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

SCHEME & SOLUTIONS

Internal Assessment Test 1 – Sept. 2018

Sub:	**Computer Networks**					Sub Code:	**15CS52**	Branch:	CSE	
Date:	07/09/2018	Duration:	90 min's	Max Marks:	50	Sem / Sec:	V / A,B,C		OBE	

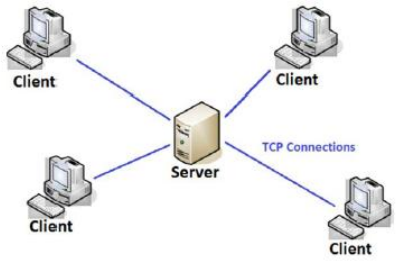
Answer any FIVE FULL Questions	MARKS	CO	RBT
Q1. Explain the different types of Network Application Architectures.	[5+5]	CO1	L2

Q1. Explain the different types of Network Application Architectures.

Two different types : Client-Server and Peer to Peer [5+5]

Client-Server:

- In client-server architecture, there is an always-on host, called the server, w services when it receives requests from many other hosts, called clients.
- Example: In Web application Web server services requests from browsers r client hosts. When a Web server receives a request for an object from a client h responds by sending the requested object to the client host.
- In client-server architecture, clients do not directly communicate with each
- The server has a fixed, well-known address, called an IP address. Because t a fixed, well-known address, and because the server is always on, a client can a contact the server by sending a packet to the server's IP address.
- Some of the better-known applications with a client-server architecture incl FTP, Telnet, and e-mail.



Peer to Peer Architecture:

- In a P2P architecture, there is minimal dependence on dedicated servers in c
- The application employs direct communication between pairs of intermitten hosts, called peers.
- The peers are not owned by the service provider, but are instead desktops ar

controlled by users, with most of the peers residing in homes, universities, and

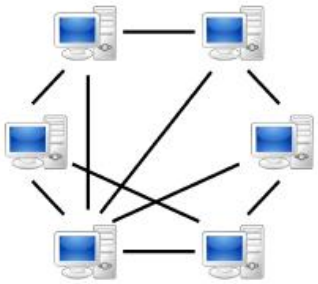
- Many of today's most popular and traffic-intensive applications are based o architectures. These applications include file sharing (e.g., BitTorrent), Internet Telephony (e.g., Skype), and IPTV (e.g., Kankan and PPstream).
- Features:

Self-scalability:

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

Cost effective:

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



Q2. With examples, explain HTTP request and response message formats. [5+5] CO2 L2

Request Header: [5 Marks]

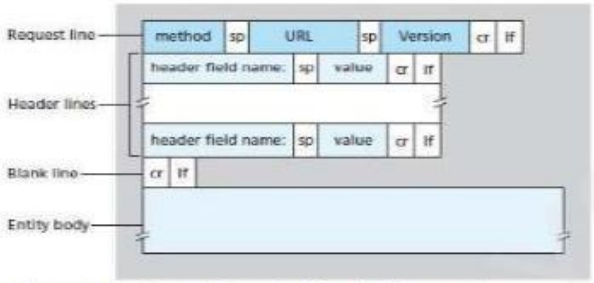


Figure 1.5: General format of an HTTP request-message

- An example of request-message is as follows:

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

- The request-message contains 3 sections (Figure 1.5):
  1) Request-line
  2) Header-line and
  3) Carriage return.
- The first line of message is called the request-line. The subsequent lines are called the header-lines.
- The request-line contains 3 fields. The meaning of the fields is as follows:
  1) Method
     - "GET": This method is used when the browser requests an object from the server.
  2) URL
     - "/somedir/page.html": This is the object requested by the browser.
  3) Version
     - "HTTP/1.1": This is version used by the browser.
- The request-message contains 4 header-lines. The meaning of the header-lines is as follows:
  1) "Host: www.someschool.edu" specifies the host on which the object resides.
  2) "Connection: close" means requesting a non-persistent connection.
  3) "User-agent:Mozilla/5.0" means the browser used is the Firefox.
  4) "Accept-language:eng" means English is the preferred language.
- The method field can take following values: GET, POST, HEAD, PUT and DELETE.
  1) GET is used when the browser requests an object from the server.
  2) POST is used when the user fills out a form & sends to the server.
  3) HEAD is identical to GET except the server must not return a message-body in the response.
  4) PUT is used to upload objects to servers.
  5) DELETE allows an application to delete an object on a server.

