Module-1

Syllabus: Module-1

Application Layer: Principles of Network Applications: Network Application Architectures, Processes Communicating, Transport Services Available to Applications, Transport Services Provided by the Internet, Application-Layer Protocols. The Web and HTTP: Overview of HTTP, Non-persistent and Persistent Connections, HTTP Message Format, User-Server Interaction: Cookies, Web Caching, The Conditional GET, File Transfer: FTP Commands & Replies, Electronic Mail in the Internet: SMTP, Comparison with HTTP, Mail Message Format, Mail Access Protocols, DNS; The Internet's Directory Service: Services Provided by DNS, Overview of How DNS Works, DNS Records and Messages, Peer-to-Peer Applications: P2P File Distribution, Distributed Hash Tables, Socket Programming: creating Network Applications: Socket Programming with UDP, Socket Programming with TCP.

(*Important topics are marked in 🟥)

Question-1: With neat diagram, explain the network application architectures

There are two different network application architecture, they are

1) Client Server Architecture

2) P2P Architecture

Client Server Architecture:

- In client-server architecture, there is an always-on host, called the server, which provides services when it receives requests from many other hosts, called clients.
- Example: In Web application Web server services requests from browsers running on client hosts. When a Web server receives a request for an object from a client host, it responds by sending the requested object to the client host.

- clients do not directly communicate with each other.
- The server has a fixed, well-known address, called an IP address.
- Some of the better-known applications with a client-server architecture include the Web, FTP, Telnet, and e-mail.

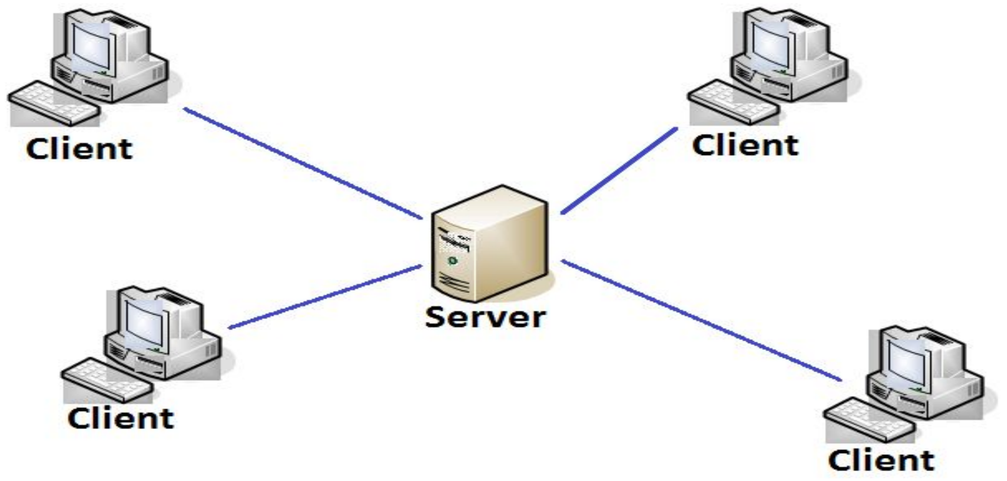


Fig. Client server Architecture

-

- In a client-server application, a single-server host is incapable of keeping up with all the requests from clients. For this reason, a data center, housing a large number of hosts, is often used to create a powerful virtual server.

Peer to Peer Applications:

- In a P2P architecture, there is minimal dependence on dedicated servers in data centers.
- In a P2P architecture, there is minimal dependence on dedicated servers in data centers.
- The peers are not owned by the service provider, but are instead desktops and laptops controlled by users, with most of the peers residing in homes, universities, and offices.

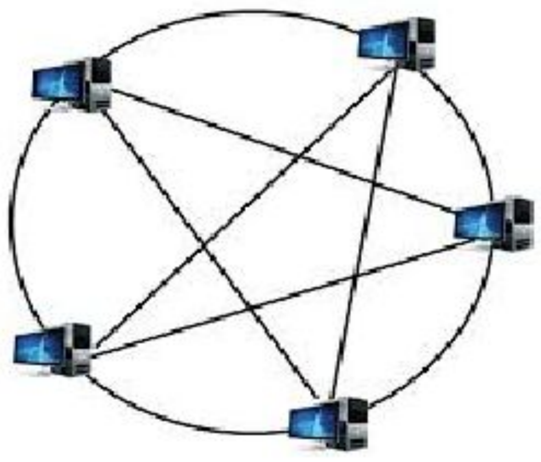


Fig. Peer to Peer Architecture

- Many of today’s most popular and traffic-intensive applications are based on P2P architectures. These applications include file sharing (e.g., BitTorrent), Internet Telephony (e.g., Skype), and IPTV (e.g., Kankan and PPstream).

- Peer -to peer applications are added with following features.
- Self-scalability:
- For example, in a P2P file-sharing application, although each peer generates workload by requesting files, each peer also adds service capacity to the system by distributing files to other peers.
- Cost effective:
- P2P architectures are also cost effective, since they normally don’t require significant server infrastructure and server bandwidth

Question-2:List and explain different transport services offered to the application layer

Transport Services Available to Applications:

1. Reliable Data Transfer:

- One important service that a transport-layer protocol can potentially provide to an application is process-to-process reliable data transfer.
- When a transport protocol provides this service, the sending process can just pass its data into the socket and know with complete confidence that the data will arrive without errors at the receiving process.
- When a transport-layer protocol doesn’t provide reliable data transfer, some of the data sent by the sending process may never arrive at the receiving process. This may be acceptable for loss-tolerant applications, most notably multimedia applications such as conversational audio/video that can tolerate some amount of data loss.

2) Throughput:

Transport-layer protocol could provide guaranteed available throughput at some specified rate.

With such a service, the application could request a guaranteed throughput of r bits/sec, and the transport protocol would then ensure that the available throughput is always at least r bits/sec. Such a guaranteed throughput service would appeal to many applications.

For example, if an Internet telephony application encodes voice at 32 kbps, it needs to send data into the network and have data delivered to the receiving application at this rate.

- If the transport protocol cannot provide this throughput, the application would need to encode at a lower rate or may have to give up.
- Applications that have throughput requirements are said to be bandwidth-sensitive applications. Many current multimedia applications are bandwidth sensitive
- Elastic applications can make use of as much, or as little, throughput as happens to be available. Electronic mail, file transfer, and Web transfers are all elastic applications.

