USN

CMRIT CMR INSTITUTE OF TECHNOLOGY, BENGALURU. ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 1 – September 2019

Sub:	COMPUTER NETWORKS			Sub Code:	17CS52	Branch:	CSE	
Date:	06/06/2019	Duration:	90 min's	Max Marks:	50	Sem / Sec:	A,B,C	OBE

Answer any FIVE FULL Questions		MARKS	CO	RBT
1 (a)	With relevant diagrams, discuss the two basic architectures defined for network applications	[10]	CO1	L2
2 (a)	Explain persistent and non persistent connections of HTTP	[06]	CO1	L2
(b)	Write a short note on Web Cache	[04]	CO1	L1
3 (a)	With example for each, discuss the Request and Response message formats for HTTP	[10]	CO1	L2
4 (a)	Explain two types of DNS query processing services	[06]	CO1	L2
(b)	Describe the working of Distributed Hash Table (DHT)	[04]	CO1	L1
5 (a)	Explain MULTIPLEXING and DEMULTIPLEXING operations at transport layer	[6]	CO2	L2
(b)	With an example, explain how UDP Checksum is computed and used for detecting transmission errors.	[4]	CO2	L2
6 (a)	Explain different services offered by the Transport Layer to the Application Layer	[6]	CO2	L1
(b)	Write a short note on Cookies	[4]	CO2	L1
7 (a)	With the help of FSM, explain the working of RDT 3.0	[10]	CO2	L2
8 (a)	With a neat diagram, explain Go-Back-N sliding window protocol. Discuss why the window size should be less than 2^m-1?	[10]	CO1	L2

USN

CMRIT CMR INSTITUTE OF TECHNOLOGY, BENGALURU. ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 1 – September 2019

Sub:	COMPUTER NETWORKS			Sub Code:	17CS52	Branch:	CSE	
Date:	06/06/2019	Duration:	90 min's	Max Marks:	50	Sem / Sec:	A,B,C	OBE

Answer any FIVE FULL Questions		MARKS	CO	RBT
1 (a)	With relevant diagrams, discuss the two basic architectures defined for network applications	[10]	CO1	L2
2 (a)	Explain persistent and non persistent connections of HTTP	[06]	CO1	L2
(b)	Write a short note on Web Cache	[04]	CO1	L1
3 (a)	With example for each, discuss the Request and Response message formats for HTTP	[10]	CO1	L2
4 (a)	Explain two types of DNS query processing services	[06]	CO1	L2
(b)	Describe the working of Distributed Hash Table (DHT)	[04]	CO1	L1
5 (a)	Explain MULTIPLEXING and DEMULTIPLEXING operations at transport layer	[6]	CO2	L2
(b)	With an example, explain how UDP Checksum is computed and used for detecting transmission errors.	[4]	CO2	L2
6 (a)	Explain different services offered by the Transport Layer to the Application Layer	[6]	CO2	L1
(b)	Write a short note on Cookies	[4]	CO2	L1
7 (a)	With the help of FSM, explain the working of RDT 3.0	[10]	CO2	L2
8 (a)	With a neat diagram, explain Go-Back-N sliding window protocol. Discuss why the window size should be less than 2^m-1?	[10]	CO1	L2

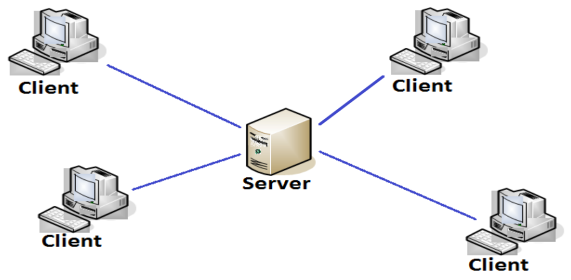
1a) Network application Architectures

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).

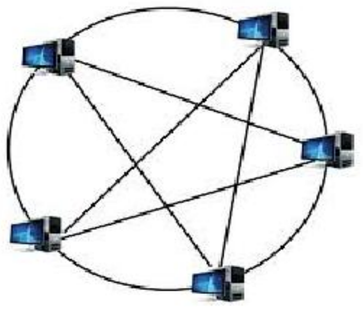
Features:

- **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.

