CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | |

Internal Assesment Test - I

Sub:	**Computer Networks**							Code:	**15CS52**
Date:	**18 / 09 / 2017**	Duration:	90 mins	Max Marks:	50	Sem:	**VA,B & C**	Branch:	**CSE**
Note: Answer any 3 questions from Module-1 and 2questions from Module-2									

	Module-1	**Marks**	**OBE**	
			CO	**RBT**
1	Explain different types of Network Application architectures	[5+5]	CO1	L2
2	Explain HTTP request and response message formats with the relevant diagrams	[5+5]	CO1	L2
3	Write short note on a). Cookies b). Web cache	[5+5]	CO1	L1
4 a).	Explain the working of 'Clever Trading Algorithm' in Bit Torrent.	[4]	CO1	L2
b).	What is DHT? Explain the working of Circular DHT	[6]	CO1	L2
5	Design and implement a network application for client server communication using sockets over TCP	[10]	CO2	L3

	Module-2	**Marks**	**OBE**	
			CO	**RBT**
6 a).	With a neat diagram, explain UDP segment structure	[4]	CO2	L2
b).	With an example, explain how to generate UDP checksum and how the generated checksum is used to detect the transmission errors at the receiver end? (Note: Take any four 16 bit data blocks as input)	[6]	CO2	L3
7	Explain the working of rdt2.2 and rdt3.0 with the help of FSMs.	[5+5]	CO2	L2
8	Explain the working of Go-Back-N for pipelined transmission. Use state transition diagram? Why the sliding window size is restricted to 2^k-1 for 'k' bit sequence number in GO-Back-N protocol?	[10]	CO1	L2

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1:	Describe the application, transport, network layer protocols and services	2							1	1			
CO2:	Implement the basic client-server communication model using TCP and UDP protocols	2	2	3					1	1			
CO3:	Design and implementation of IPCs	2	2	3					1	1			
CO4:	Design and implementation of different routing algorithms based on Link State or Distance Vector routing concepts	2	3	3					1	1			
CO5:	Explain the working of Cellular networks based on the real life scenarios	2							1	1	1		
CO6:	Describe and illustrate the concepts of multimedia networking with the guaranteed Qos support	2							1	1	1		

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

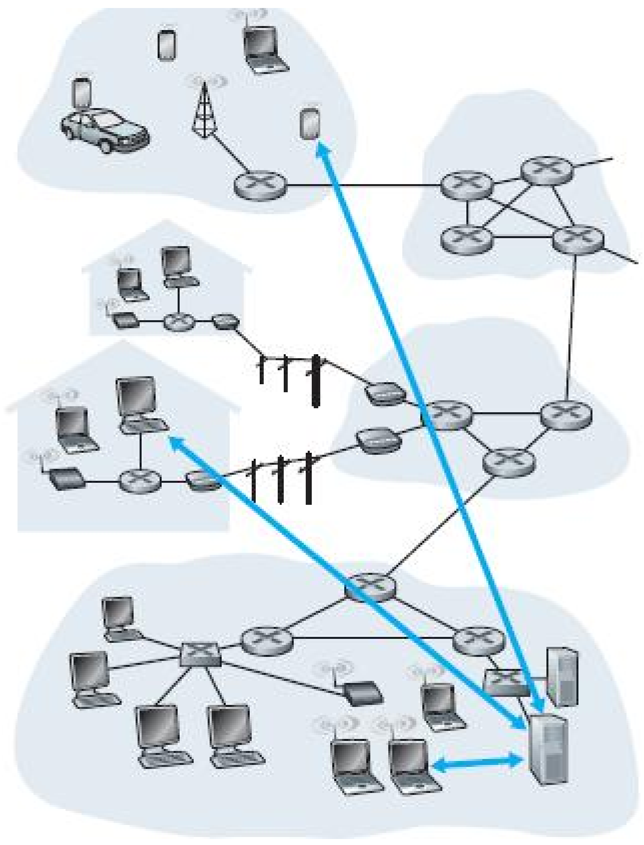
PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*;
PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*;
PO7- *Environment and sustainability*; PO8– *Ethics*; PO9 - *Individual and team work*;
PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*

