HLD

Client Server Architecture - Request Response model

→ every computer/process in a network functions as a server or client.

→ Client: any comp. that requests something from server

→ Server: any comp designed to serve the requests to client

eg → Client, webbrowser access website & sends rq to server, webserver → retrieves necessary webpages & resources & sends back to browser for display

→ online gaming → game client running on player's device interacts with game server

→ Can app server act as client?

→ Yes!

eg - DB connectivity → app server need to retrieve/update data from DB server

app server - client sending DB queries/requests to DB server

→ app servers might communicate with external services/APIs to retrieve/send data.

Communicat^n / Networking in HLD

→ Divide complex networking tasks into smaller, modular components, each responsible for specific set of functions.

→ Network protocol stack - stack of layers
- each layer provides services to the layer above it & uses services of layer below it → creating a hierarchical structure for n/w communication

→ Higher layers → app specific tasks

→ lower layers → lower level funcs closer to physical transmission of data

→ OSI → open systems interconnect^n - 7 layers } conceptual frameworks

→ TCP/IP → 4 layers - simplified, condensed model

Transport layer - TCP, UDP

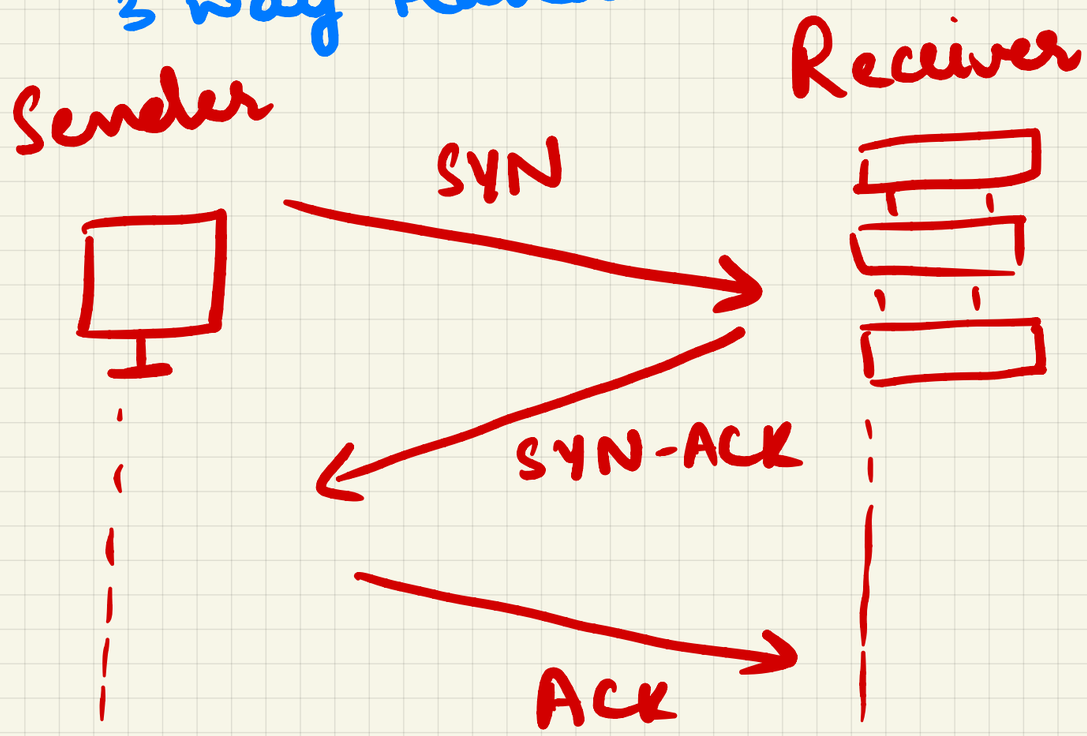
→ provides reliable, end to end data delivery
→ breaks down data into smaller segments, which can be transmitted over nlw.

