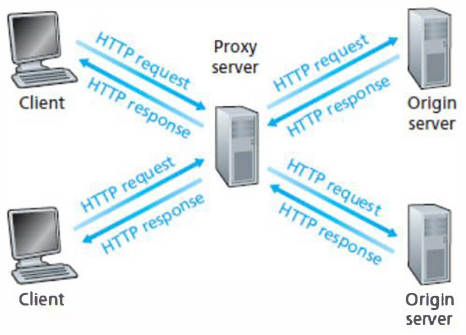


Figure 2.8: Clients requesting objects through a Web cache

- 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. As an example, 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. After receiving this request, the origin server sends the object within an HTTP response to the Web cache.

4. 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).

Note that a cache is both a server and a client at the same time.

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, particularly if the bottleneck bandwidth between the client and the origin server is much less than the bottleneck bandwidth between the client and the cache.
- Web caches can substantially reduce traffic on an institution's access link to the Internet.. Furthermore, Web caches can substantially reduce Web traffic in the Internet as a whole, thereby improving performance for all applications.

Below shown figure has two networks—the institutional network and the rest of the public Internet. The institutional network is a high-speed LAN.

- A router in the institutional network and a router in the Internet are connected by a 15 Mbps link.
- The origin servers are attached to the Internet but are located all over the globe.

A case study:

Avg. Object size = 1Mb

Avg. request rate from the institution's browsers to the origin servers = 15 requests/ sec.

LAN speed = 100Mbps

Assumption: HTTP request messages are negligibly small and thus create no traffic in the networks or in the access link (from institutional router to Internet router).

Response from internet side router to access link router on institutional network = 2 seconds (Internet delay.)

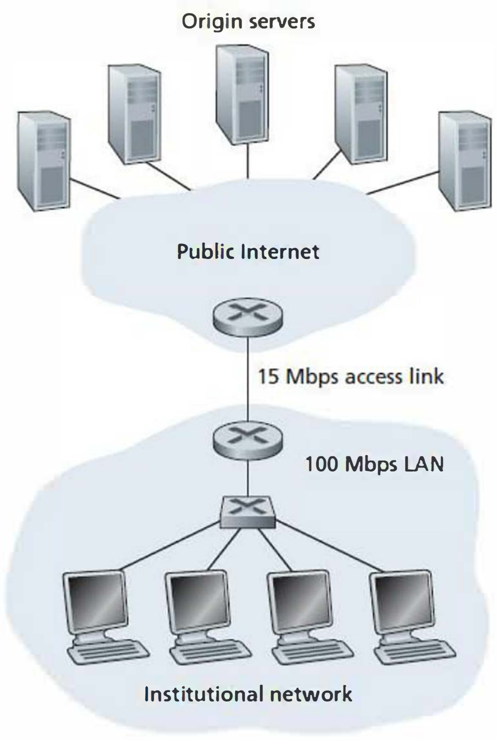


Figure 2.9: Bottleneck between an institutional network and the Internet

The total response time = LAN delay + the access delay (that is, the delay between the two routers) + the Internet delay.

The traffic intensity on the LAN is

(15 requests/sec) * (1 Mbits/request) / (100 Mbps) = 0.15 [negligible]

The traffic intensity on the access link (from the Internet router to institution router) is

(15 requests/sec) * (1 Mbits/request) / (15 Mbps) = 1 [unacceptable]

One possible solution is to increase the access rate from 15 Mbps to, say, 100 Mbps which is costlier

Now consider the alternative solution of not upgrading the access link but instead installing a Web cache in the institutional network.

Below figure illustrates the solution. A web cache has been installed in the institutional network with a good hit rate.

Hit rates—the fraction of requests that are satisfied by a cache—typically range from 0.2 to 0.7 in practice.

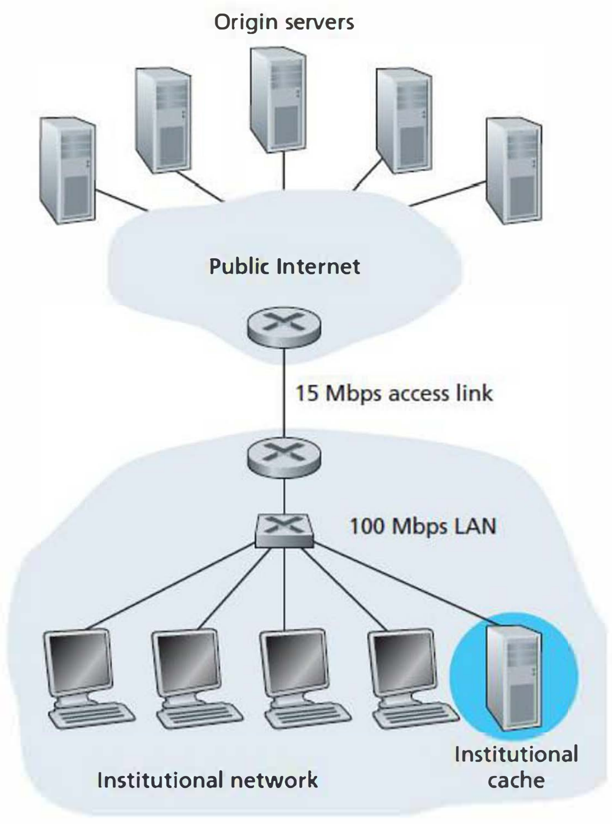


Figure 2.10: Adding a cache to the institutional network

If the rate is 0.4, 40% of the requests will be satisfied almost immediately, say, within 10 milliseconds, by the cache.

Nevertheless, the remaining 60 percent of the requests still need to be satisfied by the origin servers. But with only 60 percent of the requested objects passing through the access link, the traffic intensity on the access link is reduced from 1.0 to 0.6.

This delay is negligible compared with the two second Internet delay. Given these considerations, average delay therefore is

$$0.4 * (0.01 \text{ seconds}) + 0.6 * (2.01 \text{ seconds}) = 1.2 \text{ seconds}$$

which is just slightly greater than 1.2 seconds. Thus, this second solution provides an even lower response time than the first solution, and it doesn't require the institution to upgrade its link to the Internet. The institution does, of course, have to purchase and install a Web cache. But this cost is low—many caches use public-domain software that runs on inexpensive PCs.

Probable Question: ***Explain Web Caching with relevant examples***

2.2.6 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.
- Fortunately, 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.

To illustrate how the conditional GET operates, let's walk through an example.

On 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
```

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. 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
```

Note that the value of the If-modified-since: header line is exactly equal to the value of the Last-Modified: header line that was sent by the server one week ago. 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, 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. Including the requested object would only waste bandwidth and increase user-perceived response time, particularly if the object is large.

Probable Question: ***Describe the concept of Conditional GET***

2.3 File Transfer: FTP

- An FTP session gets initiated when a user requests FTP client through FTP interface for a file located at a remote host.
- Server then asks for his username and password for authorization.
- 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). Below figure depicts the same.

