Computer Networks (IT308)

MidSem Exam Solution

Q1.

A. Socketaddr:

struct sockaddr contains variables for family (AF_NET/AF_NET6), port no, IP address information about the connection. ..... 0.5 Marks

B. Server Pseudo Code:

a. **Socket creation:**

int sockfd = socket(**domain**, **type**, protocol)

sockfd: socket descriptor, an integer (like a file-handle)

domain: integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)

type: SOCK_STREAM for TCP or SOCK_DGRAM for UDP

protocol: Protocol value for Internet Protocol(IP), which is 0.

b. Bind:

int bind(**int sockfd**, **const struct sockaddr *addr**, socklen_t addrlen);

bind function binds the socket to the address and port number specified in addr.

c. Listen:

int listen(**int sockfd**, **int pending**);

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection.

The pending defines the maximum length of the queue of pending connections.

d. Accept:

int new_socket= accept(**int sockfd**, **struct sockaddr_in *addr**, socklen_t *addrlen);

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd and creates a new connected socket with new file descriptor.

e. receive/send

valread = receive(**new_socket** , buffer, 1024);

send(**new_socket** , hello , strlen(hello) , 0);

C. Client Pseudo Code:

a. Socket Creation:

int sockfd = socket(**domain**, **type**, protocol)

b. Connect:

int connect(**int sockfd**, **const struct sockaddr *addr**, socklen_t addrlen);

c. receive/send

valread = receive(**new_socket** , buffer, 1024);
send(**new_socket** , hello , strlen(hello) , 0); …..... 3.5 Mark

Note: The function calls must have arguments which are bold and should be explained.

D. logic for server to extract Client IP address:

On accept() call, the input argument struct sockaddr *addr contains the IP address (addr→sin_addr) of Client which needs to be returned to client on receipt of message "whoamI". …..... 1 Mark

Q2.

a) Pros & cons of layered architecture: (1 marks)

Pros: Layer interface are defined i.e we can make changes in any layer without disturbing other layers.

--- 0.5 marks

Cons: It does not allow querying an interface which is not directly below it. ---0.5 marks

b) Service provided by layers of Internet Protocol Stack: (2 marks)

Layers	Services Provided
Application layer	Protocols, logic
Transport layer	End to end reliability, congestion control, connection management
Network layer	Addressing, routing, packetization
Data link layer	Error detection and correction, framing, MAC or Sharing of medium
Physical layer	Bit stream to signal

Only layers are defined ---0.5 marks
Services are defined ---1.5 marks

c) Discuss design choices made for the services that are offered in transport layer: (2 marks)

Transport layer provides following services:

I. Multiplexing
II. Reliability
III. Congestion control
IV. Connection management --- 0.5 marks

All the above can be done in other layers also.

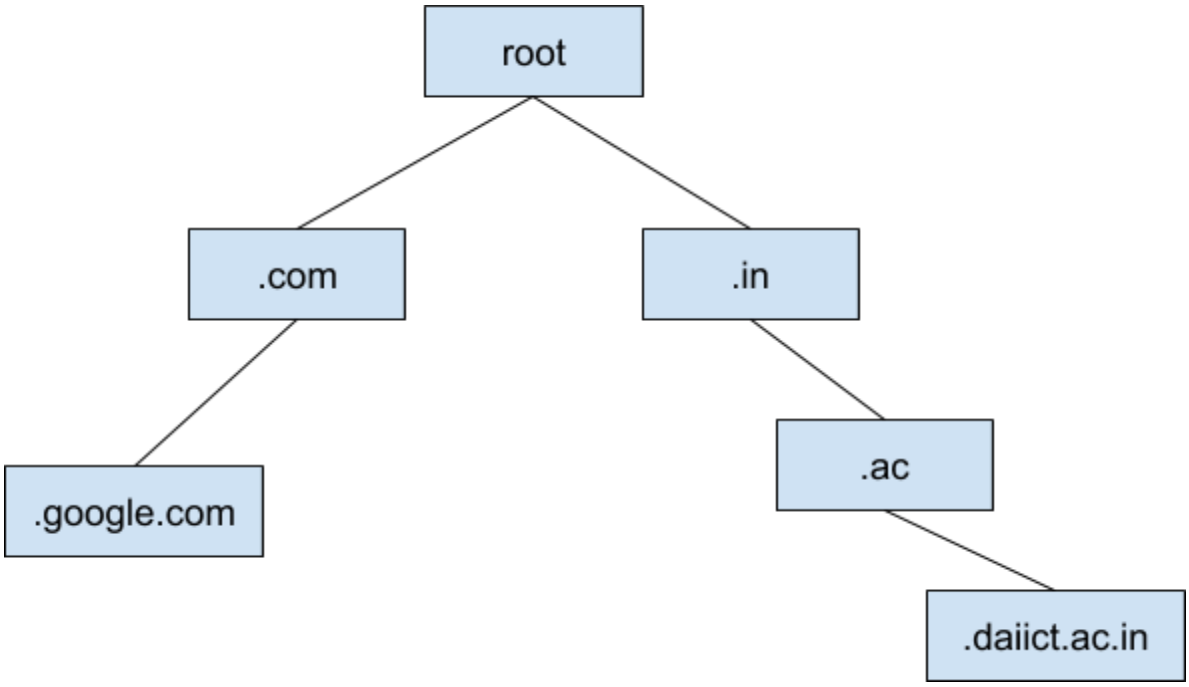
Design choices for TL are:

