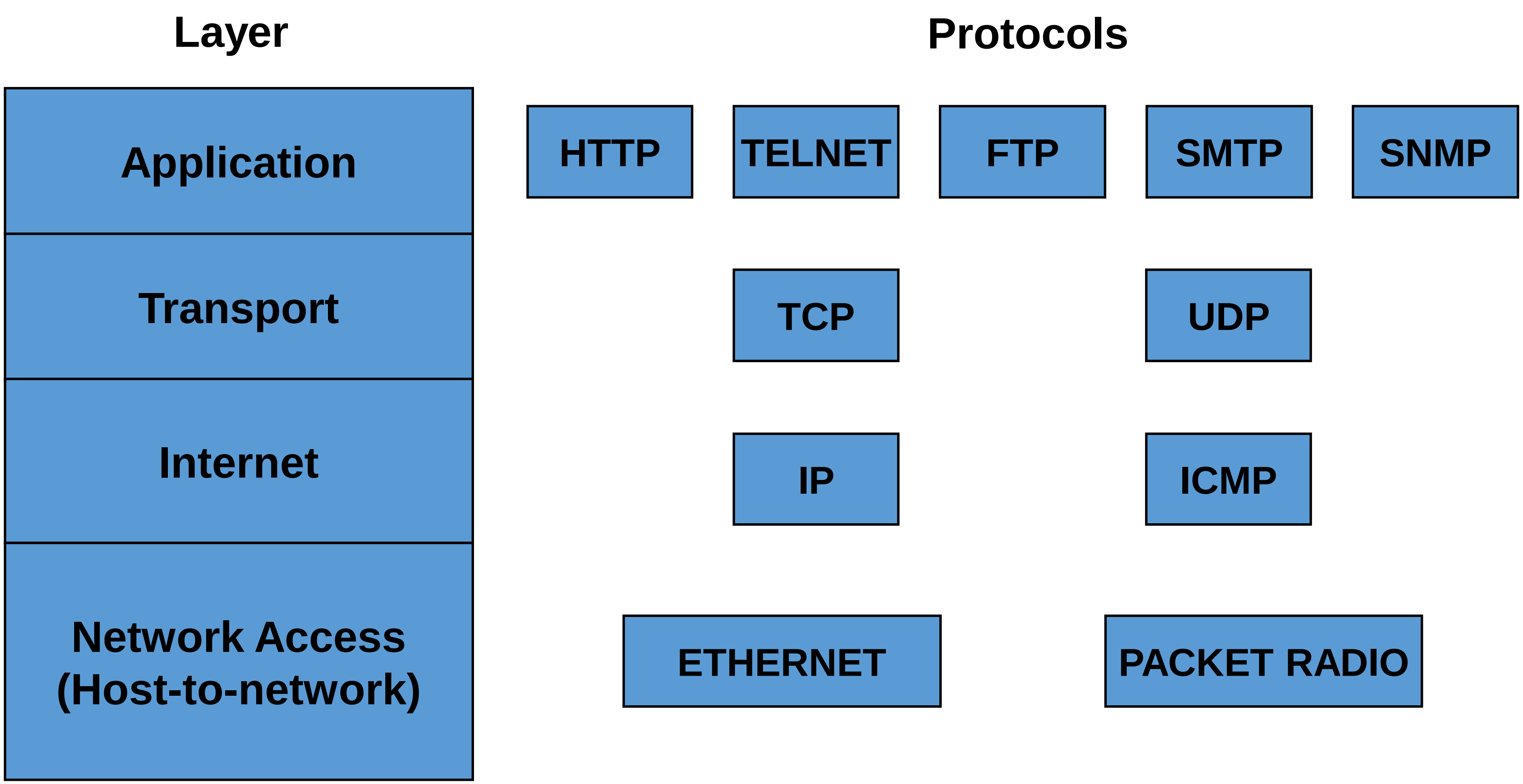
TCP/IP Internal

Learning outcome

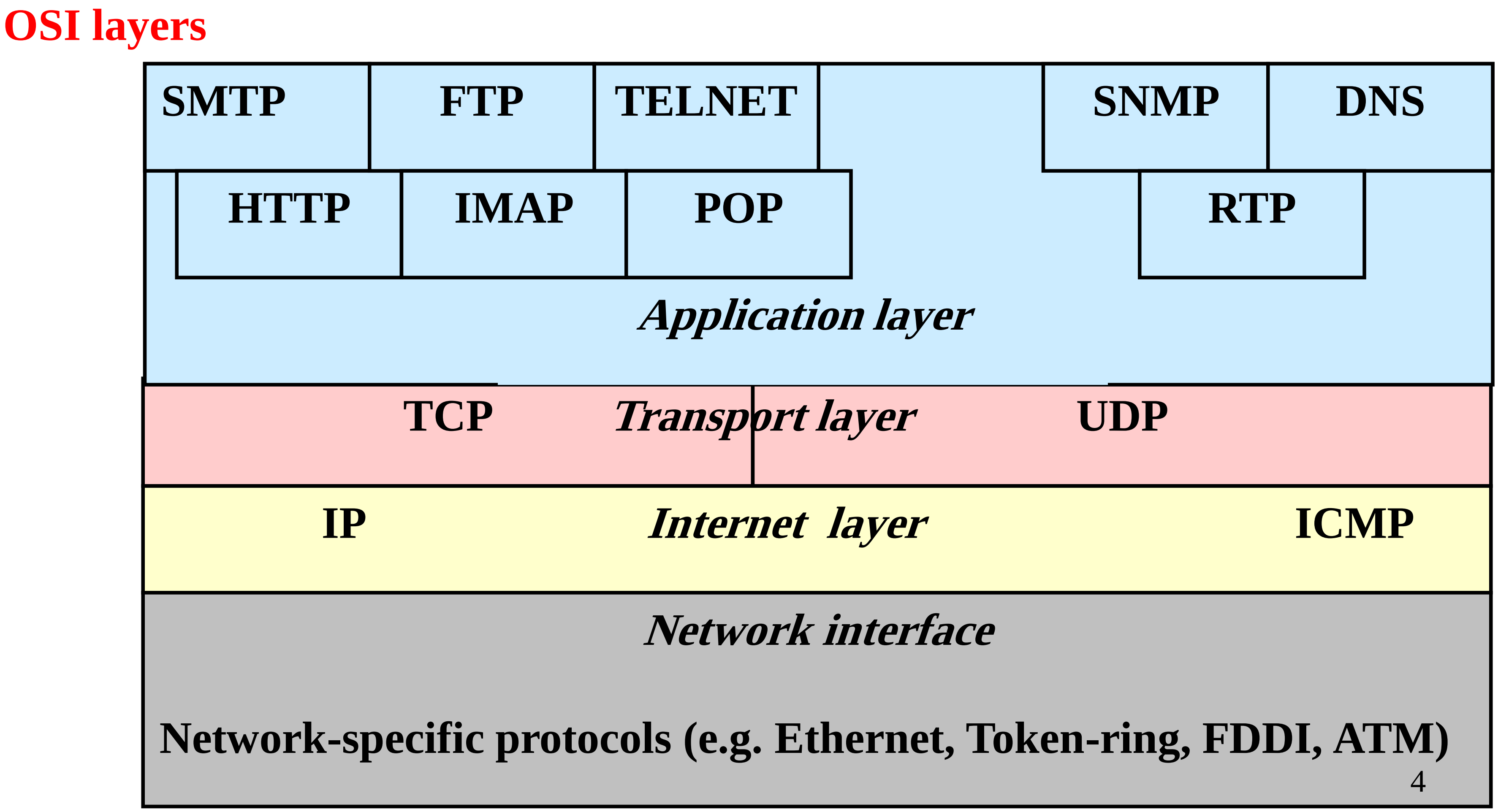
- Application layer
  - HTTP, FTP, TELNET, POP3, SMTP, IMAP, DNS protocols
- Transport layer
  - TCP and UDP
  - TCP and UDP segment
  - Opening and closing connections
  - Flow control
  - Reliable data transmission
- Internet layer
  - IP , ICMP, ARP and RARP
  - IP datagram
  - Routing

