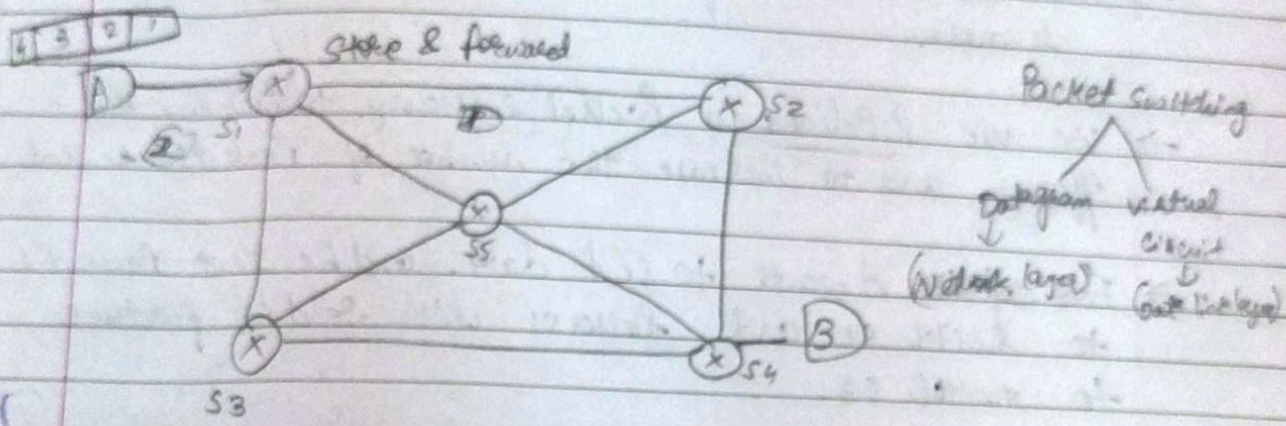
Lec-3

What's a Protocol?

→ Protocols define format, order of messages sent and received among network entities, and actions taken on message transmission, receipt.

Packet Switching :-



→ Packet switching works on data link and network layer

→ The data is transmitted in the form of packets over a network

→ for eg :- If A wants to send some data then that data is divided into packets.

→ Packet switching follows store & forward method.

→ for eg :- If A wants to send data and it is divided into different packets.

→ Then one packet is send to S1 switch it stores the packet in buffer.

→ From buffer we will finalize which path the packet will be transmitted.

→ Switch uses routing table concept to get help to decide through which path the packets should be sent.

→ And because of store & forward the efficiency of Packet switching is high compared with circuit switching.

→ And also because of store & forward the delay is more.

→ We use Pipelines in Packet switching. to increase the efficiency and to increase the number of Packets sended.

→ For eg:- A wants to send data. and he sent Packet 1 to Switch S1 and Switch S1 will send it further to switch S3.

→ By the time S1 is sending the packet to S3, A with send Packet 2 to S1 through Pipeline.

Total time = n(TT) + PD

→ Transmission time - A will transmit to switch S1, it will process & transmit to S3 further it will process & transmit to S2.

→ n is the number of switches used inside the network

→ PD = Propagation delay. (Total delay happened in each switch).

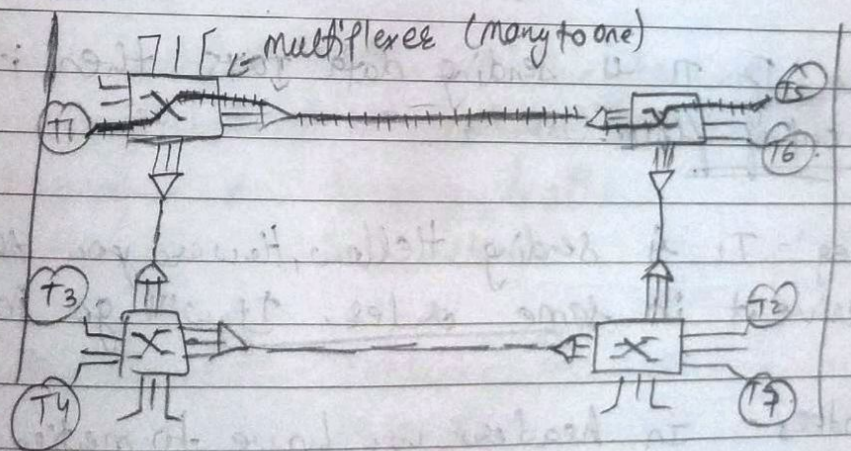
→ If the arrival rate (in bits) of link exceeds the transmission rate of link for a period of time then,

- Packets will queue, wait to be transmitted on link.
- Packets can be dropped (lost) if memory (buffer) fills up.

Points to remember:-

- Data link & Network layer
- Pipeline used
- Store & forward
- efficiency high
- Delay More.

Circuit Switching :-



→ Data is sent to T5 by T1.

→ Circuit switching was designed for telephone exchange.

→ In telephone networking in order to physically connect the devices circuit switching was used.

→ Connection is established first before sending the data.

→ The device through which the data is to be shared act as a multiplexer. (many to one).

→ many people are connected and one wire is going out on ~~senders~~ receivers side.

→ while on receivers side de-multiplexing is used (one to many).

→ Circuit switching uses dedicated path.

→ Contiguous flow - once the connection is established you can send data contiguously.

→ If T1 is sending data to T5 then it can never be out of flow. (in order)

→ for eg: T1 is sending Hellow, How are you then T5 will receive it in same order. It will go in order.

→ No Headers - In headers we have to mention the source & destination address but in Circuit switching no headers are required because we are already establish a connection, we are setting up the connection.

→ Efficiency less :- When we have reserved the resources then no one else can use that resource.

→ for eg :- You have made a reservation in hotel for 5 pm but you arrived there at 7 pm so between 5 to 7 pm the table (resource) was reserved but was not in use hence efficiency. Similarly you have reserved the resource but not sending the data leads to less efficiency.

→ Delay less :- In packets switching the routers store the message & forward it hence it leads to delay but in circuit switching once the connection is established there is no delay.

→ Circuit switching is good for telephone network but for computer network it should not be used because the efficiency is less.

Total time = Setup time + TT + PD + tear down time

Setup time = total time to get the reservation.

Transmition time = $\frac{\text{Message length}}{\text{Bandwidth}}$

Propogation time = $\frac{\text{Distance}}{\text{Speed}}$

Tear down time = When you have terminated the connection you have to free the resources

Points to Remember:-

- Physical layer
- Contiguous flow
- No headers
- Efficiency less
- delay less.

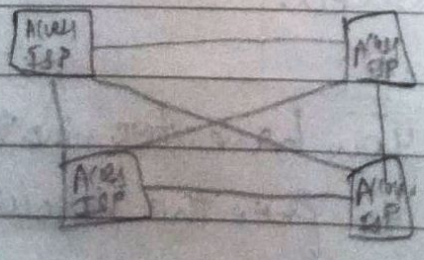
Network of Networks:-

→ End system which are your laptops, Computers etc which want to connect to the internet and they do so via Internet service providers (ISPs)

→ These ISPs could be at residential level, organisation or Company level as well as in university level.

→ Access ISPs must be connected to so that packets from one host can travel to other host in Internet.

→ These connections can result in a network that is very complex.



They must be interconnected to each other to transfer data but what if we have millions of access ISPs.

→ Connecting millions of ISP's will be very complex as it will result into $\Theta(n^2)$ connections.

→ What we can do is every ISP can connect to Global ISP.

→ There will not be only one global ISP; there will be multiple global ISP which will be connected with different ACCESS ISP.

→ These global ISP should talk to each other over a Peering link.

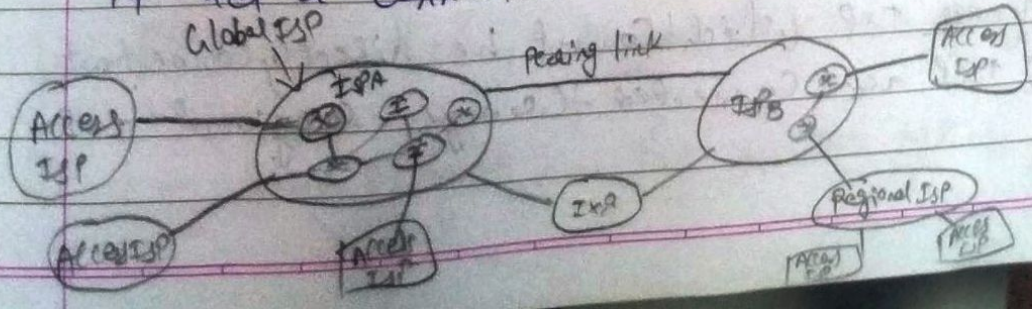
→ The peering links are managed by some kind of economic agreements related to how much traffic these global ISPs are willing to share for one another.

→ Apart for peering links there are also IXP (Internet Exchange Points)

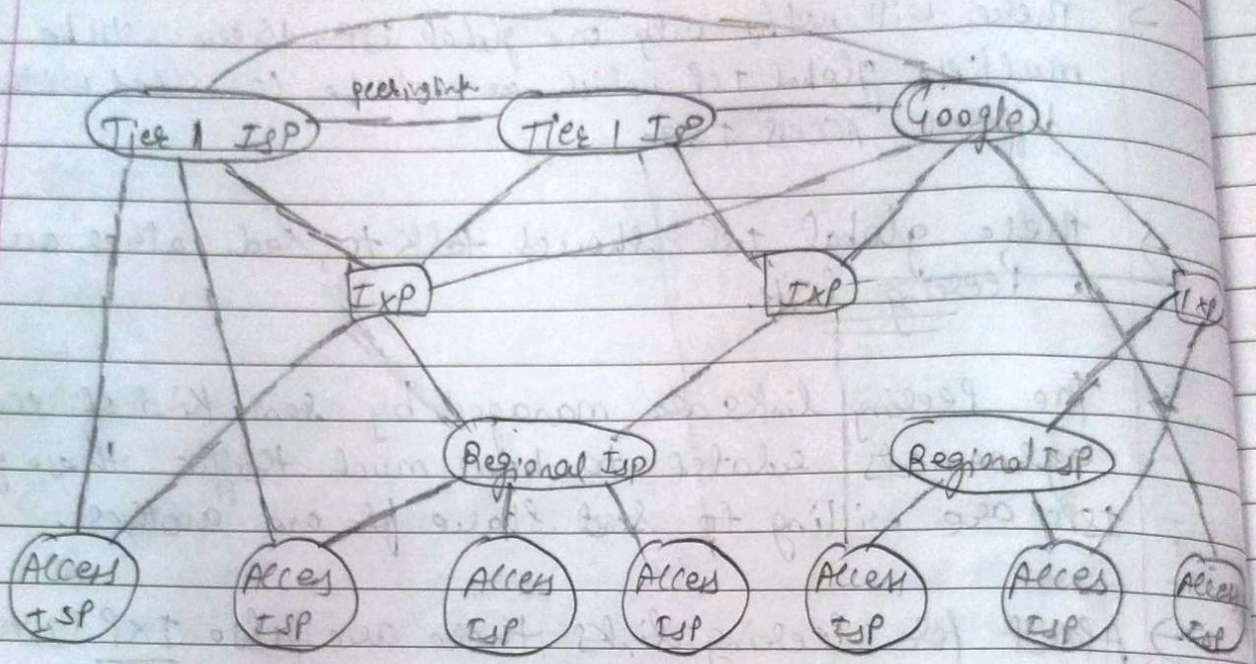
→ IXP are places like different global IXP meets and exchange packets.

→ Some Access ISPs may be located in remote areas they may not be able to directly connect to global ISPs this gives rise to create Regional ISPs.

→ These Regional ISPs are connected to global ISPs and it act a connection between Access ISP & global ISP.



→ There are also a bunch of Content Provider (eg: Google, Microsoft) may run their own network to bring services, content close to end users.



→ At the bottom level we have Access ISP which can be directly connected to a tier 1 or global ISP or it can also be connected to regional ISP

→ Some of the ISP can also be connected to the IXP

→ The tier 1 ISP or global ISP need to exchange traffic with each other & they do so with the help of peering link or through IXP.

→ Some Access ISP which cannot be directly connected to global ISP are connected through regional ISP.

How do loss & delay occur?

⇒ Let's take a example where Source A is seding a Packet to destination.

→ This Packet has to pass through multiple routers before it reaches the destination and each of these routers the Packets can get qued in the router buffer.

→ When the Packet arrival rate to a linked temporarily exceeds the outbut link capacity what's gonna happen is the Packets have to wait and the Packets experience Queuing and hence delays.

→ Let say the buffer has a size of storing 5 Packets in a queue but 6th Packet arrives and the buffer is already full and hence the 6th Packet will be lost

Four Source of Packet delay:-

$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}.$$

→ There are 4 components of delay at a particular router.

→ The first one is Processing delay.

1. Processing delay: Nodal Processing delay.

→ The amount of time the router takes for Processing

Components:-

→ Check bit errors

→ It has to determine if it has multiple outgoing links then it has to determine which among them should the packet be sent.

→ It's typically in the order of few milliseconds or sometimes even less.

2. Queuing delay:-

→ Queing delay occures if the packet arrival rate is greater than the output rate

→ Amount of time the packet has to wait in buffer before it can get transmitted.

3. Transmition delay:-

→ once the packet gets to the head of the queue that is it becomes the packet to be transmitted that means it's going to be put onto the wire.

→ The amount of time the router takes to put the packet onto the wire.

$$dtrans = \frac{L}{R}$$ (where L = packet length (bits)
R = link Bandwidth (bps).)

4. Propogation delay:-

→ The packet is put on wire now the time taken for that packet to reached from router 1 to router 2. The time taken by the packet to transmit is known as propogation delay.

→ dependent on length of wire & the speed.

$d_{prop} = \frac{d}{s}$ (where d = length of physical link.
s = propogation speed.)

Throughput:-

→ Rate at which the bits are transferred from the sender to receiver

→ Unit - Bits / Unit time

Instantaneous - Rate at any given time. It can vary over time.

# Average - Rate over longer period of time.
Rate = no. of bits that are being transmitted. Per unit time between the sender & receiver.

→ Network layers

→ OSI model

→ Network security (Left).

Lec-4.

Creating a network app:-

→ To design the network applications what we will have to do is we'll have to write Programs.

→ the Programs that run on different end systems.

→ These Programs has to Communicate with each other over network.

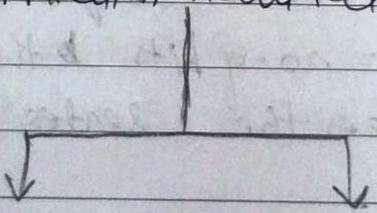
→ for eg:- There is a web server software that Communicates with the browser software

→ Application layer only written at the end host.

→ The end host Communicate with each other through application layer.

→ The app. layer at one of the host Company is going to Communicate with the app. layer of the other.

Application Architecture



Client-Server Peer-to-Peer (P2P).

Client Server :- Every App. has 2 Parts :- server & client.

Server :-

→ There is a server which we assume to be always on.

→ It has permanent IP address.