Internet architecture was designed for Store and Forward packets, datagrams and routers are stateless. This was done so that router can be simple and run fast.

Main reason is to keep Network interface simple(routers) and handle complexity at end-nodes(computers) . --- 1.5 marks

Q3.

DNS Hierarchy: [1 mark]



Obtaining IP of server1.google.com [2 marks]
(for mentioning correct message sequence - 1 mark,
for mentioning type of records in each message - 1 mark)

Local nameserver : *ns1.daiict.ac.in*

Suppose,
Host name: *pc1.daiict.ac.in*
Authoritative DNS server for google.com: *dns.google.com*
And the required DNS record is not found it cache

1. Host *pc1.daiict.ac.in* first sends a DNS query message to its local DNS server(*ns1.daiict.ac.in)*.
   The query message contains the hostname to be translated(*server1.google.com),*in the name field and A in the type field in the question section as it's a query message for getting ip address for a hostname.

2. Generally, The query from the requesting host to the local DNS server is recursive, and the remaining queries are iterative.

3. Then local DNS will send the query message to one of the root servers with *server1.google.com* in name field and A in type field.

4. The **root server** takes note of the *.com* suffix and responds with two types of records in answer section of the **response message**.
   1. **NS records** for the TLD servers for *.com*
   2. **A records** containing IP addresses of these TLD servers

5. The local DNS then sends the same query message to one of these TLD servers.

6. The **TLD server** takes note of the *google.com* suffix and responds with two types of records in answer section of this DNS response message.
    1. NS records for the authoritative DNS server *dns.google.com*
    2. A record containing IP address of the authoritative DNS server

7. Finally, the local DNS sends the query message to **authoritative server** *dns.google.com*, which responds with the IP address of *server1.google.com*(in form of **A records**).

8. Now the **local DNS** has the IP address of server1.google.com and it send the **A record** to the requesting host(*pc1.daiict.ac.in*).

DNS for Load balancing/distribution across heavily loaded application servers: [2 marks]
(rotations on set of IP addresses - 1 mark,
client generally sends request to first IP from the set - 1 mark)

DNS is also used to perform load distribution among replicated servers, such as replicated Web servers.

Busy sites, such as google.com, are replicated over multiple servers, with each server running on a different end system and each having a different IP address.

For replicated Web servers, a set of IP addresses is thus associated with one canonical hostname. The DNS database contains this set of IP addresses. When clients make a DNS query for a name mapped to a set of addresses, the server responds with the entire set of IP addresses, but **rotates the ordering of the addresses** within each reply. **Because** a client typically sends its HTTP request message to the IP address that is listed first in the set, DNS rotation distributes the traffic among the replicated servers.

4. An ack-based link protocol (waiting for ack before sending next packet) uses a timer to abort ack_wait function after the timeout period T and retransmits the same packet. This process continues till the packet is successfully transmitted before moving to a new packet.

(a). How would you define the efficiency of such a protocol. [1 mark]