TCP/IP

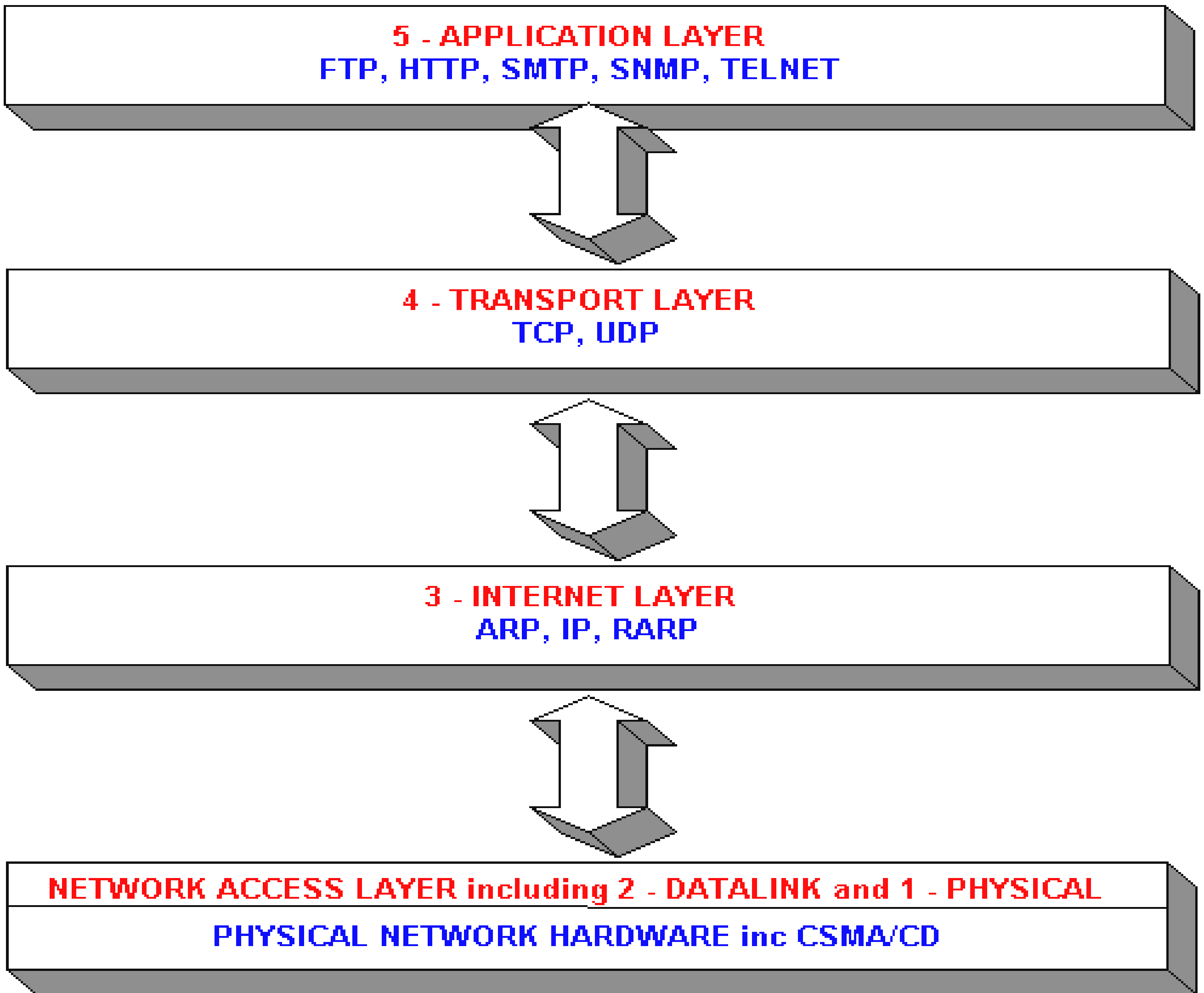
TCP/IP Reference Model



TCP/IP Protocol Suite is a four-layered protocol suite. The location of the important protocols within the TCP/IP layers is showed below



The suite of Protocols for TCP/IP



TCP/IP suite

Application Protocols

<table>
<tr><th>Protocols</th><th>Role</th><th>Ports</th></tr>
<tr><td>HTTP</td><td>Hyper Text Transfer Protocol
• browser and web server communication
1. client browser connects to HTTP server
2. client browser send a request to the HTTP server
3. HTTP server reacts by sending a response
4. HTTP server disconnects</td><td>80</td></tr>
<tr><td>FTP</td><td>File transfer protocol
1. allow people anywhere on the Internet to log in and download whatever files they have placed on the FTP server, or upload other files.
2. Port 20 for data channel and 21 for control channel</td><td>20, 21</td></tr>
</table>

TCP/IP suite

Application Protocols

Protocols	Role	Ports
DNS	Domain Name System 1. provides translation between host name and IP address 2. DNS messages are carried using UDP on port 53	53
TELNET	Remote login	23

Application Protocols (cont'd)

TCP/IP suite

Protocols	Role	Ports
POP3	Post Office Protocol 3 1. The point of POP3 is to fetch email from the remote mailbox and store it on the user's local machine to read later. 2. Downloaded emails are then deleted from the server.	110
IMAP	Internet Message Access Control 1. Retrieve emails 2. retaining e-mail on the server and for organizing it in folders on the serve	143
SMTP	Sending email 1. Sending emails 2. Establish TCP connection to port 25 of the destination machine / server 3. Start sending email message	25

HTTP (Hypertext Transfer Protocol)

Purpose:
Used for transferring web pages and other resources on the World Wide Web.