2a). persistent and Non-persistent connections:

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

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

Further suppose the URL for the base HTML file is

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

Here is what happens:

1. The HTTP client process initiates a TCP connection to the server www.someSchool.edu on port number 80, which is the default port number for HTTP. Associated with the TCP connection, there will be a socket at the client and a socket at the server.

2. The HTTP client sends an HTTP request message to the server via its socket. The request message includes the path name /someDepartment/home.index.

3. The HTTP server process receives the request message via its socket, retrieves the object /someDepartment/home.index from its storage (RAM or disk), encapsulates the object in an HTTP response message, and sends the response message to the client via its socket.

4. The HTTP server process tells TCP to close the TCP connection.

5. The HTTP client receives the response message. The TCP connection terminates. The message indicates that the encapsulated object is an HTML file. The client extracts the file from the response message, examines the HTML file, and finds references to the 10 JPEG objects.

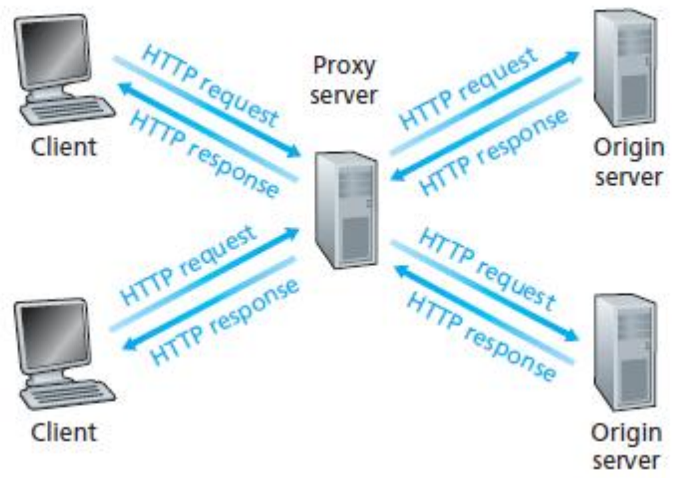
6. The first four steps are then repeated for each of the referenced JPEG objects.

HTTP Persistent Connections:

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

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.



- 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.

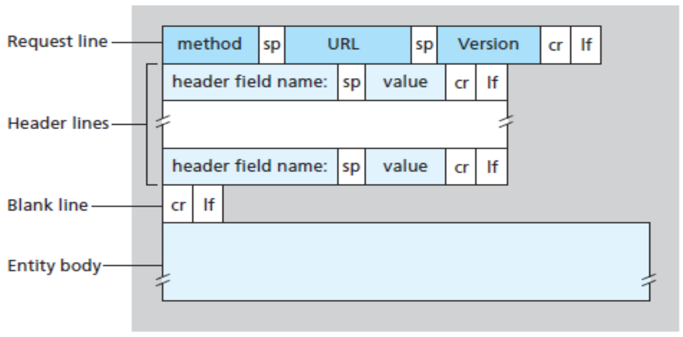
- If the Web cache does not have the object, the Web cache opens a TCP connection to the origin server, that is, to www.someschool.edu. The Web cache then sends an HTTP request for the object into the cache-to-server TCP connection.
- After receiving this request, the origin server sends the object within 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.

3a). 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:

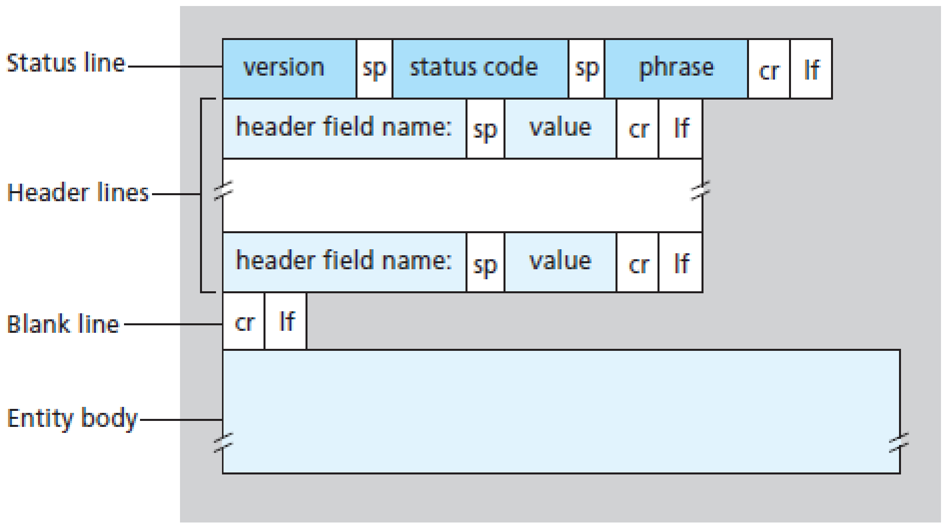
 GET: The GET method is used when the browser requests an object, with the requested object identified in the URL field.