Application layer

→ topmost layer in networking stack
→ provides services, protocols & interfaces for end-users & apps to interact over the network.
→ HTTP, HTTPS, FTP, SFTP, RTMP, SMTP, POP3, IMAP, DNS, DHCP, SNMP, SSH

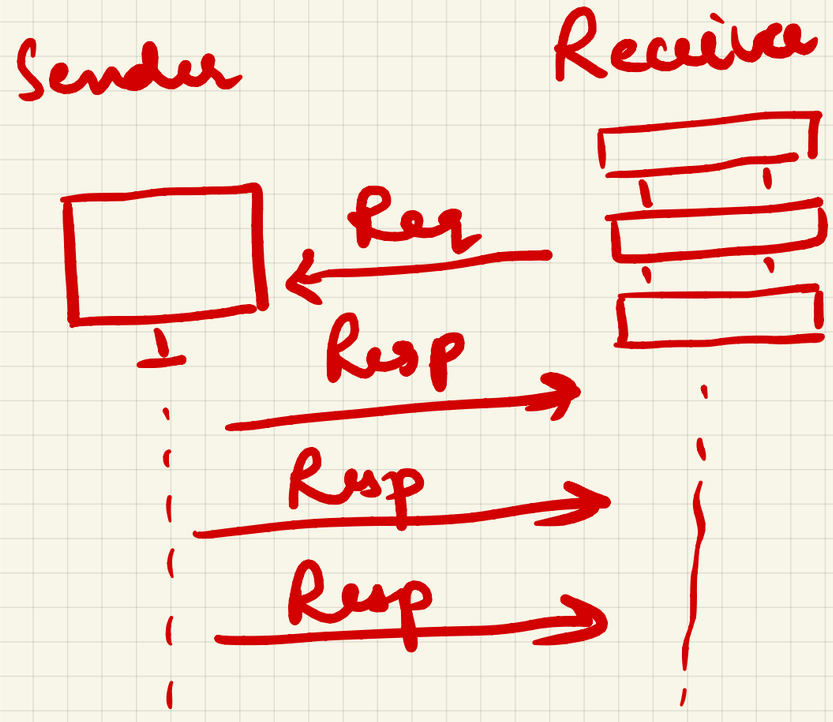
TCP

3 way handshake



→ connect° oriented protocol connect° should establish connect° before transmitting data & close after that

UDP



→ connectionless protocol

→ no overhead for opening, maintaining / terminating connect°

→ reliable, guarantees delivery of data

→ extensive error-checking mechanisms

→ sequencing of data ie. packets arrive in order at the receiver

→ Slower

→ Broadcasting not supported

→ retransmission of lost packets ✓

→ safe & trustworthy communicat° procedure

→ email, web surfing, military services

→ delivery not guaranteed, unreliable

→ basic error-checking using checksums

→ no sequencing of data, if required - manage at app layer

→ faster, simpler

→ Broadcasting supported (multicasting)

→ no retransmission

→ quick communicat° is necessary, not dependability

→ game streaming, video & music streaming

3-Way TCP Handshake - Connection Establishment

① SYN (Synchronize) → client sends SYN packet to server
   → indicating intent to establish connection
   → packet contains seq. no. to identify the data

② SYN-ACK (Synchronize - Acknowledge)
   → Ack. client's req & indicating willingness to establish connect.

③ ACK (Acknowledgement)
   → confirm receipt of SYN-ACK
   → connect established

Reliable Data Transfer

① Segmentation → divide data to TCP segments
→ each segment has seq. no, control flags, acknowledgements etc.

② Transmission → send segments

③ Acknowledgement → Receiver sends Ack packets to sender
→ if sender does not receive ACK within a certain time, it retransmits the segments

④ Flow control → "sliding window" mechanism to manage flow control

→ sender & receiver negotiate a window size which determines no. of segments that can be sent before sending ACK, prevents sender from overwhelming the receiver with data.