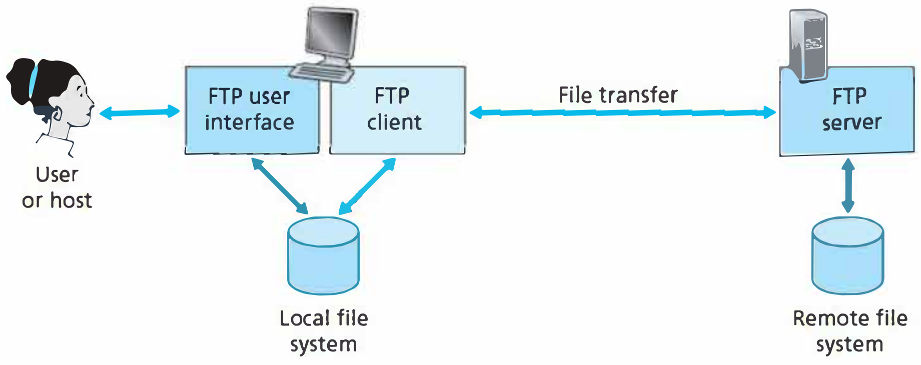


Figure 2.11: FTP moves files between local and remote file systems

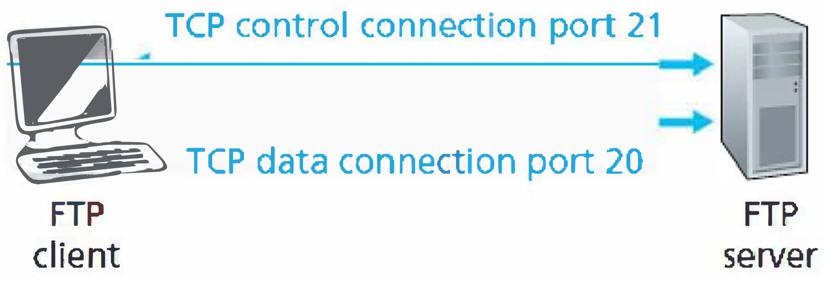


Figure 2.12: Control and data connections

- FTP too runs on top of TCP. Unlike HTTP, 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, commands to change the remote directory over this control connection.
- When the server side receives a command for a file transfer over the control connection, the server side initiates a TCP data connection to the client side.
- FTP dedicates separate data connection for every file transfer and closes after transmission.
- FTP is stateful protocol. Throughout a session, the FTP server must maintain state about the user to track user's current directory as the user wanders about the remote directory tree.

2.3.1 FTP Commands and Replies

The commands, from client to server, and replies, from server to client, are sent across the control connection in 7-bit ASCII format.

In order to delineate successive commands, a carriage return and line feed end each command. Each command consists of four uppercase ASCII characters and a few 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 a file from the current directory of the remote host.
- **STOR** filename: Used to store 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.

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

Probable Question: ***Write a note on FTP (File Transfer Protocol)***

List and explain the FTP commands and replies

2.4 Electronic Mail in the Internet

- Electronic mail was the most popular application when the Internet was in its infancy and has become more and more elaborate and powerful over the years.
- As with ordinary postal mail, e-mail is an asynchronous communication medium—people send and read messages when it is convenient for them, without having to coordinate with other people's schedules.
- In contrast with postal mail, electronic mail is fast, easy to distribute, and inexpensive.
- Modern e-mail has many powerful features, including messages with attachments, hyperlinks, HTML-formatted text, and embedded photos.
- Below figure presents a high-level view of the Internet mail system highlighting three of its major components: user agents, mail servers, and the Simple Mail Transfer Protocol (SMTP).

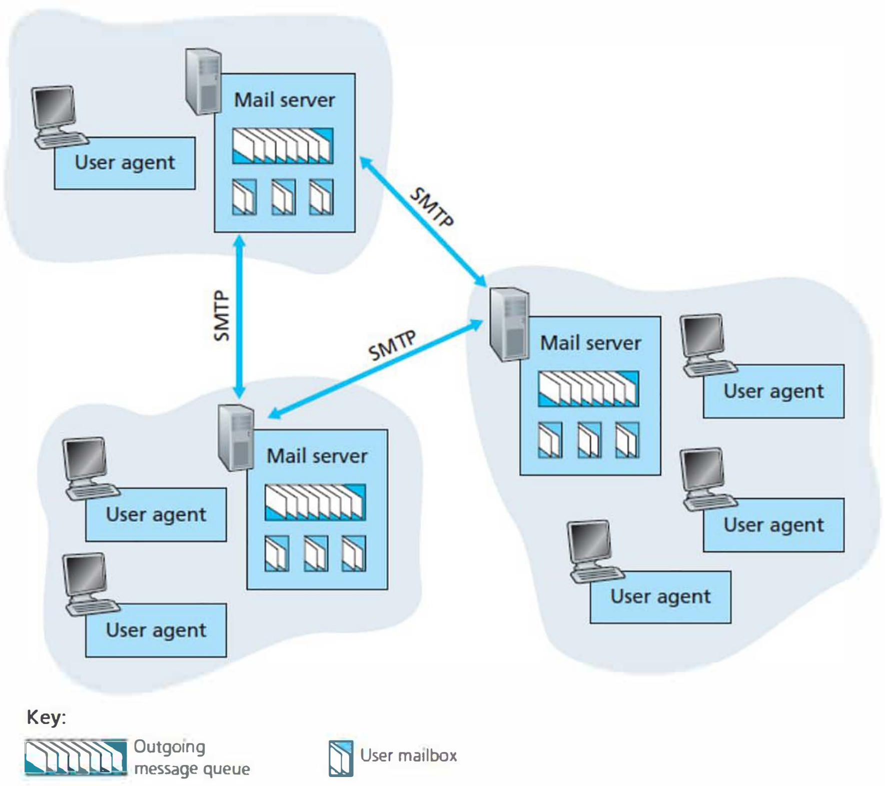


Figure 2.13: A high-level view of the Internet e-mail system

Let us try to understand email system using the above diagram.

- An email system will have a user agent which sends and receives emails for users.
- When a user finishes composing an email, the user agent stores it in outgoing message queue of sender's mail server.
- From sender's mail server, the mail will be directed towards recipient's mail server as the receiver wants to access it.
- Both sender and the receiver must authenticate with the mail server by supplying credentials.
- If sender's server cannot deliver mail to receiver's server, sender'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 with an e-mail message.

- 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.

2.4.1 SMTP

Suppose Raju wants to send Ramu a simple ASCII message.

1. Raju invokes his user agent for e-mail, provides Ramu's e-mail address (ramu@someschool.edu), composes a message, and instructs the user agent to send the message.

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

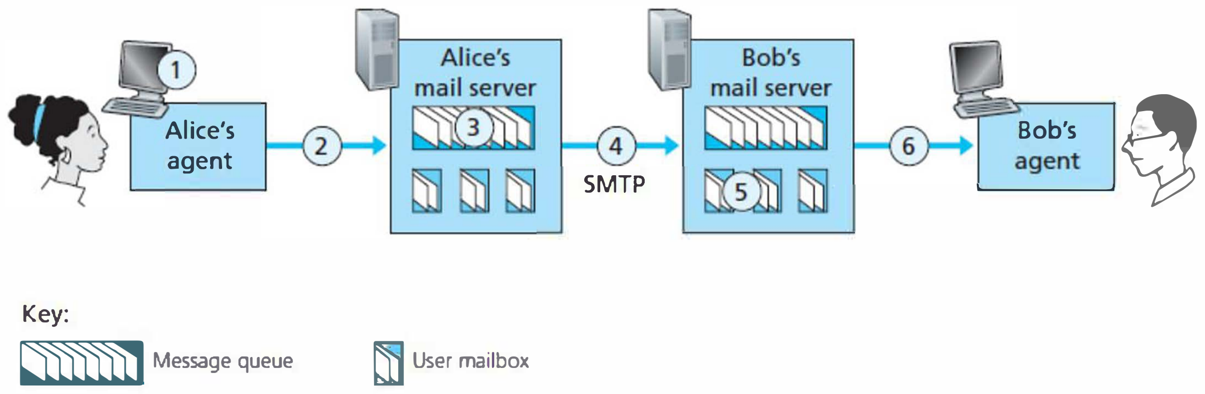
3. The client side of SMTP, running on Raju's mail server, sees the message in the message queue. It opens a TCP connection to an SMTP server, running on Ramu's mail server.