 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.

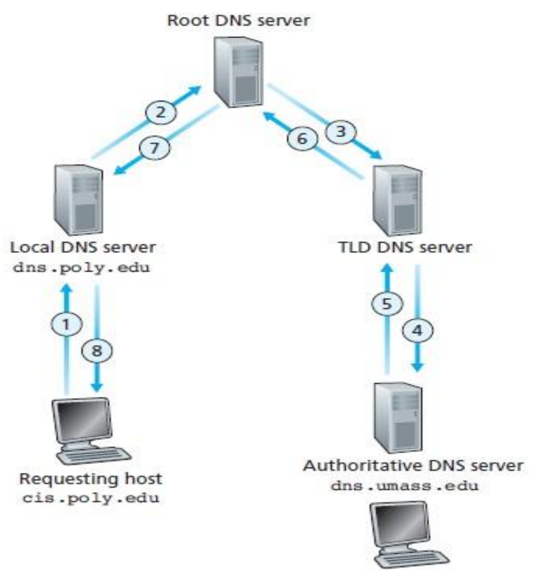
• 505 HTTP Version Not Supported: The requested HTTP protocol version is not supported by the server.

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.

4 a).

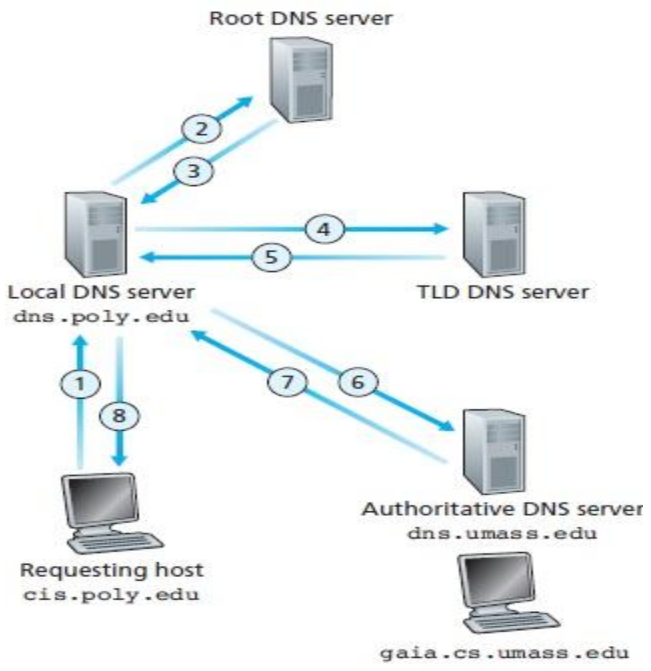
Solution: Recursive query service:



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.