Response Header: [5 Marks]

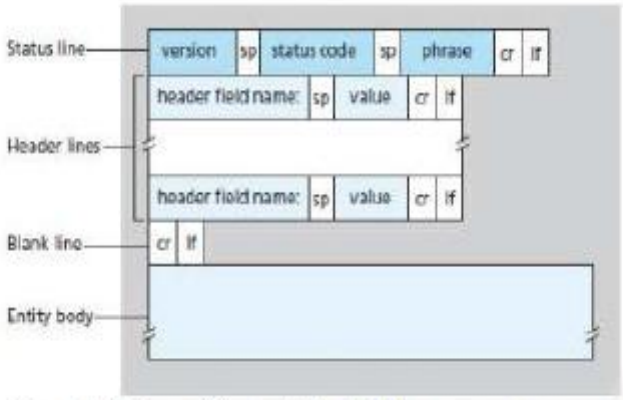


Figure 1.6: General format of an HTTP response-message

- An example of response-message is as follows:

```
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 response-message contains 3 sections (Figure 1.6):
  1) Status line
  2) Header-lines and
  3) Data (Entity body).
- The status line contains 3 fields:
  1) Protocol version
  2) Status-code and
  3) Status message.
- Some common status-codes and associated messages include:
  1) 200 OK: Standard response for successful HTTP requests.
  2) 400 Bad Request: The server cannot process the request due to a client error.
  3) 404 Not Found: The requested resource cannot be found.
- The meaning of the Status line is as follows:
  "HTTP/1.1 200 OK": This line indicates the server is using HTTP/1.1 & that everything is OK.
- The response-message contains 6 header-lines. The meaning of the header-lines is as follows:
  1) Connection: This line indicates browser requesting a non-persistent connection.
  2) Date: This line indicates the time & date when the response was sent by the server.
  3) Server: This line indicates that the message was generated by an Apache Web-server.
  4) Last-Modified: This line indicates the time & date when the object was last modified.
  5) Content-Length: This line indicates the number of bytes in the sent-object.
  6) Content-Type: This line indicates that the object in the entity body is HTML text.

Q3. Write short notes on a) Cookies b) Web Caching [5+5] CO2 L2

Cookies – 5 Marks

Web Caching- 5 Marks

1.2.4 User-Server Interaction: Cookies

- 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.

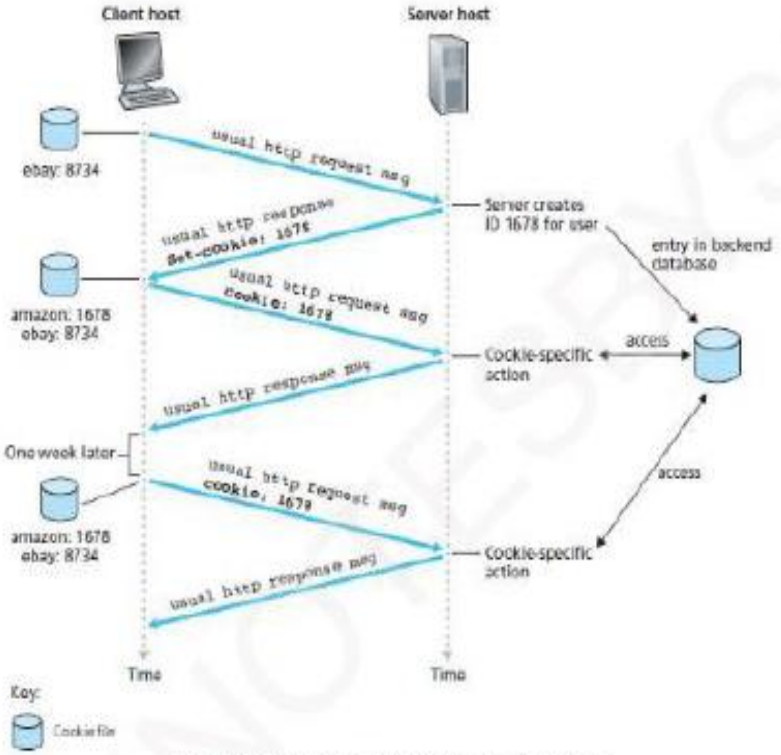


Figure 1.7: Keeping user state with cookies

- Here is how it works (Figure 1.7):