ANSWER=>

$$Efficiency = \frac{\tau}{\tau + RTT}$$

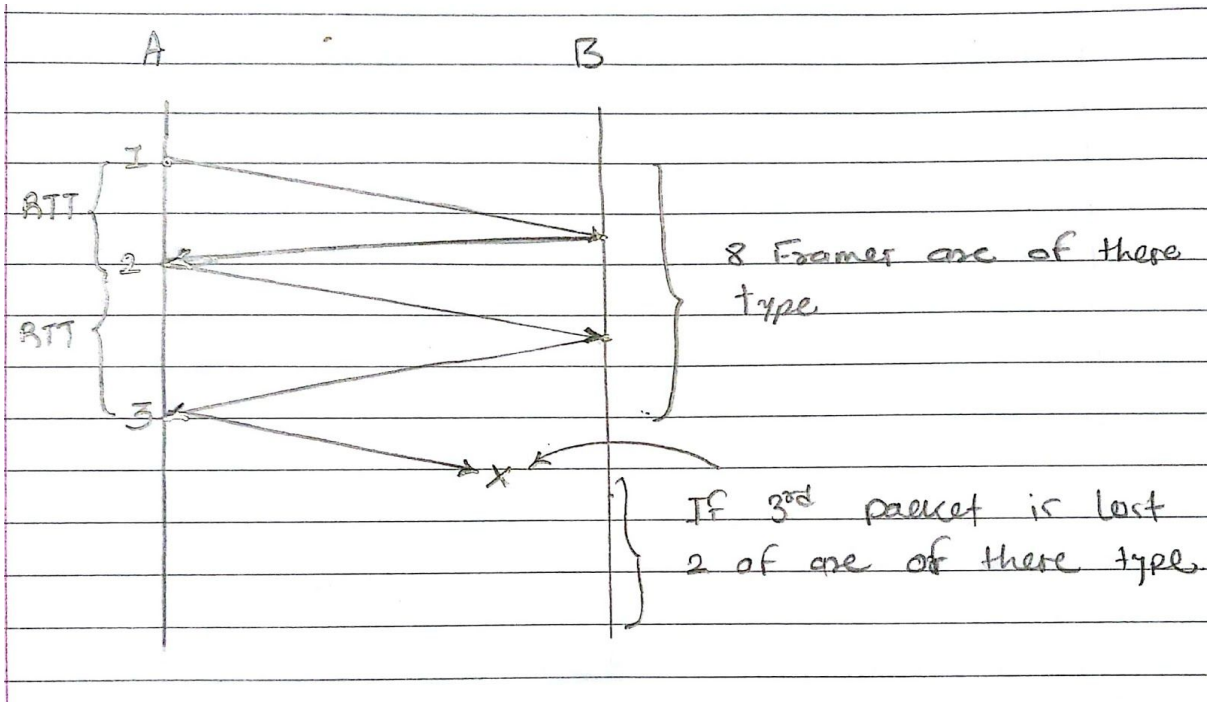
$$Efficiency = \frac{1}{1+2\alpha}$$

OR

$$Efficiency = \frac{Total\ time\ taken\ for\ transmitting\ a\ frame}{Total\ time\ including\ number\ of\ retransmissions}$$

(b). 10 packets are sent on a channel. Each packet takes 10ms to transmit. Acks take exactly 100ms to arrive after the transmission. Two of the ten packets got corrupted the first time they were sent (these needed to be retransmitted once). No other packets suffered corruption. Compute the efficiency of the protocol in each case when timeout value T is set to (i)150 ms, (ii)120 ms, (iii)80 ms.

ANSWER=>



$$T_{total} = 8 * RTT + 2(RTT + T_0) \quad \text{if , } T_0 > \text{RTT}$$

$$\eta = \frac{Transmission\ time}{Total\ time} = \frac{10 * T_{trans}}{T_{total}}$$

