USN | | | | | | | | | | |

Internal Assessment Test 1 – September 2019

Sub:	COMPUTER NETWORKS					Sub Code:	17CS52	Branch:	CSE	
Date:	11/11/2021	Duration:	90 min's	Max Marks:	50	Sem / Sec:	A,B,C		OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	Explain the different types of Network Application Architectures	[10]	CO1	L1
2	With examples, explain HTTP request and response message formats.	[10]	CO1	L2
3	Explain the following application Protocols a) SMTP b). FTP	[10]	CO1	L1
4	Write short notes on a) Cookies b) Web Caching	[10]	CO2	L2
5 (a)	Describe the iterative and recursive services for query resolution provided by DNS	[06]	CO2	L2
(b)	What is DHT? Explain in brief.	[04]	CO2	L2
6	Implement a network application for client server communication over TCP, using socket programming	[10]	CO2	L3
7 (a)	Explain the connection-oriented multiplexing and de-multiplexing	[06]	CO2	L2
(b)	With an example, demonstrate how UDP checksum is generated and how is it used to detect the transmission errors at the receiver end? (Note: Take any four 16-bit data blocksas input)	[04]	CO2	L2
8 (a)	Demonstrate the working of rdt 2.0 (sender and receiver) with a neat labeled FSM diagram	[06]	CO2	L3
(b)	Differentiate between UDP and TCP	[04]	CO2	L2

USN | | | | | | | | | | |

Internal Assessment Test 1 – September 2019

Sub:	COMPUTER NETWORKS					Sub Code:	17CS52	Branch:	CSE	
Date:	11/11/2019	Duration:	90 min's	Max Marks:	50	Sem / Sec:	A,B,C		OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	Explain the different types of Network Application Architectures	[10]	CO1	L1
2	With examples, explain HTTP request and response message formats.	[10]	CO1	L2
3	Explain the following application Protocols a) SMTP b). FTP	[10]	CO1	L1
4	Write short notes on a) Cookies b) Web Caching	[10]	CO2	L2
5 (a)	Describe the iterative and recursive services for query resolution provided by DNS	[06]	CO2	L2
(b)	What is DHT? Explain in brief.	[04]	CO2	L2
6	Implement a network application for client server communication over TCP, using socket programming	[10]	CO2	L3
7 (a)	Explain the connection-oriented multiplexing and de-multiplexing	[06]	CO2	L2
(b)	With an example, demonstrate how UDP checksum is generated and how is it used to detect the transmission errors at the receiver end? (Note: Take any four 16-bit data blocksas input)	[04]	CO2	L2

8 (a)	Demonstrate the working of rdt 2.0 (sender and receiver) with a neat labeled FSM diagram	[06]	CO2	L3
(b)	Differentiate between UDP and TCP	[04]	CO2	L2

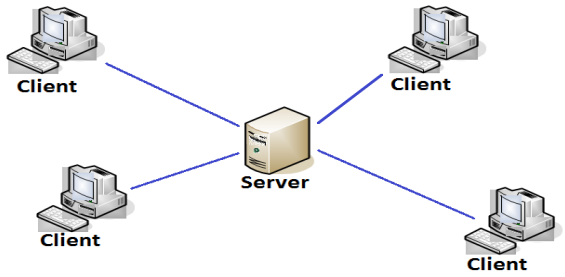
1) Network application Architectures

Client Server Architecture:

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

Example: In Web application Web server services requests from browsers running on client hosts. When a Web server receives a request for an object from a client host, it responds by sending the requested object to the client host.

- In client-server architecture, clients do not directly communicate with each other.
- The server has a fixed, well-known address, called an IP address. Because the server has a fixed, well-known address, and because the server is always on, a client can always contact the server by sending a packet to the server's IP address.
- Some of the better-known applications with a client-server architecture include the Web, FTP, Telnet, and e-mail.



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

Peer-to-peer (P2P) Architecture:

- In a P2P architecture, there is minimal dependence on dedicated servers in data centers.
- The application employs direct communication between pairs of intermittently connected hosts, called peers.
- The peers are not owned by the service provider, but are instead desktops and laptops controlled by users, with most of the peers residing in homes, universities, and offices.
- Many of today's most popular and traffic-intensive applications are based on P2P architectures. These applications include file sharing (e.g., BitTorrent), Internet Telephony (e.g., Skype), and IPTV (e.g., Kankan and PPstream).

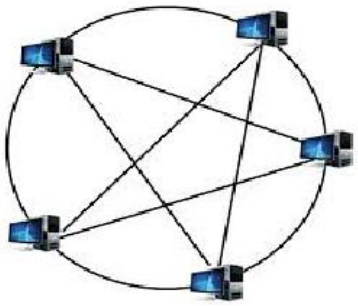
Features:

- **Self-scalability:**

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

- **Cost effective:**

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



Future P2P applications face three major challenges:

1. **ISP Friendly.** Most residential ISPs have been dimensioned for "asymmetrical" bandwidth usage, that is, for much more downstream than upstream traffic. But P2P video streaming and file distribution applications shift upstream traffic from servers to residential ISPs, thereby putting significant stress on the ISPs. Future P2P applications need to be designed so that they are friendly to ISPs

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

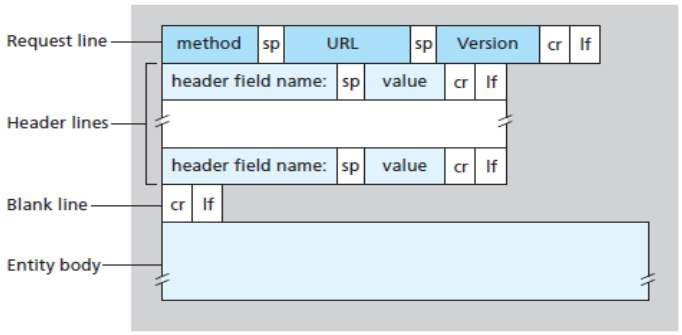
3. **Incentives.** The success of future P2P applications also depends on convincing users to volunteer bandwidth, storage, and computation resources to the applications, which is the challenge of incentive design.