  1) When a user first time visits a site, the server
     → creates a unique identification number (1678) and
     → creates an entry in its back-end database by the identification number.
  2) The server then responds to user's browser.
  ➢ HTTP response includes Set-cookie: header which contains the identification number (1678)
  3) The browser then stores the identification number into the cookie-file.
  4) Each time the user requests a Web-page, the browser
     → extracts the identification number from the cookie file, and
     → puts the identification number in the HTTP request.
  5) In this manner, the server is able to track user's activity at the web-site.

1.2.5 Web Caching

- A Web-cache is a network entity that satisfies HTTP requests on the behalf of an original Web-server.
- The Web-cache has disk-storage.
- The disk-storage contains copies of recently requested-objects.

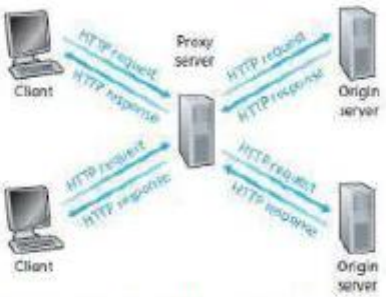
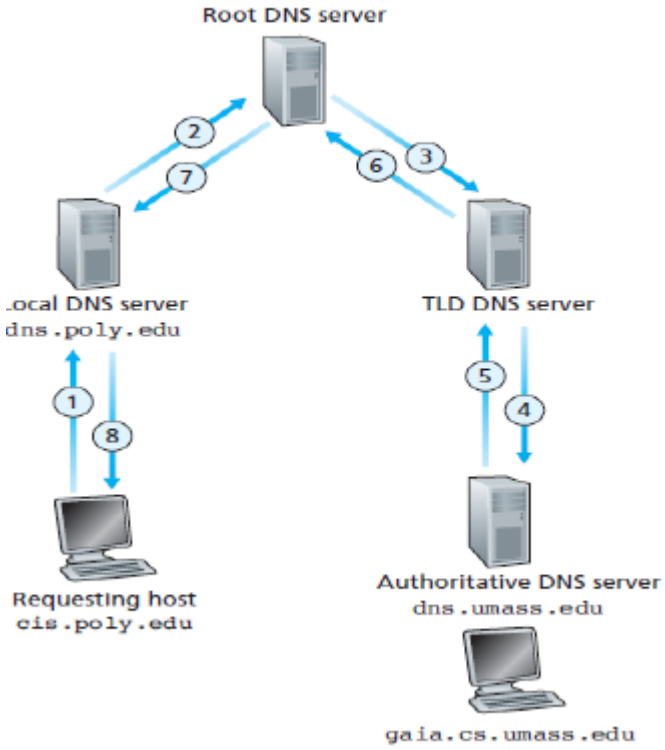


Figure 1.8: Clients requesting objects through a Web-cache (or Proxy Server)

- Here is how it works (Figure 1.8):
  1) The user's HTTP requests are first directed to the web-cache.
  2) If the cache has the object requested, the cache returns the requested-object to the client.
  3) If the cache does not have the requested-object, then the cache
     → connects to the original server and
     → asks for the object.
  4) When the cache receives the object, the cache
     → stores a copy of the object in local-storage and
     → sends a copy of the object to the client.
- A cache acts as both a server and a client at the same time.
  1) The cache acts as a server when the cache
     → receives requests from a browser and
     → sends responses to the browser.
  2) The cache acts as a client when the cache
     → requests to an original server and