3) Timing

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

4) Security:

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

Question-3:Explain HTTP request and response messages. Give example for each

HTTP Request Message: HTTP Request message gives details about the service that should be given by the specific server.

Format of HTTP Request Message:

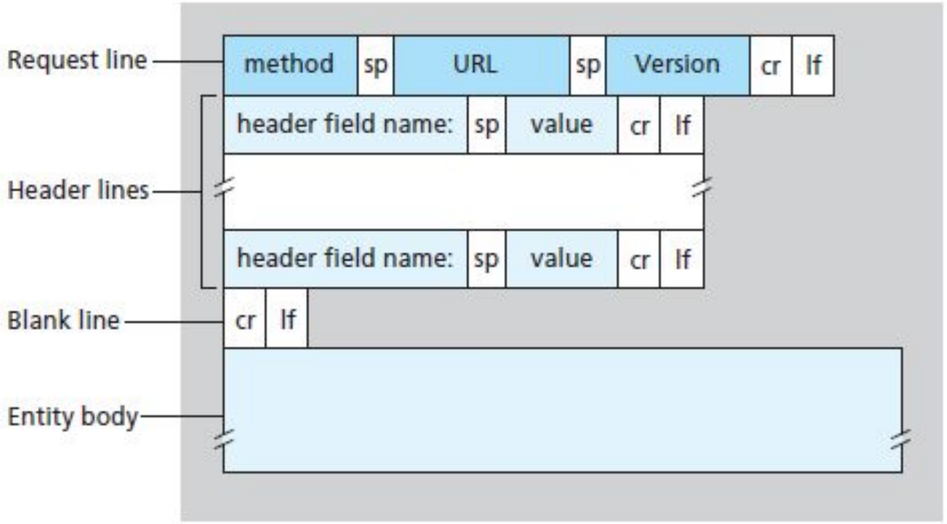


Fig. HTTP Request Message format

- 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.
- The first line of an HTTP request message is called the request line; the subsequent lines are called the header lines.
- The request line has three fields: the method field, the URL field, and the HTTP version field.
- The method field can take on several different values, including GET, POST, HEAD, PUT, and DELETE.
- The great majority of HTTP request messages use the GET method. The GET method is used when the browser requests an object, with the requested object identified in the URL field.
- An HTTP client often uses the POST method when the user fills out a form
- The HEAD method is similar to the GET method. When a server receives a request with the HEAD method, it responds with an HTTP message but

- The PUT method is also used by applications that need to upload objects to
- Web servers.
- The DELETE method allows a user, or an application, to delete an object on a Web server.
- Below we provide a typical HTTP request message:

```
GET /somedir/page.html HTTP/1.1

Host: www.someschool.edu

Connection: close

User-agent: Mozilla/5.0

Accept-language: fr
```

- The message is written in ordinary ASCII text, so that your ordinary computer-literate human being can read it

- The header line Host: www.someschool.edu specifies the host on which the object resides.
- By including the Connection: close header line, the browser is telling the server that it doesn’t want to bother with persistent connections; it wants the server to close the connection after sending the requested object.
- The User-agent: header line specifies the user agent, that is, the browser type that is making the request to the server. Here the user agent is Mozilla/5.0, a Firefox browser.
- the Acceptlanguage: header indicates that the user prefers to receive a French version of the object, if such an object exists on the server; otherwise, the server should send its default version.
- The Accept-language: header is just one of many content negotiation headers available in HTTP.

Format of HTTP Request Message:

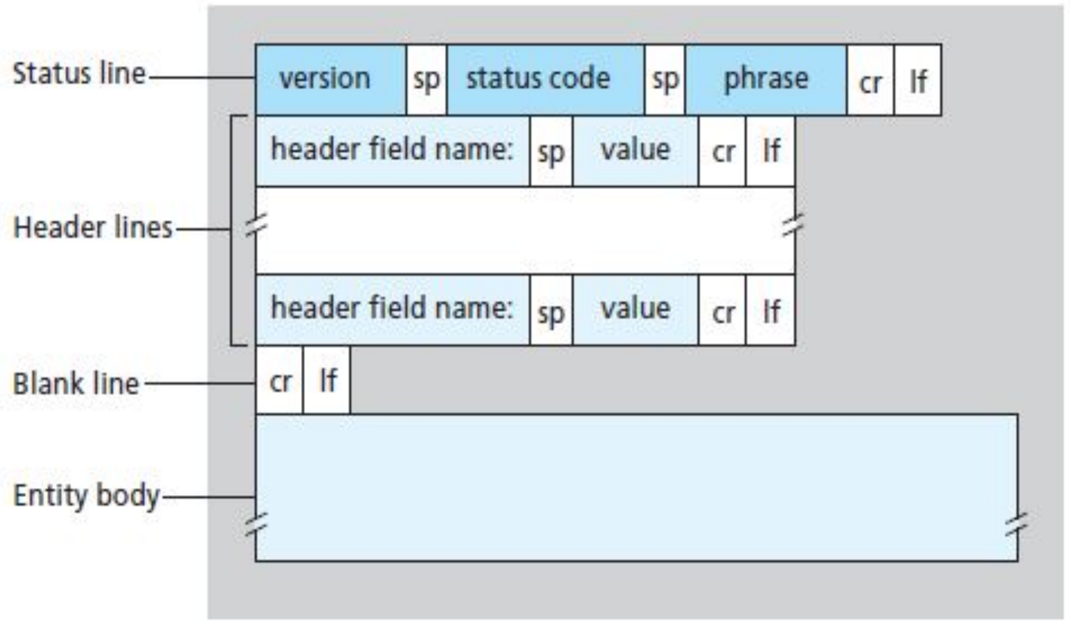


Fig. HTTP Response message

- Response message has three sections: an initial status line, six header lines, and then the entity body.
- The status line has three fields: the protocol version field, a status code, and a corresponding status message.
- The server uses the Connection: close header line to tell the client that it is going to close the TCP connection after sending the message.
- The Date: header line indicates the time and date when the HTTP response was created and sent by the server.
- The Server: header line indicates that the message was generated by an specific 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.