2b). 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.
- A user's browser can be configured so that all of the user's HTTP requests are first directed to the Web cache.

2). Message formats for HTTP Request and Reply messages

HTTP Request Message:



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

Method:

There are five HTTP methods:

 GET: The GET method is used when the browser requests an object, with the requested object identified in the URL field.

 POST: With a POST message, the user is still requesting a Web page from the server, but the specific contents of the Web page depend on what the user entered into the form fields. If the value of the method field is POST, then the entity body contains what the user entered into the form fields.

 PUT: The PUT method is also used by applications that need to upload objects to Web servers.

 HEAD: Used to retrieve header information. It is used for debugging purpose.

 DELETE: The DELETE method allows a user, or an application, to delete an object on a Web server.

URL: Specifies URL of the requested object

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

Header line:

Ex:

Host: www.someschool.edu

Connection: close

User-agent: Mozilla/5.0

Accept-language: fr

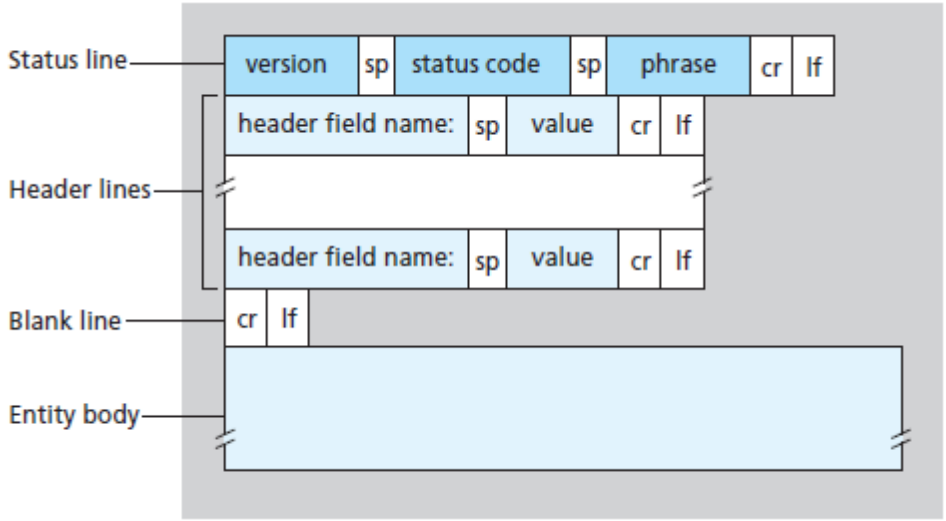
The header line **Host:www.someschool.edu** specifies the host on which the object resides.

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

The **User-agent:** header line specifies the user agent, that is, the browser type that is making the request to the server. Here the user agent is Mozilla/5.0, a Firefox browser.

The **Accept-language:** header indicates that the user prefers to receive a French version of the object, if such an object exists on the server; otherwise, the server should send its default version.

HTTP Response Message



Ex:

HTTP/1.1 200 OK

Connection: close

Date: Tue, 09 Aug 2011 15:44:04 GMT

Server: Apache/2.2.3 (CentOS)

Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT

Content-Length: 6821

Content-Type: text/html

(data data data data data ...)

The **status**

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

Version is HTTP/1.1

The status code and associated phrase indicate the result of the request. Some common status codes and associated phrases include:

• 200 OK: Request succeeded and the information is returned in the response.

• 301 Moved Permanently: Requested object has been permanently moved; the new URL is specified in Location: header of the response message. The client software will automatically retrieve the new URL.

• 400 Bad Request: This is a generic error code indicating that the request could not be understood by the server.