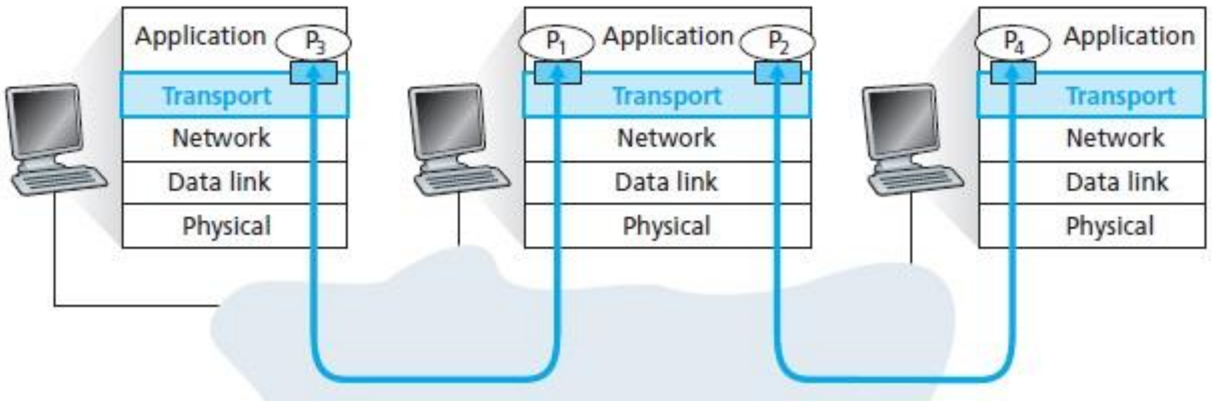
Iterative Query service:



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

4 b).Client-server architecture that stores all the (key, value) pairs in one central server. So in this section, we'll instead consider how to build a distributed, P2P version of this database that 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).

5a). 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 following 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

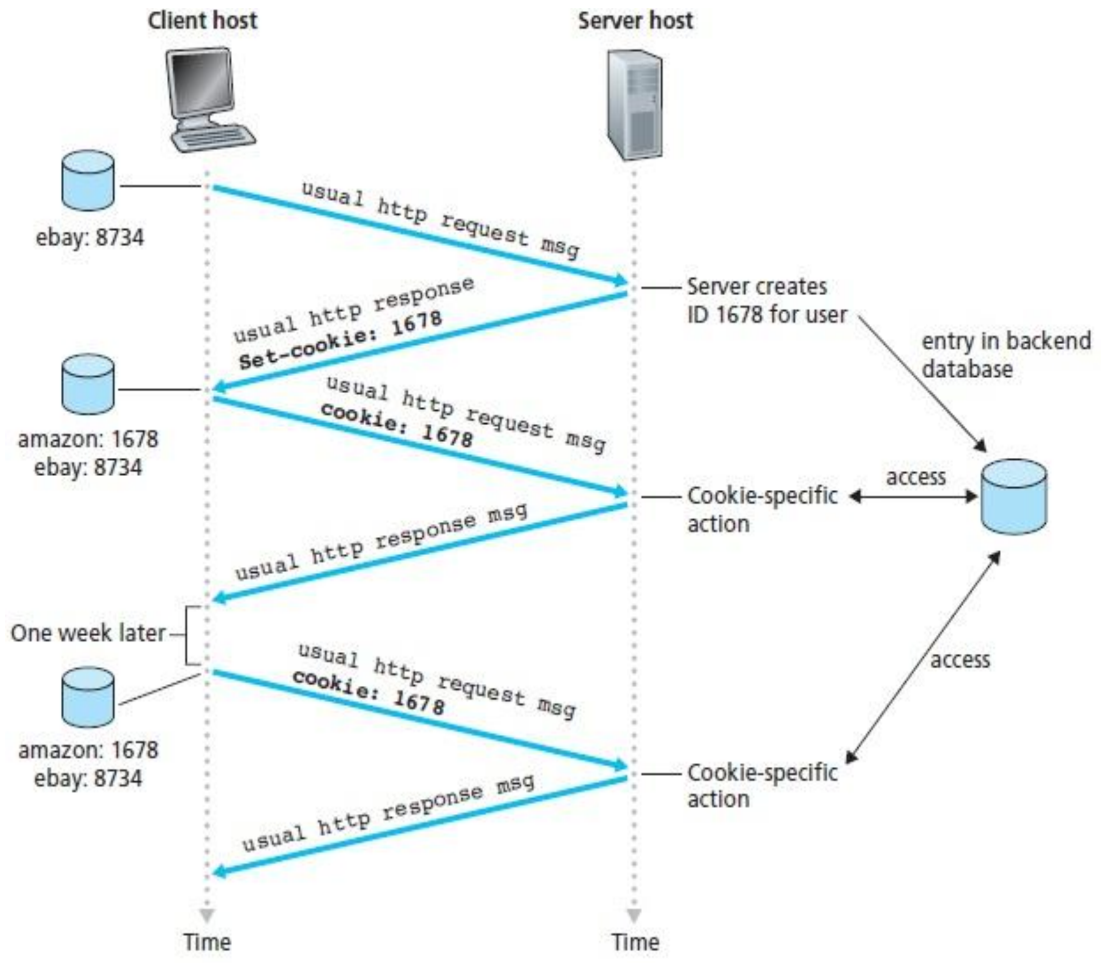
Note that this last addition had overflow, which was wrapped around. The 1s complement is obtained by converting all the 0s to 1s and converting all the 1s to 0s. Thus the 1s complement of the sum 0100101011000010 is 1011010100111101, which becomes the checksum. At the receiver, all four 16-bit words 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.