Example: 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 ...)
```

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

- Version is HTTP/1.1,200 OK: Request succeeded and the information is returned in the response.
- Header fields specify the details about the server Data and other details about the service.

Question 4: Write short note on i).Web Cache ii). Cookie

WEB Cache: also called a proxy server. It 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.
- A user's browser can be configured so that all of the user's HTTP requests are fi

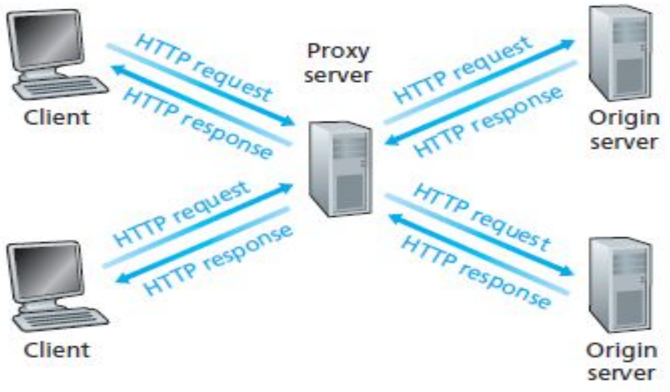


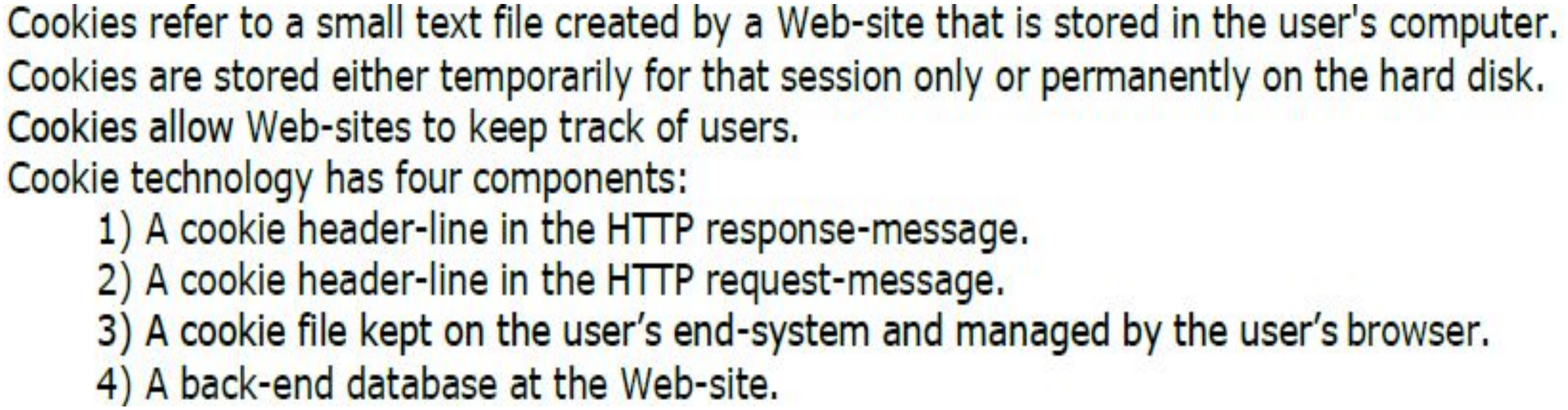
Fig. Web Cache

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

- The browser establishes a TCP connection to the Web cache and sends an HTTP request for the object to the Web cache.
- 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.
- 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.

- When the Web cache receives the object, it stores a copy in its local storage and sends a copy, within an HTTP response message, to the client browser (over the existing TCP connection between the client browser and the Web cache).
- When web cache receives requests from and sends responses to a browser, it is a server. When it sends requests to and receives responses from an origin server, it is a client.
- Typically a Web cache is purchased and installed by an ISP. For example, a university might install a cache on its campus network and configure all of the campus browsers to point to the cache. Or a major residential ISP (such as AOL) might install one or more caches in its network and pre configure its shipped browsers to point to the installed caches.
- Web caching has seen deployment in the Internet for two reasons. First, a Web cache can substantially reduce the response time for a client request. Second, Web caches can substantially reduce traffic on an

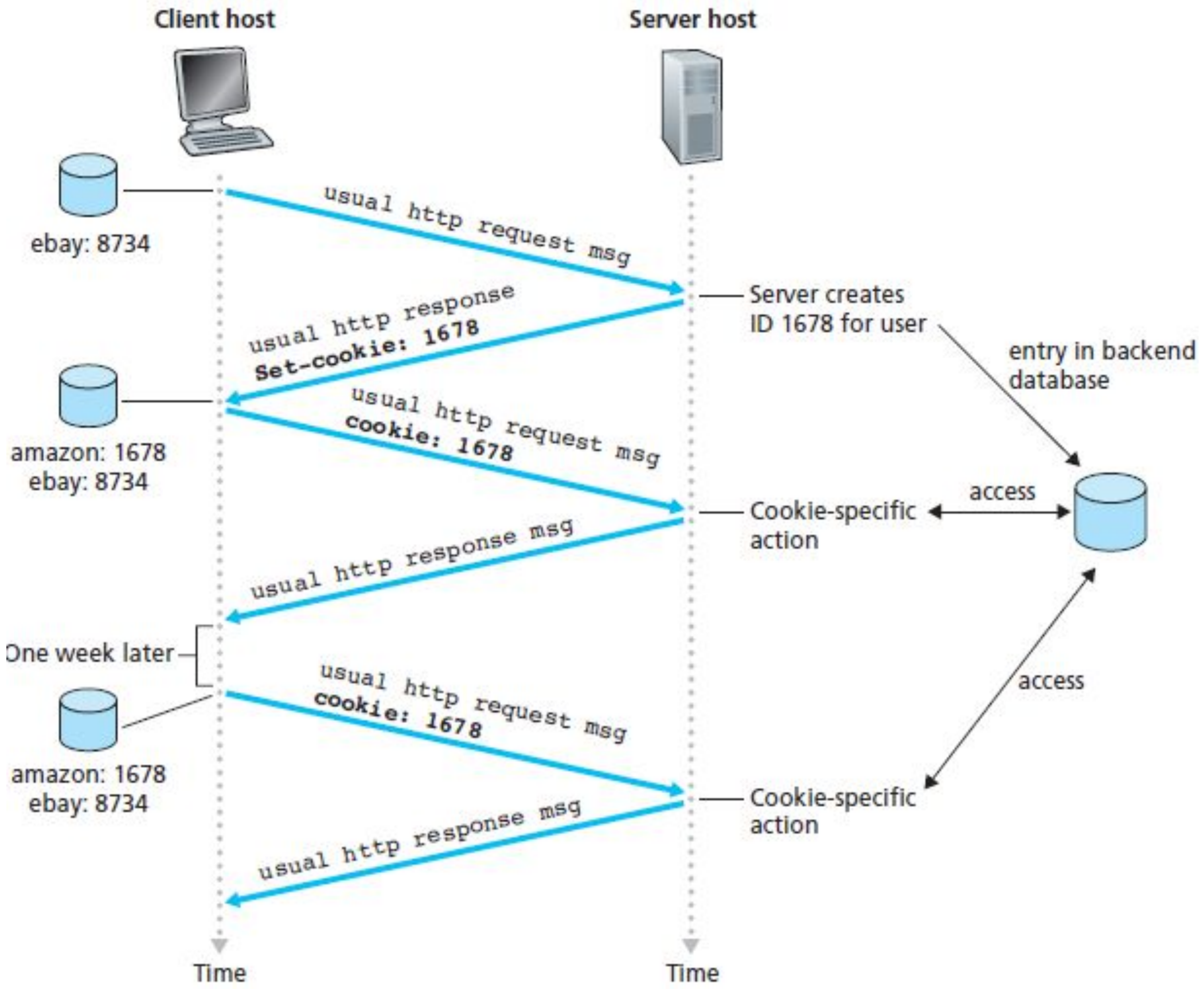
Cookie:



Cookies refer to a small text file created by a Web-site that is stored in the user's computer.
Cookies are stored either temporarily for that session only or permanently on the hard disk.
Cookies allow Web-sites to keep track of users.
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.

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



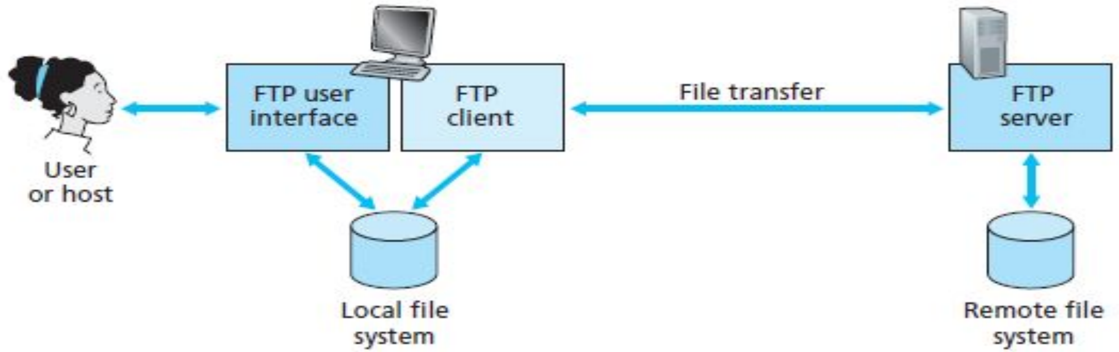
Client host
Server host
ebay: 8734
usual http request msg
Server creates ID 1678 for user
usual http response
Set-cookie: 1678
entry in backend database
amazon: 1678
ebay: 8734
usual http request msg
cookie: 1678
access
Cookie-specific action
usual http response msg
One week later
usual http request msg
cookie: 1678
access
amazon: 1678
ebay: 8734
Cookie-specific action
usual http response msg
Time
Time

- For example, the header line might be:
- Set-cookie: 1678