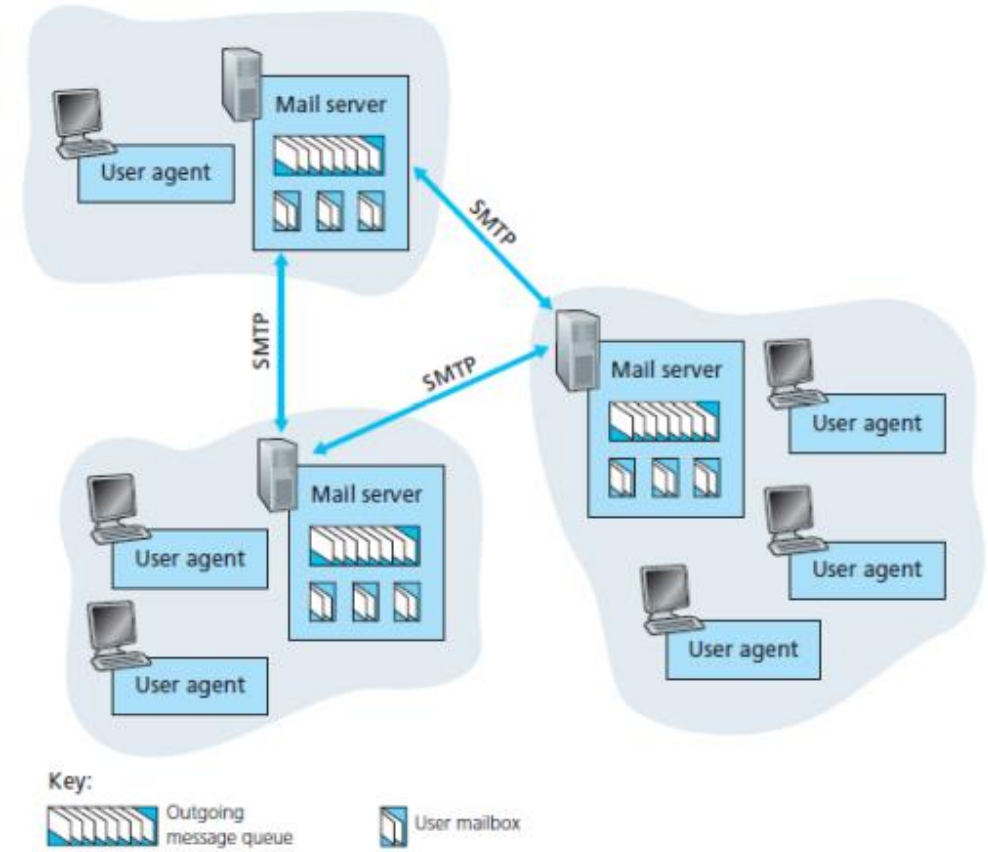
• 404 Not Found: The requested document does not exist on this server.

• 505 HTTP Version Not Supported: The requested HTTP protocol version is not supported by the server.

Header fields:

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

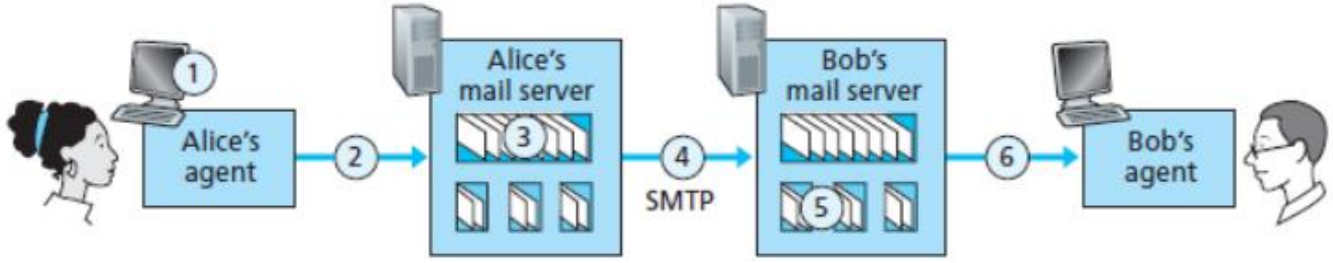
3 a) SMTP : Electronic Mail in the Internet E-mail has three major components: user agents, mail servers, and the Simple Mail Transfer Protocol (SMTP)



User agents allow users to read, reply to, forward, save, and compose messages. • Mail servers form the core of the e-mail infrastructure. Each recipient has a mailbox located in one of the mail servers. A typical message starts its journey in the sender's user agent, travels to the sender's mail server, and travels to the recipient's mail server, where it is deposited in the recipient's mailbox. • SMTP is the principal application-layer protocol for Internet electronic mail. It uses the reliable data transfer service of TCP to transfer mail from the sender's mail server to the recipient's mail server. As with most application-layer protocols, SMTP

has two sides: a client side, which executes on the sender's mail server, and a server side, which executes on the recipient's mail server. 1.4.1 SMTP SMTP transfers messages from senders' mail servers to the recipients' mail servers. It restricts the body (not just the headers) of all mail messages to simple 7-bit ASCII. Suppose Alice wants to send Bob a simple ASCII message. 1. Alice invokes her user agent for e-mail, provides Bob's e-mail address (for example, bob@someschool.edu), composes a message, and instructs the user agent to send the message.

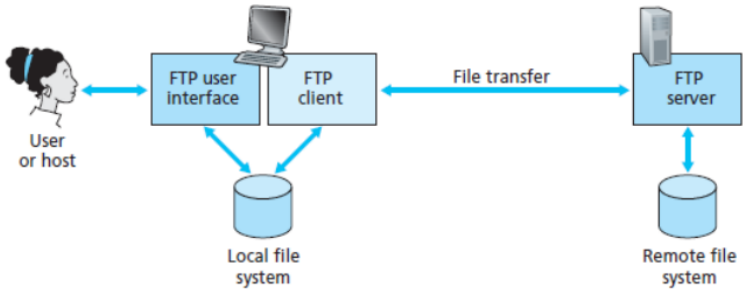
2. Alice's user agent sends the message to her mail server, where it is placed in a message queue. 3. The client side of SMTP, running on Alice's mail server, sees the message in the message queue. It opens a TCP connection to an SMTP server, running on Bob's mail server. 4. After some initial SMTP handshaking, the SMTP client sends Alice's message into the TCP connection. 5. At Bob's mail server, the server side of SMTP receives the message. Bob's mail server then places the message in Bob's mailbox. 6. Bob invokes his user agent to read the message at his convenience.