1. Explain different types of Network Application architectures
Answer:

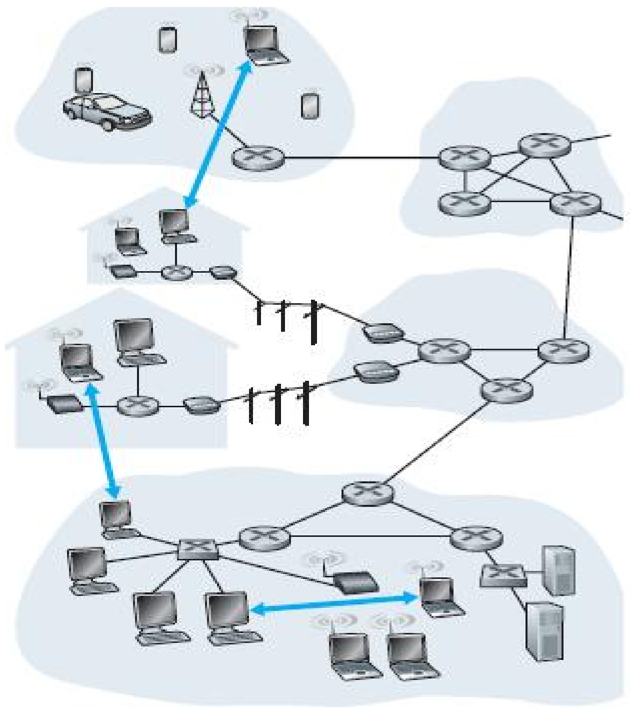
The **application architecture**, dictates how the application is structured over the various end systems. In choosing the application architecture, an application developer will likely draw on one of the two predominant architectural paradigms used in modern network applications

- Client-Server architecture
- Peer-to-peer (P2P) architecture

1. Client Server architecture:



In a **client-server architecture**, there is an always-on host, called the *server*, which services requests from many of the hosts, called *clients*. A classic example is the Web application for which an always-on 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. Note that with the client-server architecture, clients do not directly communicate with each other; for example, in the Web application, two browsers do not directly communicate. Another characteristic of the client-server architecture is that the server has a fixed, well-known address, called an IP address (which we'll discuss soon). 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.
2. Peer-to peer Architecture: The figure given below depicts the architecture of peer to peer model.



In a **P2P architecture**, there is minimal (or no) reliance on dedicated servers in data centers. Instead the application exploits direct communication between pairs of intermittently connected hosts, called *peers*.

The peers are not owned by the service provider, but are instead desktops and laptops controlled by users, with most of the peers residing in homes, universities, and offices. Because the peers communicate without passing through a dedicated server, the **architecture is called peer-to-peer**. Many of today's most popular and traffic-intensive applications are based on P2P architectures. These applications 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).

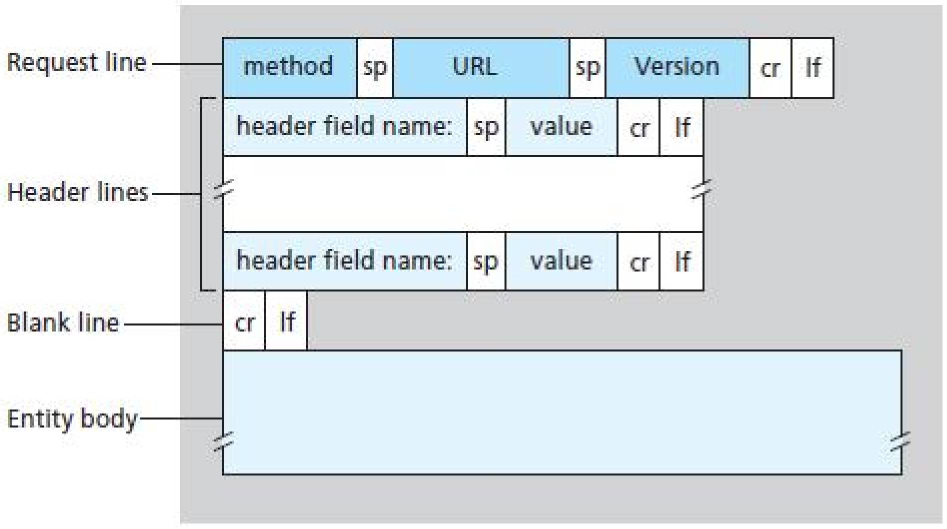
One of the most compelling features of P2P architectures is their **self-scalability**. For example, in a P2P file-sharing application, although each peer generates workload by requesting files, each peer also adds service capacity to the system by distributing files to other peers. P2P architectures are also cost effective, since they normally don't require significant server infrastructure and server bandwidth.