Connect Terminate - Graceful Connection Release

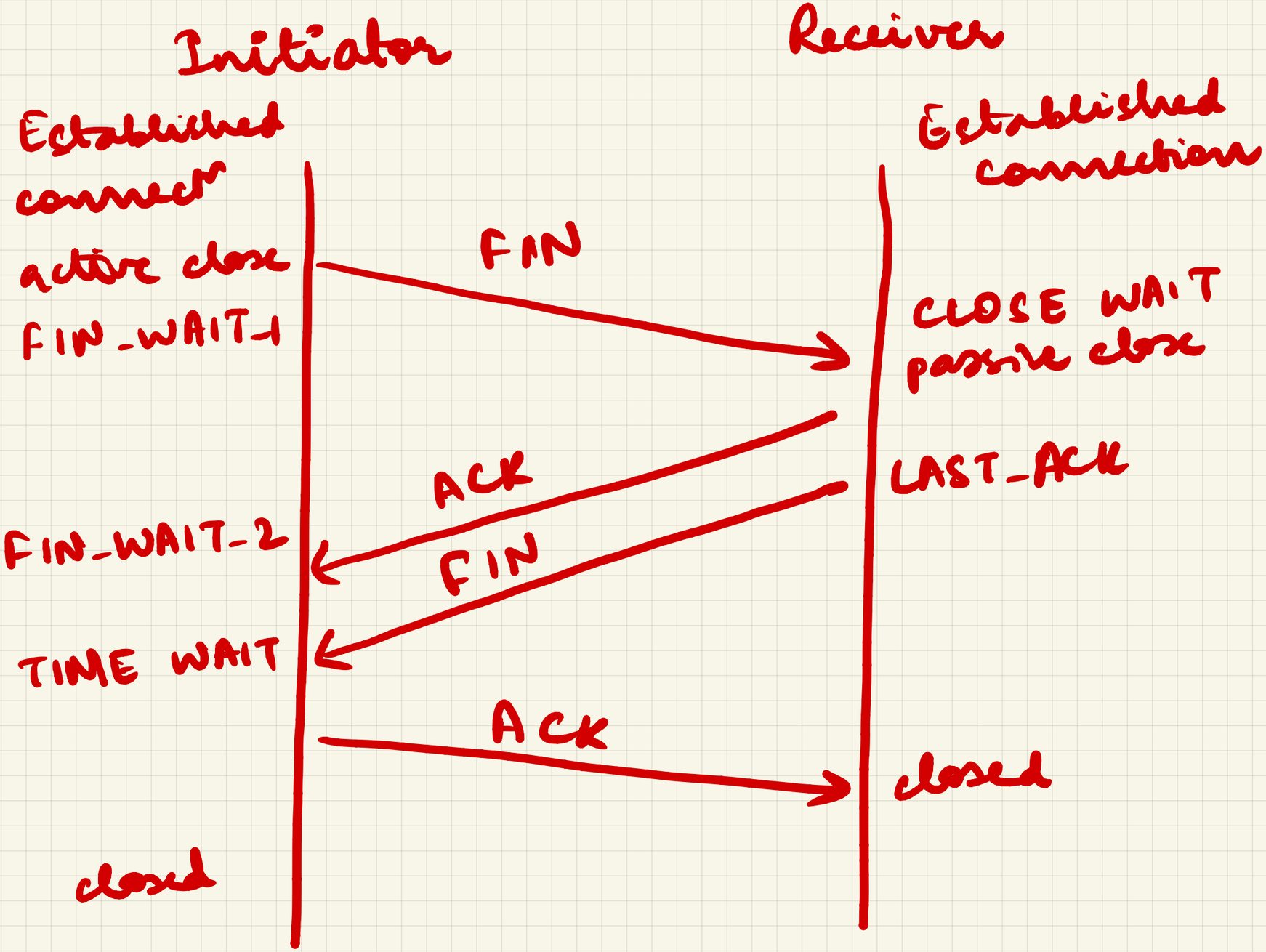
① FIN (Finish) - either client or server can initiate the connection termination process by sending a FIN packet to other party, indicating intention to close the connection

② ACK → Receiving party sends ACK packet to ack

③ FIN-ACK → Receiving party sends its own FIN packet indicating agreement to terminate the connect

④ FINAL ACK → Initiating party responds with ACK packet, acknowledging the termination

⑤ TIME-WAIT → Both sides enter a time-wait state to ensure that any delayed packets related to closed connect are handled.


Initiator
Receiver
Established connection
active close
FIN
FIN_WAIT_1
CLOSE WAIT
passive close
ACK
LAST_ACK
FIN_WAIT_2
FIN
TIME WAIT
ACK
closed
closed

→ Why does DNS use UDP and not TCP? DNS is supposed to be reliable

→ DNS reqs are v. small & fit within UDP segment - It is widely used protocol, there is significant traffic that needs to be handled by DNS servers

→ In large DNS responses i.e. when response exceeds max size of UDP packet, server may switch to TCP to transmit complete response