     → receives responses from the origin server.
- Advantages of caching:
  1) To reduce response-time for client-request.
  2) To reduce traffic on an institution's access-link to the Internet.
  3) To reduce Web-traffic in the Internet.

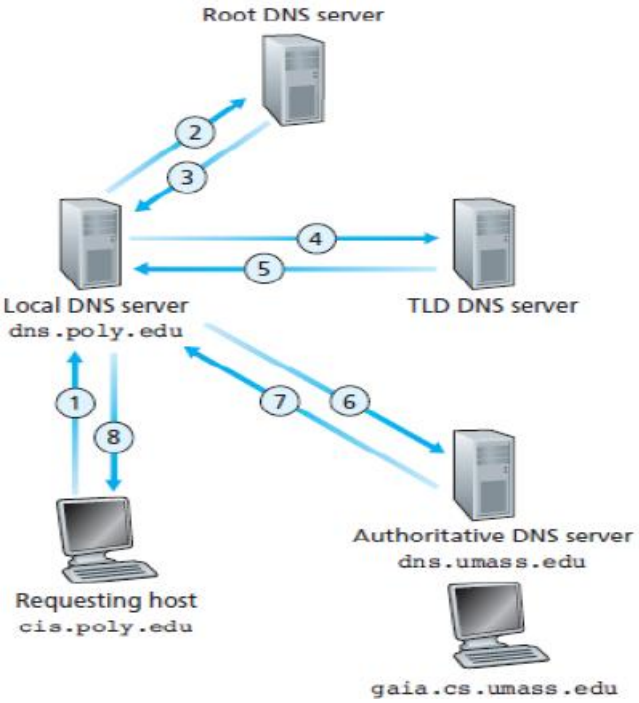
Q4. a) Describe the iterative and recursive services for query resolution provided by DNS. [3+3] CO2 L2 L2

Iterative Queries:



Here DNS query is sent to local DNS server then to root server, then to TLD server and finally to authoritative DNS server. DNS response arrives in the reverse order.

Recursive Queries:



Here DNS query will be sent to Local DNS server, then to root server. Root server sends the IP address of TLD server. Now local DNS server sends query to TLD DNS server. TLD DNS server sends the IP address of authoritative DNS server to local DNS server. Now Local DNS server sends query to authoritative DNS server. Authoritative DNS server sends the IP address of host to local DNS server. Local DNS server sends it to the host.

b) What is DHT? Explain the working of DHT. [1+3]

Distributed Hash Tables

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.

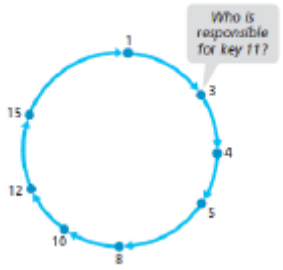
 This also require each key to be an integer in the same range.

 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 $[0, 2^n-1]$.

Problem of storing the (key, value) pairs in the DHT:

 The central issue here is defining a rule for assigning keys to peers. Given that each peer has an integer identifier and that each key is also an integer in the same range, a natural

approach is to assign each (key, value) pair to the peer whose identifier is the closest to the key.



Q5. Implement a network application for client server communication using sockets over TCP. [5+5] CO3 L3