Key Features:

- **Client → Server** (usually web browser → web server).
- The client (browser) sends a request (e.g., GET, POST) to the server, and the server responds with data such as an HTML page, image, or video.
- **Stateless** – each request is independent.

Example:
You type https://www.example.com → browser sends HTTP request → server responds with the website content.

Port Numbers:

- HTTP: **80** (default)
- HTTPS (secure): **443**

FTP (File Transfer Protocol)

Purpose:
Used to transfer files between computers over a network.

Key Features:

- Client-server architecture, two-part connection (one for commands and one for data).
- Authentication for access.
- Support for various file operations like uploading and downloading.

Example:
Uploading a file from your computer to a website's hosting server using FileZilla or command-line FTP.

Port Numbers:

- FTP (control connection): **21**
- FTP (data connection): **20**

Telnet (TELecommunication NETwork)

Purpose:
Provides remote login to another computer or network device (like a server or router).

Key Features:

- Allowing for remote command-line access and control of devices.
- It is a lightweight and platform-independent protocol.
- It is insecure as it transmits all data, including passwords, in unencrypted plaintext.
- It is now mainly used for local network tasks.
- **SSH (Secure Shell)** — same purpose but encrypted and secure.

Example:
A network administrator connects to a remote router to configure it using command-line interface commands.

Port Numbers:

- Telnet: **23**

IMAP (Internet Message Access Protocol)

Purpose:
Used by an email client (like Outlook, Gmail app, Thunderbird) to **access emails stored on the mail server**.

Key Features:

- Emails **stay on the server**.
- You can **view and manage** your mailbox from **multiple devices** (phone, laptop, etc.).
- Folder structure (Inbox, Sent, etc.) is **synchronized** across all devices.
- Requires an internet connection to read new emails (unless cached).

Example:
You read an email on your phone → it's marked "read" on your laptop too.

Port Numbers:

- IMAP: **143** (default)
- IMAPS (secure): **993**

Pop3 (Post Office Protocol v3)

Purpose:
Used by an email client to **download emails** from the mail server to the device.

Key Features:

- Emails are **downloaded and usually deleted from the server** (unless you change settings).
- Once downloaded, they exist **only on that device**.
- NOT good for multiple-device access.

Example:
You check mail using POP3 on your laptop → the emails disappear from the server → you can't see them on your phone.

Port Numbers:

- POP3: **110**
- POP3S (secure): **995**

SMTP (Simple Mail Transfer Protocol)

Purpose:
Used to **send emails** from a client to the mail server or between servers.

Key Features:

- Handles **outgoing mail** only.
- Works with IMAP or POP3 (which handle incoming mail).
- Can send messages to another email server for delivery.

Example:
When you hit "Send," your email client uses SMTP to push the message to the mail server.

Port Numbers:

