Date _____ Day _____

CHAPTER # 03
— Transport Layer —

→ TCP provides logical communication b/w application processes running on different hosts.
→ TCP is implemented at "end-systems", not in network routers.
↳ TCP on sender side → breaks application messages into segments, passes to network layer
↳ TCP on receiver side → reassembles segments into msgs, passes to application layer.

▷ Relationship Between Transport and Network Layer
→ Transport layer is communication b/w processes.
→ Network layer is communication b/w hosts
→ Within an end system, a transport layer protocol moves msgs from app processes to network edge (network layer) & vice versa.
→ Agr koi esi service like bandwidth or delay guarantee jo network layer provide nhi kr sakti to TCP tran layer protocol bhi nhi kr sakta.
→ Lek kuch services like reliable data transfer, encryption jo TLP provide krta hai lekn NL nhi krta.

Date Day

→ Transport layer actions

▷ Sender :

① TL is passed an application layer message
② determines segment header values
③ create segment
④ passes segment to NL

▷ Reciever :

① Reciever segment from NL
② checks header values
③ Extract application layer msgs
④ Demultiplexes message up to application via socket

• TCP (short intro)

↳ reliable, in order delivery
↳ congestion control
↳ flow control
↳ connection setup

• UDP

↳ unreliable, unordered delivery
↳ "best-effort" [it makes best effort to deliver segments blw two process but does not guarantee delivery services

this occurs on ... due to buffer size (occurs on intermediate nodes)

receiver's ability to receive the sender's packet.

occurs on receiver

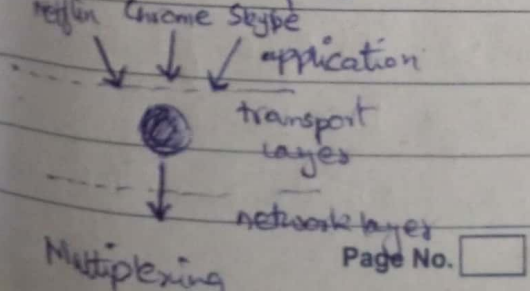
Date Day

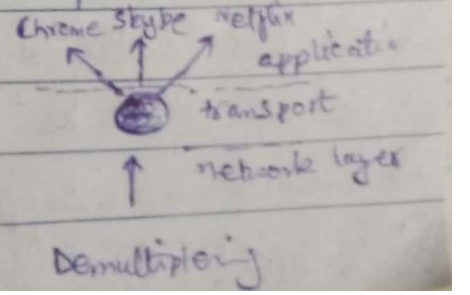
Congestion Control	Flow Control
(1) Prevents network-wide congestion (overloaded routers)	(1) Prevents sender from overwhelming the receiver
(2) Focuses on entire network traffic	(2) Focuses on individual connection b/w sender & receiver
(3) Applied when network is overloaded (packets get dropped)	(3) Applied when receiver can't handle incoming data fast enough
(4) Reduces sender's rate if congestion is detected	(4) Receiver advertises a window size (how much it can handle) - Sender adjust it's sending rate accordingly.

- Multiplexing & Demultiplexing:

→ Multiplexing – gather / collect data from multiple sockets (processes communicate with TLP using sockets) ,add transport header (like src, destination port nos, ~~ip addresses~~ etc) & send to NL

→ Demultiplexing – recieves segment from NL, uses the header info & deliver the segment to their respective sockets (process).



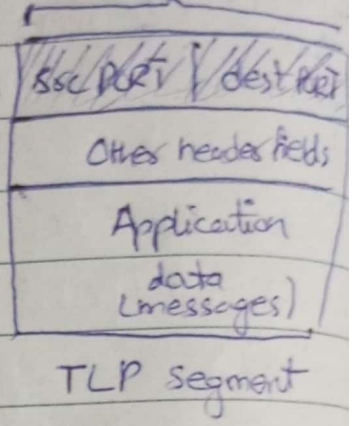


→ IP addr of src & dest is in IP datagram's header.

Date _____ Day

→ Each socket is associated with a PORT number ranging from 1024 – 65535. (16-bit number)

→ The header of a segment contains src & dest PORT number.



▷ Connectionless Multi/Demultiplexing (UDP)

▷ How Demultiplexing work?

① Host receives IP datagram
- i- datagram has src & dest IP
- ii- each datagram carries one segment
- iii- each segment has src & dest PORT

② host uses IP addr & PORT number to direct segment to appropriate socket

→ UDP socket is identified by two-tuple (destination IP, dest PORT)

→ A datagram with same dest. PORT #, but different src. IP address and/or source PORT will be directed to "same socket" at receiving host

▷ Connection Oriented Multi/Demultiplexing (TCP)

→ TCP socket is identified by a 4-tuple (src IP, src PORT, dest IP, dest PORT)

→ The host (destination wala) uses this tuple for demultiplexing

→ Kisi har socket ka tuple unique hoga hamesha.

→ Server may support many simultaneous TCP sockets

→ Each socket associated with a different connecting client.

Address Family Internet — used for TCP socket

Date Day 7

clientSocket = socket (AF_INET, SOCK_STREAM)

clientSaket.connect((serverIP, 12000)) → dest PORT #

↳ TCP client creates a socket & sends a connection establishment seg. → client sends a TCP segment with dest PORT # & a special connection bit set to 1 in TCP header. Segm also include src PORT #

At server:

connectionSocket, addr = serverSaket.accept()

↳ The server receives request & saves the 4-tuple value.

▷ Web Servers:

↳ Web server mein har ek process ke liye alag socket nhi create hota (as it will seriously effect performance)

↳ Web server often use only one ~~socket~~ process, and create a new thread with a new connection socket for each new client connection

↳ If client & server are using persistent HTTP, then throughout the duration of persistent connection the client & server exchange HTTP msgs via the same server socket.

↳ For non-persistent HTTP for each request/respon a new socket is created (a new TCP connection is created)

"Multi/Demultiplexing → happens at all layers is based on segment & datagram header field values"

Page No.