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

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

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

Below shown figure depicts the whole scenario.



Nowhere the mail gets placed in any intermediate servers/ hosts. At times, either it resides in sender's mail server or receiver's.

Let's take a look at an example transcript of messages exchanged between an SMTP client (C) and an SMTP server (S).

The hostname of the client is **"crepes.fr"** and the hostname of the server is "**hamburger.edu"**.

C: Client & S: Server.

The following transcript begins as soon as the TCP connection is established.

```
S: 220 hamburger.edu

C: HELO crepes.fr

S: 250 Hello crepes.fr, pleased to meet you

C: MAIL FROM: <alice@crepes.fr>

S: 250 alice@crepes.fr ... Sender ok

C: RCPT TO: <bob@hamburger.edu>

S: 250 bob@hamburger.edu ... Recipient ok

C: DATA

S: 354 Enter mail, end with "." on a line by itself

C: Do you like ketchup?

C: How about pickles?

C: .

S: 250 Message accepted for delivery

C: QUIT

S: 221 hamburger.edu closing connection
```

- In the example above, the client sends a message ("Do you like ketchup? How about pickles?") from mail server crepes.fr to mail server hamburger. edu. As part of the dialogue, the client issued five commands: HELO (an abbreviation for HELLO), MAIL FROM, RCPT TO, DATA, and QUIT.
- The client also sends a line consisting of a single period, which indicates the end of the message to the server. (In ASCII jargon, each message ends with CRLF.CRLF, where CR and LF stand for carriage return and line feed, respectively.)
- The server issues replies to each command, with each reply having a reply code and some (optional) English-language explanation.
- For each message, the client begins the process with a new MAIL FROM: crepes.fr, designates the end of message with an isolated period, and issues QUIT only after all messages have been sent.

2.4.2 Comparison with HTTP

- HTTP transfers files (also called objects) from a Web server to a Web client (typically a browser); SMTP transfers files (that is, e-mail messages) from one mail server to another mail server.
- HTTP is mainly a pull protocol—someone loads information on a Web server and users use HTTP to pull the information from the server at their convenience. In particular, the TCP connection is initiated by the machine that wants to receive the file.
- On the other hand, SMTP is primarily a push protocol—the sending mail server pushes the file to the receiving mail server.
- SMTP requires each message, including the body of each message, to be in 7-bit ASCII format. If the message contains characters that are not 7-bit ASCII (for example, French characters with accents) or contains binary data (such as an image file), then the message has to be encoded into 7-bit ASCII. HTTP data does not impose this restriction.
- HTTP encapsulates each object in its own HTTP response message. Internet mail places all of the message's objects into one message.

2.4.3 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 itself.

This peripheral information is contained in a series of header lines namely From: header line and a To: header line; a header may include a Subject: header line as well as other optional header lines.

The header lines and the body of the message are separated by a blank line (that is, by CRLF).

As with HTTP, each header line contains readable text, consisting of a keyword followed by a colon followed by a value.

A typical message header looks like this:

```
From: alice@crepes.fr

To: bob@hamburger.edu

Subject: Searching for the meaning of life.
```

Probable Question: ***Describe email system in the internet***

Compare SMTP with HTTP. Bring out few salient points

Write a note on Mail Message Format

2.4.4 Mail Access Protocols

- Today, mail accessing paradigm has evolved. Mail access uses a client-server architecture—the typical user reads e-mail with a client that executes on the user's end system, for example, on an office PC, a laptop, or a smartphone.
- By executing a mail client on a local PC, users enjoy a rich set of features, including the ability to view multimedia messages and attachments.
- User has his mailbox mounted virtually onto his own laptop/ PC.

- But there is a problem with this approach i.e., if an user's mail server were to reside on his local PC, then his PC would have to remain always on, and connected to the Internet, in order to receive new mail, which can arrive at any time. This is impractical for many Internet users.
- Instead, a typical user runs a user agent on the local PC but accesses its mailbox stored on an always-on shared mail server. This mail server is shared with other users and is typically maintained by the user's ISP (for example, university or company).
- As per the previous description, the email sent from a user's end will first sit on his mail server and then it will be pushed onto recipient's mail server by the SMTP.
- Why the two-step procedure? How does a recipient running a user agent on his local PC, obtain his messages, which are sitting in a mail server within his ISP?
- User agent at an end system can't use SMTP to obtain the messages because obtaining the messages is a pull operation, whereas SMTP is a push protocol.
- This can be solved by introducing a special mail access protocol that transfers messages from user's mail server to his local PC. These are known as 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.
- A mail access protocol, such as POP3, is used to transfer mail from the recipient's mail server to the recipient's user agent.

POP3

- POP3 is an extremely simple mail access protocol. 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.

Authorization: the user agent sends a username and a password (in the clear) to authenticate the user.

Transaction: the user agent retrieves messages and does some management tasks.

Update: client issues the quit command, ending the POP3 session.

- 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 (sometimes followed by server-to-client data), used by the server to indicate that the previous command was fine

-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>.
- If you Telnet directly into a POP3 server, using port 110, and issue these commands. Suppose that mailServer is the name of your mail server.

You will see something like:

```
telnet mailServer 110 [command issued]

+OK POP3 server ready

user bob

+OK

pass hungry

+OK user successfully logged on
```

- If you misspell a command, the POP3 server will reply with an -ERR message.
- Now let's take a look at the transaction phase. A user agent using POP3 can often be configured to perform either of "download and delete" or to "download and keep" modes.

- In the download-and-delete mode, the user agent will issue the list, retr, and dele commands.

Consider a scenario in which the user has two messages in his or her mailbox and the dialogue would go as below:

C: (standing for client) is the user agent and S: (standing for server) is the mail server.

```
C: list           //user agent asks the mail servers to list the size of mails

S: 1 498                //mail server lists mail1: 498

S: 2 912                //mail server lists mail2: 912

S: .                    //end

C: retr 1               //client says retrieve the first message

S: (blah blah ...             //server retrieves

S: .................

S: ..........blah)

S: .                    //end of message 1

C: dele 1               //client requests for deleting mail 1

C: retr 2

S: (blah blah ...

S: .................

S: ..........blah)

S: .

C: dele 2

C: quit                       //End of POP3 session
```

```
S: +OK POP3 server signing off
```

- A problem with this download-and-delete mode is that the recipient, may be nomadic and may want to access his mail messages from multiple machines, for example, his office PC, his home PC, and his portable computer.
- But 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; he can access a message from work and access it again later in the week from home.

IMAP

- With POP3, a user cannot create private copies of mail messages for future access from different machines.
- Though there is an option "download-and-keep", messages will be stored in any one of the machines used by the user.
- There is no option in POP3 for users to create folders in remote server and dump the messages for future use.
- To solve this and other problems, the IMAP mail access protocol was invented.
- 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 one part of a multipart MIME message. This feature is useful when there is a low-bandwidth connection.

Web-Based E-Mail

- Here, the user agent is an ordinary Web browser, and the user communicates with its remote mailbox via HTTP.
- When a recipient, such as Bob, wants to access a message in his mailbox, the e-mail message is sent from Bob's mail server to Bob's browser using the HTTP protocol rather than the POP3 or IMAP protocol.
- HTTP rather than over SMTP.