- SMTP: **25** (default, often blocked)
- SMTPS (secure): **465**

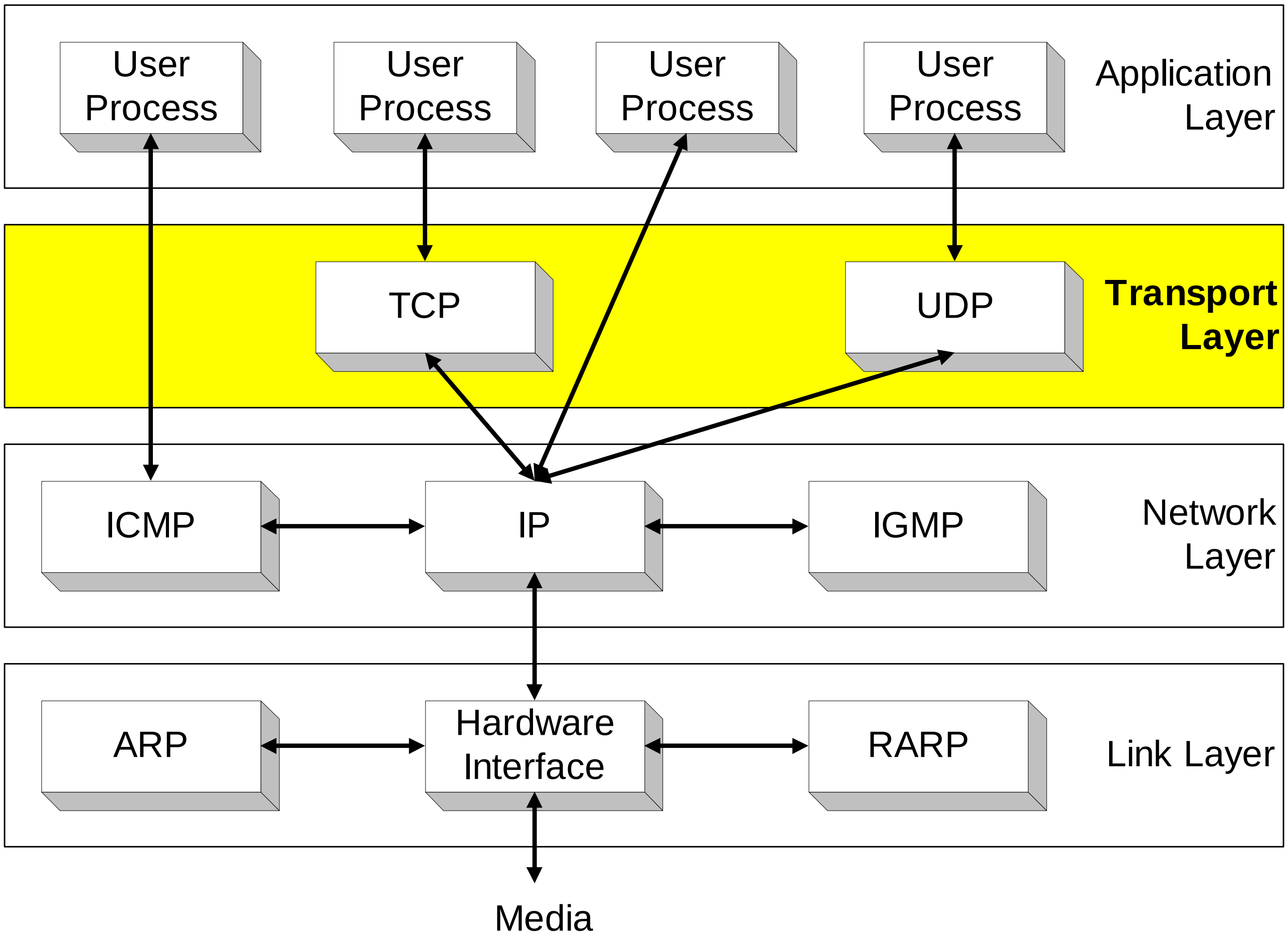
The transport layer

Transport layer

- Transport protocols
  - UDP
  - TCP
- TCP AND UDP segments

Transport Protocols

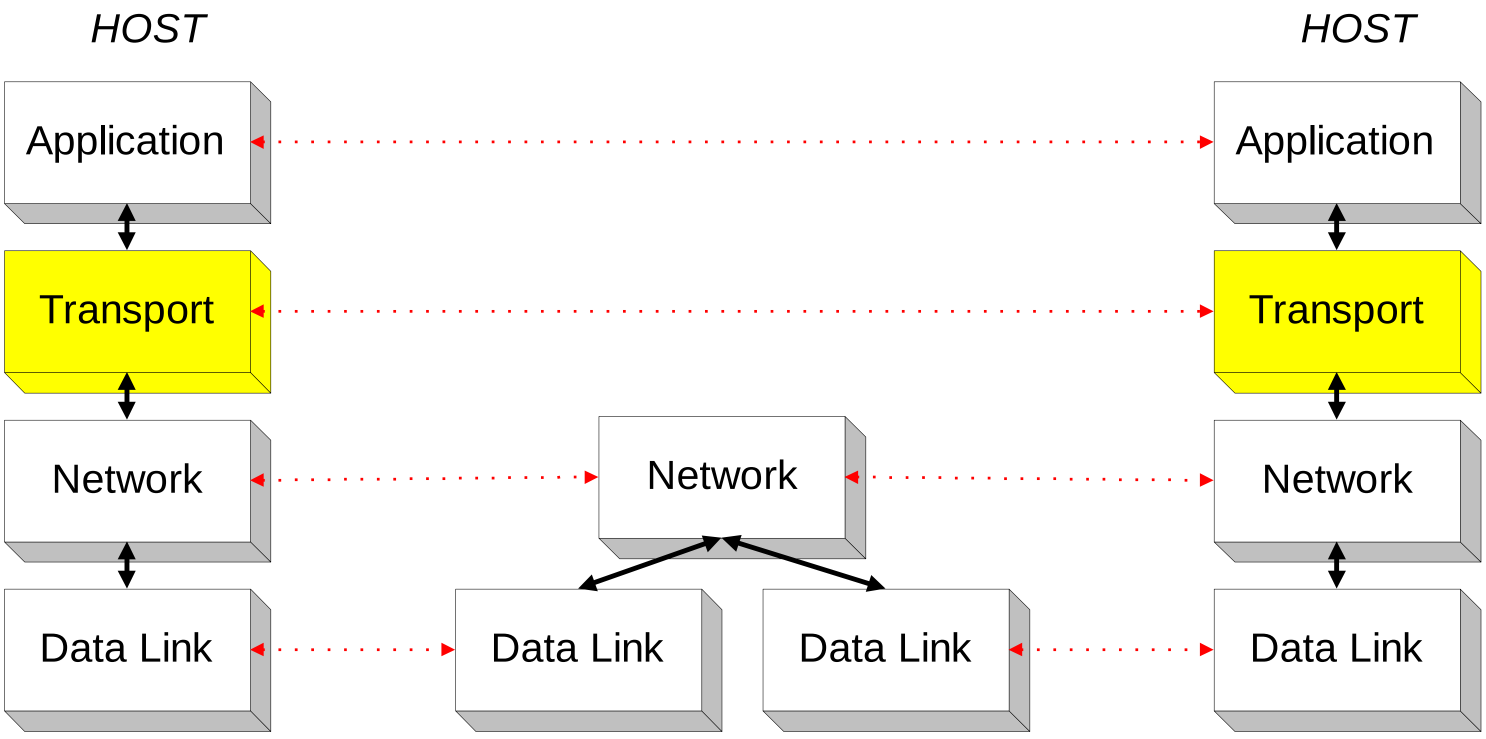
TCP/IP suite



Orientation

TCP/IP suite

- Transport layer protocols are end-to-end protocols
- They are only implemented at the hosts