Date _____ Day

▷ USER DATAGRAM PROTOCOL :

→ It is very simple transport protocol.
→ Apart of multi/demuxing & some little error checking it do nothing
→ Segments may be "lost", "delivered out of order"
→ DNS uses UDP.
  ↳ Constructs a DNS query & passes to UDP
  ↳ Without any handshaking, UDP adds header field to message & passes to network layer
  ↳ NL encapsulates UDP segment into datagram & sends to name server.
  ↳ The host DNS waits for reply

- Finner application control over what data is sent, and when :
  ↳ UDP sends data immediately without waiting for congestion control or acknowledgments
  ↳ Realtime application need low latency & can tolerate some packet loss
- No connection establishment :
  ↳ UDP does not require a 3-way handshake, unlike TCP
  ↳ Faster transmission i.e why DNS uses UDP to resolve domain query names quickly -

Page No.

Date _____ Day _____

- No connection state : (NO RTT req)
  ↳ TCP mantains buffers & control parameters, UDP doesnot
- Lower Overhead :
  ↳ TCP header = 20 bytes
  ↳ UDP header = 8 bytes

→ HTTP now also runs on UDP, providing their own error & congestion control at application layer

- UDP segment Structure →

src PORT	dest PORT
length (header+data)	checksum
Application data (message)	

- Check Sum :

1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

wrap around → 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
+1

1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0 → Sum

↳ 1's complement

0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

↳ checksum.

Date _____ Day _____

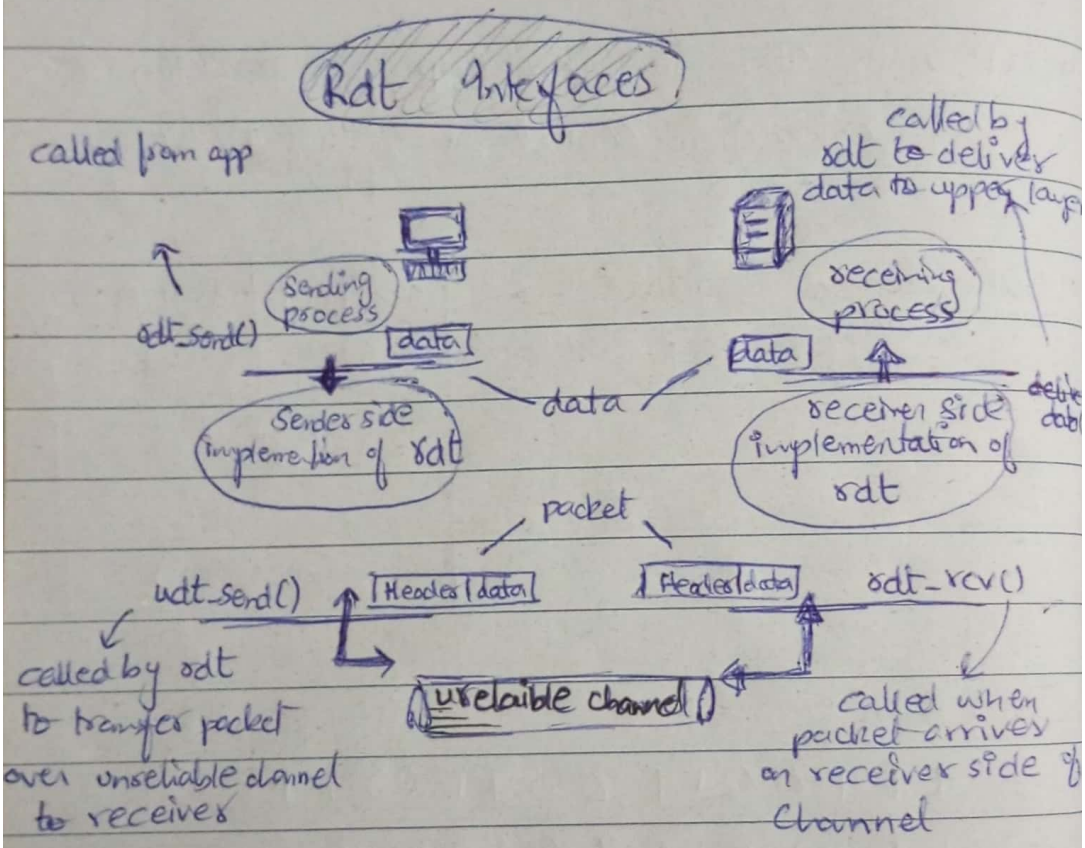
→ but there is a flaw in this as.

(sender side) 4 + 6 = 10

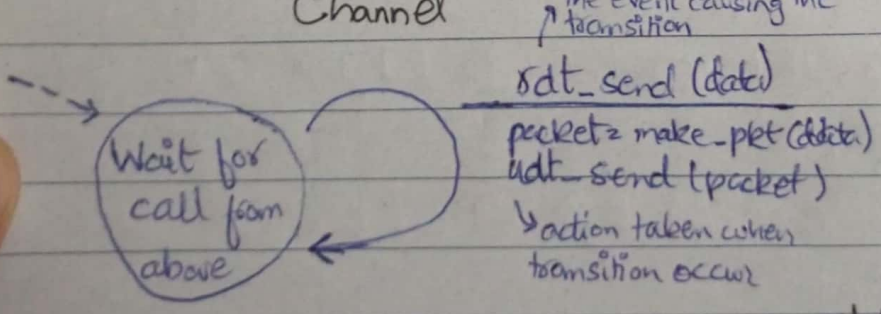
(receiving end) 7 + 3 = 10 → but no error detected as sum is same

changed the data but sum is same.

▷ PRINCIPLES OF RELIABLE DATA TRANSFER:



● rdt 1.0 : Reliable Data Transfer Over a Reliable Channel

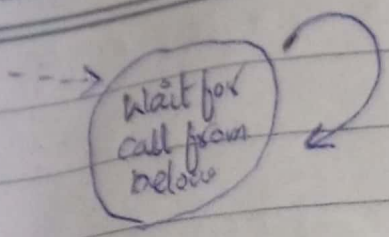


Λ → this symbol is used to represent no action / no event

Page No. ☐

▶ rdt 1.0 : sending side

Date _____ Day _____



▼ rdt 1.0 : receiving side

• rdt 2.0 : Channel with bit errors

↳ underlying channel may flip bits in packet

↳ checksum to detect bit errors.