```
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()
```

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()
```

Q6. a) Explain the connection-oriented multiplexing and de-multiplexing. [6] CO2 L2

2.2.3 Connection Oriented Multiplexing and Demultiplexing

- Each TCP connection has exactly 2 end-points. (Figure 2.4).
- Thus, 2 arriving TCP segments with different source-port-nos will be directed to 2 different sockets, even if they have the same destination-port-no.
- A TCP socket is identified by a four-tuple:
  1) Source IP address
  2) Source-port-no
  3) Destination IP address &
  4) Destination-port-no.

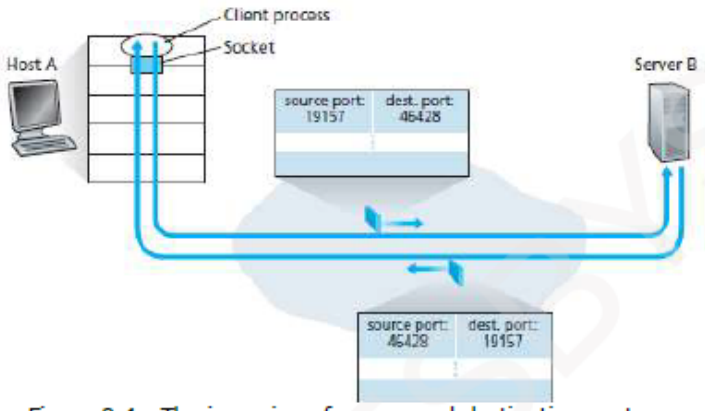


Figure 2.4: The inversion of source and destination-port-nos

- The server-host may support many simultaneous connection-sockets.
- Each socket will be
  → attached to a process.
  → identified by its own four tuple.
- When a segment arrives at the host, all 4 fields are used to direct the segment to the appropriate socket. (i.e. Demultiplexing).

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 blocks as input) [4]

Step1: Add all the data elements using binary addition (Modulo-2 addition). If you get extra bit wrap it.

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

Step 2: Take 1s complement of the result.

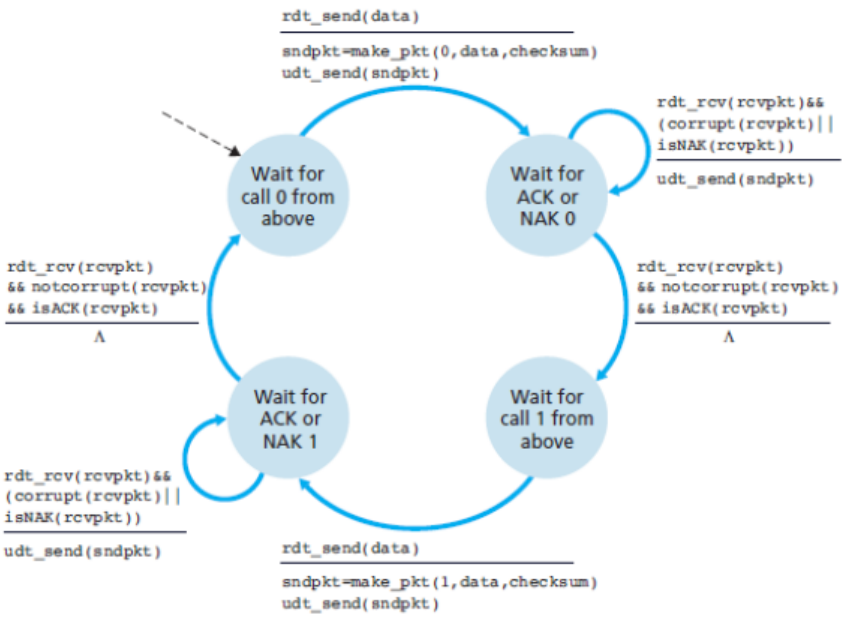
The 1s complement is obtained by converting all the 0s to 1s and converting all the 1s to 0s. Thus the 1s complement of the sum 0100101011000010 is 1011010100111101, which becomes the checksum.

Step 3: Data along with checksum is transmitted to receiver.