→ If you want to host your own server you should have a reliable IP address. which can be reached via client.

→ Sometimes, there are whole bunch of servers & the collection of servers is known as data center.

→ for eg :- A company like facebook or whatsapp might have a whole bunch of servers to which the clients connect.

→ so that is why they have to be always on because the client might want to connect to the server at any given time.

Client :-

→ Clients need not be always on., they come up intermittently and they intermittently connected. and they communicate with the server. for completion of whole bunch of task.

→ They may have dynamic IP addresses and these IP addresses can change over time.

→ Clients in general do not Communicate with each other directly for eg:- There are two Cor whatsapp clients that have to send messages to each other, the messages are first send towards observer and then from the words observer the msg is sent over to the other client.

Peer to Peer architecture :- (P2P)

→ In P2P there is no always on server.

→ The clients or the peers Communicate directly with each other.

→ Peers requestes services & Content from other Peers and they Provide Certain Service or Content to some other peers.

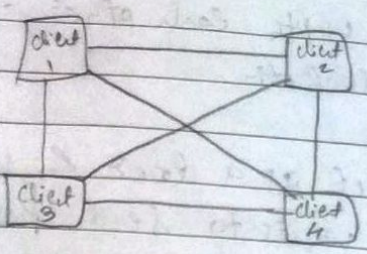
→ In this every single client or Peer Can be Client as well as server.

→ New peers being new service capacity, as well as new service demand. It brings some content which Can be share with other Peers and it also requires certain Content which it Can download from other peers.

→ Peers are Intermittently Connected & change IP address Also they Can disappear Whenever they want.

→ The managemat of P2P architecture is way more Complex than the client-server architecture

→ P2P - archi. is distributed in nature & they don't need a server that coordinates all the diff. clients.



Process Communicating :-

→ whether you want to use client server or P2P architecture you will need to write Programs that communicates with two peers or client & the server.

⇒ Now these Programs are essentially Processes that are running within a host, now if there are two Process that run within same host they communicate with each other using Inter Process communication which is defined by the OS.

→ Process that are running on different host Communicate by exchanging messages.

→ for eg:- One Process might be running on client side & other prior Process might be running on server side. So the way this Process will work is the client Process would generally initiate the communication & server Process that is waiting to be contacted would be contacted.

→ P2P also have similar Processes that communicate with each other i.e server Process & client Process.

Sockets:-

→ Application layer has whole bunch of processes & each process communicates with each other through something called sockets.

→ You can think of socket as a door & what application does is whatever it wants to send passes through the socket.

→ so app. layer have these Processes & the Processes have a whole bunch of messages that they want to send and every message has to be shut out of this door which is the socket such that it can be received by the transport layer.

→ now at the receiving end the transport layer what it does is when it receives the transport layer segments it has to assimilate them together & push it through this door of socket such that the app. layer process can actually receive it.

Addressing Processes:-

→ The Process to communicate with each other and to send messages there must be a way to identify diff Process uniquely.

→ First such identifier is IP address.

→ Every host devices has a unique 32 bit IP address.

→ But alone IP address is not sufficient for identifying the process because many process run on same host & all these processes will have same IP address

→ So along with IP address & Port number together can uniquely identify the process.

App-layer Protocol defines :-

⇒ Types of message exchanged : Request & Response

→ Message syntax :- what fields in messages & how fields are delineated.

→ message syntax should be written in such a way that who is going to receive the msg & the order (maybe).

→ Message Semantics :- Meaning of the msg, what exactly you are going to mention

→ Rules :- for when & how processes send & respond to msg.

Page No.:

Lec-5

What transport service does an app need?

1. Data Integrity:-

→ Some of the apps require 100% reliable data transfer.

→ eg:- File transfer or web transactions.

→ other apps (like: audio) can tolerate some loss.

2. timing:-

→ Some apps (eg: internet telephony, interactive games) require low delay

3. Through Put:-

→ Throughput means minimum amount of time required to send the msg from one host to another.

→ So, the Throughput time should be minimal.

4. Security:-

→ We have to apply some algorithm to Encrypt & decrypt the data and for data integrity to keep the data values secured.

TCP Internet Transport Protocol Service :

TCP	UDP
→ Transmission Control Protocol	→ User datagram Protocol.
→ It is a connection oriented Protocol. It establish a connection before sending or receiving any data.	→ It is not connection oriented Protocol. It does not form any connection & direct send or receive data through IP address.
→ It is reliable	→ It is not reliable
→ The sequence of the data sent will be same.	→ Sequence may change
→ It is slower than UDP. because it used to send data for long distance.	→ It is faster than TCP as it is used for shorter distance.
→ Header size is 20 bytes	→ Header size is 8 bytes
→ Retransmission of lost packet is possible.	→ Not possible
→ It is used as HTTP, HTTPS, SMTP, FTP, etc.	→ It is used at video conferencing, DNS etc.

→ Email - SMTP
Remote terminal Access - Telnet
Web - HTTP
File Transfer FTP
} TCP.

Streaming multimedia - HTTP, RTP
Internet Telephony - SIP, RTP
} TCP or UDP.

Lec-6

Web & HTTP :-

→ Web page consists of objects, each of which can be stored on different web servers.

→ objects can be HTML file, jpeg img, audio file etc.

→ Web page consists of base HTML-file which includes several referenced objects, each addressable by a URL.
↳ for eg: submit button, on submitting the form it will be redirected to another page. The address of the redirecting page will be managed by URL.

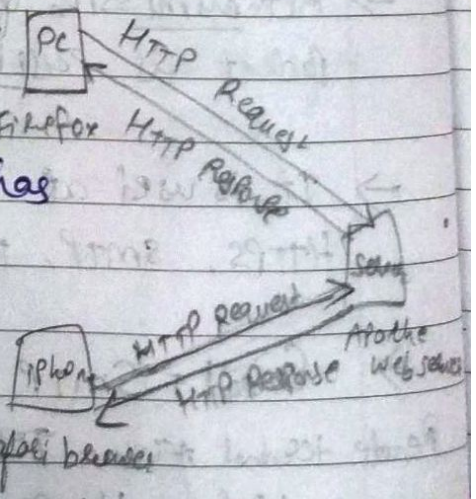
HTTP overview :-

→ HyperText Transfer Protocol.

→ HTTP is a application layer protocol.

→ HTTP provides you the architecture of client server model or client-server architecture.

→ You are requesting files from firefox browser to Apache browser.



→ The PC will be your client & first it has to send the request to the server.

→ for any host or any client it is mandatory to first connect with the server.

→ Before sending the request to server we need to send the handshake hand shaking signal.

→ once we send a handshaking signal the server will be ready to accept your request.

→ If you do not send handshaking request than server will not accept your request. and all this thing will be done by sockets.

Client :- It will send the request using HTTP protocol

Server :- web server sends objects using HTTP protocol as a response

HTTP uses tcp → As mentioned above before sending any HTTP request handshaking signal is need to be established.

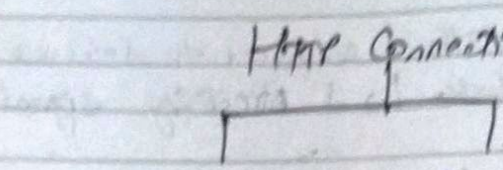
→ and TCP will create this handshaking signal.

→ And through sockets it will create the handshaking signal.

→ One socket will be there at client side & one socket will be at server side. So both the sockets has to get connected before giving or sending any kind of HTTP request.

→ Port number - 80

→ HTTP is stateless → Server will not store any info about the client by default.



Non-persistent HTTP	Persistent HTTP.
→ TCP Connection opened. (TCP is going to build connection between client & server)	→ TCP connection opened to server. [Opened only for server & not for clients. So if client want to share any msg he can]
→ At most only one obj can be sent over TCP Connection [You cannot send more than one packet.]	→ Multiple obj can be sent over single TCP Connection between client & that server [multiple obj can be sent]
→ TCP Connection closed.	→ TCP Connection closed.
→ Downloading multiple obj requires multiple TCP Connections	

Req. to 10 img.

Non-persistent

1.) A: HTTP client initiates TCP connection to HTTP server on port 80.

1.) B: HTTP server at host waiting for TCP connection at port 80 accepts connection notifying the client.

2) HTTP client sends the HTTP Req. msg into TCP connection socket. msg indicates that client want object.

3.) HTTP server receives req msg, forms response msg containing obj that was requested & send msg into socket

4) HTTP server close TCP connection

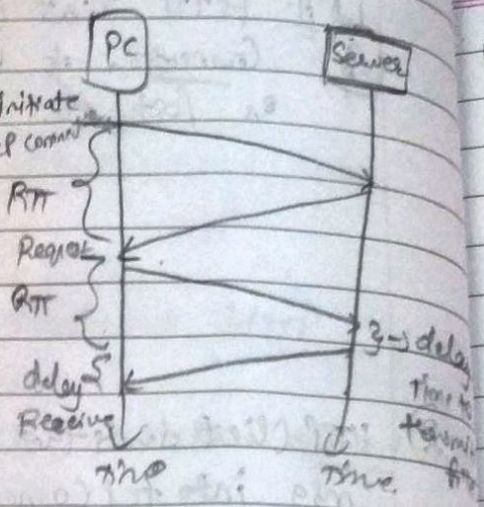
5.) HTTP client receives response msg containing HTML file.

6.) Steps 1-5 repeated for each of 10 JPeg objects.

RTT → Round Trip time :-

→ Time for a small packet to travel from client to server & back

HTTP Response time :-



→ one RTT to initiate TCP Connection.

→ One RTT for HTTP req & response.

→ obj / file transmition time.

(2 RTT / obj)

Non-Persistance Response time = 2RTT + file transmition time

HTTP Message

Request Response

→ HTTP Request message :-

→ When we send any type of HTTP message from one host to another host the msg whatever has sent from 1st host is known as request message & when we get any acknowledgement back that is known as response msg.

→ HTTP req. msg follows one readable format that is ASCII (human-readable format)

→ Request line → Get, Post, Head Commands.

req line Path Html Version Carriage return Character

Get /index.html HTTP/1.1 \r \n

\r next line

→ the first line should consist of request line

→ Using get command - It has to get the path that is mentioned after it & retrive the data.

Status (Error Codes :- (HTTP Response status code).

→ When you send a request to server you need some way to know whether the req. was successful or not. For that there exists codes.

200 → OK (Success code).

301 → Moved Permanently (Redirecting)

400 → Bad request } (Client error (something you did))

404 → Not found. }

505 → HTTP version not supported (Server Error)

other HTTP req msg :-

1. Post :- Web pages often include form input.
→ User input sent from client to server in entity body of HTTP post req. msg.
→ In a client & I'm giving something to server (eg:- username, password).

2. Get :- Used when you are going to send data directly to the server
→ Has to included in the URL field of HTTP
→ Requesting some data.

3. Head method:-

→ Req. headers (only) that would be returned if the specified URL were requested with an HTTP get method.

4. Put method:-

→ Puts data to specific location

→ Uploads new file (obj) to server.

Lec-7

Cookies & Cache.

Lec-8

P2P & Client server

Lec-8.

not made

DNS ([redacted])

Lec-9

P2P & Client server

Lec-11 (ATNDP)

Transport services & Protocols:-

⇒ This provides logical communication between app processes running on different hosts.

⇒ Different host as we know we are sending the msgs from one host to another host.

⇒ The sender & the receiving host both will have the same osi layers

⇒ so whenever the communication takes place and if they are sending it through transport services then it has to communicate only with the transport services that the reason it is known as end-to-end transport.

Transport protocols action in end systems:-

1. Sender side :- Breaks the app. msg into segments and pass it to next layer (network layer).

2. Receiver side :- Reassembles the message segments into messages & it passes on the application layer.

Two transport protocols available to Internet application:-

→ TCP, UDP.

Transport & network layer :-

Network layer :-

→ the medium for sending the msg from 1 host to another.

→ The msg received by network layer should be in segments.

Transport layer :-

→ The two host between which the msg is to be send.

Transport layer actions :-

Sender :-

→ When we send any msg it comes from application layer & it is passed by app. layer message.

→ It will determine the segments in app. msg.

→ Creats the segments.

→ Passing the segments to next layer through IP address.

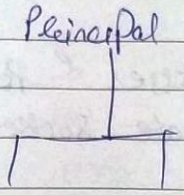
Receivers side:-

→ Transport layer will receives segments from IP

→ Check the header values.

→ extracts app. layer msg.

→ Demultiplexes msg up to app. via socket.



TCP	UDP
- Reliable	- Unreliable
- Inorder	- Unordered
- Congestion control	
- Flow control	
- Connection setup.	

multiplexing & demultiplexing:-

⇒ multiplexing is about taking several things and putting them on to one channel

⇒ Demultiplexing is taking them out & separating them back again.

⇒ Socket is like a door, a connection piece that sits in between the app. layer & transport layer & that is the mean through which process communicates.

How demultiplexing works:-

→ Host receives IP datagram

- Each datagram has
  - → Source IP address.
  - → Destination IP address.
  - → segment → source port
    - → destination port.

→ Host uses IP addresses & port numbers to direct segment to appropriate socket.

Connectionless demultiplexing:- "UDP"

⇒ UDP sockets are identified by 2 tuple

- → Destination IP address
- → Destination port number.

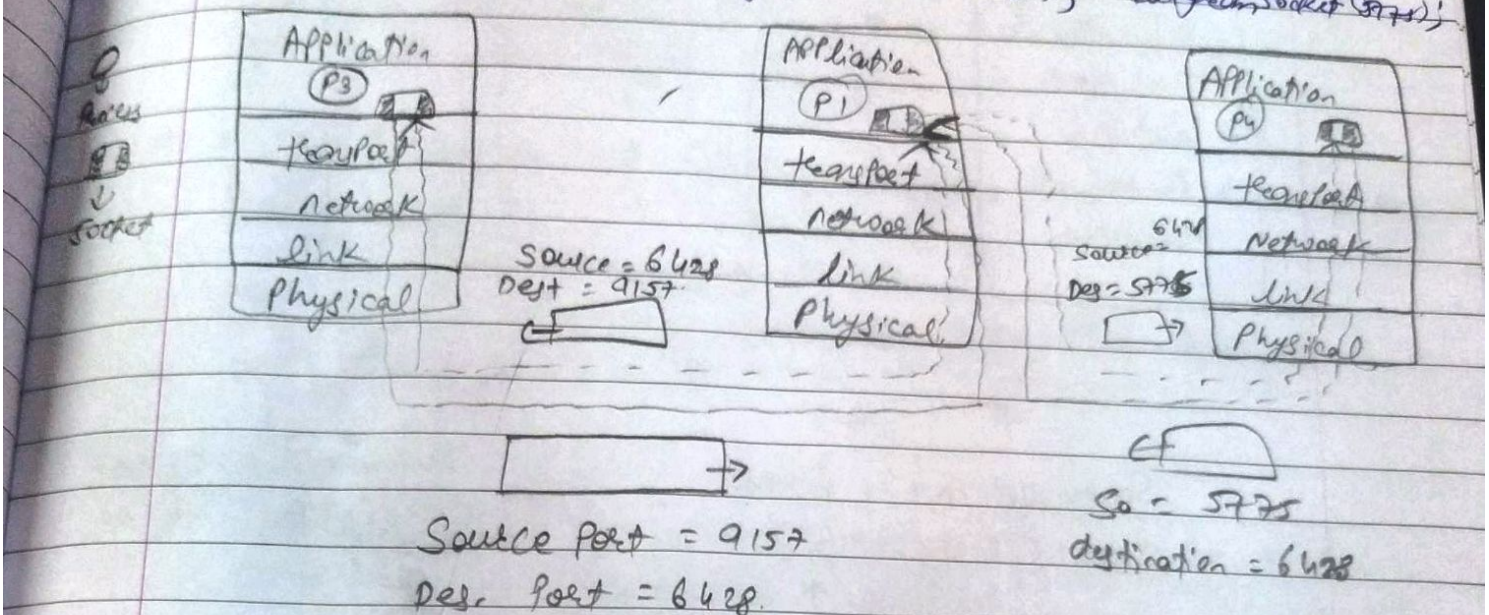
→ It checks the des. port in the segment & directs using this destination port it directs the UDP segment to the appropriate socket with the port number.