- When users browser receives the HTTP response message, it sees the Set-cookie: header. The browser then appends a line to the special cookie file that it manages. This line includes the hostname of the server and the identification number in the Set-cookie: header.
- As user continues to browse the Amazon site, each time he requests a Web page, his browser consults his cookie file, extracts his identification number for this site, and puts a cookie header line that includes the identification number in the HTTP request. Specifically, each of his HTTP requests to the Amazon server includes the header line: Cookie: 1678

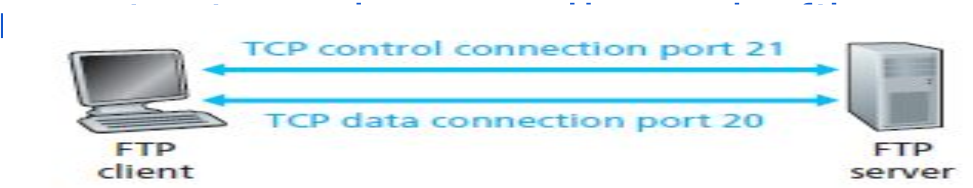
. **Question-5: Write a short note on SMTP, FTP**

FTP:

- FTP is used for transferring file from one host to another host.
- In order for the user to access the remote account, the user must provide user identification and a password. After providing this authorization information, the user can transfer files from the local file system to the remote file system and vice versa.
- The user first provides the hostname of the remote host, causing the FTP client process in the local host to establish a TCP connection with the FTP server process in the remote host.
- The user then provides the user identification and password, which are sent over the TCP connection as part of FTP commands.
- Once the server has authorized the user, the user copies one or more files stored in the local file system into the remote file system (or vice versa).



- FTP uses two parallel TCP connections to transfer a file, a control connection and a data connection.
- The control connection is used for sending control information between the two hosts information such as user identification, password, commands to change remote directory, and commands to "put" and "get" files.
- The data co



- When a user starts an FTP session with a remote host, the client side of FTP (user) first initiates a control TCP connection with the server side (remote host) on server port number 21.
- The client side of FTP sends the user identification and password over this control connection. The client side of FTP also sends, over the control connection, commands to change the remote directory.
- When the server side receives a command for a file transfer over the control connection (either to, or from, the remote host), the server side initiates a TCP data connection to the client side.
- FTP sends exactly one file over the data connection and then closes the data connection. If, during the same session, the user wants to transfer another file, FTP opens another data connection.

- Thus, with FTP, the control connection remains open throughout the duration of the user session, but a new data connection is created for each file transferred within a session (that is, the data connections are non-persistent).