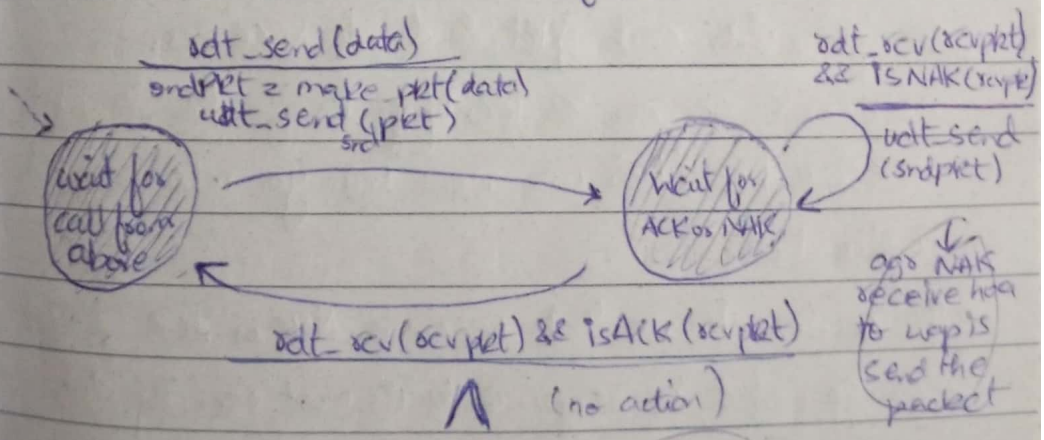
retransmission based on ACK & NAK is called ARQ protocols (Automatic Repeat reQuest)

→ acknowledgments (ACKs) : receiver explicitly tells sender that packet received is OK

→ neg acknowledgment (NAKs) : receiver explicitly tells sender that pkt had errors.

→ sender retransmits pkt on receipt of NAK

"Stop & wait" ⟹ sender sends one pkt, then waits for receiver's response



ggs NAK receive hua to wapis send the packet

ggs ACK rec hua to no action

▲ rdt 2.0 : sender

Page No. ☐

Date _____ Day _____

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)

extract(pkt,data)
deliver(data)
udt_send(ACK)

(agr packet shi to ACK send hoga)

wait for call from below

udt_rcv(rcvpkt) && corrupt(rcvpkt)

udt_send(NAK)

(agr packet corrupt to NAK send hoga)

→ rdt 2.0 has a fatal flaw:
↳ ACK or NAK can also be corrupted
↳ How can then sender know that receiver received the response or not?
↳ we can retransmitt data if ACK or NAK is corrupted
↳ Agr humein sec ki taraf se ACK receive hoga tha or wo corrupt hogya to sender wapis send krdega lekin phir sec k paas duplication hojayegi data ki.

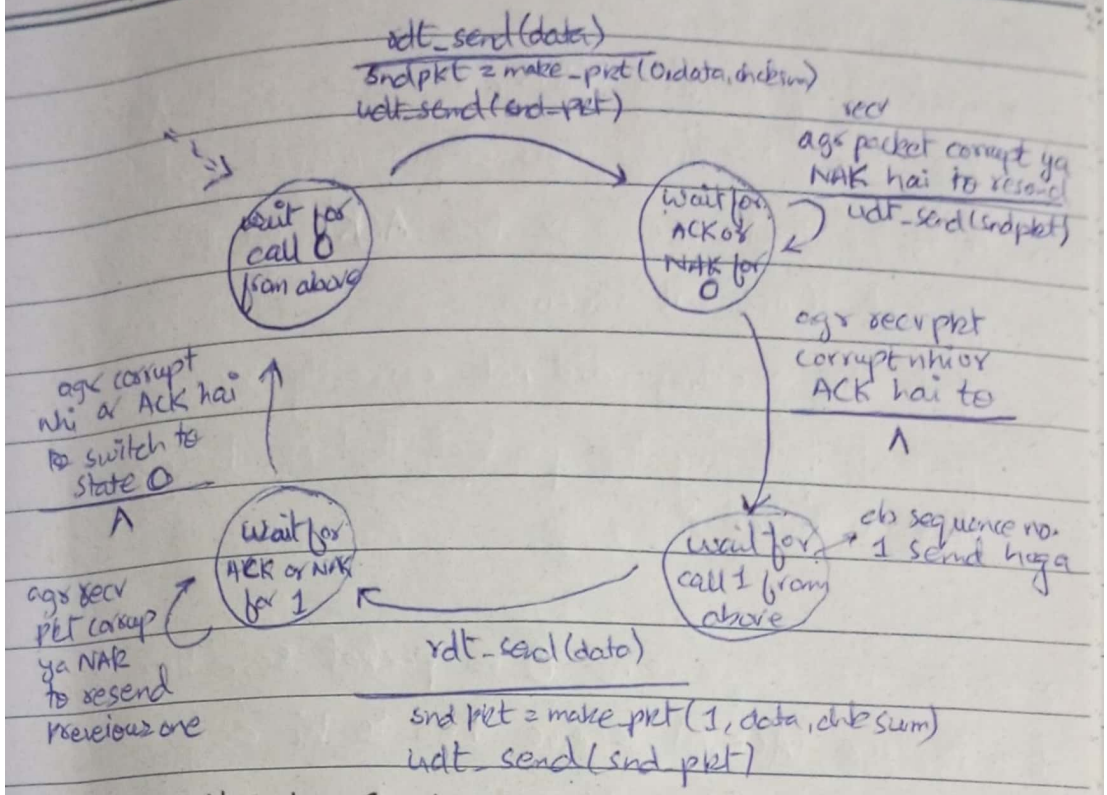
this is a 1 bit no. (0 or 1)

To deal with this problem we add sequence number with each pkt send to receiver.
↳ receiver seq no. se identify krega ke jo pkt receive hoa hai wo new hai ya retransmitted wala hai.