Probable Question: ***What are mail access protocols? Describe any one (POP3 or IMAP)***

What is the drawback in POP3 and how it has been overcome by IMAP? Explain

2.5 DNS—The Internet's Directory Service

- Mnemonics are easy to remember than any other code based identifiers. For example, remembering name of a person is easier than remembering his mobile number.
- Can you imagine saying, "Hi. My name is 132-67-9875. Please meet my father, 178-87-1146.
- Hosts can also be identified with a hostname. Hostnames—such as cnn.com, www.yahoo.com, gaia.cs.umass.edu, and cis.poly.edu—are mnemonic and are therefore appreciated by humans.
- However, hostnames provide little, if any, information about the location within the Internet of the host. (A hostname such as www.eurecom.fr, which ends with the country code .fr, tells us that the host is probably in France, but doesn't say much more.)
- Because, routers find it difficult to decode variable length hostnames, hosts are also identified by so-called IP addresses as routers prefer fixed-length, hierarchically structured data.

- An IP address consists of four bytes and has a rigid hierarchical structure. An IP address looks like 121.7.106.83, where each period separates one of the bytes expressed in decimal notation from 0 to 255.

2.5.1 Services Provided by DNS

- In order to incorporate the above idea, a directory service that translates hostnames to IP addresses is essential.
- This is the main task of the Internet's domain name system (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. The DNS protocol runs over UDP and uses port 53.

- DNS is commonly employed by other application-layer protocols—including HTTP, SMTP, and FTP—to translate user-supplied hostnames to IP addresses.
- Following tasks get executed when a user hits an address (e.g., www.someschool.edu) on his machine:

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.

5. 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.

- To reduce delay, the desired IP address is often cached in a "nearby" DNS server.

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

  • ***Host aliasing***. A host with a complicated hostname like relayl .west-coast.enterprise.com can have one or more alias names such as enterprise.com and www.enterprise.com. In this case, the hostname relay1.westcoast.enterprise.com is said to be a canonical hostname.

  • ***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. Some servers do provide complicated and less mnemonic mail addresses. Preferably, the host name and email address must have same aliases like bob@hotmail.com and www.hotmail.com.

  • ***Load distribution***. Some busy servers like cnn.com, will undergo server replication in order to respond to client queries. In this case, multiple cnn servers with the same hostname can have different IP addresses. 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.

DNS rotation is also used for e-mail so that multiple mail servers can have the same alias name.

2.5.2 Overview of How DNS Works

- The application that wants hostname-to-IP address translation would first invoke a method: gethostbyname().
- 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, 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.
- DNS is a black box providing a simple, straightforward translation service.

- But in fact, the black box that implements the service is complex, consisting of a large number of DNS servers distributed around the globe.
- A simple design for DNS would have one DNS server that contains all the mappings. But there are problems with a centralized design:

  • ***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. If the centralized DNS is geographically too far, 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.

Probable Question: ***Explain the services provided by DNS***

Give an overview of DNS working

A Distributed, Hierarchical Database

- In order to deal with the issue of scalability, the DNS uses a large number of servers, organized in a hierarchical fashion and distributed around the world.
- Hierarchy has three classes of DNS servers—root DNS servers, top-level domain (TLD) DNS servers, and authoritative DNS servers (see below figure).

Suppose a DNS client wants to determine the IP address for the hostname www.amazon.com, it first contacts one of the root servers, which returns IP addresses for TLD servers for the top-level domain com. The client then contacts one of these TLD servers, which returns the IP address of an authoritative server for amazon.com. Finally, the client contacts one of the authoritative servers for amazon.com, which returns the IP address for the hostname www.amazon.com.

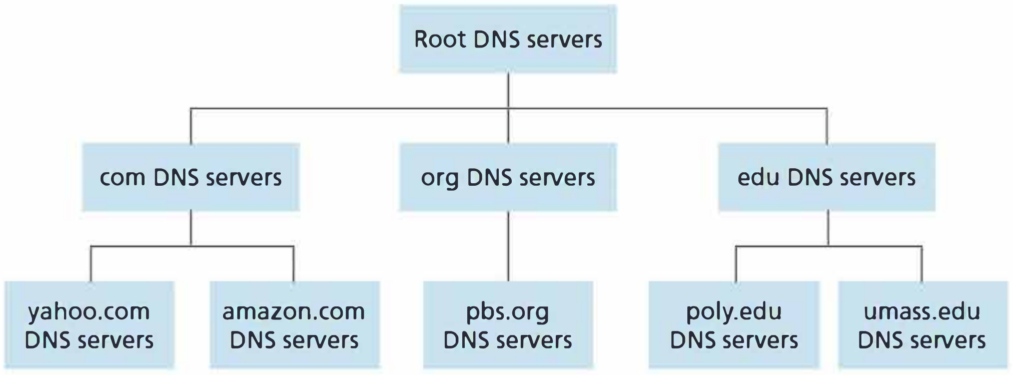


Figure 2.14: Portion of the hierarchy of DNS servers

Let's first take a closer look at these three classes of 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. Altogether, there are 247 root servers as of 2011.

• ***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.

• ***Authoritative DNS servers***. An organization's authoritative DNS server houses publicly accessible DNS records that map the names of their hosts to IP addresses. An organization can choose to implement its own authoritative DNS server to hold these records; alternatively, the organization can pay to have these records stored in an authoritative DNS server of some service provider.

- 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 (also called a default name server, ex: our mail server).

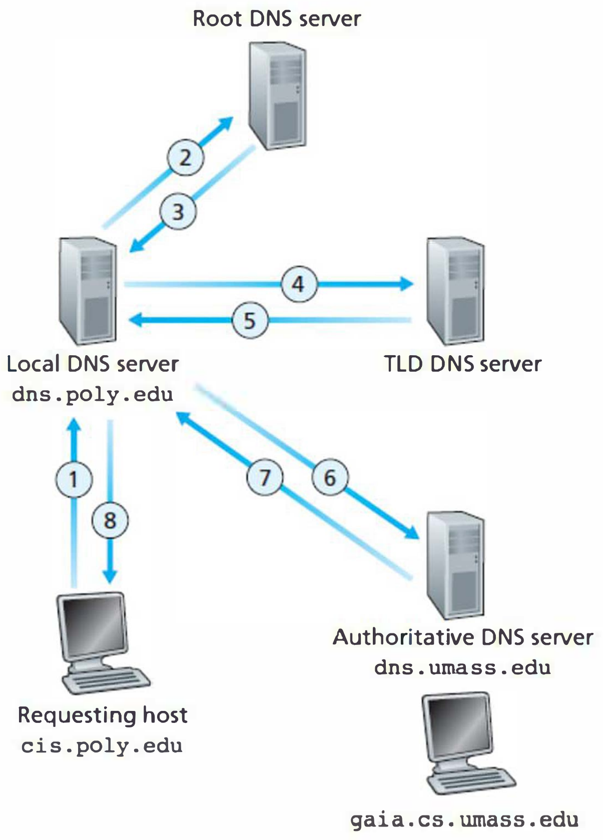


Figure 2.15: Interaction of the various DNS servers

- As shown above, the host cis.poly.edu first sends a DNS query message to its local DNS server, dns.poly.edu.
- The query message contains the hostname to be translated, namely, gaia.cs.umass.edu.
- The local DNS server forwards the query message to a root DNS server.
- The root DNS server takes note of the edu suffix and returns to the local DNS server a list of IP addresses for TLD servers responsible for edu.
- The local DNS server then resends the query message to one of these TLD servers.
- The TLD server takes note of the umass.edu suffix and responds with the IP address of the authoritative DNS server for the University of Massachusetts, namely, dns.umass.edu.