⇒ If the IP datagram all have the same dest port but diff. source IP address or source port then UDP is still going to direct it to the same socket

DatagramSocket
mySocket2 = new
DatagramSocket (9157);

DatagramSocket
serverSocket = new
DatagramSocket (6428);

DatagramSocket
mySocket1 = new
DatagramSocket (5775);



Source Port = 9157
Des. Port = 6428

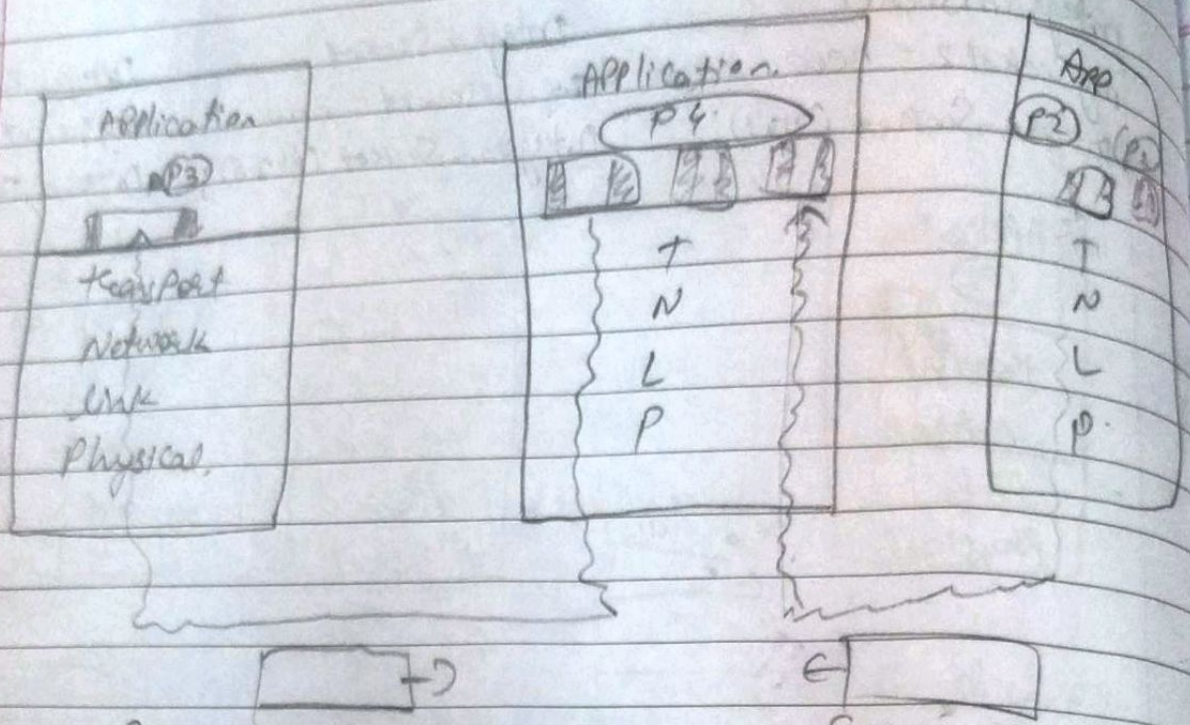
So = 5775
destination = 6428

Connection oriented protocol :- (TCP)

→ TCP - Socket identified by 4 tuple :-

- Source IP Address
- Source Port number.
- Destination IP Address.
- Destination Port number.

→ When the receiver receives all this values it uses all this value to direct the seg. to the appropriate socket.



Source IP, Port : A, 9857
dest. IP, Port : B, 80

Source IP, Port : C, 9157
dest IP, Port : B, 80

↑

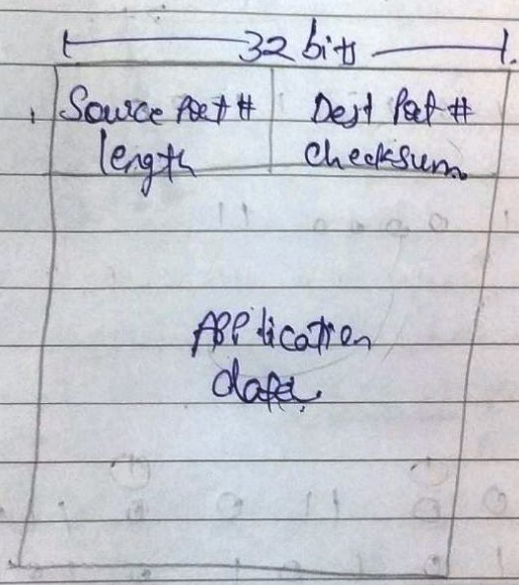
dest. Port will always be 80
because it is application for HTTP & port no of HTTP is 80.

UDP:- (Transport layer Protocol)

→ It takes messages from application layer, attach source & destination port number for multiplexing / demultiplexing & then pass it to network layer.

→ No handshaking / Hand shaking between sender & receiver in transport layer.

→ UDP used in streaming multimedia app. (Loss tolerant, rate sensitive).



length = In bytes of UDP segment, including header.

- No Connection
- small header size (8 byte).
- No congestion control. (UDP can blast away as fast as desired).

UDP Segment format

Checksum:- [It will detect errors, either discard the data or send it to app layer with a warning.]

→ Checksum is used to detect errors in transmitted segment.

→ It will only detect errors & will not help to recover the error.

Sender:

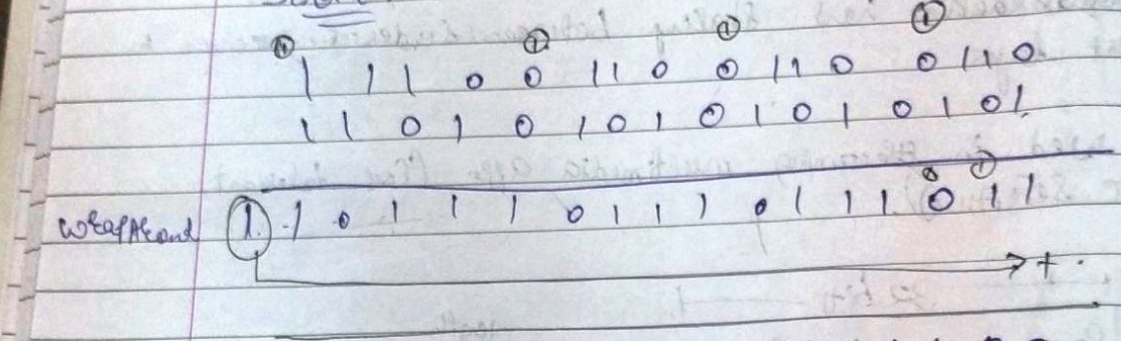
→ Treat segment contents, including header fields, as a seq of 16 bit integers.

2=10
3=11

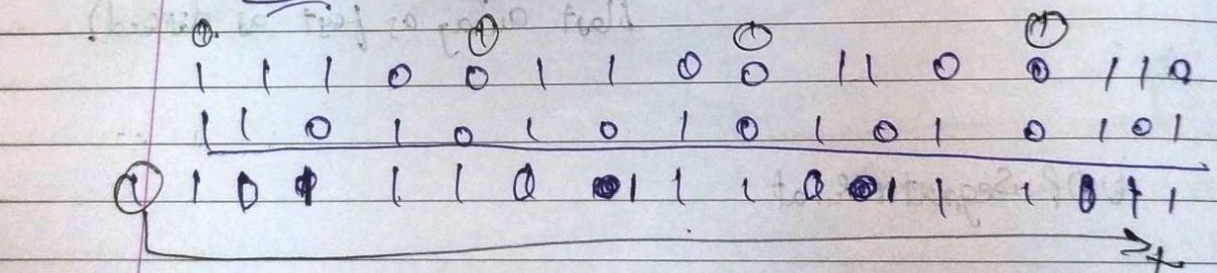
→ Checksum : Addition

Example

Sender :-



Receiver :-



Lec-12

Principle of reliable data transfer:-

→ Transport layer - divide in chunks, add header & pass on to network layer

→ App. layer generates data & will call rdt_send() (reliable data transfer send function)

→ rdt_send() will get the data & it will bring the data at transport layer.

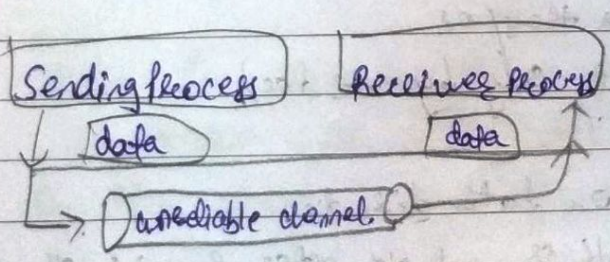
→ RDT protocol will divide it in chunks, add header also add some reliability features.

→ Now RDT protocol has to send the packet along with the reliability features added using udt_send() over an unreliable channel.

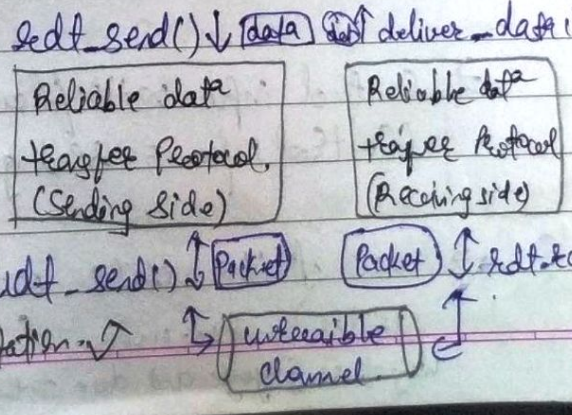
→ When the packet will be received on receiving side there is a rdt_rcv() protocol

→ The rdt_rcv() will receive the data & will check whether the data is correct or not. (By error checking, checksum etc)

→ If the data is correct then it will pass on to app. layer (deliver)



(a.) Provided service.



(b.) Service implementation

rdt_send():-

→ Called from above that is by app. layer to send the data to transport layer

udt_send():-

→ Called to transfer the packet over unreliable channel to receiver.

rdt_rcv():-

→ Called when any packet arrives

deliver_data():-

→ Called by rdt to deliver the extracted data to the upper layer that is application layer.

Getting Started:-

→ Incrementaly develop sender, receiver sides of rdt protocol.. (step by step)

→ only unidirectional data transfer.
(data can be sent only by sender).

→ control info will flow on both directions
(Control info = sending ok or not ok after receiving the data).

→ use finite state machines (FSM) to specify sender, receiver (1 state se dusre state par kisi event ki wajah se jayenge and due to that event some action will be taken.

rdt 1.0 :- Reliable transfer over a reliable channel

→ It is very simple.

→ The underlying channel is Perfectly Reliable.

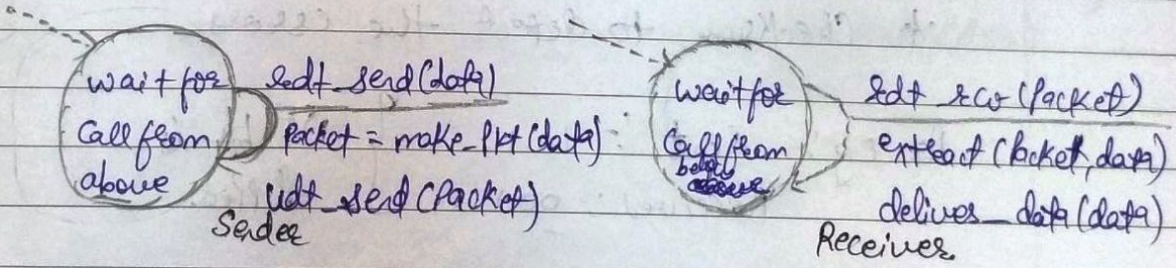
* No bit errors (No bit flip)

* No loss of Packets (No Packet loss)

Separate FSMs for Sender & receiver.

* Sender sends data into underlying channel. (Reliable)

* Receiver receives data from " " (Reliable)



⇒ Initially the rdt protocol will be in waiting state & is waiting for the app. layer to generate some data

→ When the application layer will generate some data the rdt_send(data) will be called by app. layer So the wait is over of rdt protocol.

⇒ rdt_send ke event ke response me jo action hoga me transport layer data ke packet banayega (by calling make_pkt(data)) & over rdt protocol udt_send() ko call karke packet ko underlying reliable channel receiver ko bhej dega

→ When receiver receives the packet the rdt_rcv() function will be called.

→ Event is receiving of packet so is event ke response me rdt protocol perform a action that is extracting the data which is inside the packet & that data is delivered to the app. layer.

rdt2.0 :- Channel with bit errors:-

→ Bit Complex version of rdt protocol.

→ In this the channel which is used ~~to between two~~ is not completely reliable

→ Underlying Channel may flip a bit

* Checksum to detect the errors.

for eg:- Sent :- 101 } 1st bit is flipped.
Received :- 001

feedback Procedure is used to recover from errors.