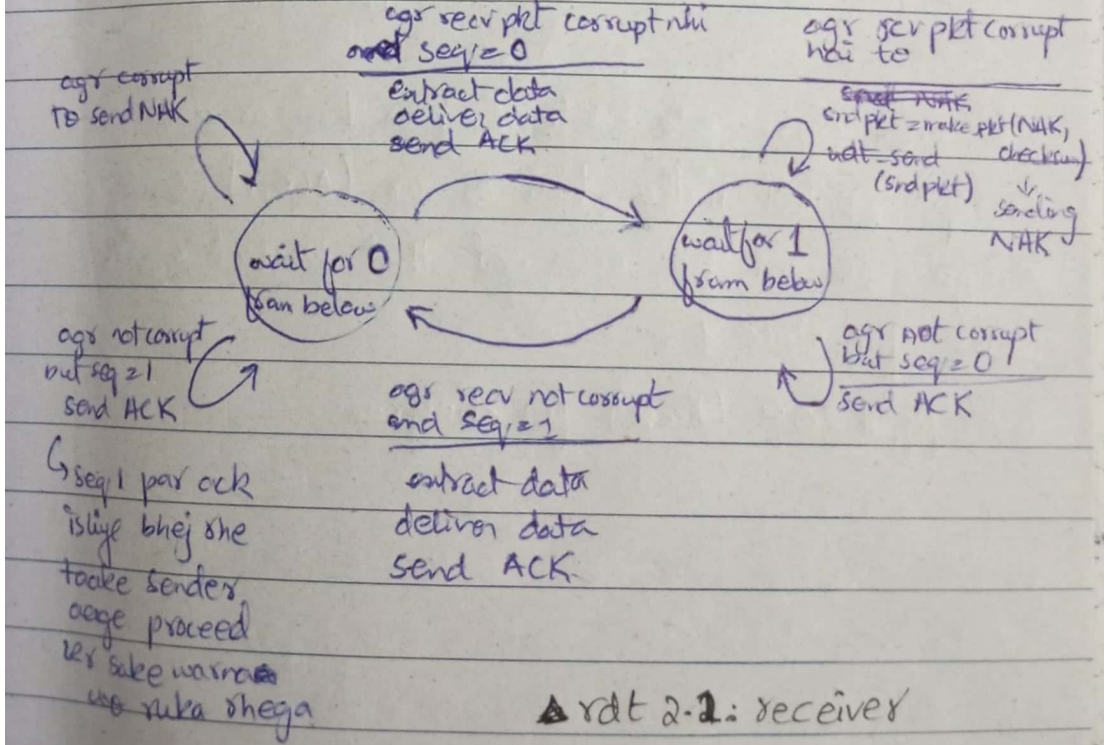
→ The sender knows that received NAK or ACK packet was generated in response to it's most recently data packet.

Page No. ☐

Date Day



▲ rdt 2.1 : Sender



↳ seq 1 par ack isliye bheja she taake sender aage proceed kr sake warna wo ruka rhega

▲ rdt 2.1: receiver

Page No.

Date _____ Day

rdt2.2: NAK free protocal

→ instead of sending NAK, receiver sends ACK for last ~~received~~ correctly received packet.

→ A sender that receieves two ACKs for the same packet (means it receives duplicate ACKs) knows that the receiver did not correctly receive the packet following the packet that is being ACKed twice.

→ sender ne sequence no 0 ka packet send kraa hai to use (ACK,0) receive hoa to hi aage bdhega agr (ACK,1) recieve ho means ne needs to retransmit. Same for seq# 1

→ receiver agr ~~sate~~ state 0 par hai or use corrupt packet milta hai to wo (ACK,1) send krega so that he can get that packet again same for state 1.

see pg# 209 (book) for diagram.

Date Day

alternating-bit protocal

rdt 3.0: Channels with error & Packet loss

↳ Sender waits for a reasonable amount of time for ACK (time could be RTT)

↳ if no ACK received retransmitts

↳ if packet or ACK is not lost but delayed

↳ retransmission will be duplicate, but seq# already handle this

↳ receiver must specify seq# of packet being ACKed.

→ When sender sends packet it starts timer & when it receives ACK it stops it

→ If sender receives an ACK, 1 when it is in state 0 it doesnot ~~resen~~ retransmitts, instead it do nothing & only retransmitts when it is timeout.

side pg #64

- Pipelined Reliable Data Transfer Protocol :

→ RTT = 30 ms

→ R = 1 Gbps

→ L = 8000 bit

→ $d_{trans} = L/R = 8000/10^9$ bit/sec $= 8\ \mu s$

it takes 8 micro sec for sender to completly send its packet on channel.

Page No.

Date _____ Day _____

→ Then he recieves an ACK after 30ms so total (t = RTT+L/R) 30.008msec time will be spent to sent just one. Then after 3.008ms the next packet will be sent.

→ Sender Utilization → fraction of time the sender is actually busy sending bits into channel.

$$U_{sender} = \frac{L/R}{RTT + L/R} = 0.00027$$

▷ Solution:

↳ Rather than operate in a stop-and-wait manner, the sender is allowed to send multiple packets without waiting for ACK.

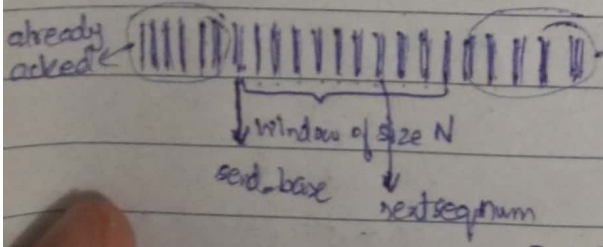
→ Agr ek waqt mein 3 packets saath bhej rhe hai to

$$U_{sender} = \frac{3L/R}{RTT + L/R} = \frac{0.024}{30.008} = 0.00081$$

Go-Back-N:

• Sender:

is mein ek window size of N hota hai that tells, consecutive transmitted but unAcked pkts



Date Day

Ack(n): Acks all packets upto, including seq#n or sabke ack reclere hojaye 1st packet ki to window ek seqnumber aage move kr jayegi.

→ Agar timeout hojaye to n^{th} packet or uske age jitne bhi window mein packet honge sabko retransmitt krdega.

• receiver k end par agr ko packet nhi aata to jo last acknoledged packet hoga uski ack wapis send hojayegi

out-of-order packets deal nhi honge.

→ sender ko jab duplicate ACK receive hogi to wo uske ignore krdega or jo ~~wind~~ window mein jo packets honge unko transmitt krdega.

Selective Repeat:

- Sender har packet ka timer maintain kr rha hota hai.
- agr ~~to~~ timeout hogaya to sirf wohi packet resend hoga.
- reciever ka par agr out of order reciere hoe to wo last acked pkt k baad waale saare packets ko buffer mein store krleta hai.

like sender ne 0,1,2,3 send kra 0,1 pohch gaya or ack bhi rec hogai but 2 loss hogaya. Acha receiver ka par 0,1 or 3 aya. 0 or 1 ka ack to wo send krchuka tha leken 2 nhi aya or 3 aagya to usne 3 ko buffer kr li. Sender ka jo 2 ka timeout hoga to usko wapis resend or uski ACK send krdi

Page No.