- Finally, the local DNS server resends the query message directly to dns.umass.edu, which responds with the IP address of gaia.cs.umass.edu. Note that in this example, in order to obtain the mapping for one hostname, eight DNS messages were sent: four query messages and four reply messages!

The example shown in figure 2.15 makes use of both recursive queries and iterative queries.

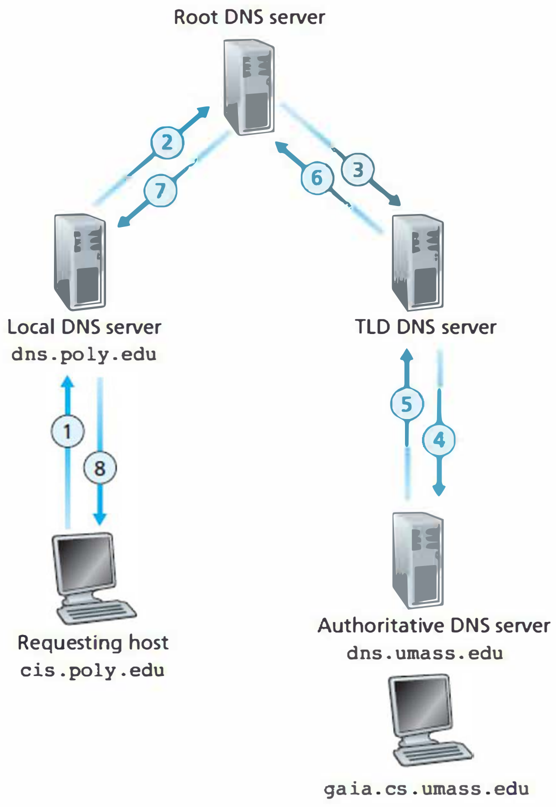


Figure 2.16: Recursive queries in DNS

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. In theory, any DNS query can be iterative or recursive. For example, Figure 2.16 shows a DNS query chain for which all of the queries are recursive. In practice, the queries typically follow the pattern in Figure 2.15: The query from the requesting host to the local DNS server is recursive, and the remaining queries are iterative.

Probable Question: ***List the DNS Server classes. Explain how interaction takes place among them***

DNS Caching

- DNS caching like web caching (discussed earlier), improves the delay performance and reduces the number of DNS messages wandering around the Internet.
- The idea behind DNS caching is very simple. In a query chain, when a DNS server receives a DNS reply (containing, for example, a mapping from a hostname to an IP address), it can cache the mapping in its local memory.
- This makes any authoritative DNS server to provide with the desired IP address if it has cached information.
- 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).

2.5.3 DNS Records and Messages

- DNS database store a kind of information called as Resource Records (RRs). RRs provide hostname to IP address mappings. Each DNS reply message carries one or more resource records.

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. 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.

• 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.

- MX records allow the hostnames of mail servers to have simple aliases. With MX records, a company can have the same aliased name for its mail server and for one of its other servers (such as its Web server).
- If a DNS server is authoritative for a particular hostname, then the DNS server will contain a Type A record for the hostname.
- If a server is not authoritative for a hostname, then the server will contain a Type NS record for the domain that includes the hostname.

As an example, suppose an edu TLD server is not authoritative for the host gaia.cs.umass.edu. Then this server will contain a record for a domain that includes the host gaia.cs.umass.edu, for example, (umass.edu, dns.umass.edu, NS). The edu TLD server would also contain a Type A record, which maps the DNS server dns.umass.edu to an IP address, for example, (dns.umass.edu, 128.119.40.111, A).

DNS Messages

DNS messages can be DNS query and reply messages. Format is same for both message types and is shown below.

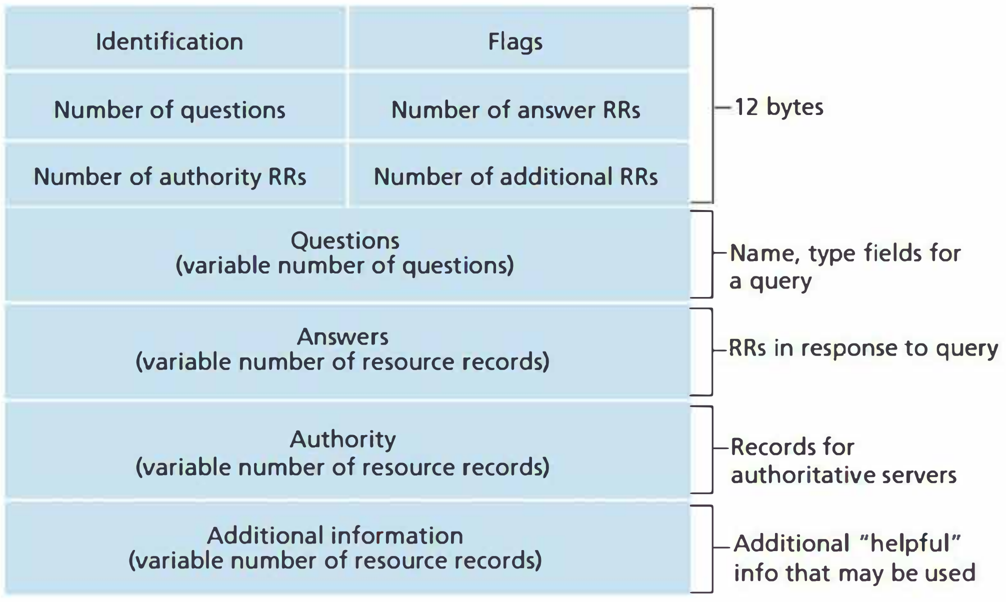

Identification
Flags
Number of questions
Number of answer RRs
Number of authority RRs
Number of additional RRs
12 bytes
Questions (variable number of questions)
Name, type fields for a query
Answers (variable number of resource records)
RRs in response to query
Authority (variable number of resource records)
Records for authoritative servers
Additional information (variable number of resource records)
Additional "helpful" info that may be used

Figure 2.17: DNS message format

The semantics of the various fields in a DNS message are as follows:

• **Identification:** 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.

• **Flags: (a)** A 1-bit ***query/reply*** flag indicates whether the message is a query (0) or a reply (1).

(b) A1-bit ***authoritative flag*** is set in a reply message when a DNS server is an authoritative server for a queried name.

(c) 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.

(d) 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.

- 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—for example, a host address associated with a name (Type A) or the mail server for a name (Type MX).

- In a reply from a DNS server, the answer section contains the resource records (for example, A, NS, CNAME, and MX), the Value, and the TTL. A reply can return multiple RRs in the answer, since a hostname can have multiple IP addresses (for example, for replicated Web servers, as discussed earlier in this section).
- The authority section contains records of other authoritative servers.
- The additional section contains other helpful records.