- Throughout a session, the FTP server must maintain state about the user. In particular, the server must associate the control connection with a specific user account, and the server must keep track of the user's current directory as the user wanders about the remote directory tree

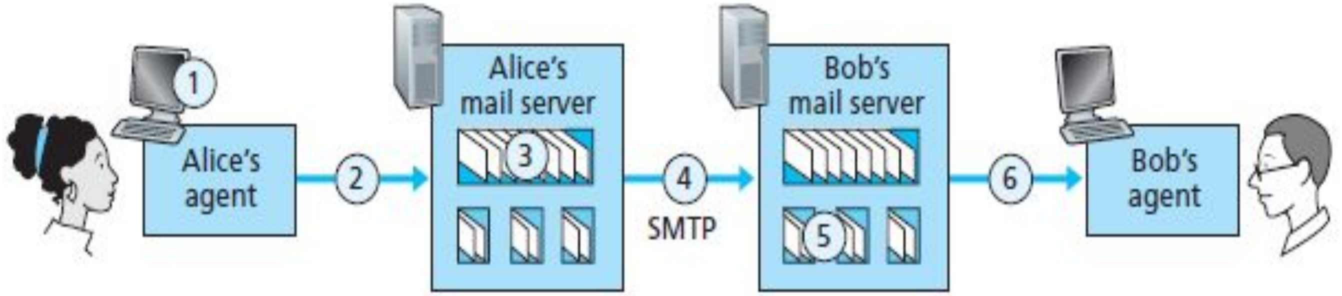
SMTP:

- SMTP is the principal application-layer protocol for Internet electronic mail. It uses the reliable data transfer service of TCP to transfer mail from the sender's mail server to the recipient's mail server. As with most application-layer protocols, SMTP has two sides: a client side,

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

Suppose Alice wants to send Bob a simple ASCII message.

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



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

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

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

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

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

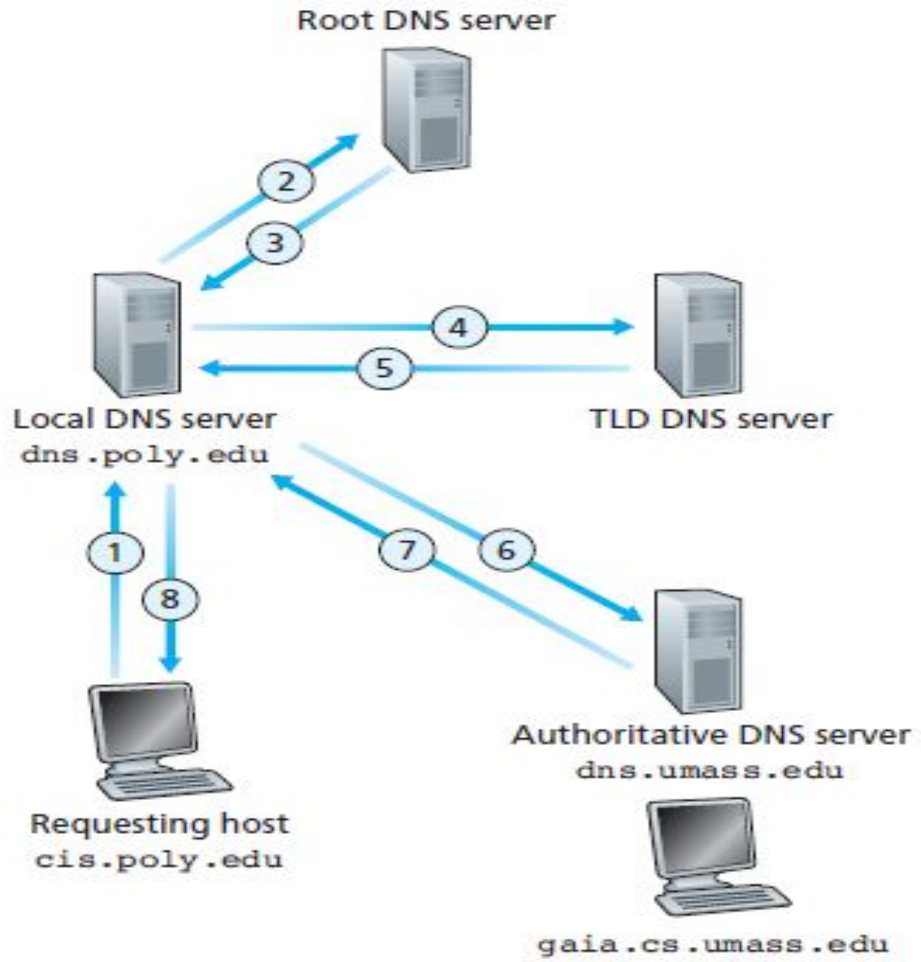
Question 5: Explain iterative and recursive query processing services offered by DNS

- The DNS is a distributed database implemented in a hierarchy of DNS servers, and an application-layer protocol that allows hosts to query the distributed database.
- Uses either iterative or recursive approach for query processing.

Iterative approach:

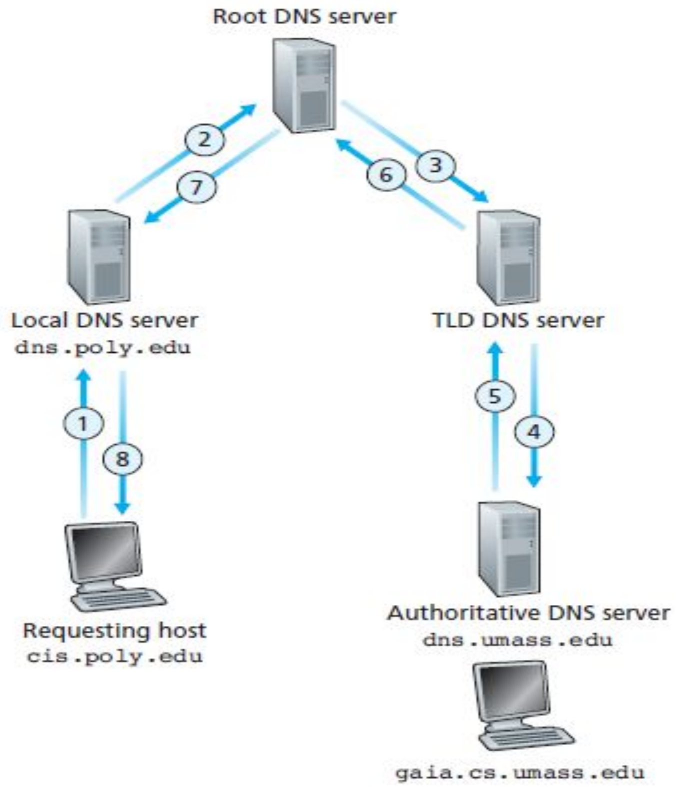
The query message contains the hostname to be translated,namely, gaia.cs.umass.edu.

- 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


Root DNS server
2
3
4
5
Local DNS server
dns.poly.edu
TLD DNS server
1
8
7
6
Authoritative DNS server
dns.umass.edu
Requesting host
cis.poly.edu
gaia.cs.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.

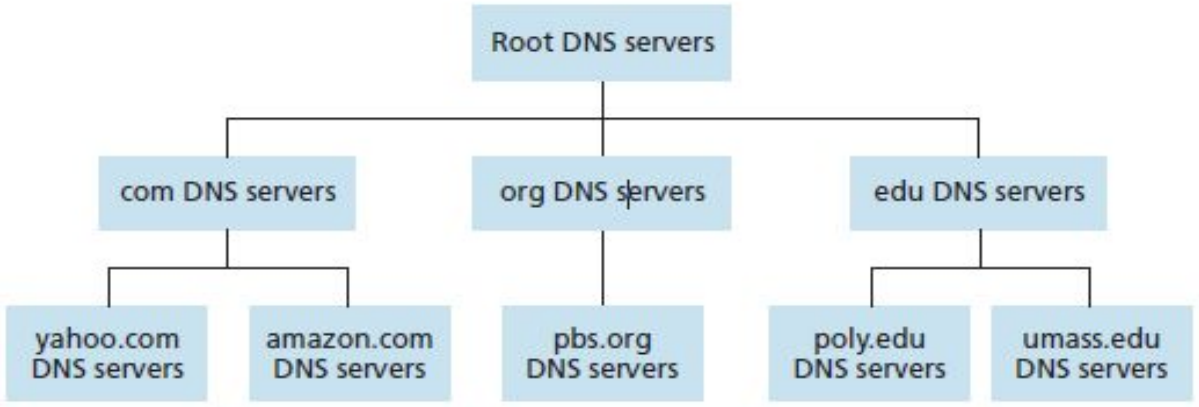
Recursive Query service:



- Here DNS query will be sent to Local DNS server, then to root server.
- Root server sends the query to TLD DNS server
- TLD DNS server sends the query to authoritative DNS server
- Authoritative DNS server sends the IP address of host to local DNS server through a chain communication with TLD server, root server and the client’s Local DNS server which sends it to the host.

Question: Write short note on DNS system Hierarchy

- In order to deal with the issue of scale, the DNS uses a large number of servers, organized in a hierarchical fashion and distributed around the world.
- There are three classes of DNS servers—root DNS servers, top-level domain (TLD) DNS servers, and authoritative DNS servers—organized in a hierarchy



- Root DNS servers. In the Internet there are 13 root DNS servers (labeled A through M), most of which are located in North America.
- Although we have referred to each of the 13 root DNS servers as if it were a single server, each "server" is actually a network of replicated servers, for both security and reliability purposes. All together, there are 247 root servers.