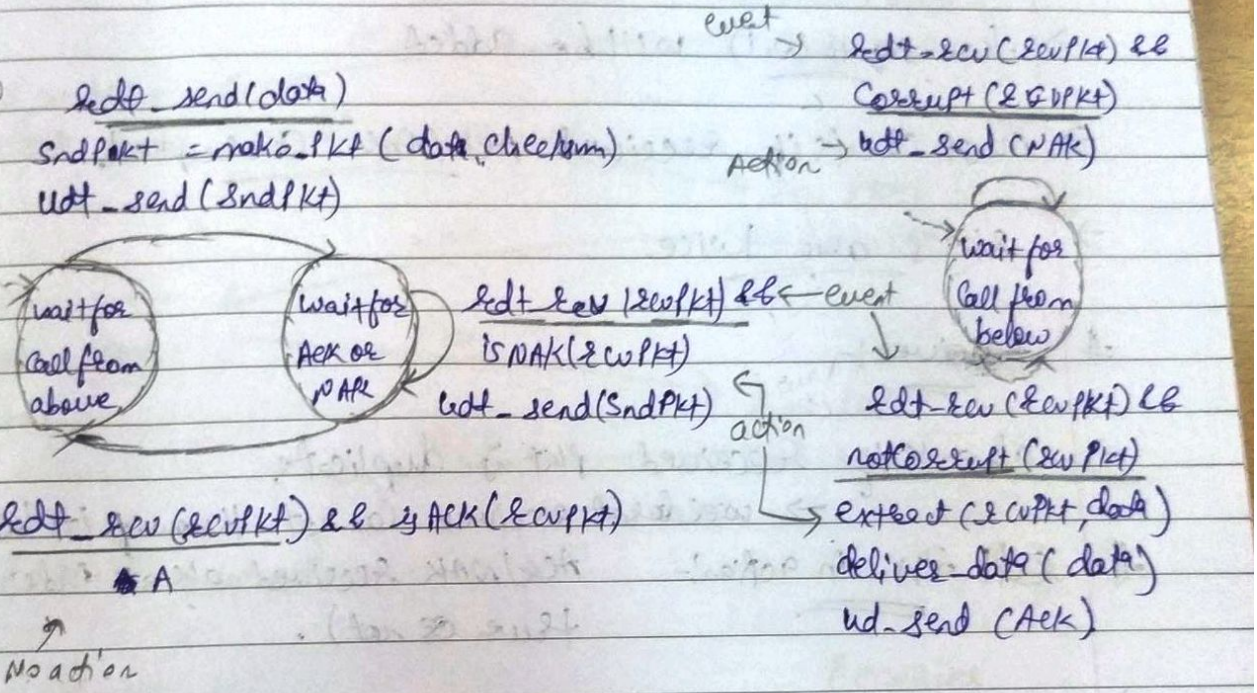
1. Acknowledgements (ACKs) - receiver explicitly tells sender that pkt received OK. [0 = OK]

2. Negative Acknowledgements (NAKs) - receiver ~~u~~ ~~u~~ ~~u~~ that pkt ~~was~~ had errors. [1 = ~~no~~ NAK].

3. Sender retransmits pkt on receipt of NAK.

New mechanism in rdt 2.0 (beyond rdt 1.0):-

- Error detection : checksum
- feedback : control msgs (ACK, NAK) from receiver to sender.



Handling duplicates:-

→ sender retransmits current pkt if ACK/NAK corrupted.

→ sender adds seq. number to each pkt.

→ receiver discards (doesn't deliver) duplicate pkt.

Stop & wait:-

→ Sender sends one packet then wait for receiver response.

Rdt 2.1:-

Sender:-

→ Seq # added to PKT.

→ two seq # (0,1) will be added

→ must check if received Ack/NAK corrupted.

→ states are twice.

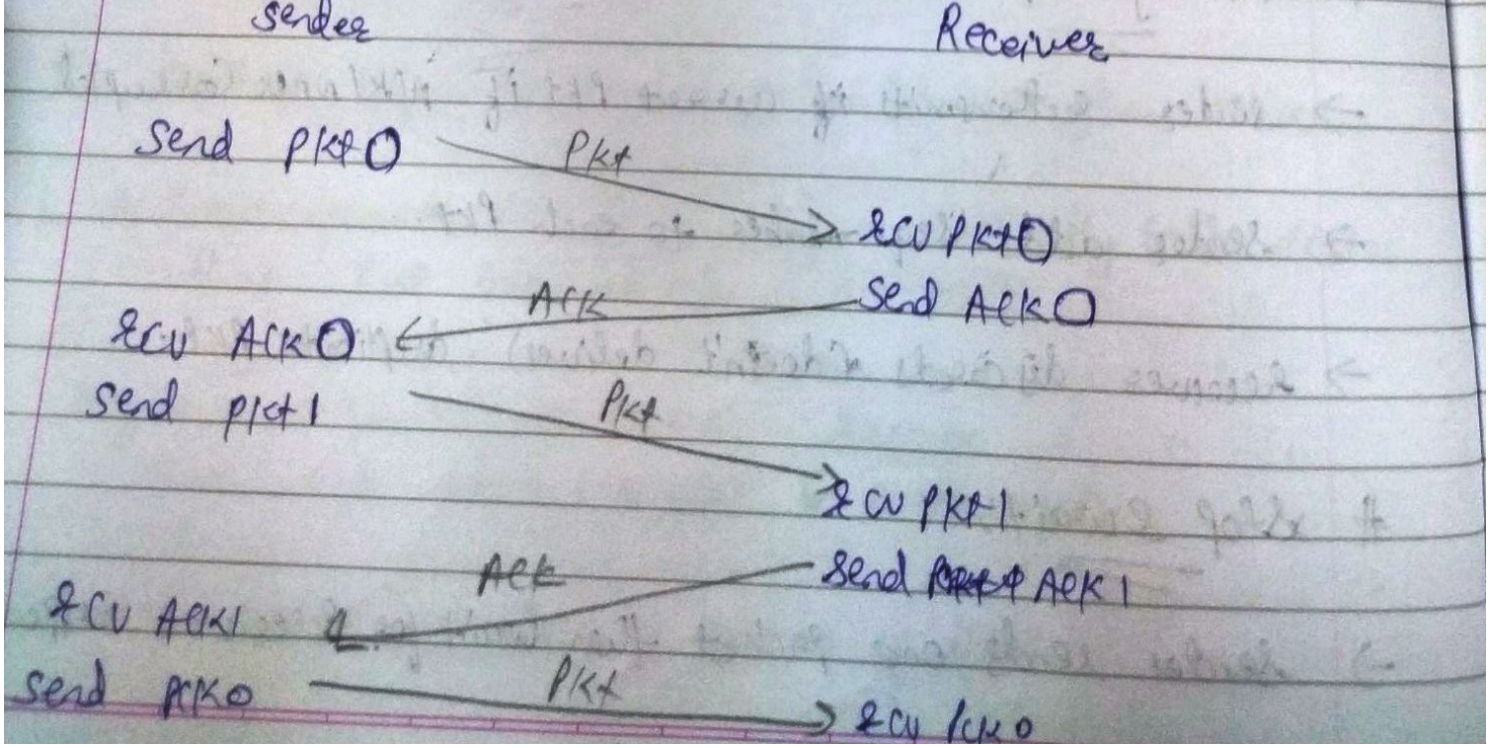
Receiver:-

→ check if received PKT is duplicate.

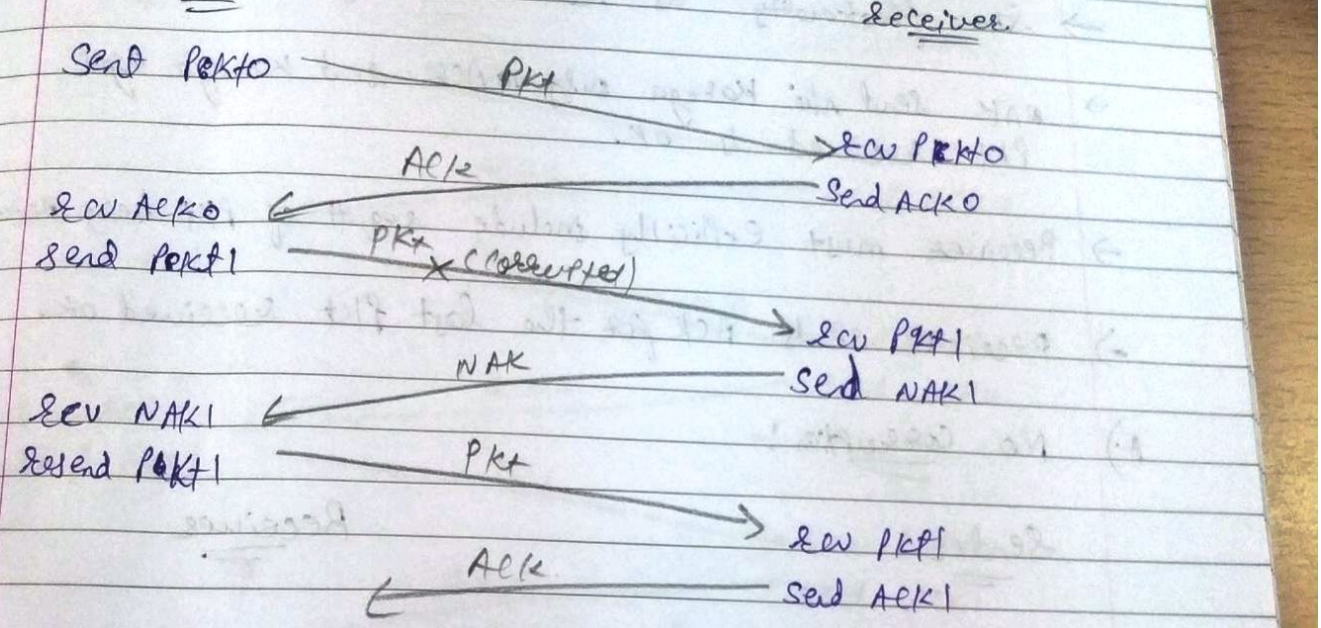
→ receiver can not know if its last Ack/NAK received ok at sender (delivered true or not).

Rdt 2.1 in action:-

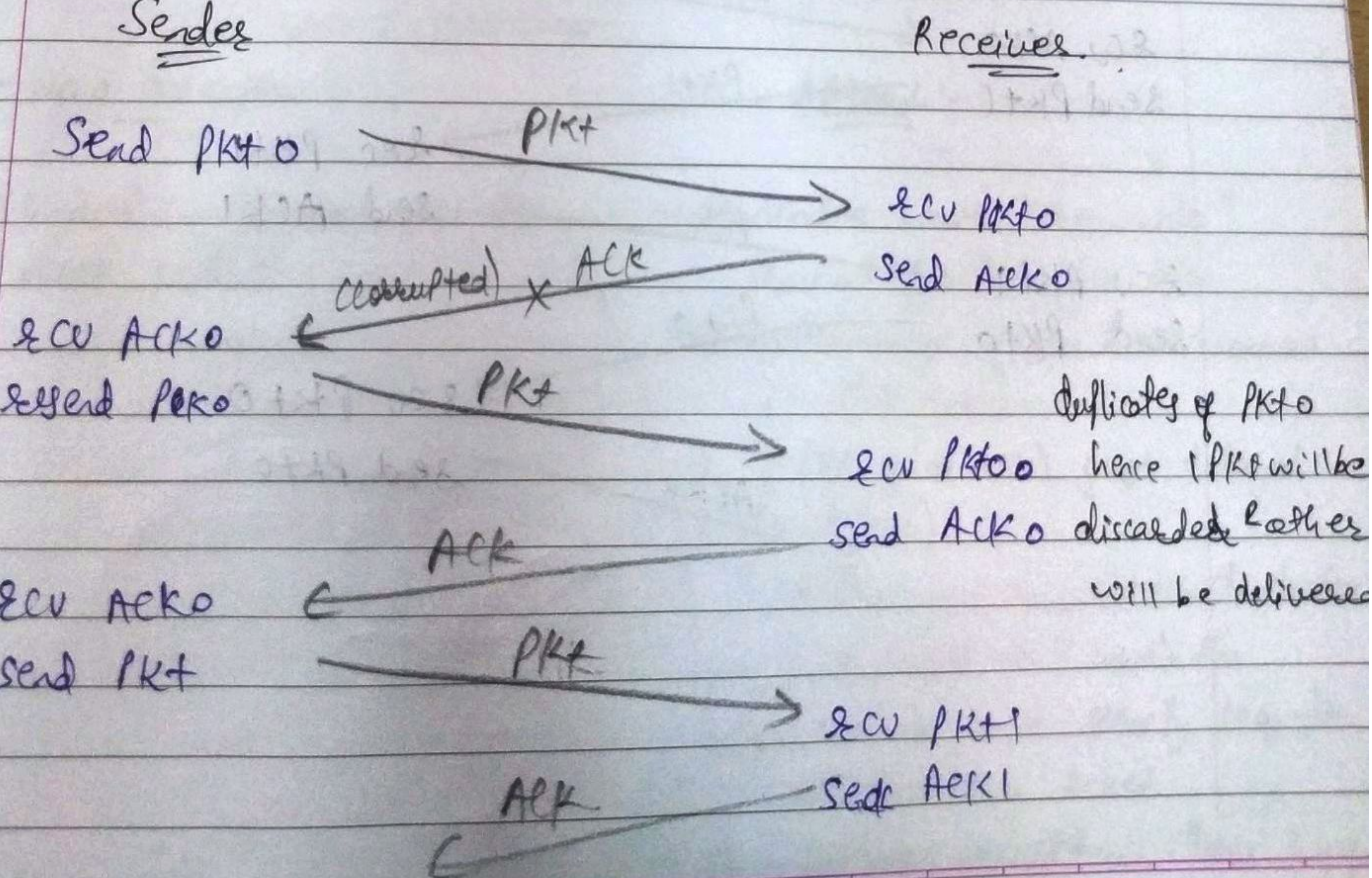
A) No Corruption.



B.) Packet Corrupted.