Probable Question: ***Explain DNS Record Resource.***

Explain the DNS message format with a neat diagram

Inserting Records into the DNS Database

- Suppose you have just created a startup company called Network Utopia.
- You would want people around the world to see your company over the internet.
- Hence, your company domain name networkutopia.com has to be registered with a registrar. A registrar is a commercial entity that verifies the uniqueness of the domain name, enters the domain name into the DNS database and collects a small fee from you for its services.
- Prior to 1999, a single registrar, Network Solutions, had a monopoly on domain name registration for com, net, and org domains. But now there are many registrars competing for customers, and the Internet Corporation for Assigned Names and Numbers (ICANN) accredits the various registrars.
- After this, you need to provide the registrar with the names and IP addresses of your primary and secondary authoritative DNS servers (e.g., dnsl.networkutopia.com, dns2.networkutopia.com, 212.212.212.1, and 212.212.212.2).
- For each of these two authoritative DNS servers, the registrar would then make sure that a Type NS and a Type A record are entered into the TLD **com** servers.

- Specifically, 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)

- You'll also have to make sure that the Type A resource record for your Web server www.networkutopia.com and the Type MX resource record for your mail server mail.networkutopia.com are entered into your authoritative DNS servers.
- Once all of these steps are completed, people will be able to visit your Web site and send e-mail to the employees at your company.

Probable Question: ***What is the way for inserting records into DNS Database? Elaborate***

2.6 Peer-to-Peer Applications

- In a P2P architecture, pairs of intermittently connected hosts, called peers, communicate directly with each other.
- The peers are not owned by a service provider, but are instead desktops and laptops controlled by users.
- Let us examine two different applications based on P2P architecture. File distribution, where the application distributes a file from a single source to a large number of peers (e.g., Bittorrent).
- Here one can notice the self-scalability characteristic of P2P architectures.
- The second P2P application we'll examine is a database distributed over a large community of peers (e.g., Distributed Hash Table (DHT)).

2.6.1 P2P File Distribution

- The file might be a new version of the Linux operating system, a software patch for an existing operating system or application, an MP3 music file, or an MPEG video file.

- In client-server file distribution, the server must send a copy of the file to each of the peers—placing an enormous burden on the server and consuming a large amount of server bandwidth.
- 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.

Scalability of P2P Architectures

- Let's examine self-scalability by considering a simple quantitative model for distributing a file to a fixed set of peers for both architecture types.

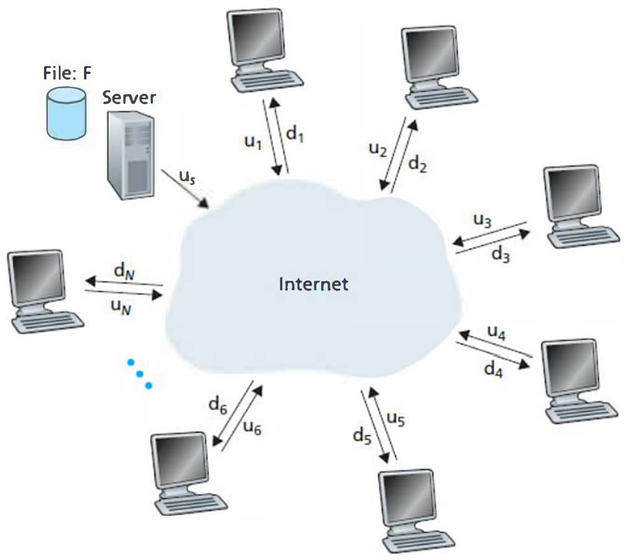


Figure 2.18: An illustrative file distribution problem

As shown in above figure, the server and the peers are connected to the Internet with access links.

u_s : be upload rate of the server's access link,

u_i : the upload rate of the i^{th} peer's access link,

d_i : the download rate of the i^{th} peer's access link

F: size of the file to be distributed (in bits)

N: the number of peers that want to obtain a copy of the file.

Let's first determine the distribution time for the client-server architecture, which we denote by D_{cs}. 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 u_s, the time to distribute the file must be at least NF/u_s.

• Let d_{min} denote the download rate of the peer with the lowest download rate, that is,

$d_{min} = \min\{d_1, d_p, \ldots, d_N\}$.

The peer with the lowest download rate cannot obtain all F bits of the file in less than F/d_{min} seconds. Thus the minimum distribution time is at least F/d_{min}.

Putting these two observations together, we obtain

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

This provides a lower bound on the minimum distribution time for the client-server architecture. In the homework problems you will be asked to show that the server can schedule its transmissions so that the lower bound is actually achieved. So let's take this lower bound provided above as the actual distribution time, that is,

$$D_{cs} = \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\} \qquad (2.1)$$

We see from Equation 2.1 that for N large enough, the client-server distribution time is given by NF/u_s. Thus, the distribution time increases linearly with the number of peers N.

Let's now go through a similar analysis for the P2P architecture, where each peer can assist the server in distributing the file. In particular, when a peer receives some file data, it can use its own upload capacity to redistribute the data to other 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.

Nevertheless, a simple expression for the minimal distribution time can be obtained [Kumar 2006]. To this end, we first 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/u_s. (Unlike the client-server scheme, a bit sent once by the server may not have to be sent by the server again, as the peers may redistribute the bit among themselves.)

• 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/d_{min} seconds. Thus the minimum distribution time is at least F/d_{min}.

• 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,

$u_{total} = u_s + u_1 + \dots + u_N$.

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 u_{total}. Thus, the minimum distribution time is also at least

$NF/(u_s + u_1 + \dots + u_N)$.

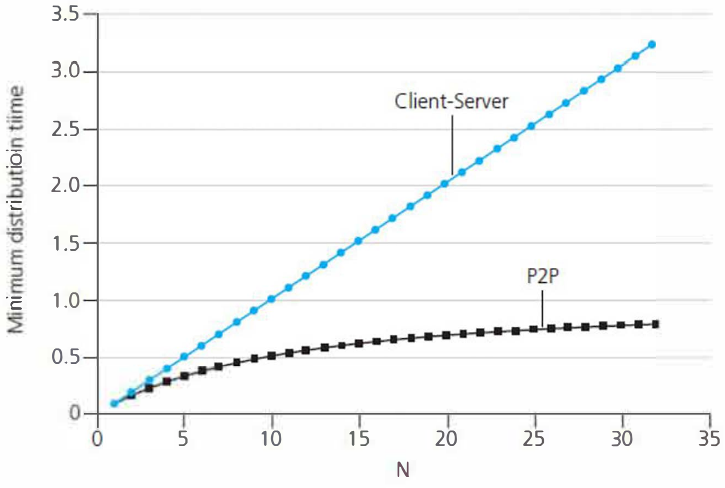
Putting these three observations together, we obtain the minimum distribution time for P2P, denoted by DP2P.

$$D_{P2P} \geq \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \tag{2.2}$$

Equation 2.2 provides a lower bound for the minimum distribution time for the P2P architecture. It turns out that if we imagine that each peer can redistribute a bit as soon as it receives the bit, then there is a redistribution scheme that actually achieves this lower bound [Kumar 2006]. (We will prove a special case of this result in the homework.) In reality, where chunks of the file are redistributed rather than individual bits, Equation 2.2 serves as a good approximation of the actual minimum distribution time. Thus, let's take the lower bound provided by Equation 2.2 as the actual minimum distribution time, that is,

$$D_{P2P} = \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \tag{2.3}$$

Figure 2.25 compares the minimum distribution time for the client-server and P2P architectures assuming that all peers have the same upload rate u. In Figure 2.25, we have set F/u = 1 hour, u_s = 10u, and $d_{min} \geq$ us. Thus, a peer can transmit the entire file in one hour, the server transmission rate is 10 times the peer upload rate, and (for simplicity) the peer download rates are set large enough so as not to have an effect. We see from Figure 2.25 that for the client-server architecture, the distribution time increases linearly and without bound as the number of peers increases. However, for the P2P architecture, the minimal distribution time is not only always less than the distribution time of the client-server architecture; it is also less than one hour for any number of peers N. Thus, applications with the P2P architecture can be self-scaling. This scalability is a direct consequence of peers being redistributors as well as consumers of bits.