Transport Protocols in the Internet

TCP/IP suite

- The Internet supports 2 transport protocols

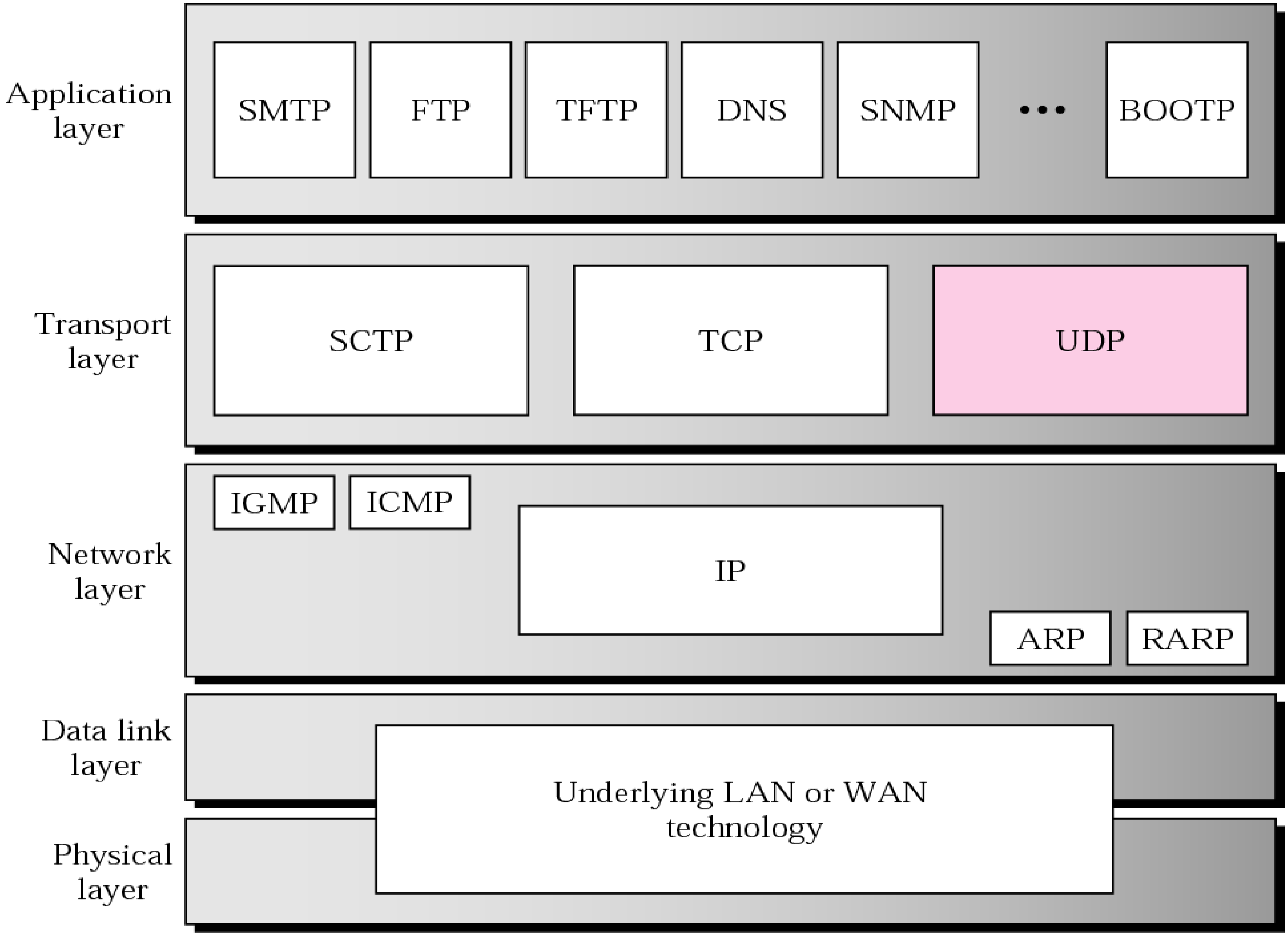
UDP - User Datagram Protocol

- datagram oriented
- unreliable, connectionless
- No acknowledgment
- simple
- unicast and multicast
- useful only for few applications, e.g., multimedia applications
- used a lot for services
  - network management (SNMP), routing (RIP), naming (DNS), etc.

TCP - Transmission Control Protocol

- stream oriented
- reliable, connection-oriented
- complex
- only unicast
- used for most Internet applications:
  - web (HTTP), email (SMTP), file transfer (FTP), terminal (TELNET), etc.

Position of UDP in the TCP/IP protocol suite

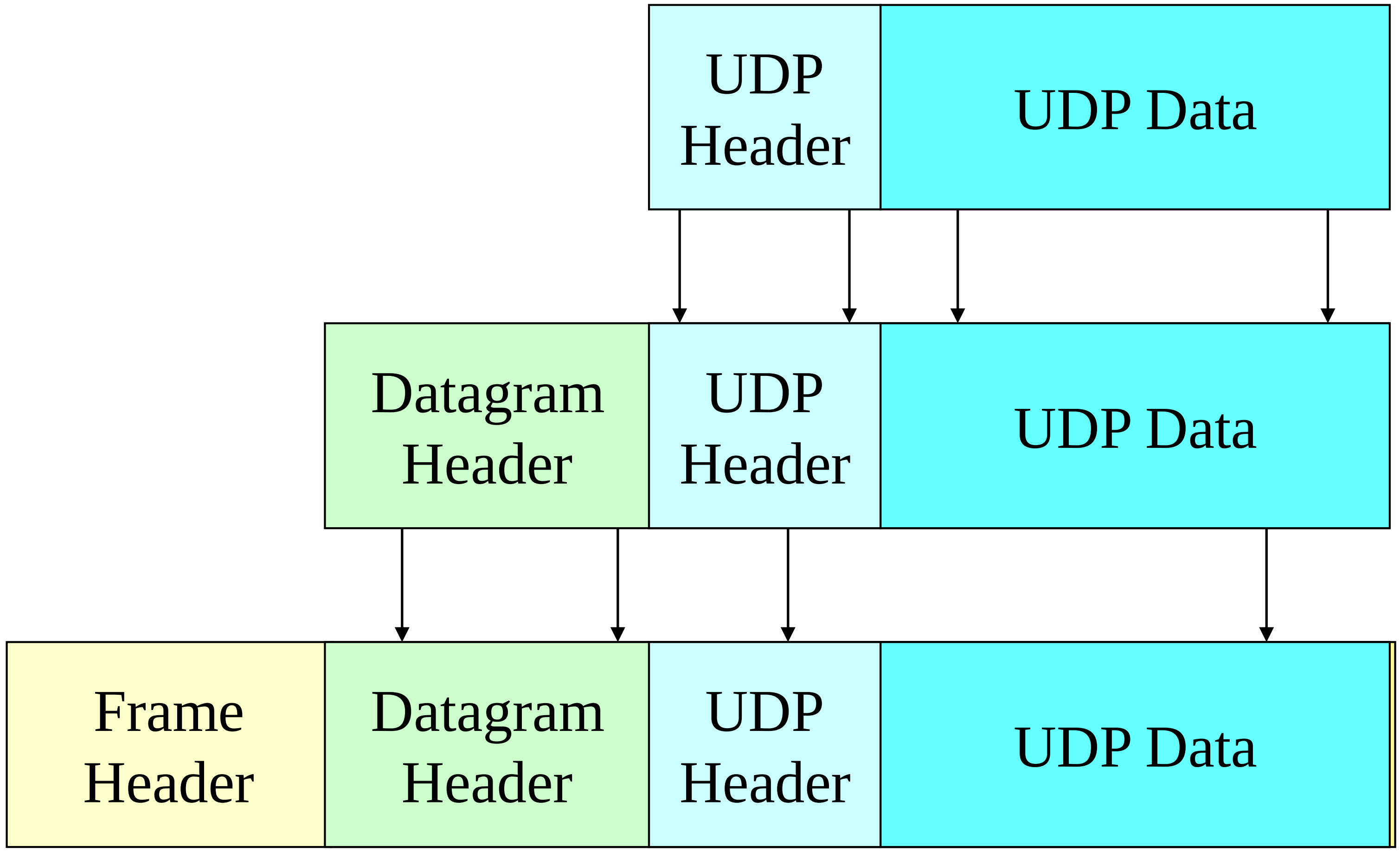


User Datagram Protocol

- Uses IP to transport message from source to destination
- Unreliable, connectionless datagram delivery
- No acknowledgements
- Messages can be lost, duplicated, or arrive out of order.