6b). 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

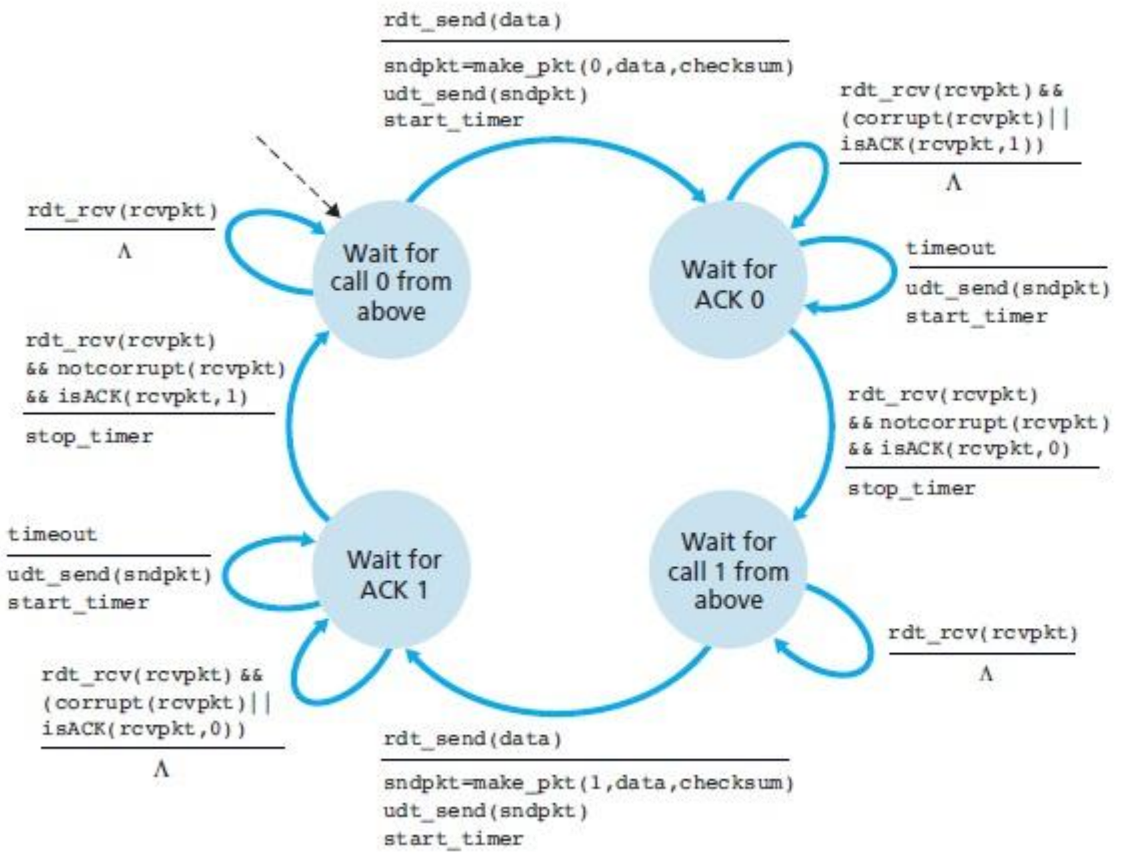
When Susan's browser receives the HTTP response message, it sees the Setcookie: header. The browser then appends a line to the special cookie file that it manages. This line includes the hostname of the server and the identification number in the Set-cookie: header. Note that the cookie file already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