2. Explain HTTP request and response message formats with the relevant diagrams

Answer:

The HTTP specifications [RFC 1945; RFC 2616] include the definitions of the HTTP message formats. There are two types of HTTP messages, request messages and response messages, both of which are discussed below.

HTTP Request Message Format:



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.

Example Message request in HTTP:

GET /somedir/page.html HTTP/1.1

Host: www.someschool.edu

Connection: close

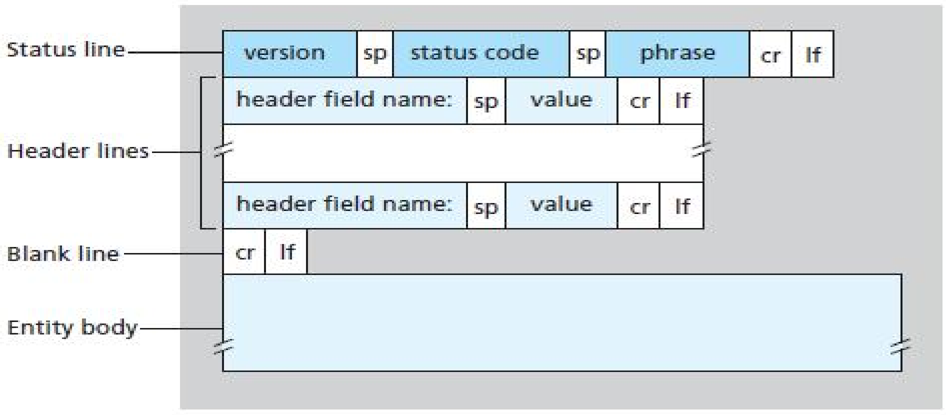
User-agent: Mozilla/5.0

Accept-language: fr

In this **Request line**, the browser is requesting the object /somedir/page.html. The version is self-explanatory; in this example, the browser implements version HTTP/1.1.

Now let's look at the header lines in the example. 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. This header line is useful because the server can actually send different versions of the same object to different types of user agents. Finally, 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. The Accept-language: header is just one of many content negotiation headers available in HTTP.

HTTP Response Message Format: Below we provide a typical HTTP response message.



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 ...). The status line has three fields: the protocol version field, a status code, and a corresponding status message. Entity body contains the actual data.

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 ...)