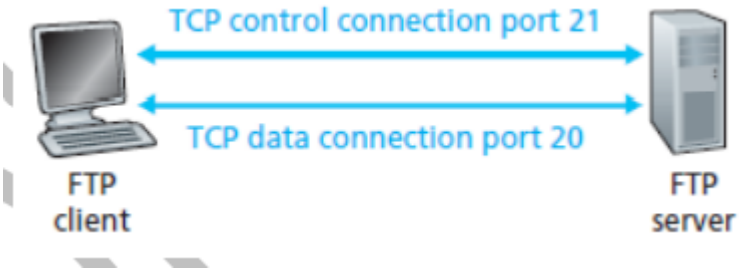
b) FTP : File Transfer: FTP

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

Once the server has authorized the user, the user copies one or more files stored in the local file system into the remote file system (or vice versa)



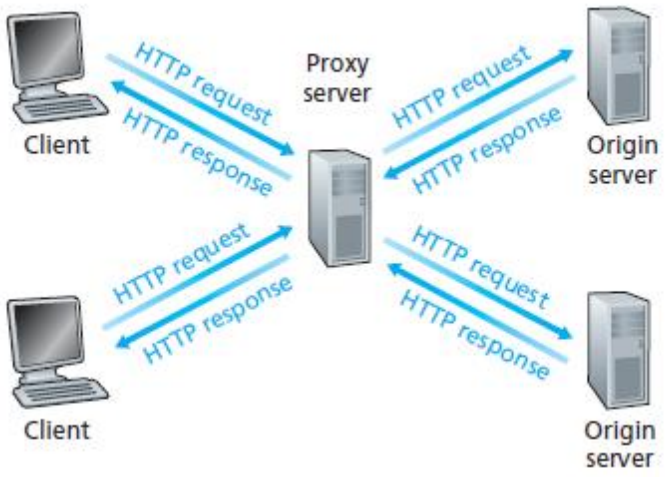
FTP uses two parallel TCP connections to transfer a file, a control connection and a data connection. • The control connection is used for sending control information between the two hosts— information such as user identification, password, commands to change remote directory, and commands to “put” and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



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

FTP sends exactly one file over the data connection and then closes the data connection. If, during the same session, the user wants to transfer another file, FTP opens another data connection. • Thus, with FTP, the control connection remains open throughout the duration of the user session, but a new data connection is created for each file transferred within a session (that is, the data connections are non-persistent). • Throughout a session, the FTP server must maintain state about the user. In particular, the server must associate the control connection with a specific user account, and the server must keep track of the user’s current directory as the user wanders about the remote directory tree.