7a). Reliable data transfer protocol : rdt 3.0

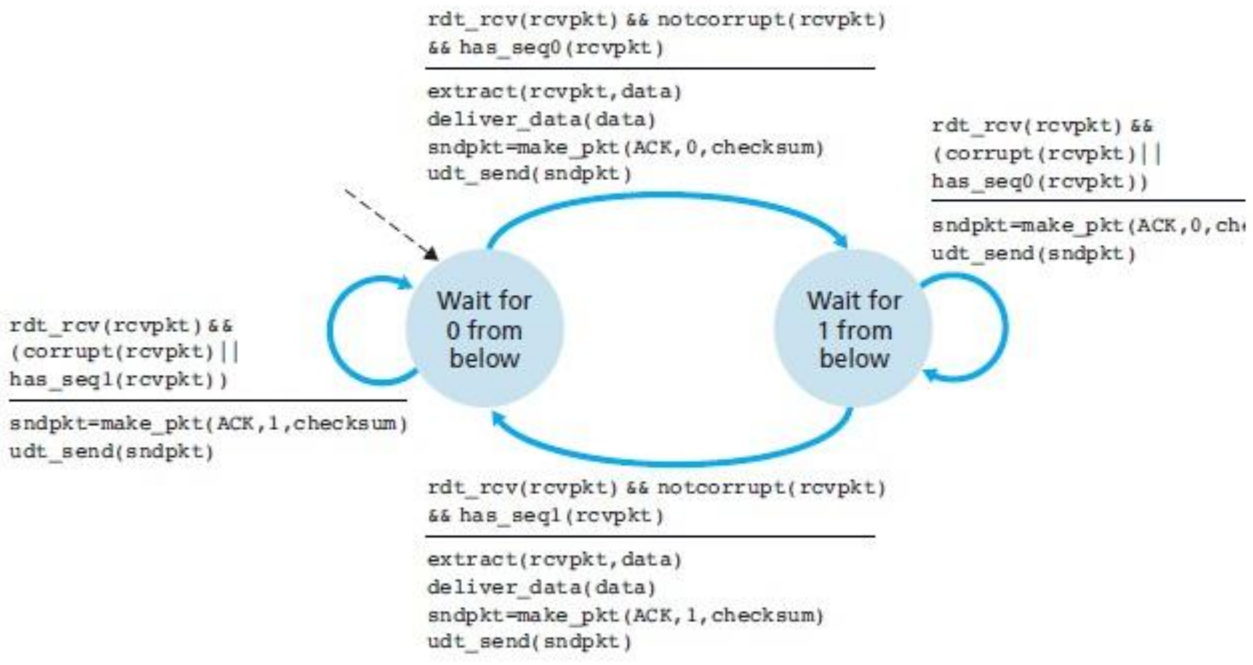
Rdt 3.0 Sender:


rdt_send(data)
sndpkt=make_pkt(0,data,checksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)|| isACK(rcvpkt,1))
Λ
rdt_rcv(rcvpkt)
Λ
Wait for call 0 from above
Wait for ACK 0
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1)
stop_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
stop_timer
timeout
udt_send(sndpkt)
start_timer
Wait for ACK 1
Wait for call 1 from above
rdt_rcv(rcvpkt)
Λ
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)|| isACK(rcvpkt,0))
Λ
rdt_send(data)
sndpkt=make_pkt(1,data,checksum)
udt_send(sndpkt)
start_timer

Above given figure shows the sender FSM for rdt3.0, a protocol that reliably transfers data over a channel that can corrupt or lose packet. This protocol works by using sequence numbers, positive or negative acknowledgements as applied in rdt 2.1 and 2.2. Addition to this, it also defines how to detect packet loss and what to do when packet loss occurs. The use of checksumming, sequence numbers, ACK packets, and retransmissions—the techniques already developed in rdt2.2 will allow us to answer the latter concern. Handling the first concern will require adding a new protocol mechanism.