Date _____ Day

kdia. Plus receiver ko jab 2 receive hua to usne 2 ki ACK bhejdi or pkt 2 & 3 ko application layer ko deliver kr dia.

→ Application layer ko kabhi bhi out of order packets send nhi kr sakte.

"Connection-Oriented Transport"

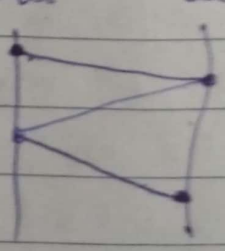
▷ TCP Connection :

→ TCP connection provides a full duplex service.
↳ If there is a connection b/w A & B then both can send & rec data.

→ TCP connection is always point to point i.e b/w a single sender & a single receiver.

→ Multicasting (transfer of data from one sender to many receivers) is not possible in TCP.

→ ~~TCP~~ Tota 3 segments are sent b/w sender/rec therefor this connection establishment is called 3 three way handshake



Date _____ Day _____

→ Application layer se data TCP k send buffer mein store hojata hai.

→ Isi buffer se data network layer ko pass hota hai

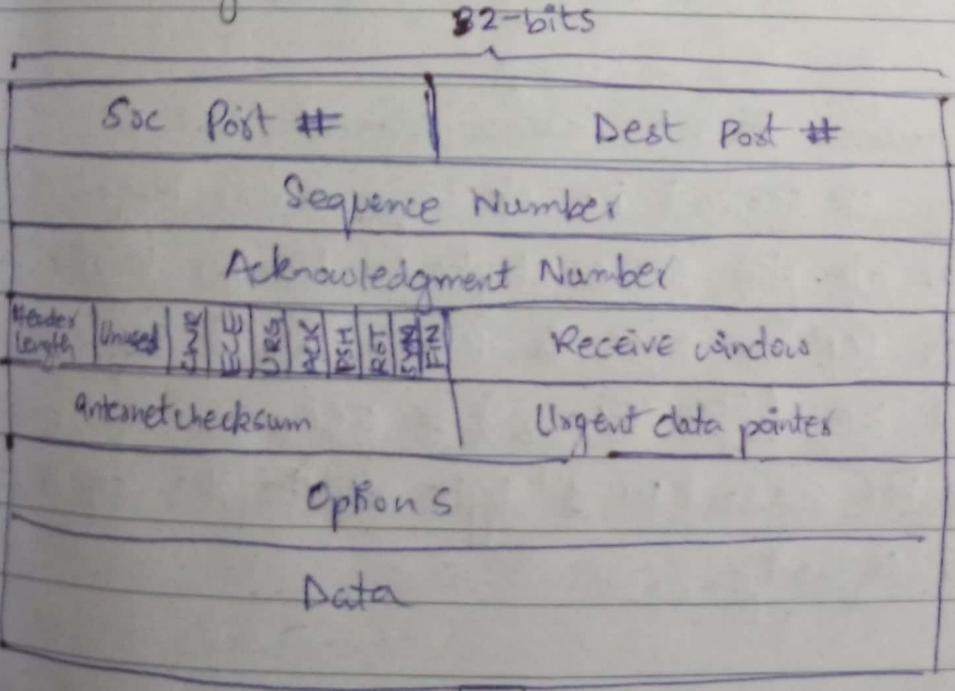
→ Ek segment mein max kitna data ja sakta hai that is refferred to as maximum segment size (MSS)

→ Ye MSS hamare link layer k frame k size par depend karta hai that is called maximum transmission unit (MTU)

→ Note: MSS is the max amount of application layer data in the segment, not the max size of TCP segment including headers.

→ TCP pairs each chunk of client data with a TCP header, thereby forming TCP segment

▷ TCP Segment Structure



Date _____ Day _____

kyuki phir receiver ko jab 2 receive hoga
usne 2, 3 dono ki ACK bhej di.

- receiver window (16-bit) used for flow control
- header length field (4-bit) specifies length of TCP header
- options field
- flag field (6 bits)
  - ↳ ACK used to indicate that the value carried in the ack field is valid; i.e the segment contains ack of a seg that has sucessfuly received
  - ↳ RST,
  - ↳ SYN & FIN is used for connection setup & teardo
  - ↳ CWR & ECE used for congestion notification
  - ↳ PSH indicates the receiver should pass data to upper layer immediately
  - ↳ URG indicates that there is data in this seg that the sending layer has marked as "urgent"
  - ↳ the location of last byte of this urgent data is indicated by urgent data pointer field.

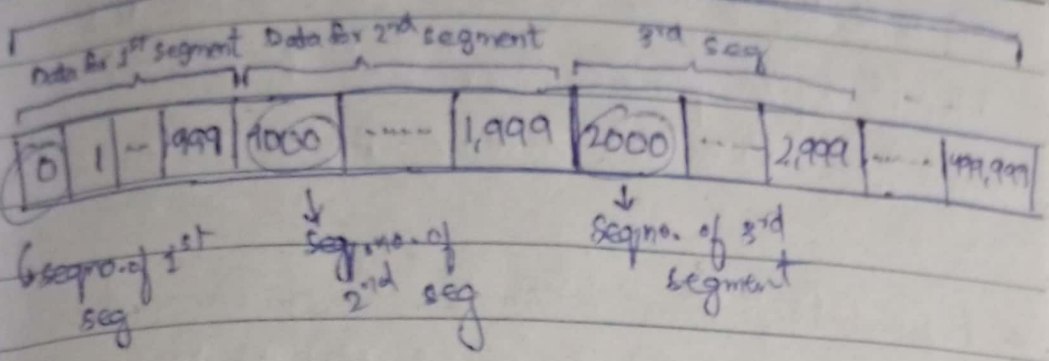
- Sequence No. & Acknowledgment No.s

→ Har ek segment ka ek sequence no. hoga.

→ Agr ek file 500,000 bytes ki transfer krni hai over TCP or MSS is 1000. Ttotal segments will be (500,000/1000) = 500.

→ Ab har segment ki 1st byte will be the seq. no of that segment.