TCP/IP

User Datagram Protocol



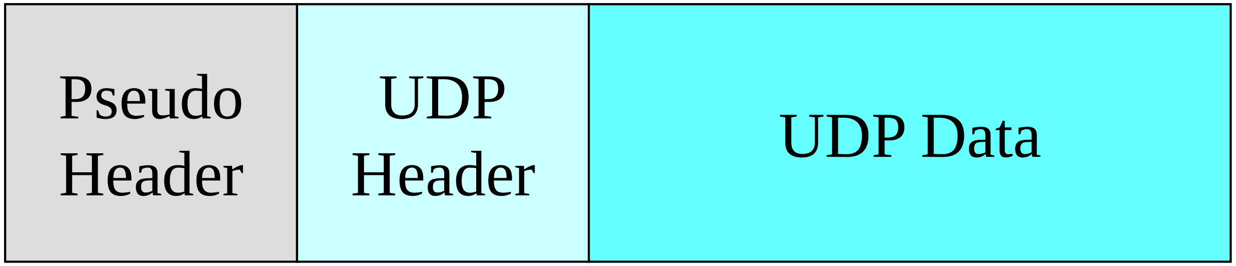
User Datagram Protocol

- Source port (optional - zero if not used)
- Length - Count of octets including header and data (minimum is 8)
- Checksum (optional - zero if not used)

UDP Source Port	UDP Destination Port
UDP Message Length	UDP Checksum
Data . . .	

User Datagram Protocol

- IP checksum does not include data
- UDP checksum is only way to guarantee that data is correct
- UDP checksum includes pseudo-header



UDP Pseudo-Header

Source IP Address			
Destination Address			
Zero	Protocol	UDP Length	
UDP Source Port		UDP Destination Port	
UDP Message Length		UDP Checksum	
Data . . .			

UDP Pseudo-Header

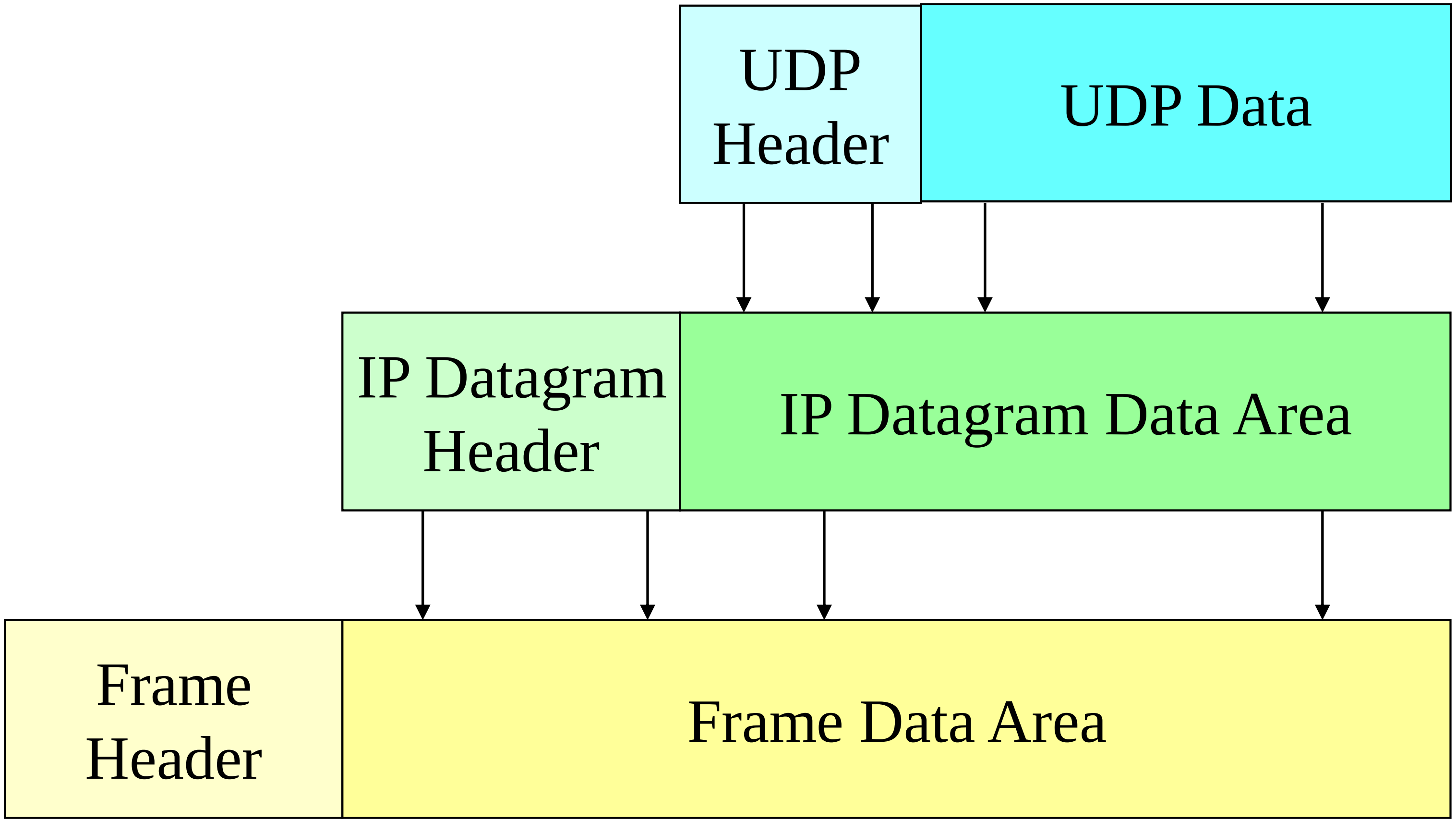
- Prefixed to the front of datagram
- Verifies that datagram reached correct destination
- UDP header only includes port numbers
- Pseudo-header includes IP addresses

TCP/IP Layers

Conceptual Layers are independent

Application
User Datagram (UDP)
Internet (IP)
Network Interface

TCP/IP Layers



TCP/IP Layers

- UDP checksum includes pseudo-header which includes source and destination IP address
- Source IP address depends on route chosen (multiple interfaces)
- UDP layer builds IP datagram

User Datagram Protocol

Summary

- Uses ports on source and target
- Does not add significantly to IP
- Unreliable connectionless packet delivery
- Interacts strongly with IP layer
- Low overhead

TCP Lingo

- When a client requests a connection, it sends a "**SYN**" segment (a special TCP segment) to the server port.
- **SYN** stands for *synchronize*. The **SYN** message includes the client's **ISN**.
- **ISN** is Initial Sequence Number.

More…

TCP/IP

- Every TCP segment includes a ***Sequence Number*** that refers to the first byte of *data* included in the segment.
- Every TCP segment includes a *Request Number* (***Acknowledgement Number***) that indicates the byte number of the next data that is expected to be received.
- All bytes up through this number have already been received.