4 b) Web Caching

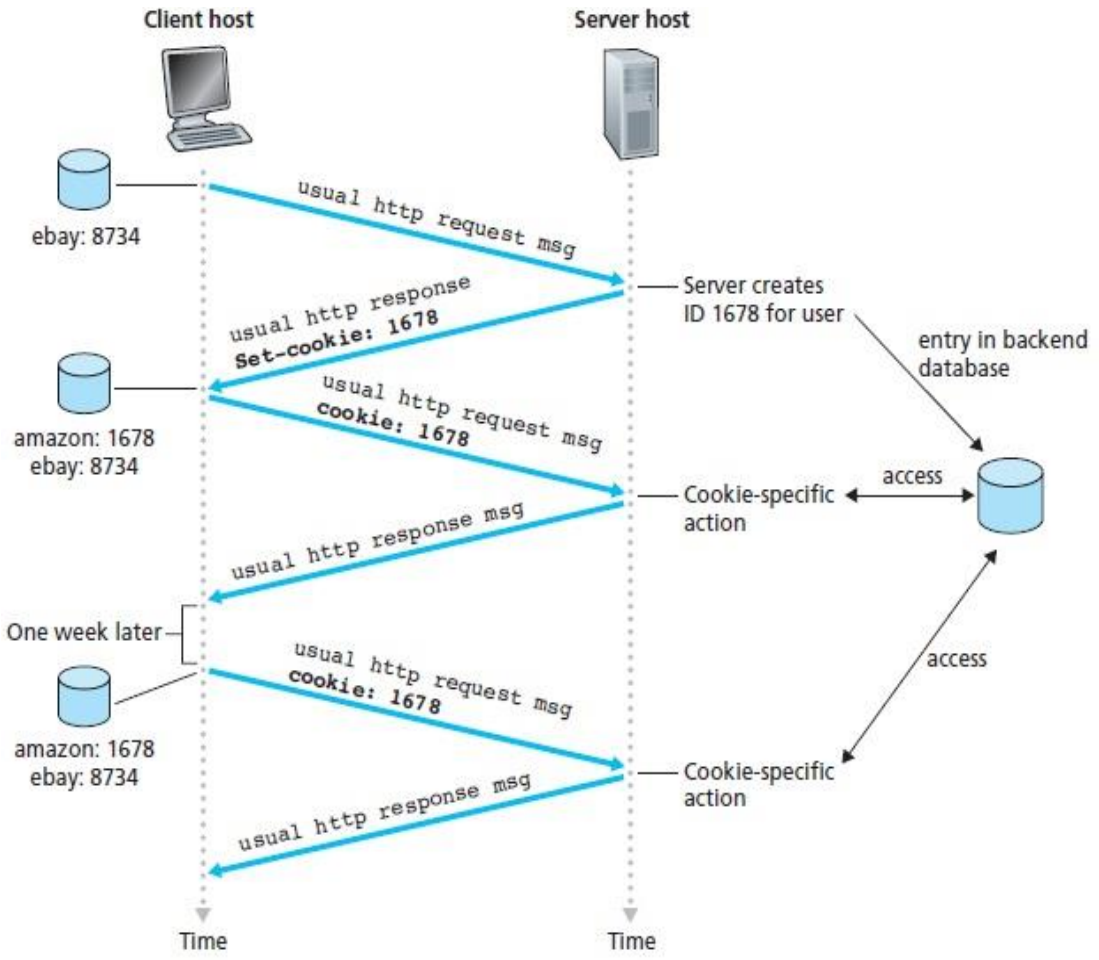


- Ex: Suppose a browser is requesting the object http://www.someschool.edu/campus.gif. Here is what happens:
- The browser establishes a TCP connection to the Web cache and sends an HTTP request for the object to the Web cache.
- The Web cache checks to see if it has a copy of the object stored locally. 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 institution's access link to the Internet.

4 a). **Cookies:** Are the small text files maintained at the web sites and web browsers at client end.

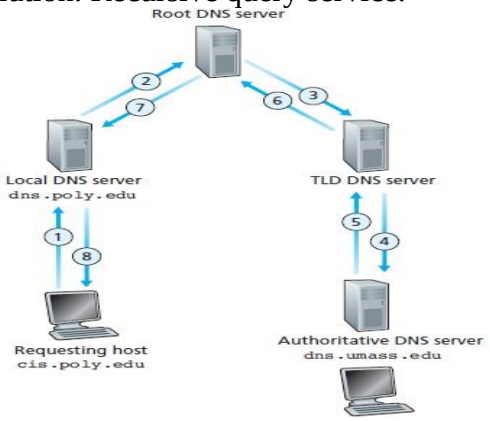


- Cookie technology has four components: (1) a cookie header line in the HTTP response message; (2) a cookie header line in the HTTP request message; (3) a cookie file kept on the user's end system and managed by the user's browser; and (4) a back-end database at the Web site. Using Figure 2.10, let's
- walk through an example of how cookies work. Suppose Susan, 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 she has already visited the eBay site. When the request comes into the Amazon Web server, the server creates a unique identification number and creates an entry in its back-end database that is indexed by the identification number. The Amazon Web server then responds to Susan's browser, including in the HTTP response a Set-cookie: header, which contains the identification number. For example, the header line might be: Set-cookie: 1678
- When Susan's browser receives the HTTP response message, it sees the Setcookie: 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. Note that the cookie file already has an entry for
- eBay, since Susan has visited that site in the past. As Susan continues to browse the Amazon site, each time she requests a Web page, her browser consults her cookie file, extracts her identification number

for this site, and puts a cookie header line that includes the identification number in the HTTP request. Specifically, each of her HTTP requests to the Amazon server includes the header line:

- `Cookie: 1678`
- In this manner, the Amazon server is able to track Susan's activity at the Amazon site. Although the Amazon Web site does not necessarily know Susan's name, it knows exactly which pages user 1678 visited, in which order, and at what times! Amazon uses cookies to provide its shopping cart service Amazon can maintain a list of all of Susan's intended purchases, so that she can pay for them collectively at the end of the session.

5 a).
Solution: Recursive query service:

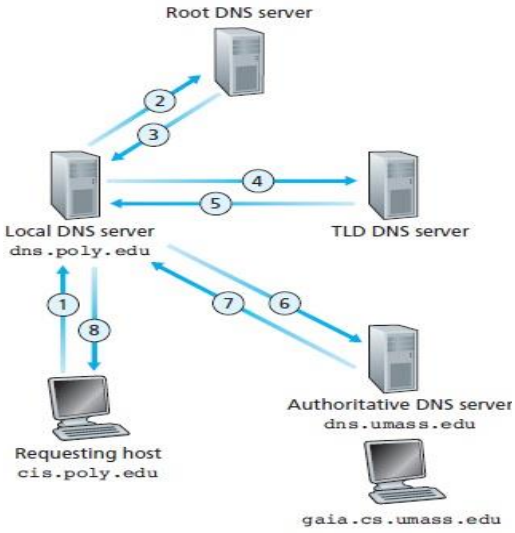


Suppose the host cis.poly.edu desires the IP address of gaia.cs.umass.edu. Also suppose that Polytechnic's local DNS server is called dns.poly.edu and that an authoritative DNS server for gaia.cs.umass.edu is called dns.umass.edu. As shown in above given figure. The host cis.poly.edu first sends a DNS 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.umas .edu, which responds with the IP address of gaia.cs.umass.edu. Our previous example assumed that the TLD server knows the authoritative DNS server for the hostname. In general this not always true. Instead, the TLD server may know only of an intermediate DNS server, which in turn knows the authoritative DNS server for the hostname. For example, suppose again that the University of Massachusetts has a DNS server for the university, called dns.umass.edu. Also suppose that each of the departments at the University of Massachusetts has its own NS server, and that each departmental DNS server

is authoritative for all hosts in the department. In this case, when the intermediate DNS server, dns.umass.edu, receives a query for a host with a hostname ending with cs.umass.edu, it returns

to dns.poly.edu the IP address of dns.cs.umass.edu, which is authoritative for all hostnames ending with cs.umass.edu. The local DNS server dns.poly.edu then sends the query to the authoritative DNS server, which returns the desired mapping to the local DNS server, which in turn returns the mapping to the requesting host. 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 1(a) shows a DNS query chain for which all of the queries are recursive.