C.) ACK Corrupted.



rdt 2.2 :- A NAK free protocol :-

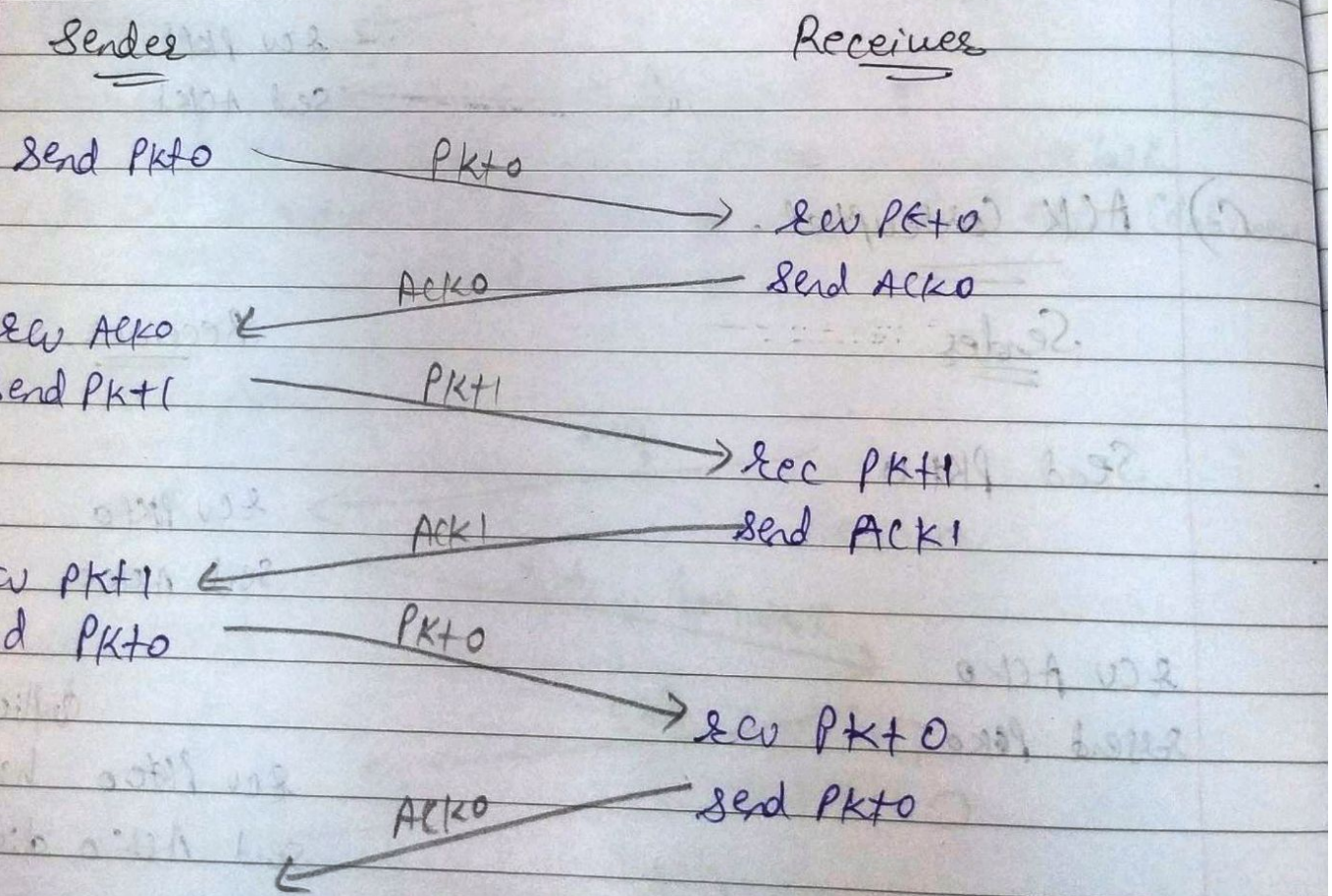
→ Same functionality as rdt 2.1, using ACKs only.

→ NAK send nhi karega only ACK send karega agar Pkt received is OK.

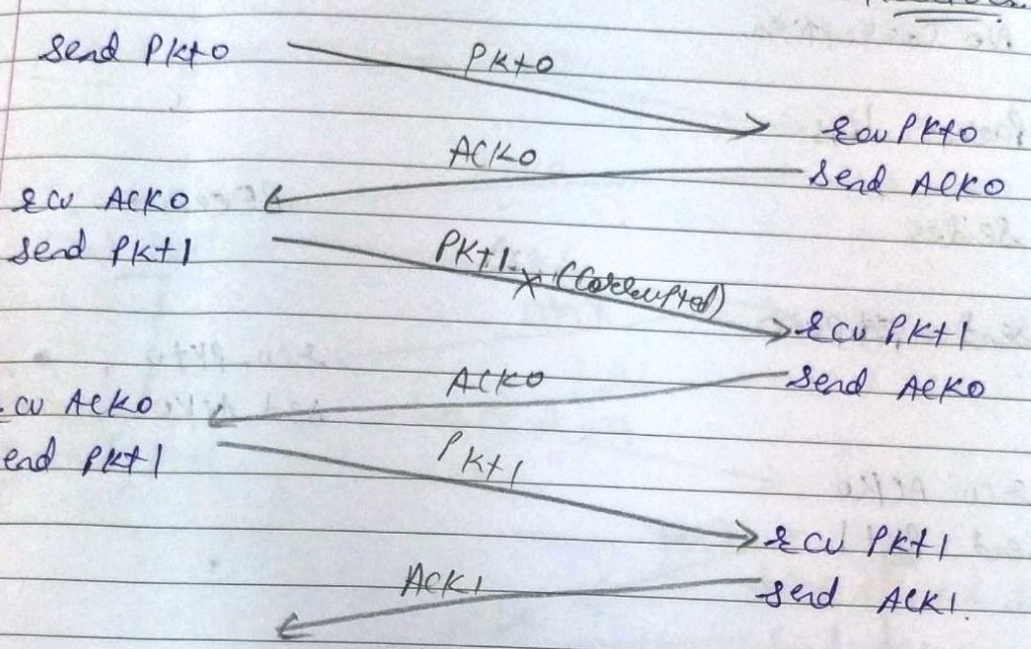
→ Receiver must explicitly include seq # of Pkt being acked.

→ Receiver sends Ack for the last Pkt received OK.

A.) No Corruption :-



B.) Packet Corrupted:-



rdt 3.0 :- Channels with errors & loss.

New assumption :

→ Underlying Channel can also loss packets

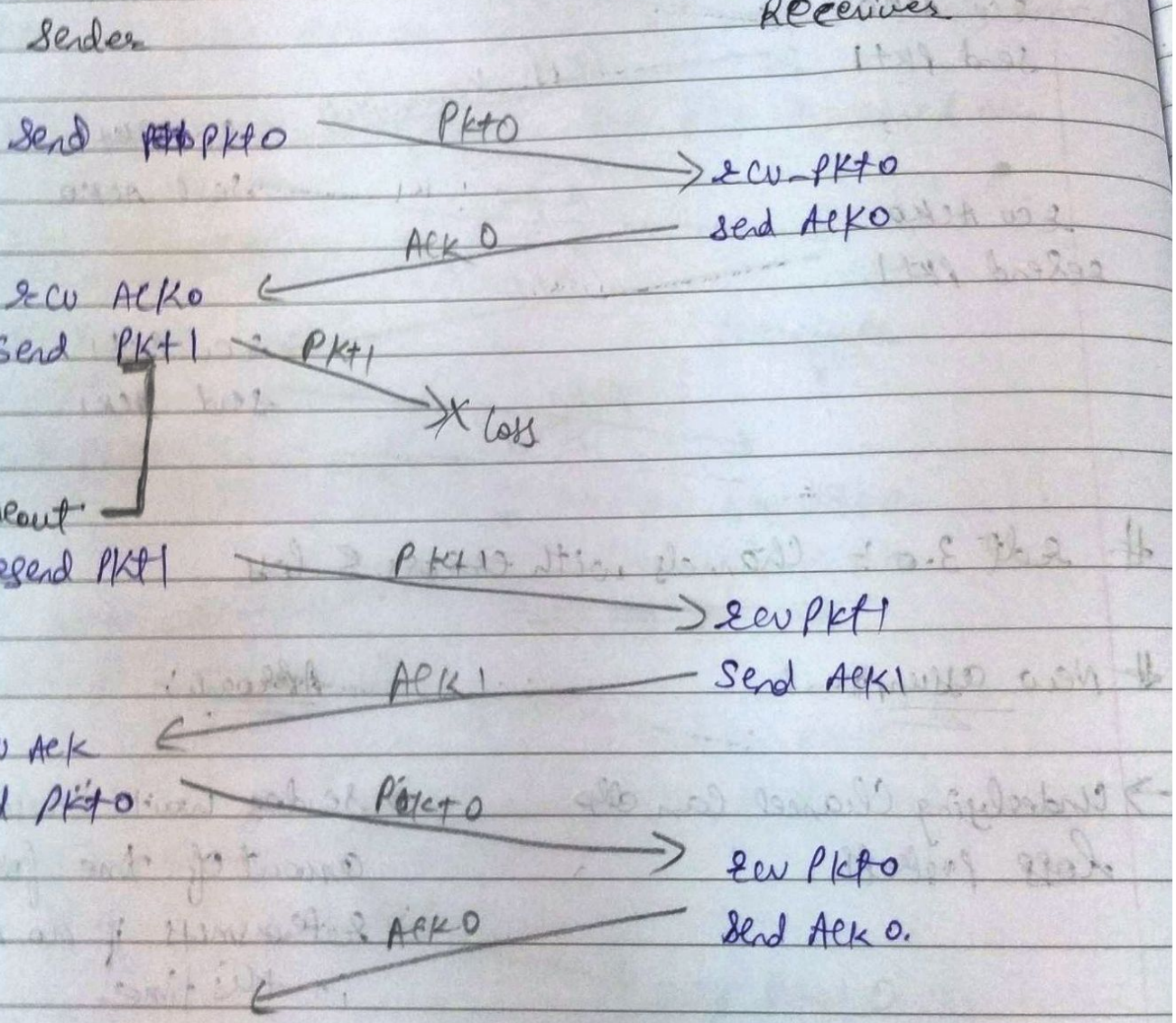
Approach :

→ Sender waits "reasonable" amount of time for Ack.
→ Retransmits if no Ack received in this time.
→ if Pkt (or Ack) just delayed (not loss):
  ↳ Retransmission will be duplicate, but seq # will handle this
  → Receiver must specify seq # of Pkt being Acked
→ ~~Good Condition~~ Countdown timer req.

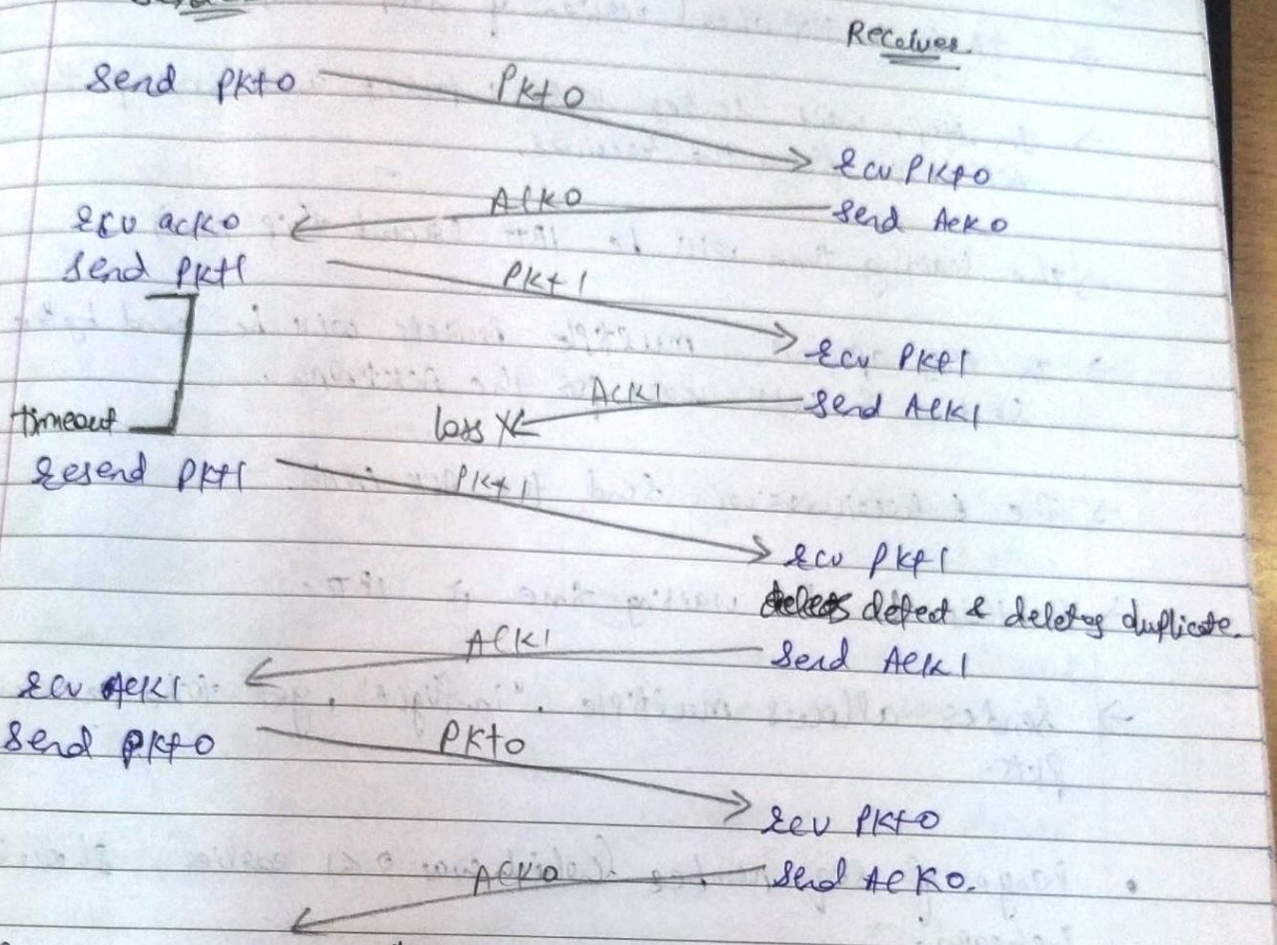
rdt 3.0 in action:

A.) No Corruption.

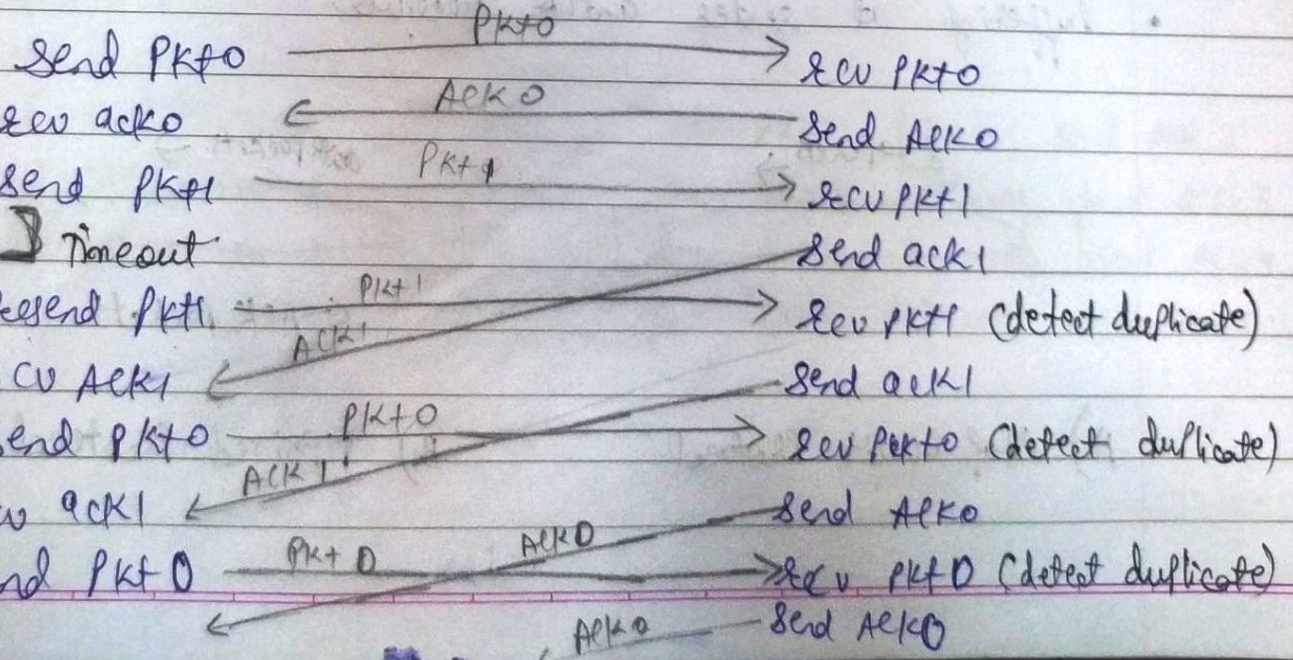
B.) Packet loss



c) ACK loss:-



d. Premature time out / Delayed ACK:-



Lec - 18.