(i) 150ms: **[1 marks]**

$T_0 = 150ms$

$\tau = 10ms$

RTT = 100ms

$$\eta = \frac{10 * 10\ ms}{1300\ ms} = \frac{1}{13} = 7.69\%$$

$T_{total} = 8*100 + 2(100+150)$

$= 1300$ ms

$$\eta_{eff} = \frac{100}{1300} = 7.69\%$$

(ii) 120ms: **[1 marks]**

$T_0 = 120ms$

$T_{total} = 8*100 + 2(100+120)$

$= 1240$ ms

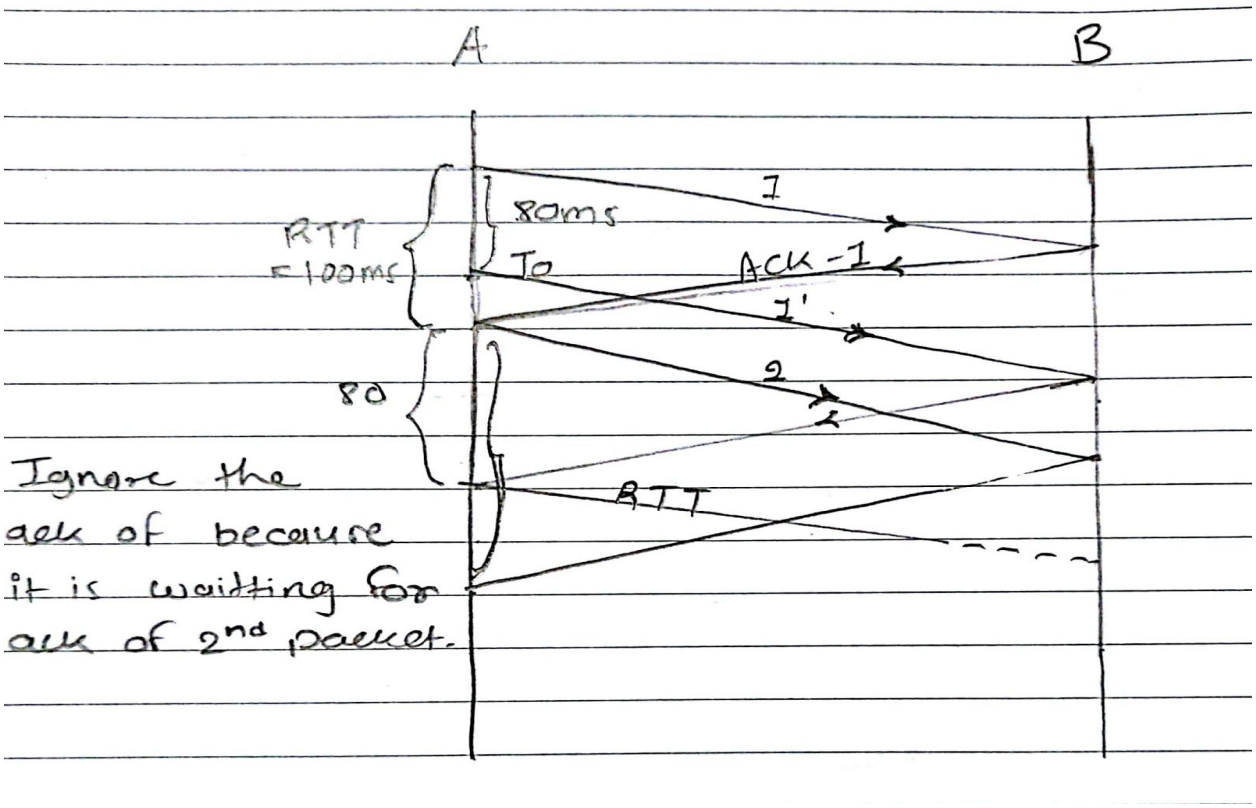
$$\eta_{eff} = \frac{100}{1240} = 8.06\,\%$$

(iii) 80ms: **[1.5 marks]**

$T_0 = 80ms$ ($T_0 < RTT$)

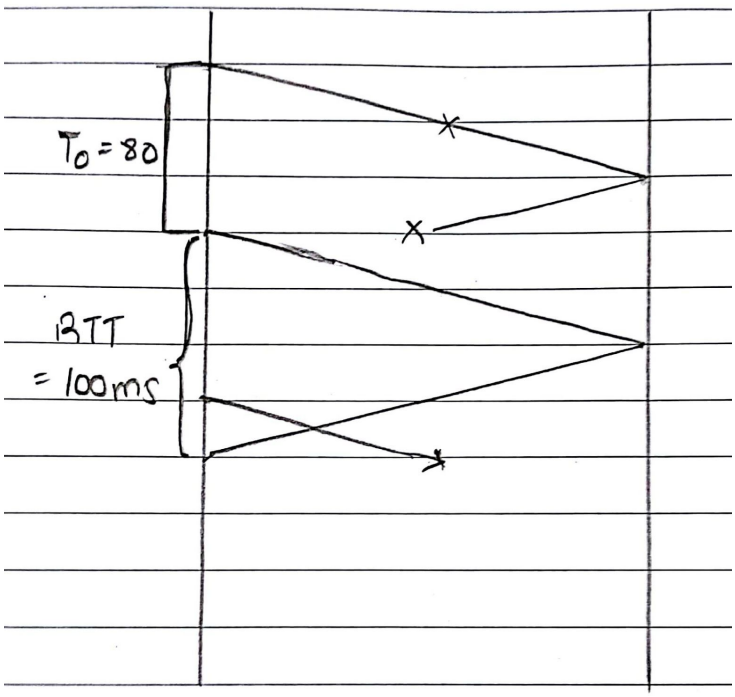
RTT is bigger than T_0 , so all the packets will get retransmitted.

(a). If no packet get lost ,



- Here, we consider T_0 = RTT.
- All packets get retransmitted, but need not to wait for ack, so no additional time waste.

(b). If 2 packets will lost then , [0.5 marks]



- Here, if packet lost, then it retransmits before RTT because T_0 is lesser, so it retransmits it.

$$T_{total} = 8 * T_0 + 2(RTT + T_0)$$

$$= 8*80 + 2(100+80)$$

$$= 1000 \text{ ms}$$

$$\eta = \frac{10 * 10 \ ms}{1000 \ ms} = \frac{1}{10} = 10\%$$