Iterative Query service:



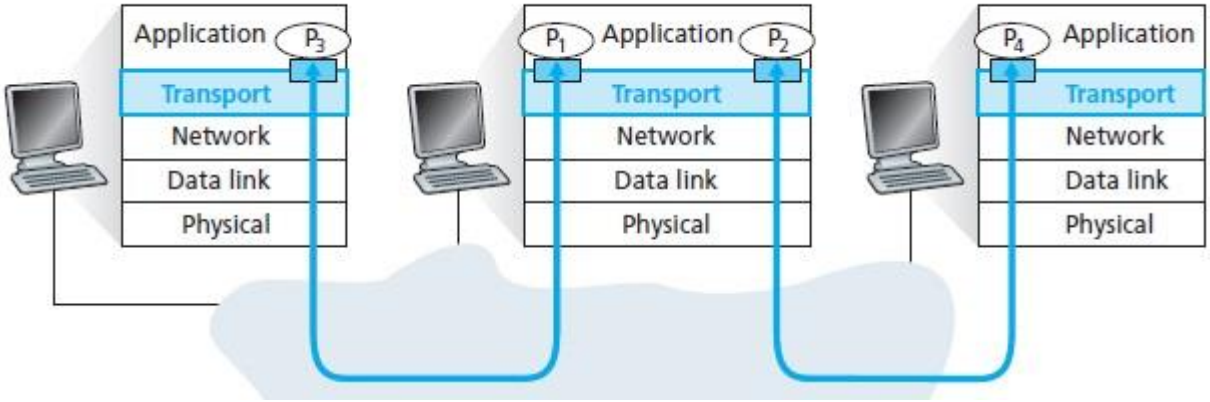
In this case, the request and service proceeds through a series of servers based on the intermediate reply from the server at that level.

5 b).

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

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

7a). Multiplexing and Demultiplexing at transport Layer:



Now let's consider how a receiving host directs an incoming transport-layer segment to the appropriate socket. Each transport-layer segment has a set of fields in the segment for this purpose. At the receiving end, the transport layer examines these fields to identify the receiving socket and then directs the segment to that socket. This job of delivering the data in a transport-layer segment to the correct socket is called demultiplexing. The job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information (that will later be used in demultiplexing) to create segments, and passing the segments to the network layer is called multiplexing. Note that the transport layer in the middle host in Fig- ure 3.2 must demultiplex segments arriving from the network layer below to either process P 1 or P 2 above; this is done by directing the arriving segment's data to the corresponding process's socket. The transport layer in the middle host must alsogather outgoing data from these sockets, form transport-layer segments, and pass these segments down to the network layer. Although we have introduced multiplex-
ing and demultiplexing in the context of the Internet transport protocols, it's impor- tant to realize that they are concerns whenever a single protocol at one layer (at the transport layer or elsewhere) is used by multiple protocols at the next higher layer. To illustrate the demultiplexing job, recall the household analogy in the previ ous section. Each of the kids is identified by his or her name. When Bill receives a batch of mail from

the mail carrier, he performs a demultiplexing operation by observing to whom the letters are addressed and then hand delivering the mail to his brothers and sisters. Ann performs a multiplexing operation when she collects letters from her brothers and sisters and gives the collected mail to the mail person.

5b)

3.3.2 UDP Checksum

The UDP checksum provides for error detection. That is, the checksum is used to determine whether bits within the UDP segment have been altered (for example, by noise in the links or while stored in a router) as it moved from source to destination. UDP at the sender side performs the 1s complement of the sum of all the 16-bit words in the segment, with any overflow encountered during the sum being wrapped around. This result is put in the checksum field of the UDP segment. Here we give a simple example of the checksum calculation. You can find details about efficient implementation of the calculation in RFC 1071 and performance over real data in [Stone 1998; Stone 2000]. As an example, suppose that we have the follow-
ing three 16-bit words:

0110011001100000

0101010101010101

1000111100001100

The sum of first two of these 16-bit words is

0110011001100000

0101010101010101

1011101110110101

Adding the third word to the above sum gives

1011101110110101

1000111100001100

0100101011000010

Note that this last addition had overflow, which was wrapped around. The 1s complement is obtained by converting all the 0s to 1s and converting all the 1s to 0s. Thus the 1s complement of the sum 0100101011000010 is 1011010100111101, which becomes the checksum. At the receiver, all four 16-bit words are added, including the checksum. If no errors are introduced into the packet, then clearly the
sum at the receiver will be 1111111111111111. If one of the bits is a 0, then we know that errors have been introduced into the packet.