Probable Question: ***Explain scalability factor of P2P Architecture with proper substances***

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 from other peers and also contributes chunks to them.
- 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.
- Any peer may leave the torrent at any time with only a subset of chunks, and later rejoin the torrent.
- In Bittorrent, each torrent has a node called tracker to which every peers register with and periodically informs the tracker that it is still in the torrent.

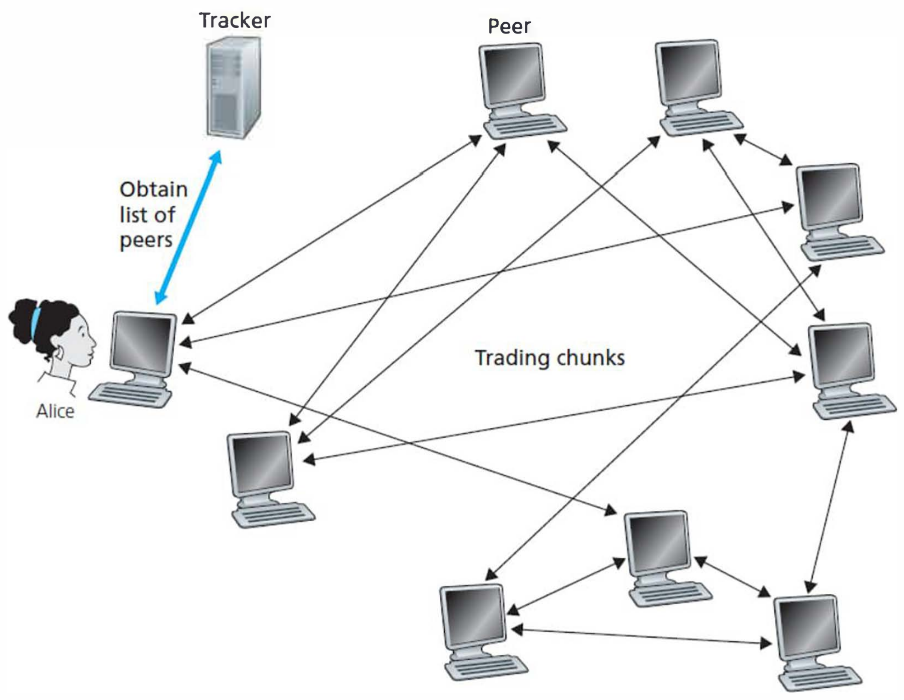


Figure 2.19: File distribution with BitTorrent

- As shown above, 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.

- New peer attempts to establish concurrent TCP connections with all the peers on this list, now they are referred to 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 with this peer.
- If a peer has L different neighbors, it will obtain L lists of chunks.
- This peer will have two important decisions to make. First, which chunks should she request first from her neighbors? And second, to which of her neighbors should she send requested chunks?

Solution for the first, is a technique called **rarest first**. The idea is to determine, from among the chunks she does not have, the chunks that are the rarest among her neighbors (that is, the chunks that have the fewest repeated copies among her neighbors) and then request those rarest chunks first.

Solution to the next is another technique, **a clever trading algorithm**. The basic idea is that the peer gives priority to the neighbors that are currently supplying her data at the highest rate. Specifically, for each of her neighbors, the peer continually measures the rate at which it receives bits and determines the four peers that are feeding bits to it at the highest rate. It then reciprocates by sending chunks to these same four peers. Every 10 seconds, it recalculates the rates and possibly modifies the set of four peers.

Read further to know more….

In BitTorrent lingo, these four peers are said to be unchoked. Importantly, every 30 seconds, she also picks one additional neighbor at random and sends it chunks. Let's call the randomly chosen peer Bob. In BitTorrent lingo, Bob is said to be optimistically unchoked. Because Alice is sending data to Bob, she may become one of Bob's top four uploaders, in which case Bob would start to send data to Alice. If the rate at which Bob sends data to Alice is high enough, Bob could then, in turn, become one of Alice's top four uploaders. In other words, every 30 seconds, Alice will randomly choose a new trading partner and initiate trading with that partner. If the two peers are satisfied with the trading, they will put each other in their top four lists and continue trading with each other until one of the peers finds a better partner. All other neighboring peers besides these *five peers (four "top" peers and one probing peer) are "choked," that is, they do not receive any chunks from Alice.*

Probable Question: ***Explain how file distribution takes place with BitTorrent. Also comment on the hurdles in it.***

2.6.2 Distributed Hash Tables (DHTs)

- Consider a centralized version of this simple database, which will simply contain (key, value) pairs.
- For example, the keys could be social security numbers and the values could be the corresponding human names; in this case, an example key-value pair is (156-45-7081, Johnny Wu). Or the keys could be content names (e.g., names of movies, albums, and software), and the value could be the IP address at which the content is stored; in this case, an example key-value pair is (Led Zeppelin IV, 128.17.123.38).
- We query the database with a key and it returns the associated value in the form of contents.
- Building such a database is straightforward with a client-server architecture that stores all the (key, value) pairs in one central server.
- In the P2P system, each peer will only hold a small subset of the totality of the (key, value) pairs. If a peer queries for a key, distributed database will then locate the peers that have the corresponding (key, value) pairs and return.
- 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).

Let us take an example before we ought to learn about creating a DHT.

Here, a key is the content name (e.g., Linux) and the value is the IP address of a peer that has a copy of the content.

So, if Bob and Charlie each have a copy of the latest Linux distribution, then the DHT database will include the following two key-value pairs: (Linux, IP_{Bob}) and (Linux, $IP_{Charlie}$).

Since the DHT database is distributed over the peers, some peer, will have the key value pair, using which the querying peer can reach the source.

Now suppose Alice wants to obtain a copy of Linux. She first needs to know which peers have a copy of Linux before she can begin to download it.

She queries the DHT with "Linux" as the key. The DHT then determines that (some peer) Dave is responsible for the key "Linux."

The DHT then contacts peer Dave, obtains from Dave the key-value pairs (Linux, IP_{Bob}) and (Linux, $IP_{Charlie}$), and passes them on to Alice.

Alice can then download the latest Linux distribution from either IP_{Bob} or $IP_{Charlie}$.

For creating a DHT to address problems in general, a simple approach 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.

- So 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.
- But this approach is completely un-scalable, as it would require each peer to not only know about all other peers, but even worse, have each query sent to *all* peers.
- An elegant approach would be 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* represented using some bits.
- Let's 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].
- A hash function is a many-to-one function for which two different inputs can have the same output (same integer).
- The hash function is assumed to be available to all peers in the system.

For example, if the original key is "ABCXYZ," the key used in the DHT will be the integer that equals the hash of "ABCXYZ."

- Let's now consider the 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, we'll need to define what is meant by ***"closest",*** for which many conventions are possible.
- For convenience, let's define the closest peer as the ***closest successor of the key***. Suppose n=4 so that all the peer and key identifiers are in the range [0, 15].
- If there are 8 peers in the system with identifiers 1, 3, 4, 5, 8, 10, 12, and 15 and if we want to store the (key, value) pair e.g., (11, Johnny Wu) in one of the 8 peers using our closest convention, it will be the peer with Id 12 since peer 12 is the closest successor for key 11.
- Definition of **closest**: if the key is exactly equal to one of the peer identifiers, we store the (key, value) pair in that matching peer and if the key is larger than all the peer identifiers, we use a modulo-2^n convention, storing the (key, value) pair in the peer with the smallest identifier.