- Top-level domain (TLD) servers: These servers are responsible for top-level domains such as com, org, net, edu, and gov, and all of the country top-level domains such as in,uk, fr, ca.
- Authoritative DNS servers: Every organization with publicly accessible hosts on the Internet must provide publicly accessible DNS records that map the names of those hosts to IP addresses. An organization's

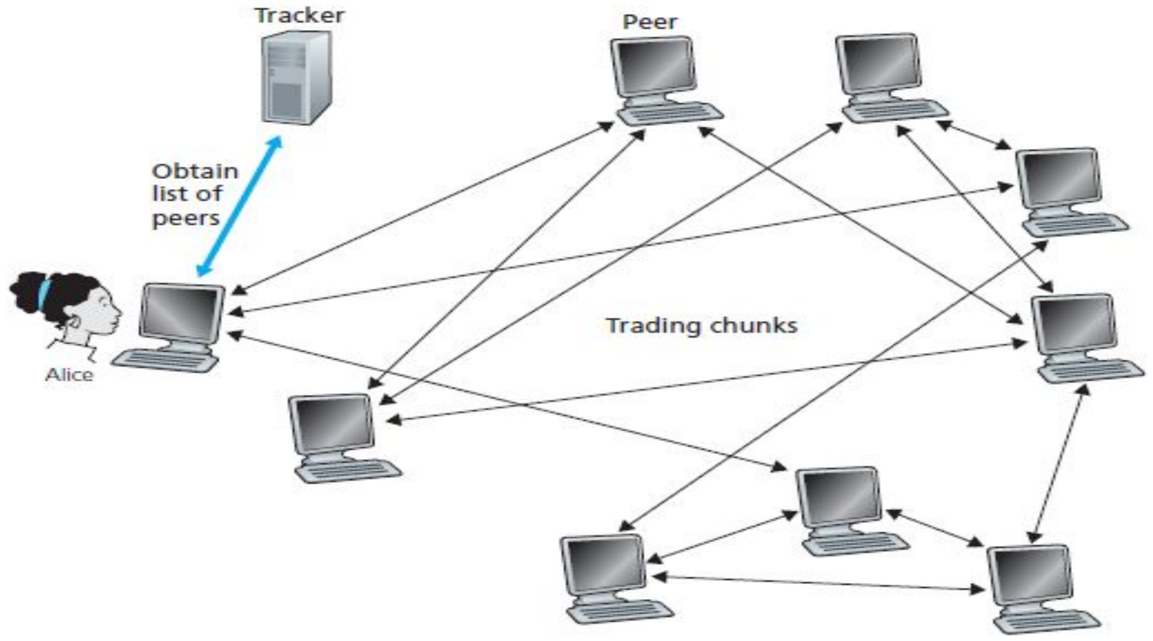
- There is another important type of DNS server called the local DNS server. A local DNS server does not strictly belong to the hierarchy of servers but is nevertheless central to the DNS architecture. Each ISP—such as a university, an academic department, an employee’s company, or a residential ISP—has a local DNS server.

Question:Explain the working of Bit Torrent application

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

- Each torrent has an infrastructure node called a tracker.
- When a peer joins a torrent, it registers itself with the tracker and periodically informs the tracker that it is still in the torrent. In this manner, the tracker keeps track of the peers that are participating in the torrent.
- When a new peer joins the torrent, the tracker randomly selects a subset of peers (for concreteness, say 50) from the set of participating peers, and sends the IP addresses of these 50 peers to new peer.
- Possessing this list of peers, new peer attempts to establish concurrent TCP connections with all the peers on this list. All the peers with which new peer succeeds in establishing a TCP connection will be called as “neighboring peers.”
- As time evolves, some of these peers may leave and other peers (outside the initial 50) may attempt to establish TCP connections.
- Periodically, peer will ask each of its neighboring peers (over the TCP connections) for the list of the chunks they have. If peer has L different neighbors, it will obtain requests (again over the TCP connections) for chunks

- In deciding which chunks to request, peer uses a technique called rarest first. The idea is to determine, from among the chunks peer does not have, the chunks that are the rarest among its neighbors and then request those rarest chunks first. In this manner, the rarest chunks get more quickly redistributed, aiming to equalize the numbers of copies of each chur
-



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

Question:Write a short note on Clever trading algorithm

- The peer nodes in Bittorrent, uses a technique called rarest first for geeting the rarest chunks from neighbors. The idea is to determine, from among the chunks the node 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. In this manner, the rarest chunks get more quickly redistributed, aiming to (roughly) equalize the numbers of copies of each chunk in the torrent.