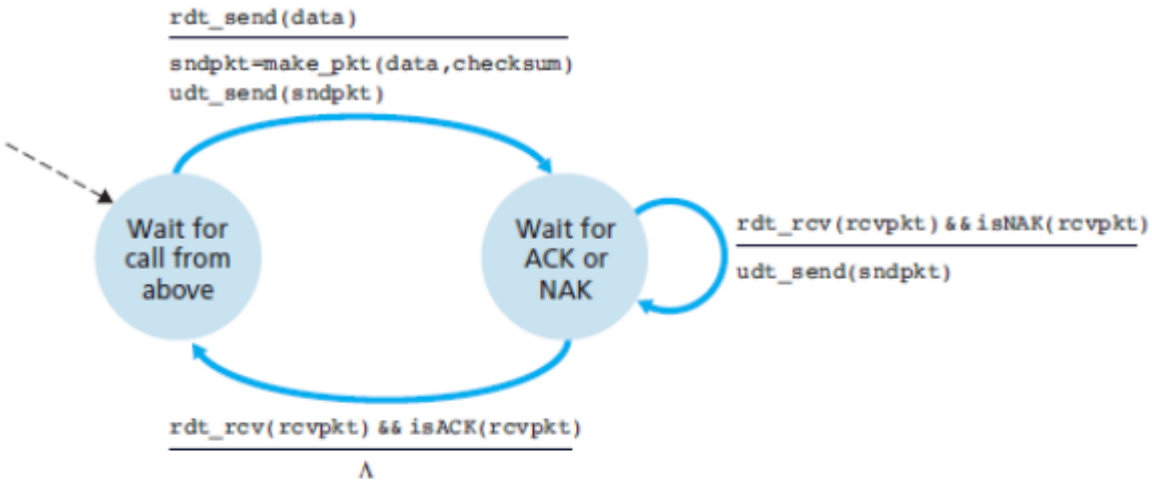
6a)

Transport services to applications: A transport-layer protocol provides for **logical communication** between application processes running on different hosts. By *logical communication*, we mean that from an application's perspective, it is as if the hosts running the processes were directly connected; in reality, the hosts may be on opposite sides of the planet, connected via numerous routers and a wide range of link types. Application processes use the logical communication provided by the transport layer to send messages to each other, free from the worry of the details of the physical infrastructure used to carry these messages. Transport-layer protocols are implemented in the end systems but not in network routers. On the sending side, the transport layer converts
the application-layer messages it receives from a sending application process into transport-layer packets, known as transport-layer **segments** in Internet terminology. This is done by (possibly) breaking the application messages into smaller chunks and adding a transport-layer header to each chunk to create the transport-layer segment. The transport layer then passes the segment to the network layer at the sending end system, where the segment is encapsulated within a network-layer packet (a datagram) and sent to the destination. It's important to note that network routers act only on the network-layer fields of the datagram; that is, they do not examine the fields of the transport-layer segment encapsulated with the datagram. On the receiving side, the network layer extracts the transport-layer segment from the datagram and passes
the segment up to the transport layer. The transport layer then processes the received segment, making the data in the segment available to the receiving application. More than one transport-layer protocol may be available to network applications. For example, the Internet has two protocols—TCP and UDP. Each of these protocols provides a different set of transport-layer services to the invoking application.

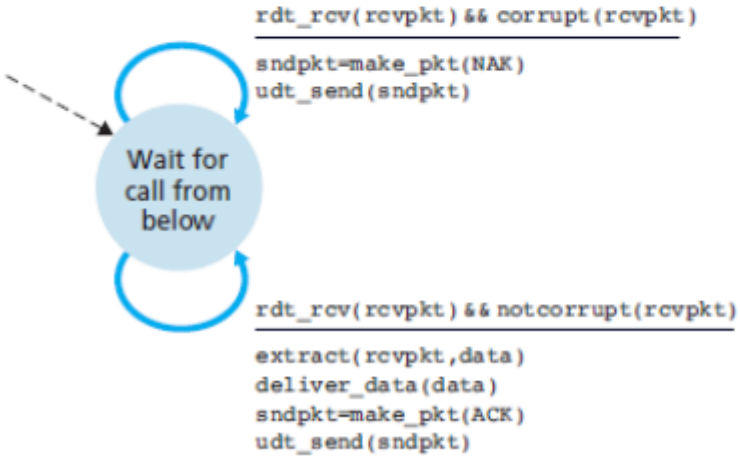
8a). Reliable data transfer protocol : rdt 2.0

Rdt 2.0 Sender:

A more realistic model of the underlying channel is one in which bits in a packet may be corrupted. Three additional protocol capabilities are required in (Automatic Repeat Request) ARQ protocols to handle the presence of bit errors: • Error detection. First, a mechanism is needed to allow the receiver to detect when bit errors have occurred. • Receiver feedback. Since the sender and receiver are typically executing on different end systems, possibly separated by thousands of miles, the only way for the sender to learn of the receiver's view of the world is for the receiver to provide explicit feedback to the sender. The positive (ACK) and negative (NAK) acknowledgment replies in the message-dictation scenario are examples of such feedback. • Retransmission. A packet that is received in error at the receiver will be retransmitted by the sender