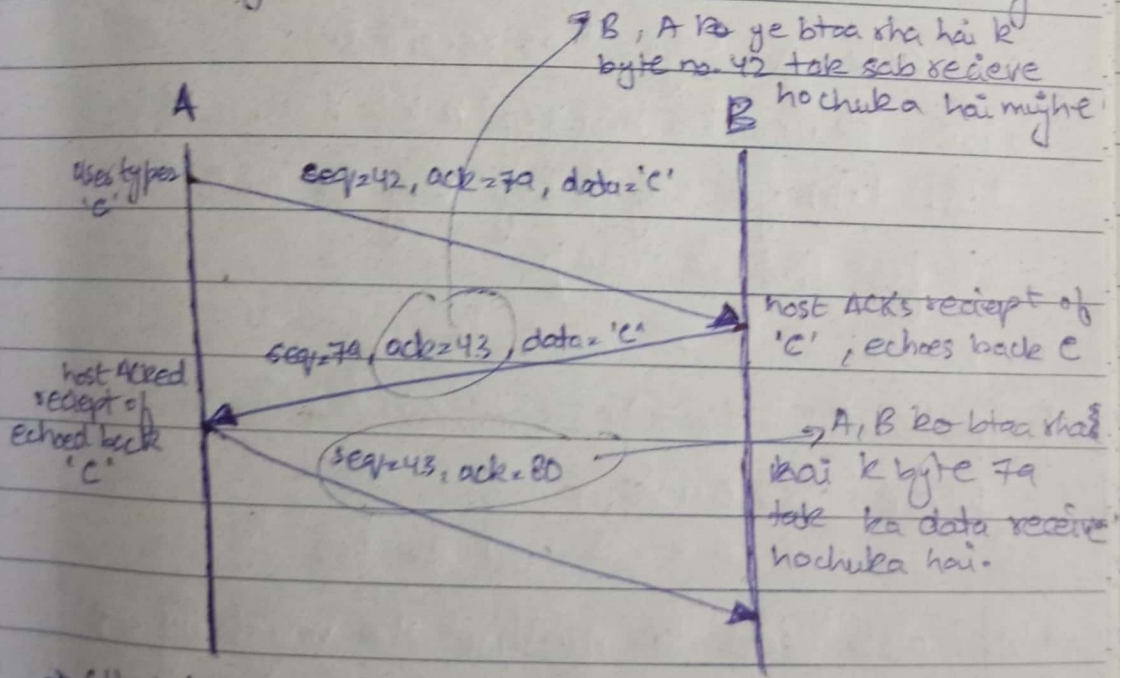
Date _____ File _____ Day _____



Acknowledgment Number:

The ack no. that HostA puts in its segment is the sequence no. of the next byte Host A is expecting from Host B.

Example: Agr A, B ko 0-999 bytes wala segment send kreg B ko to wo ack mein next segment ka seq no. i.e 1000 send krega.



→ Client to server jo data bhej rhe hai uska ack server-to-client wale segment mein reciere hota hai. This is called piggybacked acknowledgment

Date _____ Day _____

→ Agr out of order segments receive ho TCP me... to its upr programmer that how to deal with it. i) either discard ii) or buffer

- Round-Trip Time Estimation and Timeout :

→ Timeout ki value hamesha RTT se badi honi chahiye warna unnessary retransmissions hongi

→ Agr timeout ki value bohot zyada rakhdi to agr segment loss hogaya to retransmission will be very slow.

- SampleRTT, for a segment is the amount of time between when the segment is sent & when an ack for segment is received.

G isko ek martaba kisi seg k liye measure ki jata hai. Lekin ye seg to seg vary kr sakta hai so for that we use ↓

$$EstimatedRTT = (1-\alpha) * EstimatedRTT + \alpha * SampleRTT$$

Where $\alpha = 0.125$

→ We also need to measure the variability of RTT

$$DevRTT = (1-\beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT|$$

Where $\beta = 0.25$

Date _____ Day _____

→ Now the final formula for timeout is

TimeoutInterval = EstimatedRTT + 4*DevRTT

(4*DevRTT: safety margin)

→ See slide# 19

• Doubling The Timeout Interval

→ Jab bhi timeout expire hota hai to new timeout is twice the previous value (not calculated thru formula)

→ Agr data application se receive hua ho ya acknowledgment recieve hoi ho tab hi sirf formula se value derive hoti hai.

• Fast Retransmitt : (before timeout)

if sender recieves 3 additional ACKs for same data resend unACKed segment with smallest seq#

→ TCP Comparasion with Go-back N & Selective Repeat

↳ TCP's error recovery mechanism is a hybrid of Go-Back-N & SR.

1. TCP is similar to GBN because
   - It uses cumulative acknowledgments
   - The sender tracks only the smallest unack seq#
   - out of order seg are not immediately acked

Date _____ Day

2. TCP differs from GBN because
- Many TCP ~~application~~ implementations buffer out of order segments, unlike GBN
- It doesnot always retransmitt all subsequent packets when a seg is lost.
  ↳ Example: Sender ne 1, 2, 3 -- N segments send kre & the arrived correctly. But ACK for n^{th} segment is lost. Agr GBN hota to timeout hote hi n^{th}, $n+1$, $n+2$, $n+N$ packets retransmit hote. Lekin TCP sirf n^{th} packet transfer krega timeout pr. Jab tak n^{th} ki ACK receive nhi hogi $n+1$ seg ki retransmission nhi hoga. Jab Ack reciere hogi to usmein bhogi $n+1$ -- $n+N$ sabki cummulative ACK aajayegi ~~(illegible)~~. And then it would not retransmit $n+1$ seg.
- Lekn agr timeout hone se phele $n+1$ ACK aaygi to n^{th} seg bhi retransmit nhi krna pareega q k $n+1$ ACK ka mtlab hai k usse phele k tamam seg recev ho chuke hai.

Sender ko agr ACK $(n+1)$ mil gya hai to iska matlab ye k k usse phele tamam seg rec ho chuke hai. LEKN & GBN ACK $(n+1)$ ko ignore krdeta hai.

Date _____ Day _____

3- TCP is similar to SR

- The reciever ACKs out of order segments individually
- The sender retransmits only lost seg rather than unessecary ones.

▷ Flow Control :

→ TCP provides a "flow-control service" to its application to eliminate the possibility of the sender overflowing the receiver's buffer.