- To determine which requests the node responds to, BitTorrent uses a clever trading algorithm. The basic idea is that node gives priority to the neighbors that are currently supplying her data at the highest rate.
- Specifically, for each of her neighbors,the node continually measures the rate at which it receives bits and determines the four peers that are feeding her bits 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.
- In BitTorrent lingo, these four peers are said to be unchoked. Importantly, every 30 seconds, it also picks one additional neighbor at random and sends it chunks.
- Node will randomly choose a new trading partner and initiate trading with that partner.

Question:Write a short note on DHT

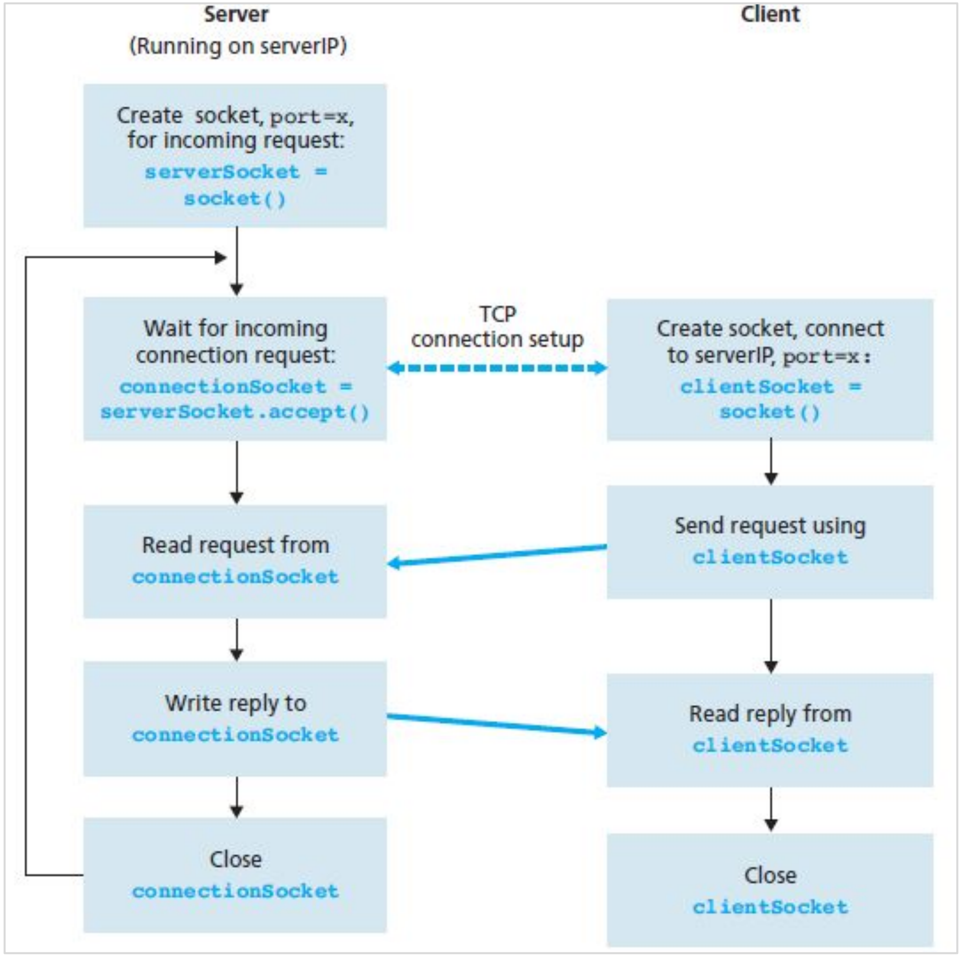
Distributed Hash Tables (DHTs):

- Centralized version of this simple database will simply contain (key, value) pairs. We query the database with a key. If there are one or more key-value pairs in the database that match the query key, the database returns the corresponding values.
- Building such a database is straightforward with client-server architecture that stores all the (key, value) pairs in one central server.
- P2P version of this database will store the (key, value) pairs over millions of peers.
- In the P2P system, each peer will only hold a small subset of the totality of the (key, value) pairs. We'll allow any peer to query the distributed database with a particular key. The distributed database will then locate the peers that have the corresponding (key, value) pairs and return the key-value pairs to the querying peer

- Any peer will also be allowed to insert new key-value pairs into the database. Such a distributed database is referred to as a distributed hash table (DHT).
- One naïve approach to building a DHT is to randomly scatter the (key, value) pairs across all the peers and have each peer maintain a list of the IP addresses of all participating peers.
- In this design, the querying peer sends its query to all other peers, and the peers containing the (key, value) pairs that match the key can respond with their matching pairs.
- Such an approach is completely unscalable as it would require each peer to know about all other peers and have each query sent to all peers.
- An elegant approach to designing a DHT is to first assign an identifier to each peer, where each identifier is an integer in the range [0, 2n-1] for some fixed n.
- This also require each key to be an integer in the same range.
- To create integers out of such keys, we will use a hash function that

Question:Write and explain the algorithm for TCP enabled socket communication

- TCP is a connection-oriented protocol. This means that before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection.
- One end of the TCP connection is attached to the client socket and the other end is attached to a server socket.