And more...

There are a bunch of control flags:

- URG: urgent data included.
- ACK: this segment is (among other things) an acknowledgement.
- RST: error - abort the session.
- SYN: synchronize Sequence Numbers (setup)
- FIN: polite connection termination.

And more…

- **MSS**: Maximum segment size (A TCP option)
- **Window**: Every ACK includes a Window field that tells the sender how many bytes it can send before the receiver will have to throw it away (due to fixed buffer size).

TCP Connection Creation

- Programming details later - for now we are concerned with the actual communication.
- A *server* accepts a connection.
  - Must be looking for new connections!
- A *client* requests a connection.
  - Must *know* where the server is!

Client Starts

A client starts by sending a SYN segment with the following information:

- Client's ISN (generated pseudo-randomly)
- Maximum Receive Window for client.
- Optionally (but usually) MSS (largest datagram accepted).

Server’s Response

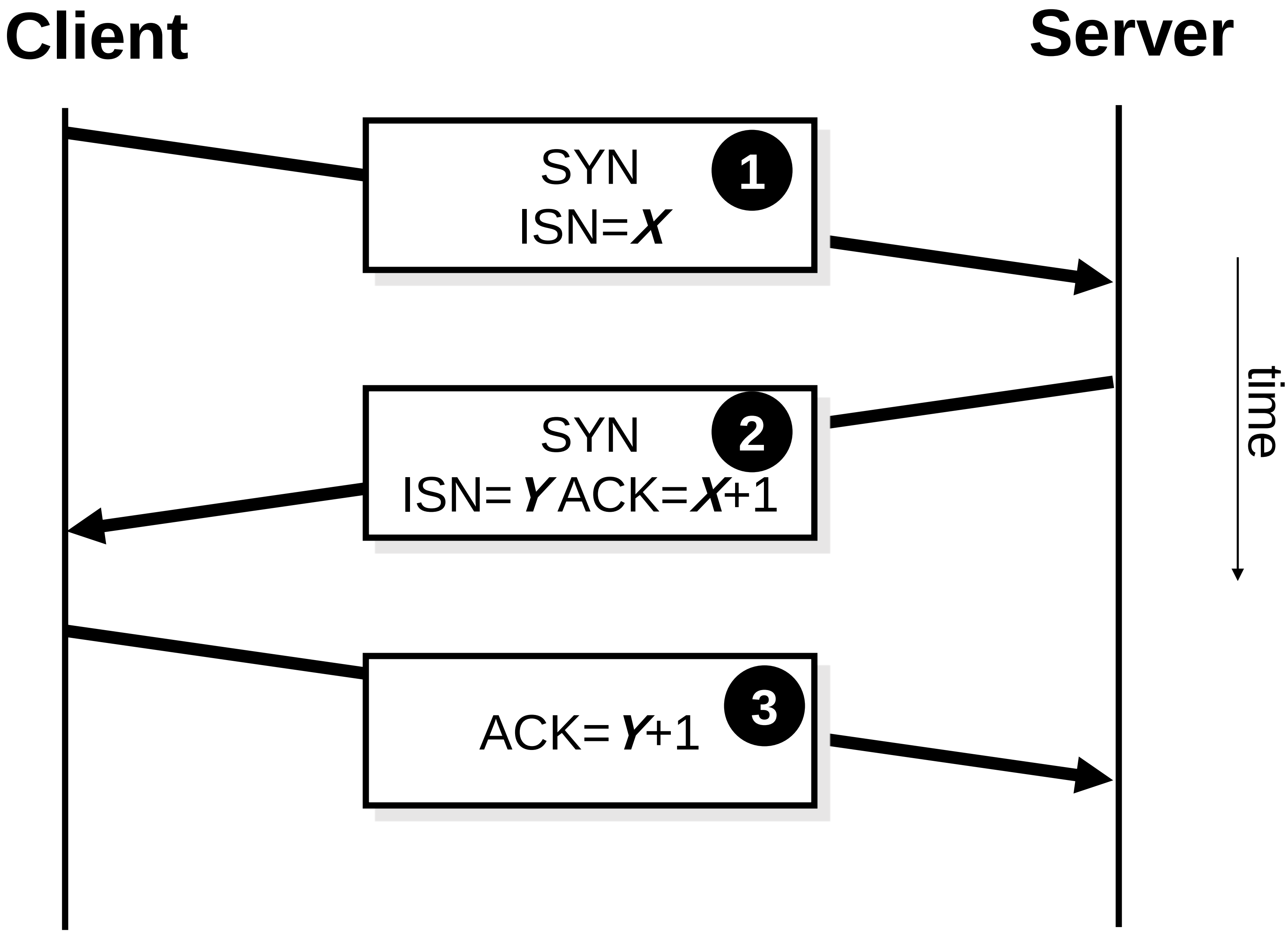
When a waiting server sees a new connection request, the server sends back a SYN segment with:

- Server’s ISN (generated pseudo-randomly)
- Request Number is Client ISN+1
- Maximum Receive Window for server.
- Optionally (but usually) MSS

Finally

When the Server's SYN is received, the client sends back an ACK with:

- Request Number is Server's ISN+1



TCP 3-way handshake

TCP 3-way handshake

1. Client: “I want to talk, and I’m starting with byte number ***X***”.

2. Server: “OK, I’m here and I’ll talk. My first byte will be called number ***Y***, and I know your first byte will be number ***X+1***”.

3. Client: “Got it - you start at byte number ***Y+1***”.

Why 3-Way?

- Why is the third message necessary?

- HINTS:

  - TCP is a reliable service.

  - IP delivers each TCP segment.

  - IP is not reliable.