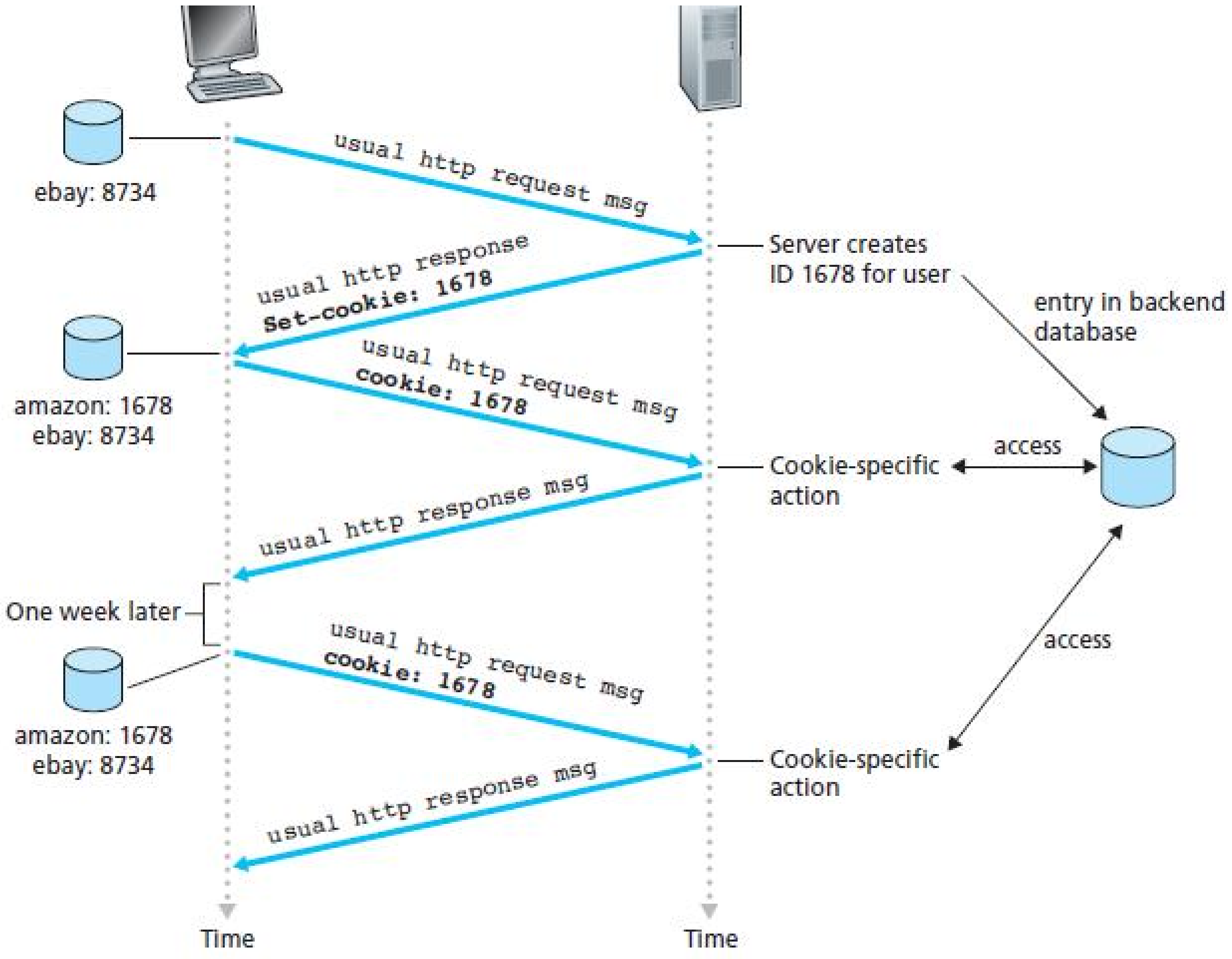
In this example, the status line indicates that the server is using HTTP/1.1 and that everything is OK (that is, the server has found, and is sending, the requested object). Now let's look at the header lines. The server uses the Connection: close header line to tell the client that it is going to close 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. Note that this is not the time when the object was created or last modified; it is 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; 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 Last-Modified: header, which we will soon cover in more detail, is critical for object caching, both in the local client and in network cache servers (also known as proxy servers). 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. (The object type is officially indicated by the Content-Type: header and not by the file extension.)

3. Write short note on a). Cookies b). Web cache

a). Cookies

It is often desirable for a Web site to identify users, either because the server wishes to restrict user access or because it wants to serve content as a function of the user identity. For these purposes, HTTP uses cookies**.** Cookie is a small piece of information the website makes the broser to store in the client system for further reference.

The following diagram depicts how the cookie works.



As shown in the above given 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; and

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

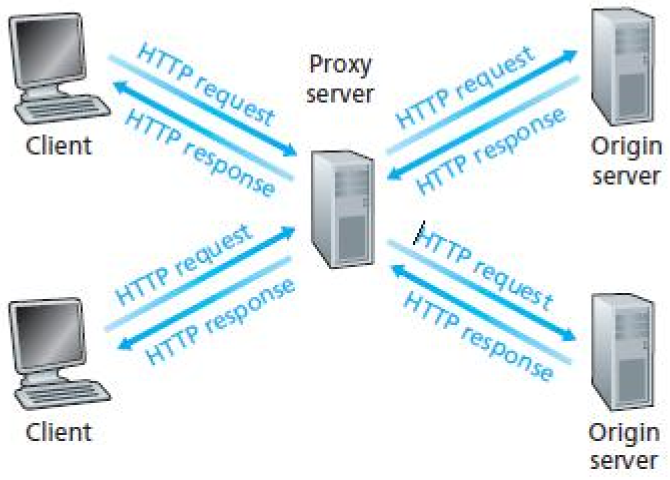
Using the figure given above, let's walk through an example of how cookies work. Suppose Susan, who always accesses the Web using Internet Explorer from her home PC, contacts Amazon.com for the first time. Let us suppose that in the past she has already visited the eBay site. When the request comes into 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.

The first time a user visits a site, the user can provide a user identification (possibly his or her name). During the subsequent sessions, the browser passes a cookie header to the server, thereby identifying the user to the server. Cookies can thus be used to create a user session layer on top of stateless HTTP. For example, when a user logs in to a Web-based e-mail application (such as Hotmail), the browser sends cookie information to the server, permitting the server to identify the user throughout the user's session with the application.

b). Web cache

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. Once a browser is configured, each browser request for an object is first directed to the Web cache.

The following figure illustrates the working of a 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. When it 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.