Pipelined Protocol:-

→ It is a stop ahead version of stop & wait.

→ In stop n wait, sender sends 1 packet and wait for the ACK / NAK from the receiver.

→ The waiting time will be 1RTT (Round trip time)

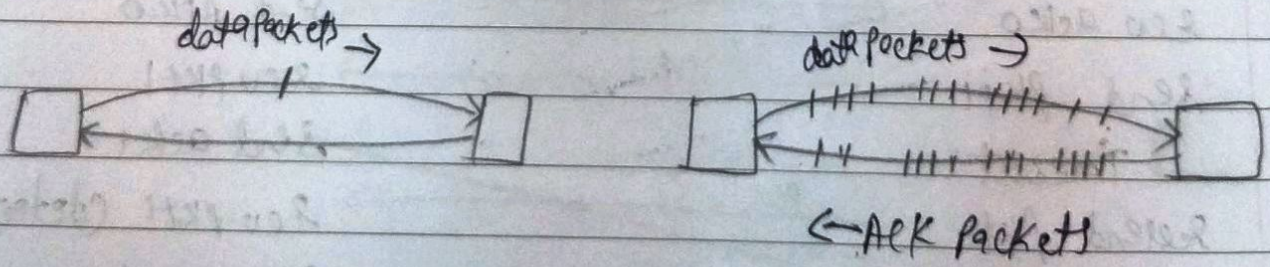
→ In this approach, multiple packets will be send by the sender & will wait for the ACK / NAK.

→ The receiver will send the ACK / NAK.

→ In this also the waiting time is 1RTT.

→ Sender allows multiple, "in-flight", yet-to-be-acknowledged pkts.

- Range of seq. number (which was 0 & 1 earlier) should be increased.
- buffering at sender and/or receiver



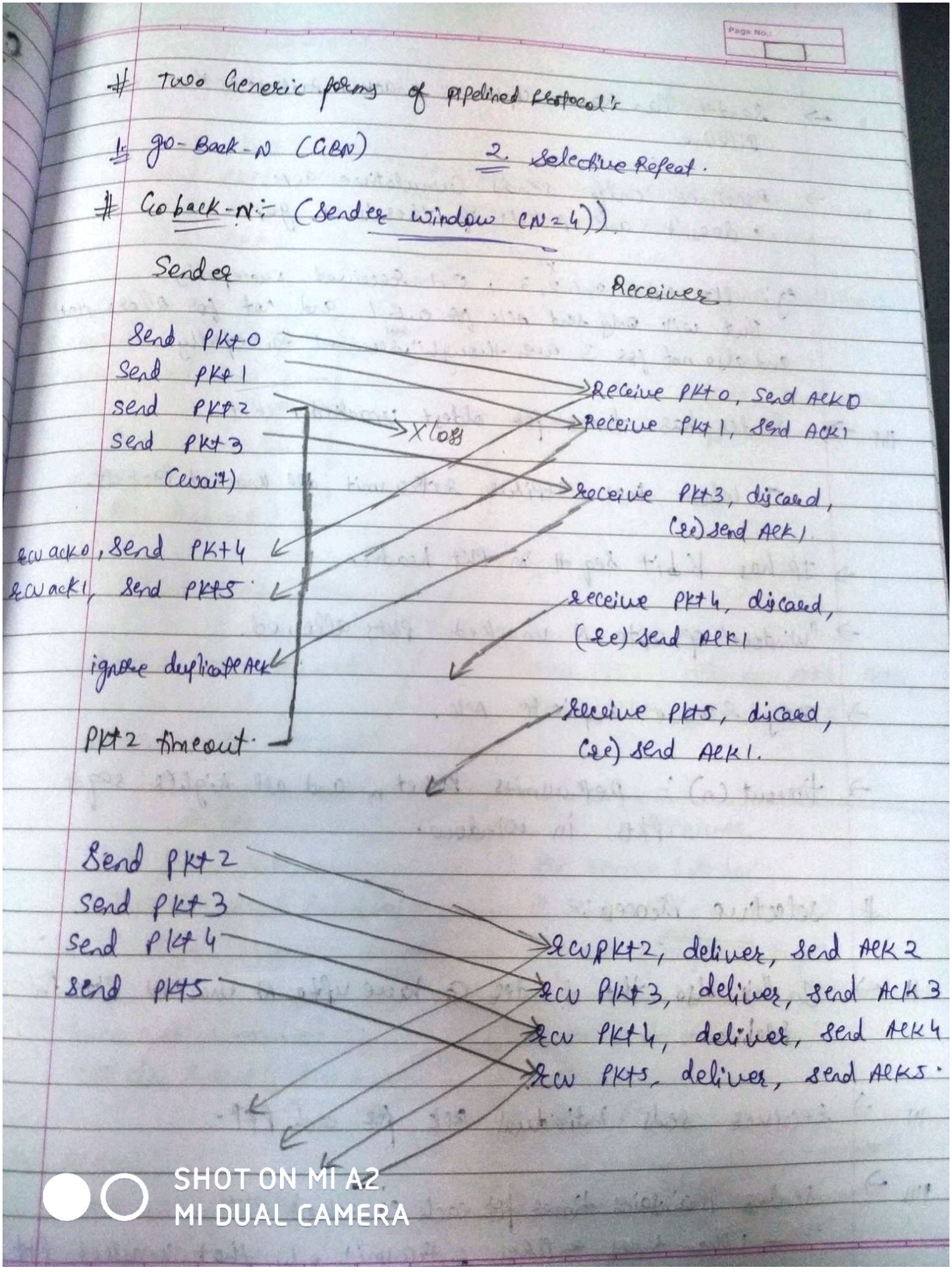
A) Stop & wait Protocol

B) Pipelined Protocol.

Two Generic forms of pipelined protocol's

1. go-Back-N (GBN)
2. Selective Repeat.

Go back-N:- (Sender window (N=4))



I → Sender can have upto N unacked packets in the pipeline

II → Receiver only sends cumulative ack.
- doesn't ack packets if there's a gap.

eg:- Pkt send → 0, 1, 2, 3, 0, 1 received successfully.
But will only send Ack for 0 & 1 and not for 2 (corrupted) and also not for 3 even though it may received successfully.

III → Sender has timer for oldest unacked packets.
- when timer expires, retransmit all unacked packets.

→ It has K bit seq # in Pkt header.

→ "Window" of upto N unacked Pkts allowed.

→ may receive duplicate Ack.

→ timeout (n) :- Retransmits packet n and all higher seq # Pkts in window.

selective Reapeat :-

II → In this also the sender can have upto N unacked Pkts in Pipeline.

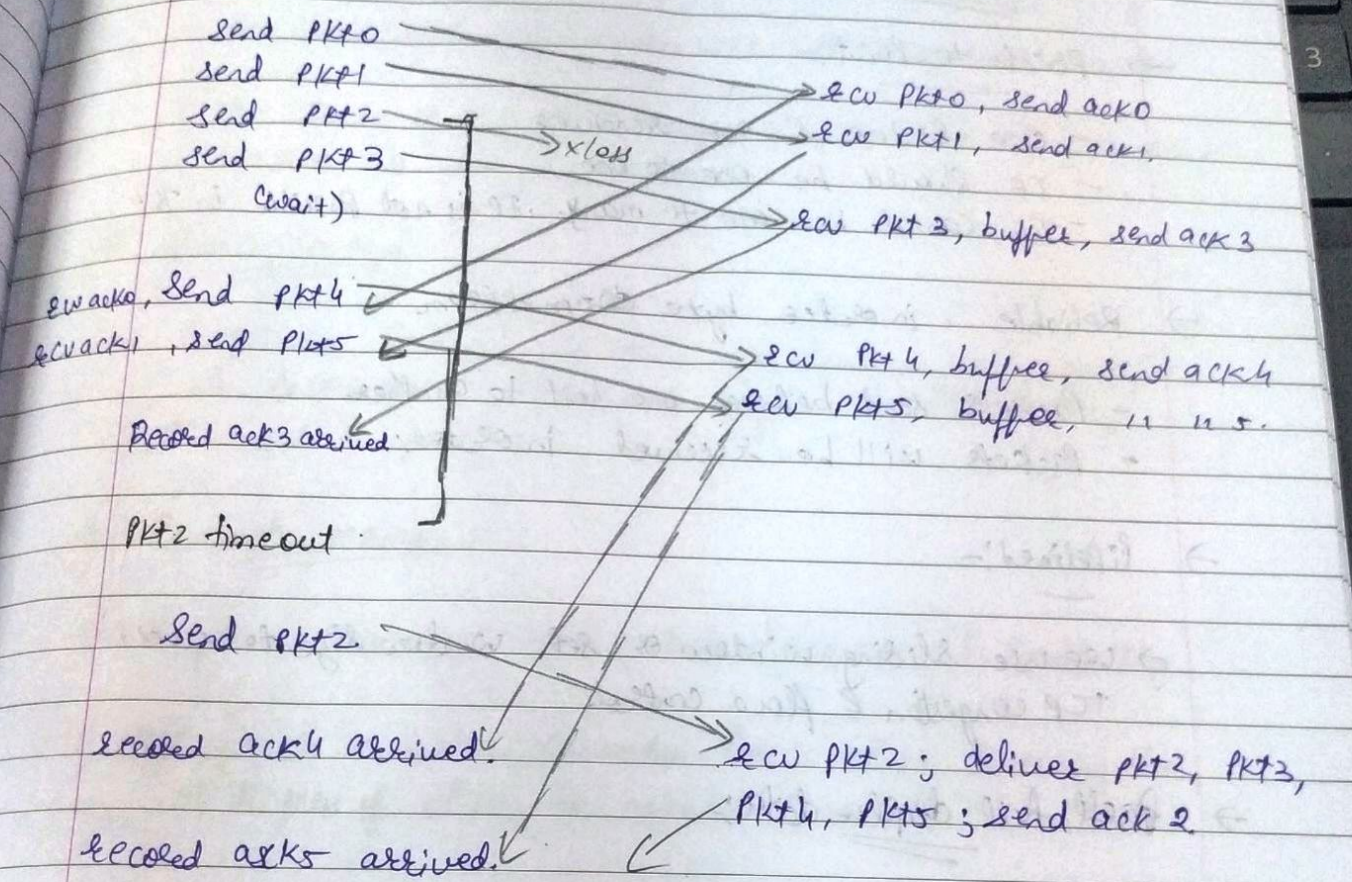
→ receiver sends individual ack for each Pkt.

→ Sender maintains timer for each unacked Pkts.
When timer expires, retransmit only that unacked Pkt.

[window n=4]

→ Sender

Receiver



Sender

→ data from above
• Available seq# Packet hai to send kiya jayega

→ timeout(n):
→ or else resend if time out.

→ Ack(n):
→ Ack mil gyi to received mark
→ or else (n received ack karte hai & re transmit.

Receiver

→ Pkt receive hota hai
→ Send Ack (n)
→ if out-of-order : Buffer.
→ in-order : deliver. Also deliver buffered, in order pkts.

⇒ Ack(n)

→ ignore (buffered)

Lec-14

TCP overview:-

→ Point-to-Point:-

- One Sender & one receiver
- It should be one to one
- It cannot be one to many. It is not possible in TCP.

→ Reliable, in order byte ~~stream~~ stream

- Packets send between one host to another
- Packets will be received in order

→ Pipelined:-

→ We use Sliding window or set window size to avoid TCP congestion & flow control.

→ ~~Full~~ full duplex data:-

- Bidirectional data flow at same time over a single link.
  eg:- ~~mobile call~~ Phone call.

- When data comes from app layer it is divided into segments, the size of the segment is given by MSS (maximum segment size). MSS include the size of the data & not the size of header (which is 20 bytes in TCP).

→ Connection oriented:-

- 3 way handshaking.
- Control msg will be exchanged which should be agreed by both sender & receiver in order to transfer the data or communicate.

→ flow Control :-

- The rate of the sender will not overwhelm receiver.
- The rate of the data transfer should in such a way that the receiver is able to process it.

flow Control :-

⇒ receiver controls sender, so sender won't overflow or overwhelm receiver's buffer by transmitting too much & too fast.

Connection management :-

→ Before exchanging data, sender/receiver do handshaking

→ Agrees to establish connection. (each other knowing the willingness of other to establish connection)

→ Agree on connection parameters.

Lec-15

Principles of Congestion control:-

* Congestion: [Too much traffic]

→ too many sources sending too fast & too much data for network to handle.

→ TO many sources sending too much data too fast for network to handle.

→ It is different from flow control.

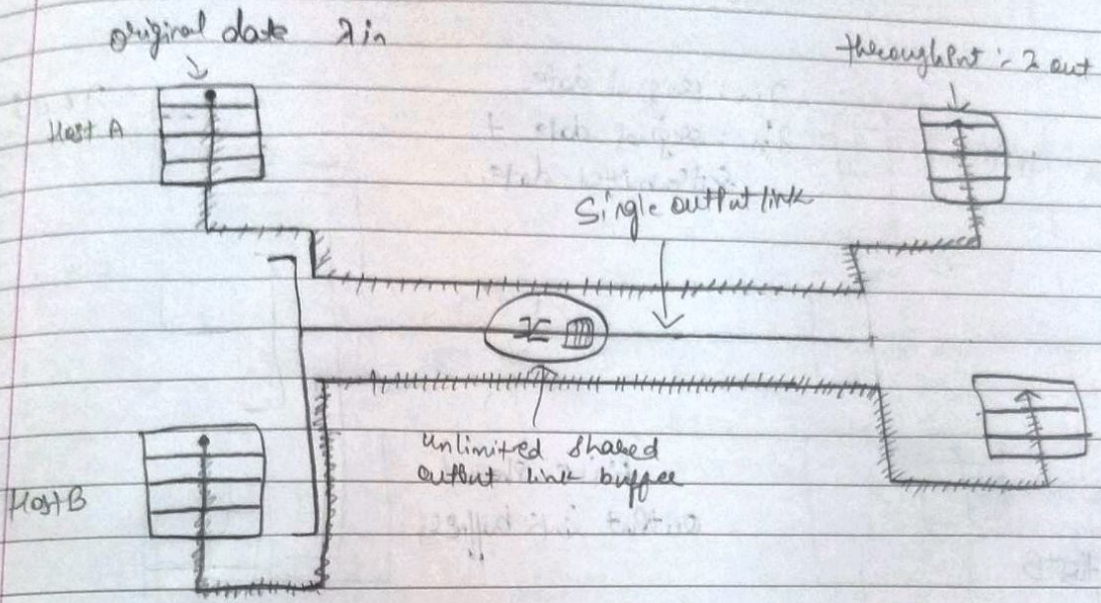
→ In flow control, the receiver inform the sender that at which rate the data should come.