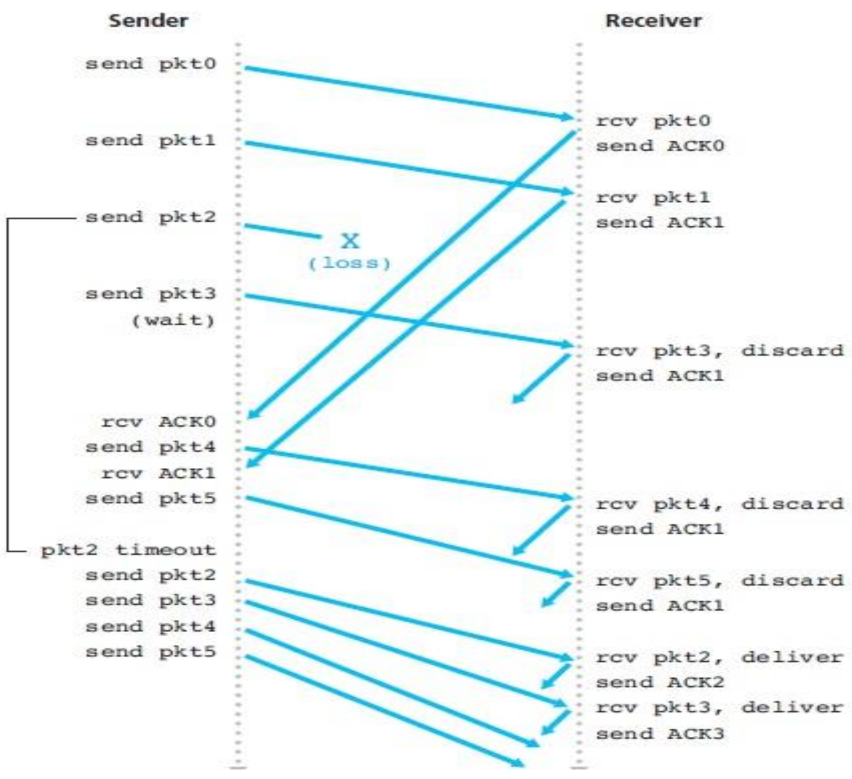
Here, we'll put the burden of detecting and recovering from lost packets on the sender. Suppose that the sender transmits a data packet and either that packet, or the receiver's ACK of that packet, gets lost. In either case, no reply is forthcoming at the sender from the receiver. If the sender is willing to wait long enough so that it is *certain* that a packet has been lost, it can simply retransmit the data packet. You should convince yourself that this protocol does indeed work. But how long must the sender wait to be certain that something has been lost? The sender must clearly wait at least as long as a round-trip delay between the sender and receiver (which may include buffering at intermediate routers) plus whatever amount of time is needed to process a packet at the receiver. In many networks, this worst-case maximum delay is very difficult even to estimate, much less know with certainty. Moreover, the protocol should ideally recover from packet loss as soon as possible; waiting for a worst-case delay could mean a long wait until error recovery is initiated. The approach thus adopted in practice is for the sender to judiciously choose a time value such that packet loss is likely, although not guaranteed, to have happened. If an ACK is not received within this time, the packet is retransmitted. Note that if a packet experiences a particularly large delay, the sender may retransmit the packet even though neither the data packet nor its ACK have been lost. This introduces the possibility of **duplicate data packets** inthe sender-to-receiver channel. Happily, protocol rdt2.2 already has enough functionality (that is, sequence numbers) to handle the case of duplicate packets. From the sender's viewpoint, retransmission is a panacea. The sender does not know whether a data packet was lost, an ACK was lost, or if the packet or ACK was simply overly delayed. In all cases, the action is the same: retransmit. Implementing a time-based retransmission mechanism requires a **countdown timer** that can interrupt the sender after a given amount of time has expired. The sender will thus need to be able to (1) start the timer each time a packet (either a first-time packet or a retransmission) is sent, (2) respond to a timer interrupt (taking appropriate actions), and (3) stop the timer.

Rdt 3.0 Receiver:



The receiver must now include the sequence number of the packet being acknowledged by an ACK message (this is done by including the ACK,0 or ACK,1 argument in make_pkt() in the receiver FSM), and the sender must now check the sequence number of the packet being acknowledged by a received ACK message (this is done by including the 0 or 1 argument in is ACK()in the sender FSM).

8a). In a **Go-Back-N (GBN) protocol**, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, *N*, of unacknowledged packets in the pipeline.