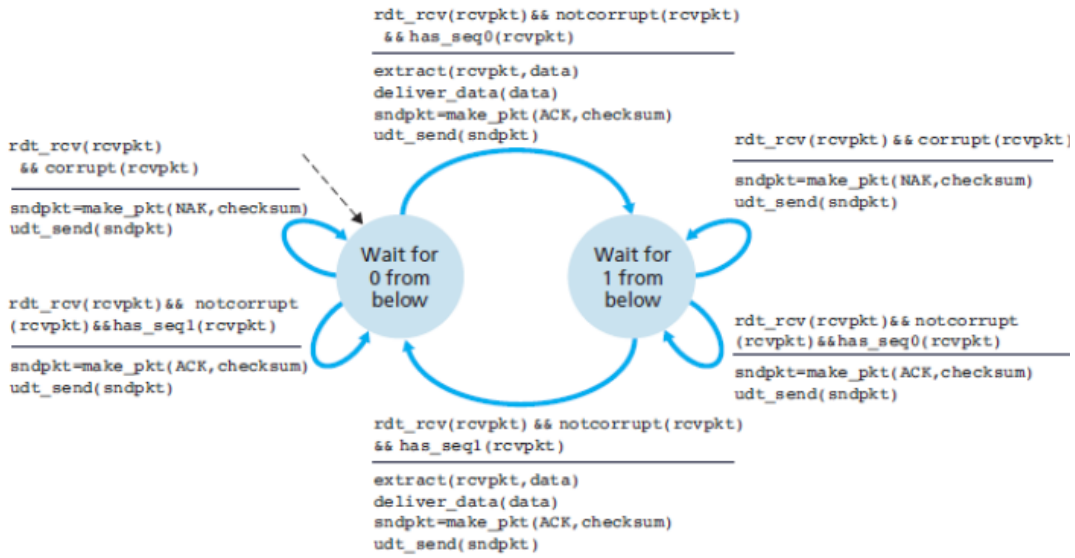
Step 4: at the receiver side add all the data and checksum using binary addition. Wrap the extra bit and take 1s complement of the result. This will be the checksum. If checksum is all 0's receiver has received error free data otherwise it has received corrupted data.

Q7. a) Illustrate rdt 2.1 (sender and receiver) with a neat labeled FSM diagram. [4+2] CO2 L2

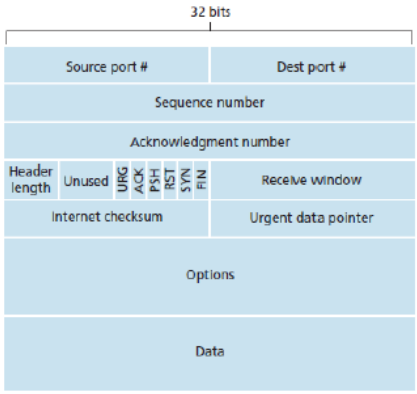
Sender:



Receiver:



b) With a neat diagram, describe TCP Segment Structure. [4]



- The TCP segment consists of header fields and a data field.
- The data field contains a chunk of application data.
- The minimum length of TCP header is 20 bytes.
- The header includes **source and destination port numbers**, which are used for multiplexing/demultiplexing data from/to upper-layer applications.
- The header includes a **checksum field** for error detection.

- The 32-bit **sequence number field** and the 32-bit **acknowledgment number** field are used by the TCP sender and receiver in implementing a reliable data transfer service.
- The 16-bit **receive window** field is used for flow control. It is used to indicate the number of bytes that a receiver is willing to accept.
- The 4-bit **header length** field specifies the length of the TCP header in 32-bit words. The TCP header can be of variable length due to the TCP options field.
- The **optional and variable-length options field** is used when a sender and receiver negotiate the maximum segment size (MSS) or as a window scaling factor for use in high-speed networks.
- The **flag field** contains 6 bits.
- The **ACK bit** is used to indicate that the value carried in the acknowledgment field is valid; that is, the segment contains an acknowledgment for a segment that has been successfully received.
- The **RST, SYN, and FIN bits** are used for connection setup and teardown.
- Setting the **PSH bit** indicates that the receiver should pass the data to the upper layer immediately.
- Finally, the **URG bit** is used to indicate that there is data in this segment that the sending-side upper-layer entity has marked as "urgent."
- The location of the last byte of this urgent data is indicated by the 16-bit **urgent data pointer field**. TCP must inform the receiving- side upper-layer entity when urgent data exists and pass it a pointer to the end of the urgent data.