Read further to know more….

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.

But how does Alice determine the peer that is closest to the key? 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.

Probable Question: ***Explain the concept of Distributed Hash Tables***

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).
- Below shown figure depicts an example with n=4 and there are the same 8 peers from the previous example. 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.
- 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.

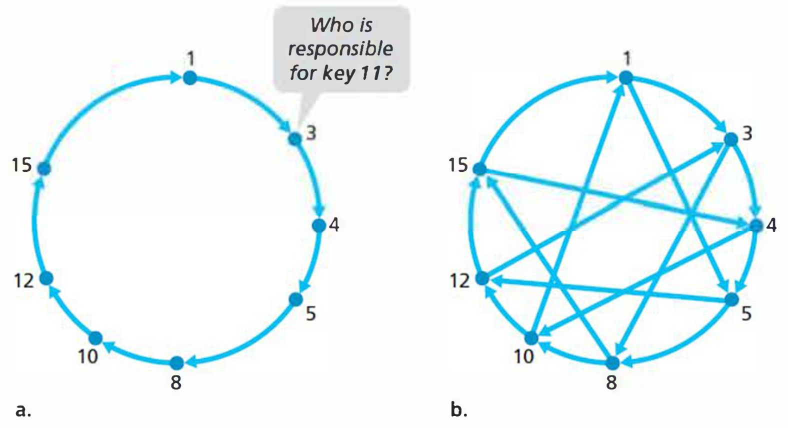


Figure 2.20: (a) A circular DHT. Peer 3 wants to determine who is responsible for key 11. (b) A circular DHT with shortcuts

- The links in an overlay network are not physical links, but are simply virtual liaisons between pairs of peers.
- 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.

Previous problem is solved as each peer needs only to be aware of two peers, its immediate successor and its immediate predecessor.

- But this solution introduces yet a new problem. In the worst case, all N nodes in the DHT will have to forward a message around the circle that figures up to ***N/2*** messages on average.
- A better solution is to upgrade circular DHT with some extra peers reached via shortcuts.
- As an example, when peer 4 receives the message asking about key 11, it determines that the closest 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.
- If we consider the circular DHT, to handle peer churn, its required for each peer to track (that is, know the IP address of) its first and second successors.
- We also require each peer to periodically verify that its two successors are alive (for example, by periodically sending ping messages to them and asking for responses).

For example, peer 4 now tracks both peer 5 and peer 8.

- Let's now consider how the DHT is maintained when (say) peer 5 leaves abruptly. 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.
- What happens when a peer wants to join the DHT? 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.

Probable Question: ***Explain the concept of Circular Hash Tables. How it has overcome the drawbacks of DHT? Comment on Peer Churn concept***

2.7 Socket Programming: Creating Network Applications

- As we know that client and server are basically programs, they become process when executed.
- 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.
- The developers of both these programs must adhere to a standard (RFC preferably).

2.7.1 Socket Programming with UDP

Let's understand this:

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.

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

UDPClient.py

```
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)
```

Below diagram explains the client – server application over UDP.

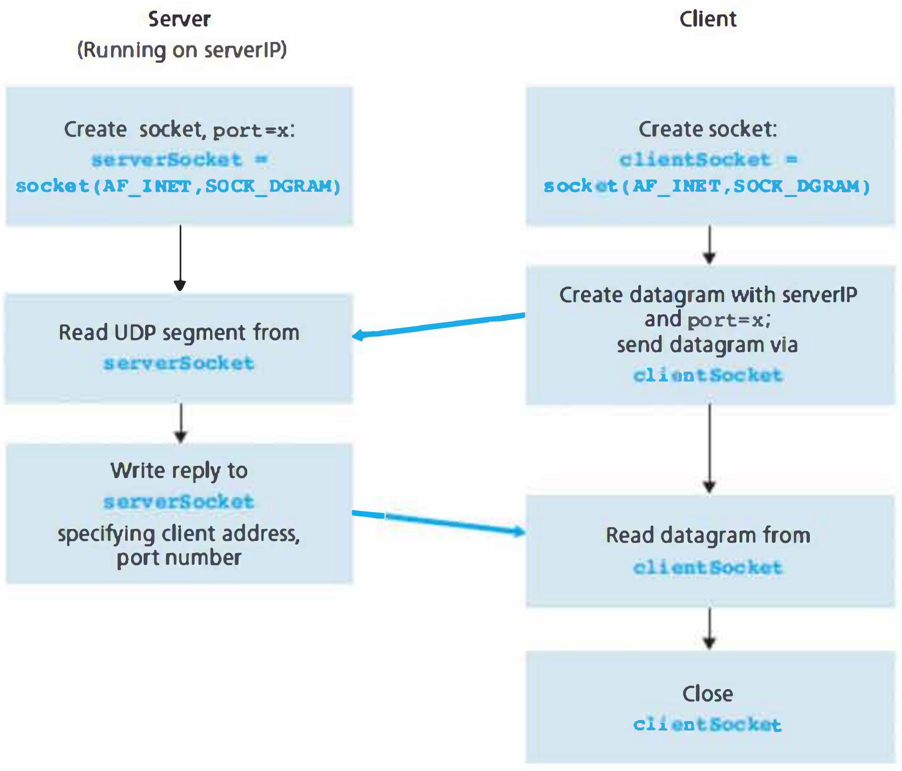

Server
(Running on serverIP)
Client
Create socket, port=x:
serverSocket = socket(AF_INET,SOCK_DGRAM)
Create socket:
clientSocket = socket(AF_INET,SOCK_DGRAM)
Create datagram with serverIP and port=x; send datagram via clientSocket
Read UDP segment from serverSocket
Write reply to serverSocket specifying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

Figure 2. : The client – server application using UDP

TCPClient.py

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

```
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

Now let’s take a look at the server program.

```
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()
```

Below diagram explains the client – server application over TCP.

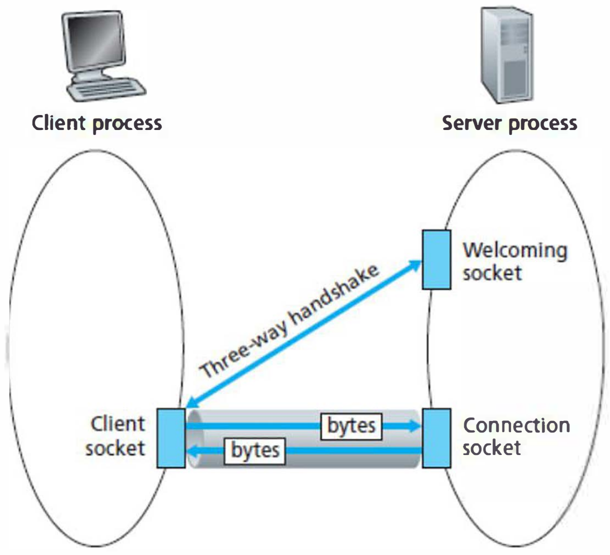


Figure 2. : The client – server application over TCP

You might have observed from the above two forms of communication that the basic difference is in UDP SOCK_DGRAM as it adopts block transfer style and in TCP SOCK_STREAM as it adopts stream transfer.

Probable Question: Explain Socket programming over TCP and UDP with neat diagrams

Write client side and server side socket programs for application over TCP/ UDP