→ It is like speed matching service "matching the rate at which the sender is sending against the rate at which receiving application is ready"

→ The receivewindow (rwnd) field in TCP header is used to control flow control.

→ it gives idea to sender that how much free space is available at receiver

# of bytes receiver willing to accept

→ 4096 bytes

→ The receiver allocates a RcvBuffer to a connection, therefore inorder to permit overflow of this buffer

$$LastByteRcvd - LastByteRead \leq RcvBuffer$$

Now, the rcwnd is set to the amount of spare room in the buffer

$$rwnd = RcvBuffer - [LastByteRcvd - LastByteRead]$$

Page No. ☐

Date _____ Day

→ rwnd is dynamic variable as the size changes with each req.

→ Initiall the receiver sends rwnd = RecvBuffer

→ The sender mantains,

LastByteSent - LastByteAcked ≤ rwnd

The amount of unACKed data that A has sent into the connection

Scenario: Agr receiver ne rwnd=0 send krdia ~~or~~ sender ko or ab uske baad uske paas koi data nhi sender ko bhejne k liye (jis wajase new value of rwnd is not provided to sender)

Ab sender koi data nhi bhejega q k rwnd=0 hai. He is now blocked until the receiver sends a non-zero value (but receiver has nothing to send now).

In this case TCP requires sender to send segments with one data byte when rwnd=0. These seg will be ACKed by receiver. Eventually buffer will began to empty & ACK will contain a non-zero rwnd value.

Date Day

▷ TCP Connection Management :

- Initiating a Connection

(1) Client Sends SYN

i- The client sends a TCP SYN segment to the server to initiate a connection

ii- The SYN bit = 1, and the client randomly selects an initial seq # (client_isn)

iii- The segment is encapsulated in an IP datagram & sent to server

(2) Server Responds with SYN-ACK Segment

i- The server allocates buffers & variables for the connection

ii- Sends a SYN-ACK segment back to the client

iii- The SYN bit = 1, the ACK = client isn+1 & the server sends its own server_isn

(3) Client Sends ACK

i- The client acknowledges the server's SYN-ACK by sending ACK segment

ii- The ACK = server_isn + 1, and SYN bit = 0 (as connection is established)

iii- This segment may carry application data.

Date _____ Day

→ Now all future segments carrying app data will have SYN bit = 0.

→ This is called TCP Three-Way Handshake

Q: Why is a three-way handshake needed instead of a Two-way?

① Avoids Half Open Connection:
A two way connection may lead to situations where one side thinks a connection is open while the other doesnot

② Prevents Spoofing Attacks:
The 3rd step confirms that both parties rec each other's seq#, reducing the risk of malicious connection hijacking.

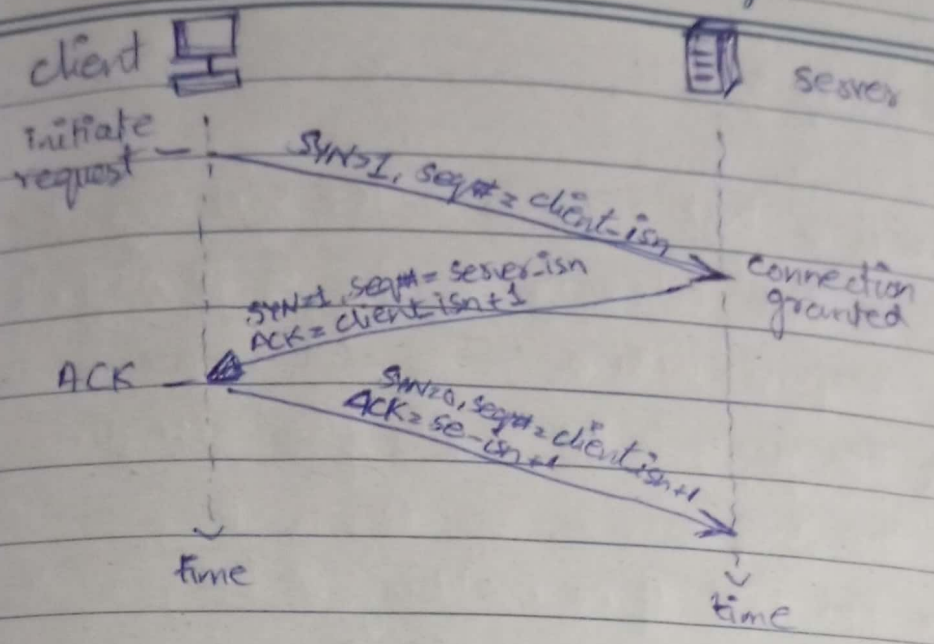
③ A 2-way han would not confirm that server's response was received, leading to potential connection issues & packet loss.

Q: Why are initial seq# needed?

① ~~Unique initial seq# prevents~~ Prevents duplicate segments from prev connections from being mistaken as valid data.

② Ensures each TCP connection starts with unique seq#

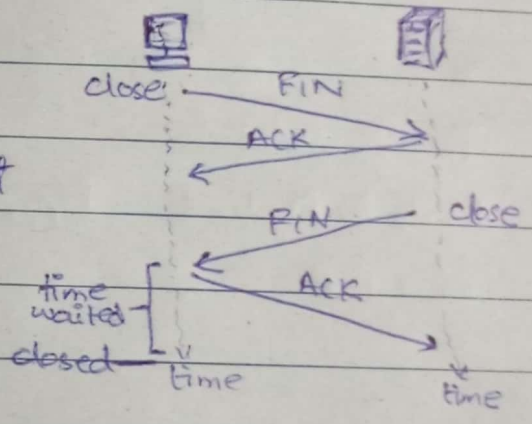
Date Day



- Closing a connection

① Client sends a FIN (Finish) segment to the server, indicating it wants to close the connection

② The server responds with an ACK segment confirming it rec the FIN request



③ Now the server sends a FIN seg to client, signaling that it is also ready to close conn.

④ Client sends a final ACK, confirming the termination

⑤ Both client & server deallocates resources, & connection terminated completely.

Date _____ Day

Scenario:

- If a host receives a TCP SYN segment for a Port where no service is running, it responds with a RST (reset) segment
- This prevents the sender from resending the req
- For UDP if no matching socket exist the host sends an ICMP & "Port unreachable" message.