→ flow control is always between sender & receiver but. Congestion Control is between. it deals with network components like bridges, routers and buffers. in those cases like whether the data send by the sender is received or not.

Whenever Congestion occurs, there are going to be 2 cases.

1. Lost Packets: Buffer overflow at routers
2. long delay: queuing in router buffer

Causes/cost of congestion: scenario 1.



→ Two senders, two receivers.

→ One router, infinite buffer.

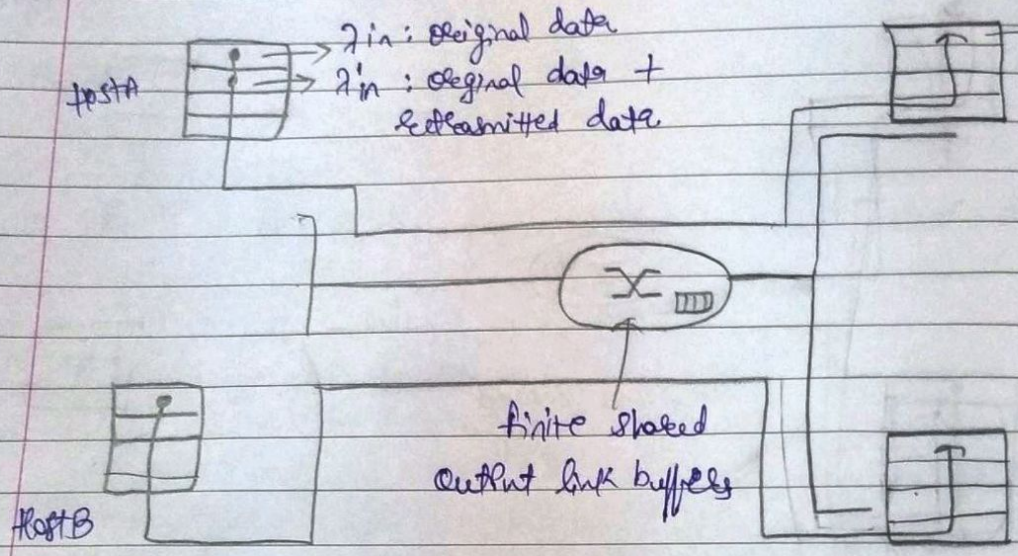
→ Having infinite buffers means the packet will be never lost.

→ output link capacity: R, therefore it will be divided among both the host equally. Each will get R/2.

→ No retransmission as data packets will never be lost.

→ There can be delay; because all the packets sendby host A & B are stored in buffer forming a queue. So there can be long delay if packets send are multiple.

Causes / costs of Congestion : Scenario 2 :-



⇒ One router, finite buffer.

→ Sender retransmits when queue pkt is timed out.

App layer input = app layer output : $\lambda_{in} = \lambda_{out}$

• Jitna data send kiya utna receive hoga

→ Transport layer input includes retransmission : $\lambda'_{in} \geq \lambda_{in}$

for eg :- Original data = 10 Packets

Retransmission = 15 Packets

receiver will get = 10 packets.

Idealization:

→ Sender sends only when buffer is available

→ Known loss:- Packets can be dropped at router due to full buffers.

→ Sender only resends if packet known to be lost.

→ Realistic: duplicates. Sender times out prematurely, sending 2 copies, both of which are delivered.

→ Packet took time in reaching to the destination & sender thought that the packet is lost resulting in sending the duplicate packet.

TCP Congestion Control:- [Additive increase multiplicative decrease]
↑ Algorithm

→ Approach:- Sender increases transmission rate (window size), ~~probably~~ Probing for user bandwidth, until loss occurs.

→ You will keepon increasing the transmission rate until loss occurs. when loss occurs, you will start decreasing.

- Additive Increase :- increase Cwnd (Congestion window) by 1 MSS every RTT until loss detected.
  ↳ max. Segment size

Cwnd :- Congestion window constraints or limits the sender on what rate should he send the data through which there will be no congestion.

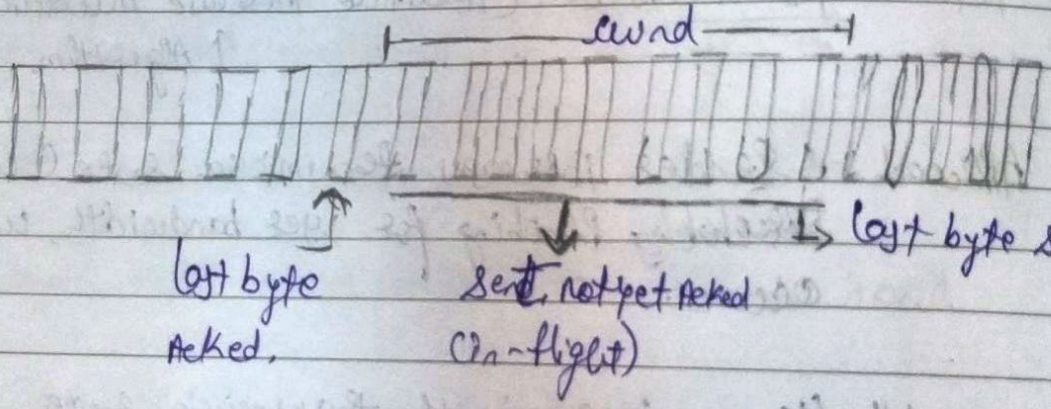
→ whenever you send 1 MSS and whenever you receive the ack for the same then in return you will increase the cwnd by 1.

→ for eg:- If you have sent 2 packets & after you have received the ack for both the pkts you will increase it by 1. So you will increase them by 2 times. Basically double.

→ Multiplicative decrease:-

→ whenever loss occurs cut cwnd in half.

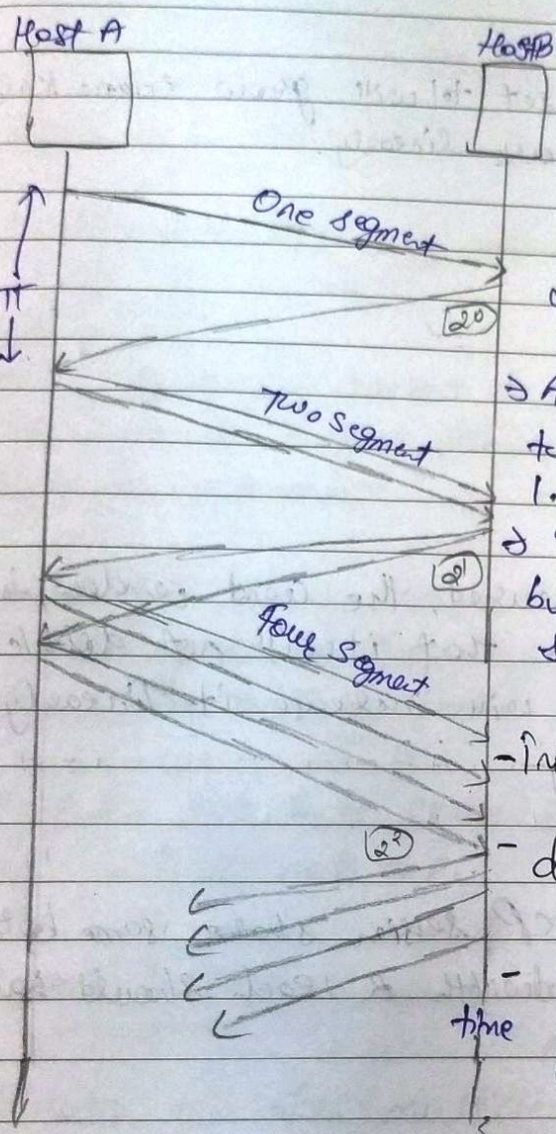
TCP congestion Control: Details.



Sender's transmission:-

LastByteSent - LastByteAcked ≤ cwnd.

TCP Slow rate :-



→ When connection begins, increase rate exponentially until first loss event.

→ After receiving the ack, the tcp will increase the MSS by 1.

→ Earlier it was sending 1 PKT but after receiving ack it will send 2 PKTs.

- Initially cwnd = 1 MSS.
- double cwnd every RTT.
- done by incrementing cwnd for every Ack received.

→ whenever the error is detected, the cwnd is cut to half.

Summary :-

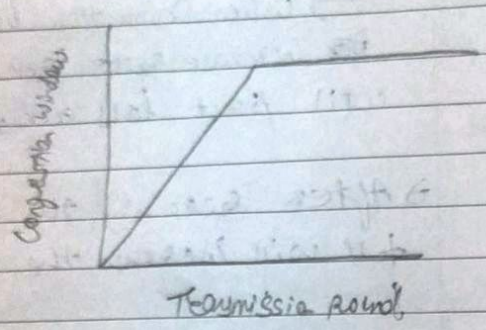
→ Initial rate is slow but ramps up exponentially fast.

detecting & reacting to loss :-

→ loss is indicated whenever there is timeout & no ack is received.

→ Also whenever the loss is indicated whenever we receive 3 duplicate ACK.

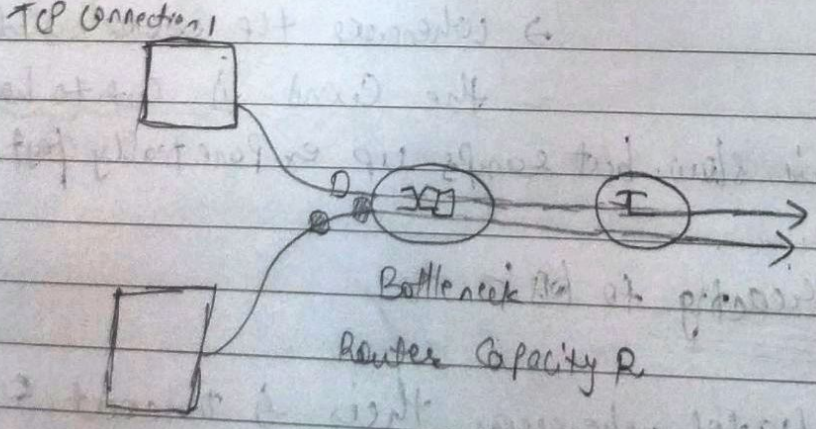
The window initially set to 1 will grows exponentially to threshold, then grows linearly.



→ Whenever the loss is occured, the cond window is cut to half and after that it will not double the cond but instead it will increase it linearly.

TCP fairness :-

fairness Goal :- If K TCP session share same bottleneck link of bandwidth R, each should have avg rate of $\frac{R}{K}$.



So every connection should get $\frac{R}{K}$
in this case it will get $\frac{R}{2}$

Why is TCP fair?

→ I DK...!

fairness & UDP:-

→ multimedia apps do not use TCP, they use UDP
(multimedia NPs)

→ they require constant data rate & they will do not decrease or reduce the data rate for anybody else in network.

→ while in TCP, if their is congestion in network then it will generate the nyg to all sender to reduce the data rate to control the data congestion.

→ To keep the data rate constant & high in multimedia app we use UDP & not TCP.

fairness in TCP:-

→ Application can open multiple parallel connections between two host

→ As we know the web browser uses or multiple parallel connections.

→ for eg:- There are 4 connections open so the bandwidth we have will be divided by $\frac{R}{4}$. So, even in multiple connections the fairness is maintained.

Lec-16

Network layer

→ The data comes from aff. layer to transport layer and add headers. & transport layer divides the data into segments. The main purpose of network layer is to transport the segments from sending host to receiving host.

→ On the sending side the network layer will add its header on the segment this is known as encapsulation

→ It encapsulates the segments into datagram

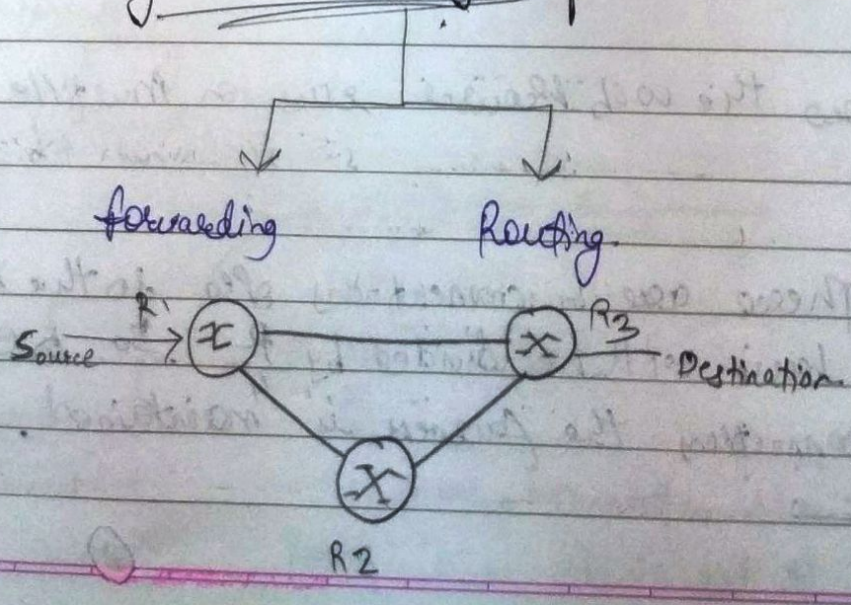
→ On receiving side, it delivers segments to transport layer

→ Network layer protocol is in every host & router whereas the transport L. Protocol was only on host

→ Routers passes the ~~datagram~~ datagram / Packet based on their IP address through the headers

→ On host 5 layers are implemented, whereas on routers only 3 layers are implemented (Network, Physical, datalink)

Two Key network-layer functions



Forwarding :- (Data plane)

→ Whenever the packet arrives from source to router, then the process of putting the packet from incoming port to outgoing port is known as forwarding.

Routing :- (Control Plane)

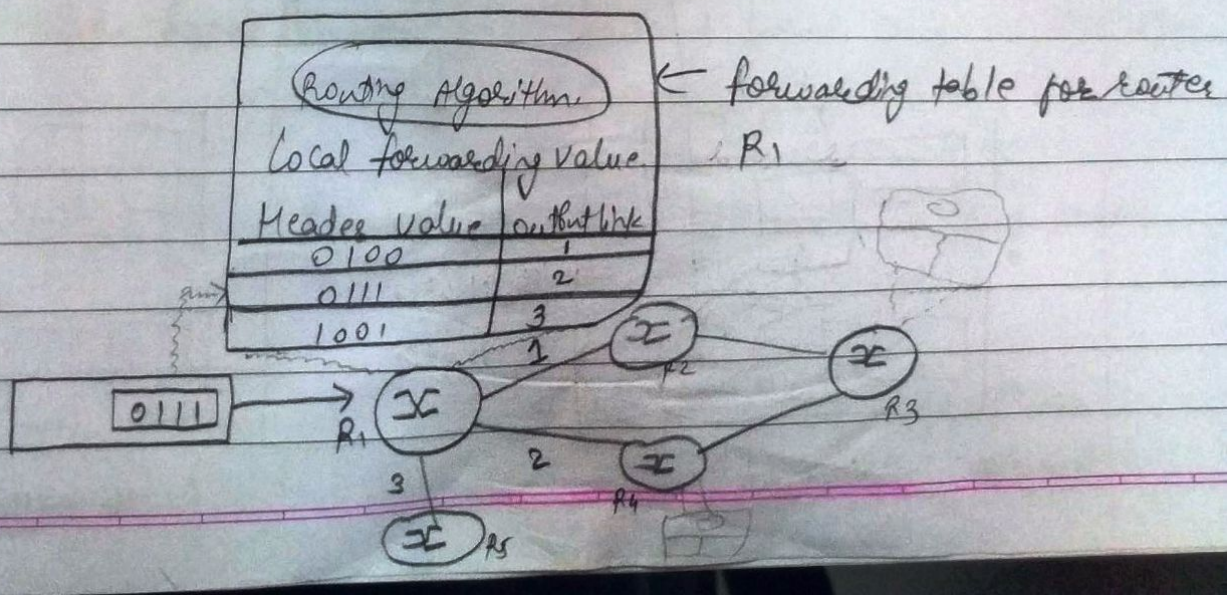
→ Routing is determining the path that is followed by the packets.