a. rdt2.0: sending side



b. rdt2.0: receiving side

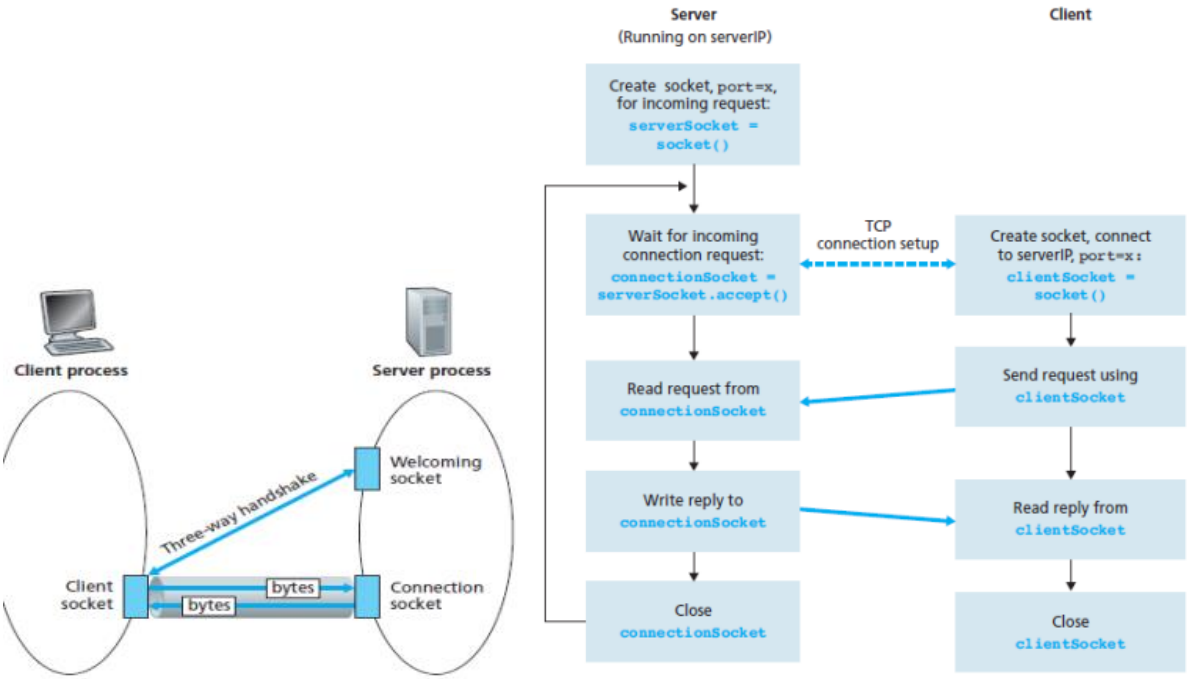
The sender side has two states. In the leftmost state, the send-side protocol is waiting for data to be passed down from the upper layer. • When the rdt_send(data) event occurs, the sender will create a packet (sndpkt) containing the data to be sent, along with a packet checksum and then send the packet via the udt_send(sndpkt) operation. • In the rightmost state, the sender protocol is waiting for an ACK or a NAK packet from the receiver. If an ACK packet is received (the notation rdt_rcv(rcvpkt) && isACK (rcvpkt)), the sender knows that the most recently transmitted packet has been received correctly and thus the protocol returns to the state of waiting for data from the upper layer. • If a NAK is received, the protocol retransmits the last packet and waits for an ACK or NAK to be returned by the receiver in response to the retransmitted data packet. • When the sender is in the wait-for-ACK-or-NAK state, it cannot get more data from the upper layer; that is, the rdt_send() event can not occur; that will happen only after the sender receives an ACK and leaves this state. Thus, the sender will not send a new piece of data until it is sure that the receiver has correctly received the current packet. Because of this behavior, protocols are known as stop-and-wait protocols.

8 b) Difference between TCP and UDP:

Transmission control protocol (TCP)	User datagram protocol (UDP)
TCP is a connection-oriented protocol. Connection-orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data.	UDP is the Datagram oriented protocol. This is because there is no overhead for opening a connection, maintaining a connection, and terminating a connection. UDP is efficient for broadcast and multicast type of network transmission.
TCP is reliable as it guarantees the delivery of data to the destination router.	The delivery of data to the destination cannot be guaranteed in UDP.
TCP provides extensive error checking mechanisms. It is because it provides flow control and acknowledgement of data.	UDP has only the basic error checking mechanism using checksums.
Acknowledgement segment is present.	No acknowledgement segment.
Sequencing of data is a feature of Transmission Control Protocol (TCP). this means that packets arrive in-order at the receiver.	There is no sequencing of data in UDP. If the order is required, it has to be managed by the application layer.
TCP is comparatively slower than UDP.	UDP is faster, simpler, and more efficient than TCP.
Retransmission of lost packets is possible in TCP, but not in UDP.	There is no retransmission of lost packets in the User Datagram Protocol (UDP).
TCP has a (20-60) bytes variable length header.	UDP has an 8 bytes fixed-length header.
TCP is heavy-weight.	UDP is lightweight.
Uses handshakes such as SYN, ACK, SYN-ACK	It’s a connectionless protocol i.e. No handshake
TCP doesn’t support Broadcasting.	UDP supports Broadcasting.
TCP is used by HTTP, HTTPs, FTP, SMTP and Telnet.	UDP is used by DNS, DHCP, TFTP, SNMP, RIP, and VoIP.

6) Socket Programming with TCP

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



TCPClient.py

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

```
print ‘From Server:’, modifiedSentence
clientSocket.close()
```

TCPServer.py:

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