Date Day

CHAPTER # 02 (Remaining Topics)

▷ PEER TO PEER FILE DISTRIBUTION

→ The distribution time is the time it takes to get a copy of the file to all N peers.

→ Peers req service from other peers, provide service in return to other peers.

Self-scalability — new peers bring new service capacity and new service demands

- File distribution Time : client-server
  - server transmission : must sequentially send (upload) N file copies:
    - time to send one copy → F/U_s (F: No. of bits of File; U_s: server upload rate)
    - time to send N copies → NF/U_s

- each client must download file copy

  d_{min} = min client download rate

  min client download time → F/d_{min}

Time to distribute F bits to N clients using client-server approach

$D_{cs} \geq \max\left\{\frac{NF}{U_s}, \frac{F}{d_{min}}\right\}$

$\frac{NF}{U_s}$ increases linearly with no. of peers/clients

Date _____ Day

- File distribution time : P2P
  - Server transmission : must upload at least one copy
    time to send one copy → $\frac{F}{U_s}$
  - client : each client must download file copy
    min client download time → F/d_{min}
  - clients : The system must deliver (upload) F bits to each of the N peers, thus delivering total of NF bits so,
    max uploadrate → $NF/(U_s + \Sigma U_i)$

ye bhi linearly grow krega

Time to distribute F bits to N clients using P2P

$$D_{P2P} \geq \max\left\{\frac{F}{U_s}, \frac{F}{d_{min}}, \frac{NF}{U_s + \sum_{i=1}^{N} U_i}\right\}$$

ye bhi grow krega to wske ~~nullify~~ reduce krdega

q/k wo uploading service bhi sath layega.

See Slide #86 For BitTorrent File Distribution

Date Day

▷ VIDEO STREAMING :

→ Video is stored on an HTTP server as a regular file with a URL
→ The client sends an HTTP GET request to fetch the video
→ The server responds by sending the video as fast as network conditions allow
→ The client buffers the received bytes and starts playback once a threshold is reached
→ The streaming application decompresses and displays video frames as they arrive

- Limitations of Basic HTTP Streaming

→ Fixed encoding : All clients rec same video quality regardless of bandwidth difference
→ Bandwidth variation issues : If a user's connection is slow, buffering & playback interruptions occur

- DASH (Dynamic Adaptive Streaming over HTTP)

→ Video is encoded into multiple versions at different bitrates & quality levels
→ Client request video in chunks of a few second instead of whole file
→ Clients dynamically chooses higher or lower bitrates based on available bandwidth.

Date _____ Day

→ Users with high speed internet get better quality

• Implementation

→ Each version of the video is stored on the HTTP server with a different URL

→ The server provides a manifest file containing URLs & bitrates of all versions

→ The client 1st downloads the manifest file, then selects chunks based on avail. bandw.

→ Client measures received bandwidth & run a rate determination algo to decide the next chunk.

Content Distribution Network (CDNs)

• Challenges in Centralized Video Streaming

→ If data center goes down, streaming stops entirely (Single point of Failure)

→ More links incr the risk of freezing delays due to bandwidth constraints

→ Popular videos are repeatedly sent over the same network, increasing cost

• Role of CDN

→ CDN stores copies of videos in multiple geographically

Date _____ Day _____

distributed servers.

→ The reduce latency by serving users from the closest available server

→ CDNs can be private

- CDN Server Placement Strategies
  - Enter Deep
    - ↳ Deploys small server clusters inside ISPs world wide to minimize distance to users
    - ↳ Improves performance but is complex to maintain due to thousands of locations
  - Bring Home
    - ↳ Deploys fewer, larger clusters at Internet Exchange Points (IXPs) instead of inside ISPs
    - ↳ Easier to manage but may increase delay & lower throughput
- Content Replication & Caching in DNS

→ Not all videos are stored in every cluster to optimize storage.

→ Uses a **pull strategy**: A requested video not in a cluster is fetched from another cluster or a central repository

→ Least frequent used videos are removed when storage is full.

Date _____ Day

- Cluster Selection Strategies

1. Geographical Proximity based Selection
   - ↳ Uses geo-location databases to find the close server by distance
   - ↳ Drawbacks:
     - ↳ The nearest server geographically may not be the best in terms of network latency or hops
     - ↳ Some users use remote DNS server, leading to incorrect mapping

2. Real Time Network Performance-Based Selection
   - ↳ CDNs periodically measure delay & packet loss b/w clusters & Local DNS server
   - ↳ Uses ping, DNS queries to dynamically choose best cluster
   - ↳ Drawbacks:
     - ↳ Many LDNS servers block probes, making it hard to collect accurate data.

a small test message sent by CDN to measure network performance

Page No.

Date Day

Case Study : Netflix

→ Uses Amazon Cloud for web services
→ Uses it's own private CDN for video distribution

- Amazon Cloud Functions
  - Content Ingestion : Uploads studio master versions of movies
  - Content Processing : Converts movies into multiple formats for different devices & bitrates
  - Uploading to CDN : Processed versions are sent to Netflix's CDN

- Netflix's Private CDN
  → Servers are installed in IXPs & ISPs
  → Over 200 IXP locations & hundreds of ISPs host netflix server
  → Server contains entire or most popular videos
  → Push caching : content is proactively pushed to servers during off peak hrs.

- Client - Server Interaction
  → Web browsing is handled by Amazon Cloud Servers
  → When a user selects a movie :
    - The best CDN server is choosen based on loc & availbly.

Date _____ Day

- If a user's ISP has a Netflix CDN rack, it is preferred, otherwise a nearby IXP CDN is used
- The client sec the server's IP addrs. and a manifest file
- The client interacts directly with the CDN server using a proprietary version of DASH
- Uses HTTP GET byte-range request to fetch 4-second video chunk
- Client measures throughput and adjust video quality dynamically

- Netflix CDN design Advantage
  - The client is directly assigned a CDN server instead of resolving via DNS
  - Unlike general purpose CDN, it is dedicated to video delivery, simplifying it's design.

CASE STUDY : YOUTUBE

→ Uses it's own private CDN
→ Pull caching is used, meaning videos are fetched on demand & stored temporarily.
→ DNS redirection helps balance traffic by assigning users to optimal server based on RTT
→ Uses HTTP streaming but doesnot support DASH