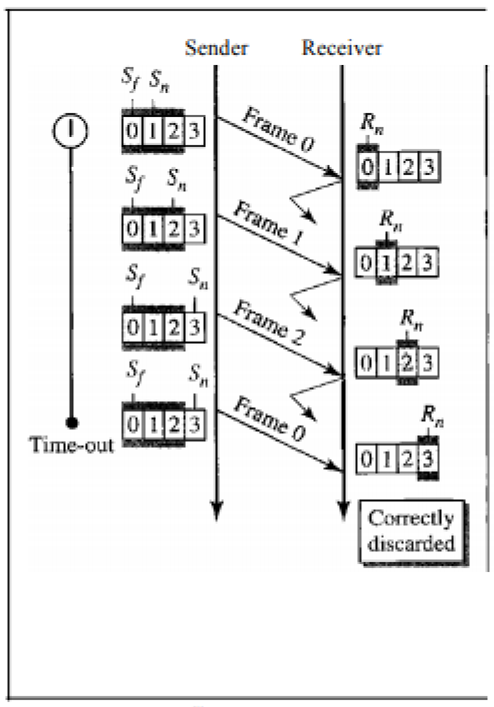


Above given figure shows the operation of the GBN protocol for the case of a window size of four packets. Because of this window size limitation, the sender sends packets 0 through 3 but then must wait for one or more of these packets to be acknowledged before proceeding. As each successive ACK (for example, ACK0 and ACK1) is received, the window slides forward and the sender can transmit one new packet (pkt4 and pkt5, respectively). On the receiver side, packet 2 is lost and thus packets 3, 4, and 5 are found to be out of order and are discarded.

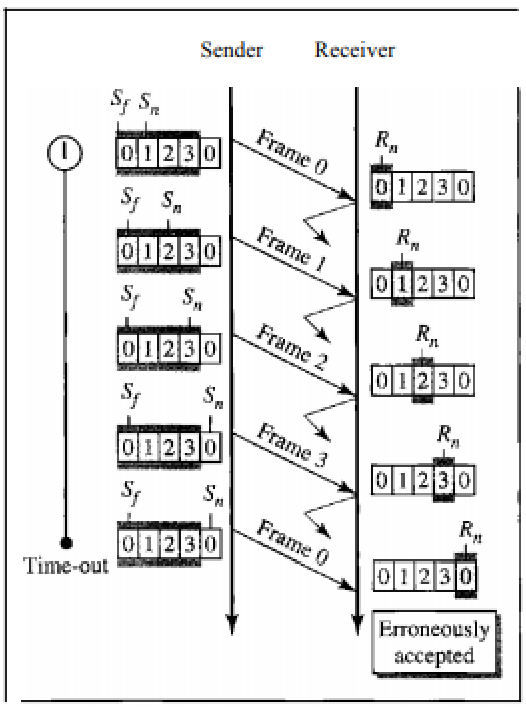
The GBN protocol allows the sender to potentially "fill the pipeline" with packets, thus avoiding the channel utilization problems we noted with stop and wait protocols. There are, however, scenarios in which GBN itself suffers from performance problems. In particular, when the window size and bandwidth-delay product are both large, many packets can be in the pipeline. A single packet error can thus cause GBN to retransmit a large number of packets, many unnecessarily. As the probability of channel errors increases, the pipeline can become filled with these unnecessary retransmissions.

Why the window size is restricted to 2^{m-1}?

Cosider the following case of data transmission using window size <2^{m-1} and window size=2^{m-1}



a. Window size < 2^m



b. Window size = 2^m

We can now show why the size of the send window must be less than 2^k. As an example, we choose k =2, which means the size of the window can be $^{2m-1}$, or 3. Figure compares a window size of 3 against a window size of 4. If the size of the window is 3 (less than 2^2) and all three acknowledgments are lost, the frame ° timer expires and all three frames are resent. The receiver is now expecting frame 3, not frame 0, so the duplicate frame is correctly discarded. On the other hand, if the size of the window is 4 (equal to 2^2) and all acknowledgments are lost, the sender will send a duplicate of frame 0. However, this time the window of the receiver expects to receive frame 0, so it accepts frame 0, not as a duplicate, but as the first frame in the next cycle. This is an error.