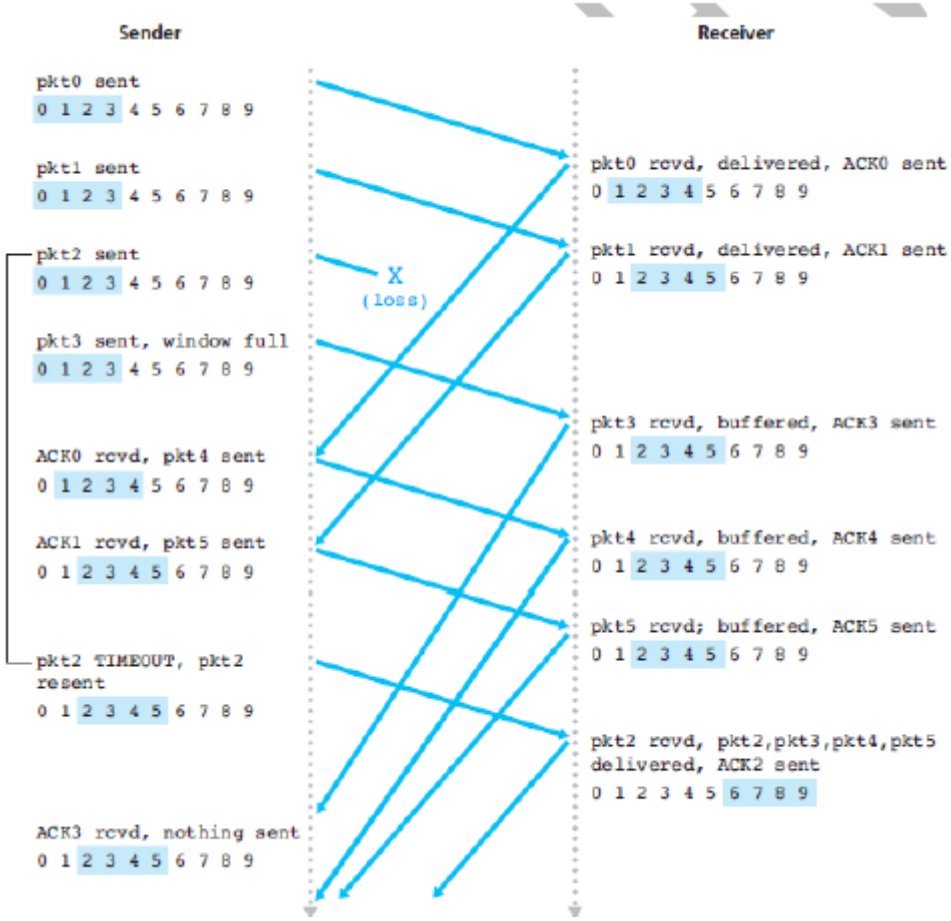
8. Explain Selective Repeat (SR) Protocol. [10]

2.4.4 Selective Repeat (SR)

- Limitation of GBN: 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.
- As the name suggests, selective-repeat protocols avoid unnecessary retransmissions by having the sender retransmit only those packets that it suspects were received in error (that is, were lost or corrupted) at the receiver.
- This individual, as needed, retransmission will require that the receiver individually acknowledge correctly received packets.
- A window size of N will again be used to limit the number of outstanding, unacknowledged packets in the pipeline.
- The SR receiver will acknowledge a correctly received packet whether or not it is in order. Out-of-order packets are buffered until any missing packets (that is, packets with lower sequence numbers) are received, at which point a batch of packets can be delivered in order to the upper layer.

1. *Data received from above.* When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise it is either buffered or returned to the upper layer for later transmission, as in GBN.
2. *Timeout.* Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers [Varghese 1997].
3. *ACK received.* If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to `send_base`, the window base is moved forward to the unacknowledged packet with the smallest sequence number. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

SR sender events and actions



1. *Packet with sequence number in* `[rcv_base, rcv_base+N-1]` *is correctly received.* In this case, the received packet falls within the receiver's window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window (`rcv_base` in Figure 3.22), then this packet, and any previously buffered and consecutively numbered (beginning with `rcv_base`) packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer. As an example, consider Figure 3.26. When a packet with a sequence number of `rcv_base=2` is received, it and packets 3, 4, and 5 can be delivered to the upper layer.
2. *Packet with sequence number in* `[rcv_base-N, rcv_base-1]` *is correctly received.* In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.
3. *Otherwise.* Ignore the packet.

SR receiver events and actions