→ In the eg :- To send the packet from S to D we have 2 path. (S-R1-R3-D) (S-R1-R2-R3-D), here routing will determine which path should be taken

→ When we do routing, routing algorithm also takes place & in accordance to that forwarding table is generated.

→ Every router has its own forwarding table.

→ whenever it receives a packet it looks for the field in the header & that value will be already present in forwarding table & according to the value it is mapped to the output link.



⇒ Routing determines algorithm determines the overall path from one end to another end.

→ forwarding table determines the path from one router to another

Connection setup:- (Not in PDF) :-

→ In network layer - Between two host (may involve intervening routers in case of VCs)

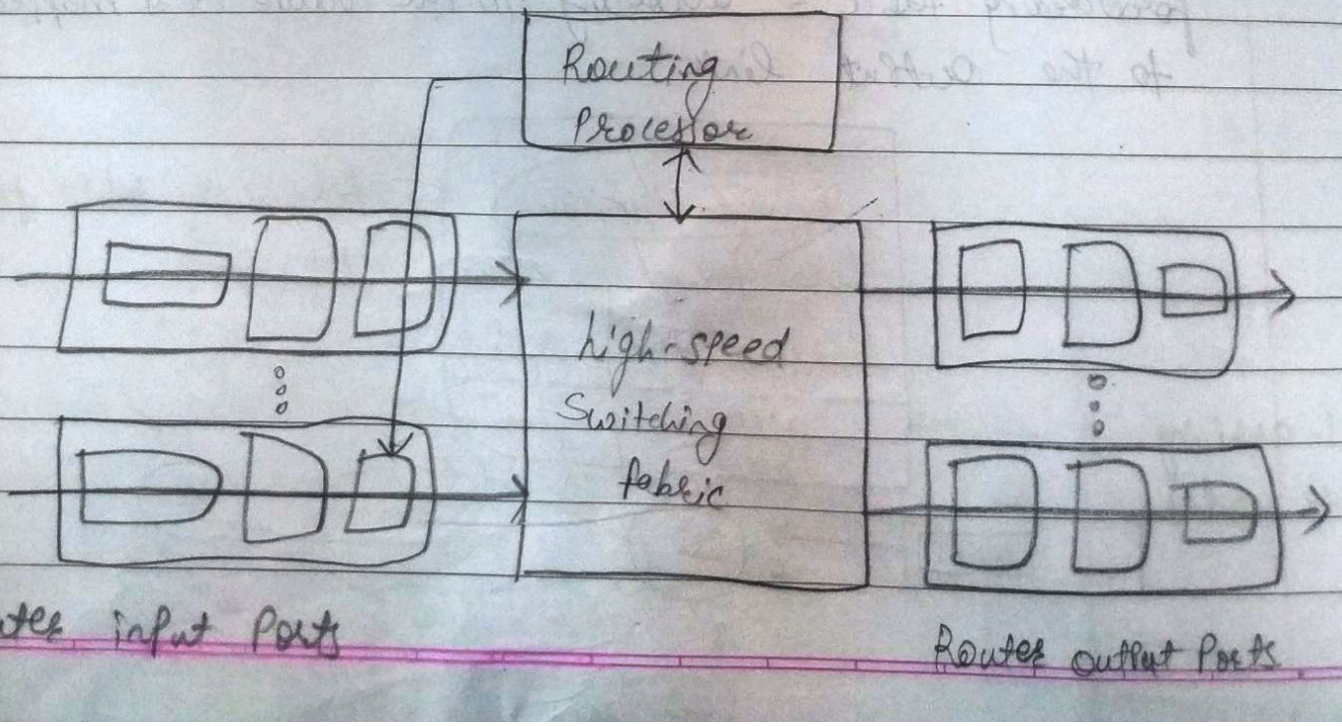
→ In transport layer - between two processes.

→ Similar to Transport lay., net. layer also have connection & connectionless service

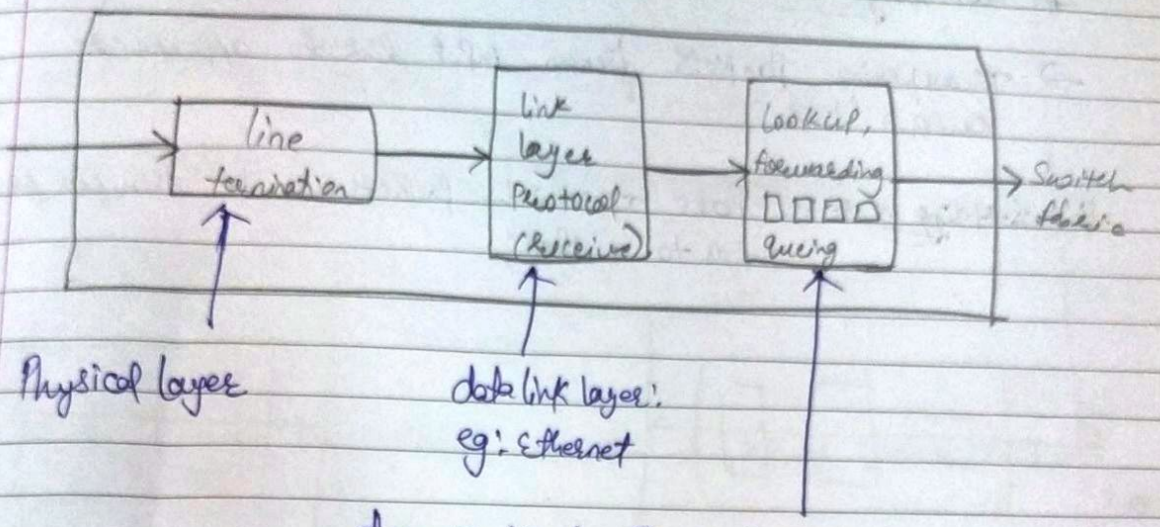
1. Datagram :- (Connectionless)

2. virtual-circuit :- (Connection oriented)

Router Architecture overview :-



Input port functions



decentralized switching :-

→ using header field values, lookup output port using forwarding table in input port memory

→ goal : Complete input port processing at 'line speed'.

→ queuing :- if datagram arrives faster than forwarding rate into switch fabric

→ destination-based forwarding :- forward based only on destination IP address

→ Generalized forwarding :- forward based on any set of header field values

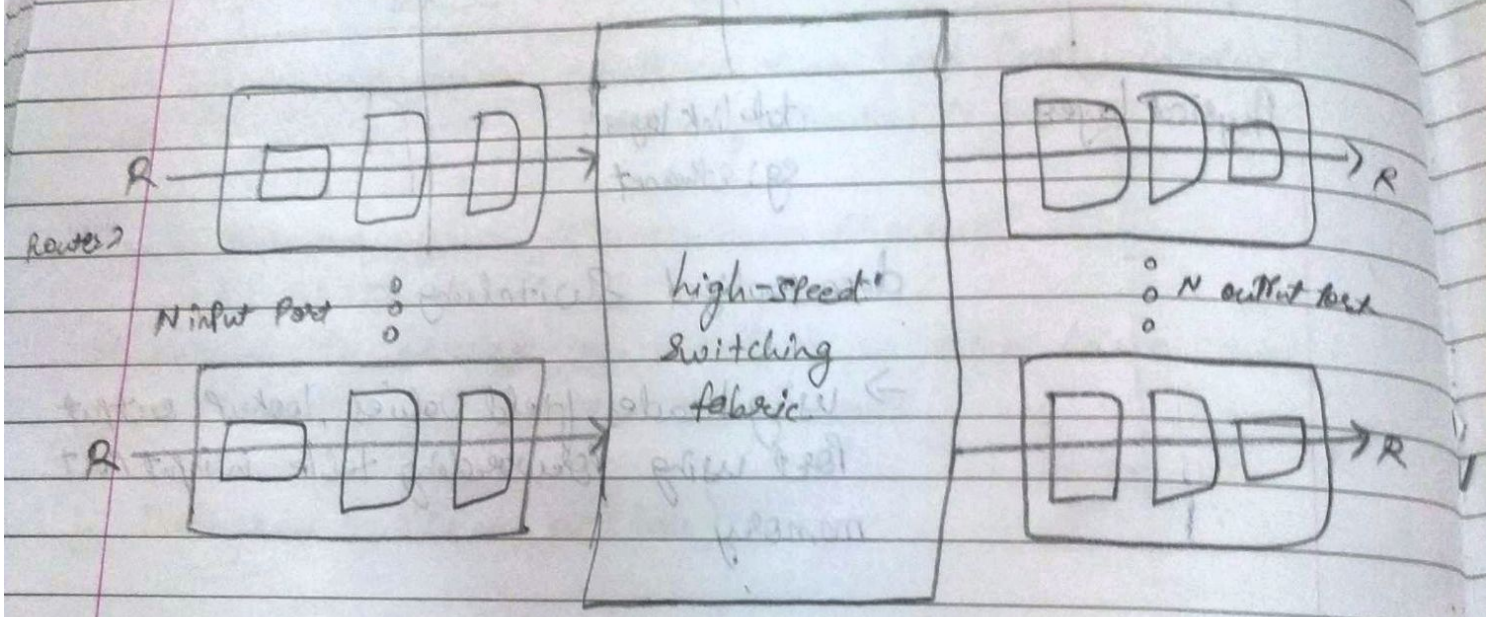
Destination-based forwarding & longest prefix match from PDF itself

Lec-17

Switching fabrics:

→ Transferring packets from input link to appropriate output link.

Switching rate: Rate at which packets can be transfer from input to output.



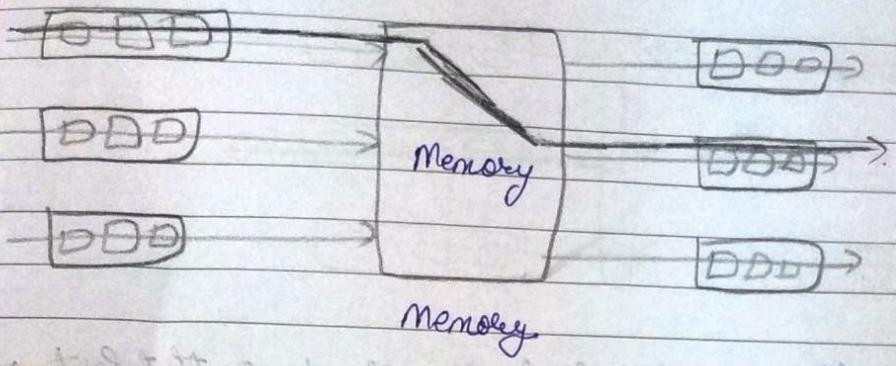
Three major types of switching fabrics:-

1.) Memory

2.) Bus

3.) Interconnection Network.

Switching via memory:-



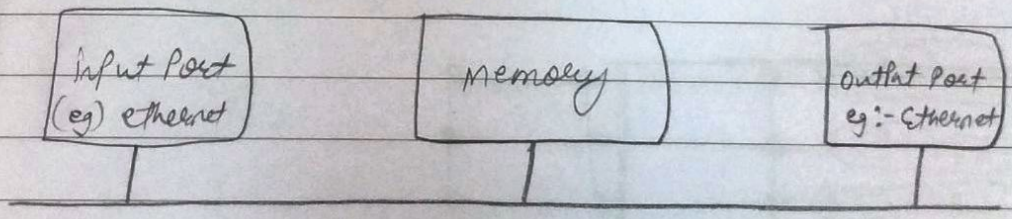
Memory

→ Traditional Computers with switching under direct control of CPU.

→ Packet is copied to system's memory.

→ Everything is connected through systems Bus.

→ Speed limited by memory bandwidth

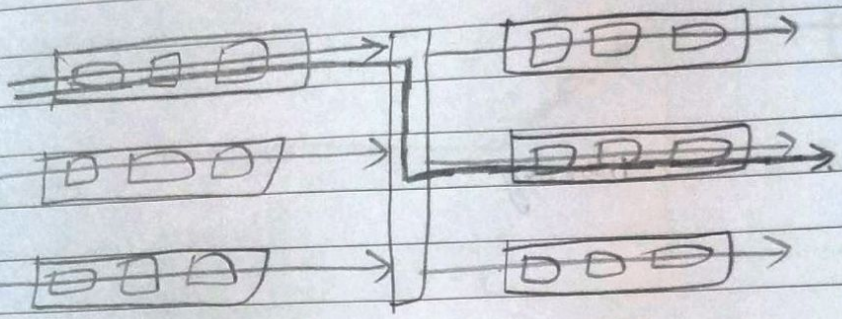


→ When you want to send the packet from input port to o/p port it will be send through memory.

→ Input port will send the packet to memory through System Bus.

→ Then further memory will send the packet to o/p port.

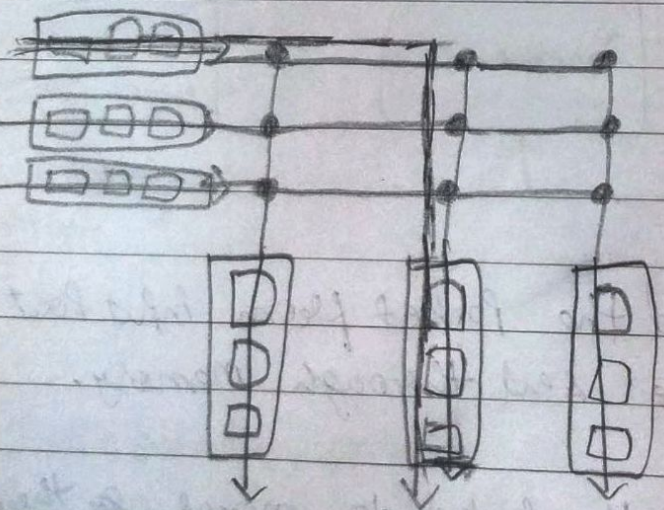
Switching Via Bus :-



→ Data flow from input port memory to output port memory via a shared bus.

→ The switching speed should not be more than by bandwidth otherwise the data will not be received properly.

Switching Via interconnection network :-



3×3 Cross bar

→