- When creating the TCP connection, we associate with it the client socket address (IP address and port number) and the server socket address (IP address and port number). With the TCP connection established, when one side wants to send data to the other side, it just drops the data into the TCP connection via its socket. This is different from UDP, for which the server must attach a destination address to the packet before dropping it into the socket.
- During the three-way handshake, the client process knocks on the welcoming door of the server process. When the server "hears" the


Server
(Running on serverIP)
Client
Create socket, port=x, for incoming request:
serverSocket = socket()
Wait for incoming connection request:
connectionSocket = serverSocket.accept()
TCP connection setup
Create socket, connect to serverIP, port=x:
clientSocket = socket()
Read request from connectionSocket
Send request using clientSocket
Write reply to connectionSocket
Read reply from clientSocket
Close connectionSocket
Close clientSocket

TCP Client Program

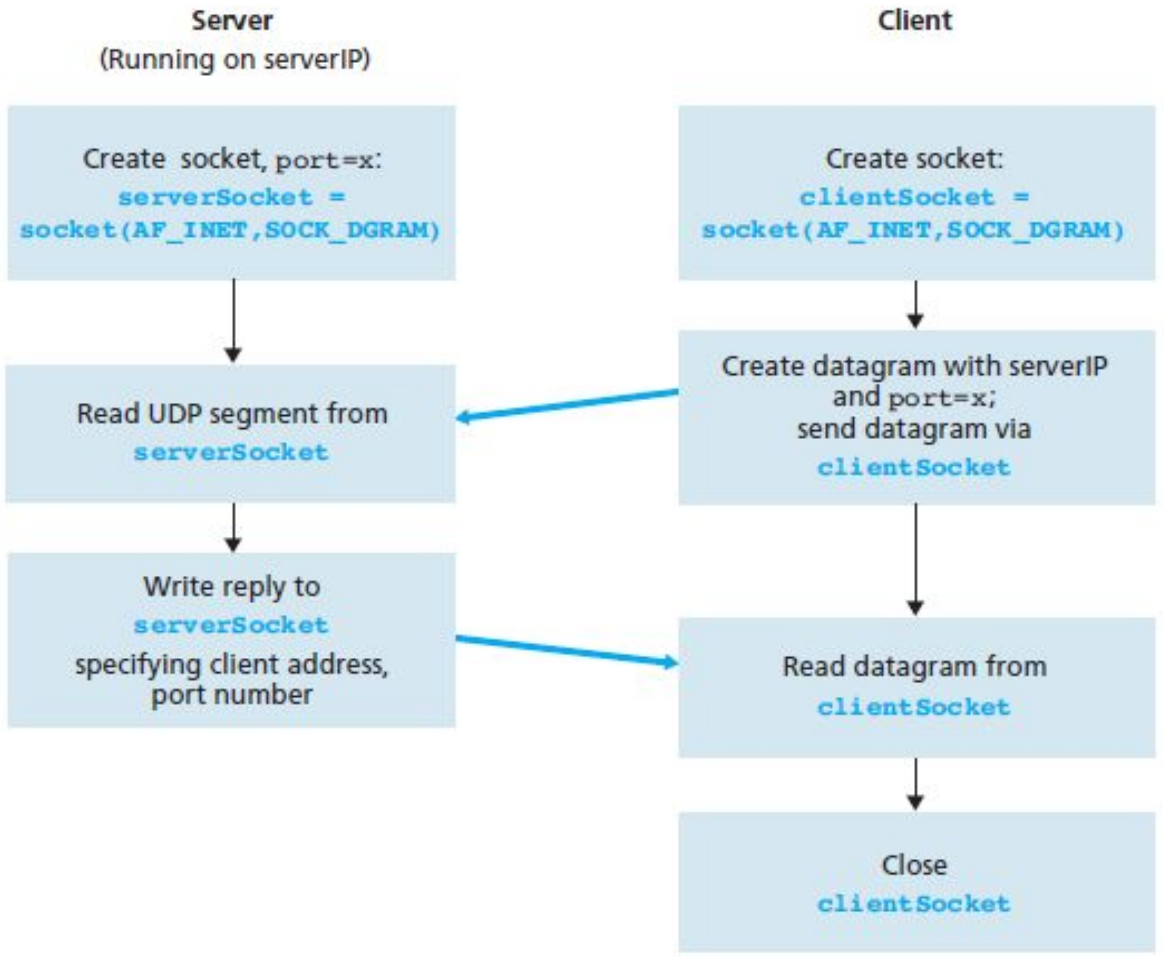
```
from socket import *
serverName = ’servername’
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input(‘Input lowercase sentence:’)
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print ‘From Server:’,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

TCP 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)
```

Question: UDP Sockets

- Before the sending process can push a packet of data out the socket door, when using UDP, it must first attach a destination address to the packet.
- After the packet passes through the sender’s socket, the Internet will use this destination address to route the packet through the Internet to the socket in the receiving process.
- When the packet arrives at the receiving socket, the receiving process will retrieve the packet through the socket, and then inspect the packet’s contents and take appropriate action.
- Example application:
- 1. The client reads a line of characters (data) from its keyboard and sends the data to the server.
- 2. The server receives the data and converts the characters to uppercase.
- 3. The server sends the modified data to the client.


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

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

```
from socket import *

serverName = ‘hostname’

serverPort = 12000

clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)

message = raw_input(’Input lowercase sentence:’)

clientSocket.sendto(message,(serverName, serverPort))

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

print modifiedMessage

clientSocket.close()
```

UDP Server:

```
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)
```

“YOU DON’T HAVE TO SEE THE WHOLE STAIRCASE,
JUST TAKE THE FIRST STEP.”
MARTIN LUTHER KING JR.