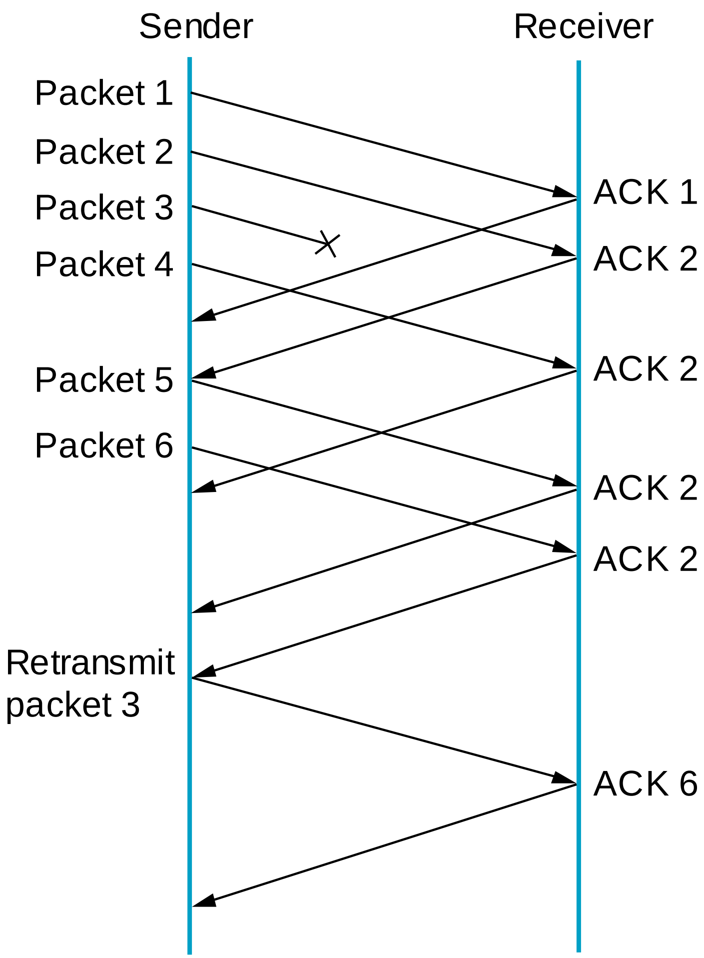
TCP Data and ACK

- Once the connection is established, data can be sent.
- Each data segment includes a sequence number identifying the first byte in the segment.
- Each segment (data or empty) includes a request number indicating what data has been received.

TCP Fast Retransmit

- Another enhancement to TCP congestion control
- Idea: When sender sees 3 duplicate ACKs, it assumes something went wrong
- The packet is immediately retransmitted instead of waiting for it to timeout

TCP Fast Retransmit

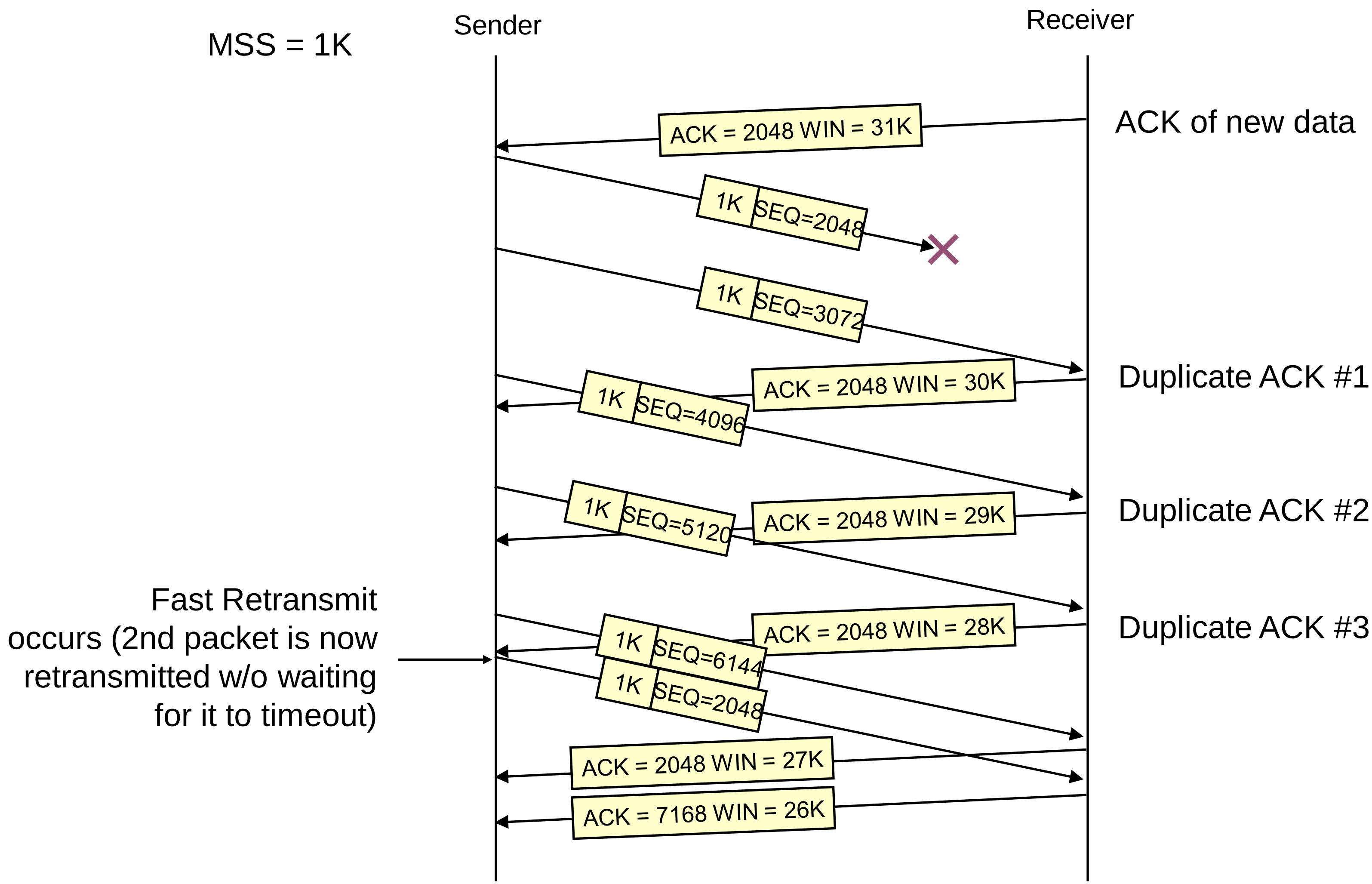


Fast Retransmit

Based on <u>three</u> duplicate ACKs

TCP Fast Retransmit

Example



Buffering

- Keep in mind that TCP is (usually) part of the Operating System. It takes care of all these details *asynchronously*.
- The TCP layer doesn’t know when the application will ask for any received data.
- TCP buffers incoming data so it’s ready when we ask for it.

TCP Buffers

- Both the client and server allocate buffers to hold incoming and outgoing data
  - The TCP layer takes care of this.
- Both the client and server announce with every ACK how much buffer space remains (the Window field in a TCP segment).

Send Buffers

- The application gives the TCP layer some data to send.
- The data is put in a send buffer, where it stays until the data is ACK’d.
  - it has to stay, as it might need to be sent again!
- The TCP layer won’t accept data from the application unless (or until) there is buffer space.

ACKs

- A receiver doesn’t have to ACK every segment (it can ACK many segments with a single ACK segment).
- Each ACK can also contain outgoing data (piggybacking).
- If a sender doesn’t get an ACK after some time limit it resends the data.

TCP Segment Order

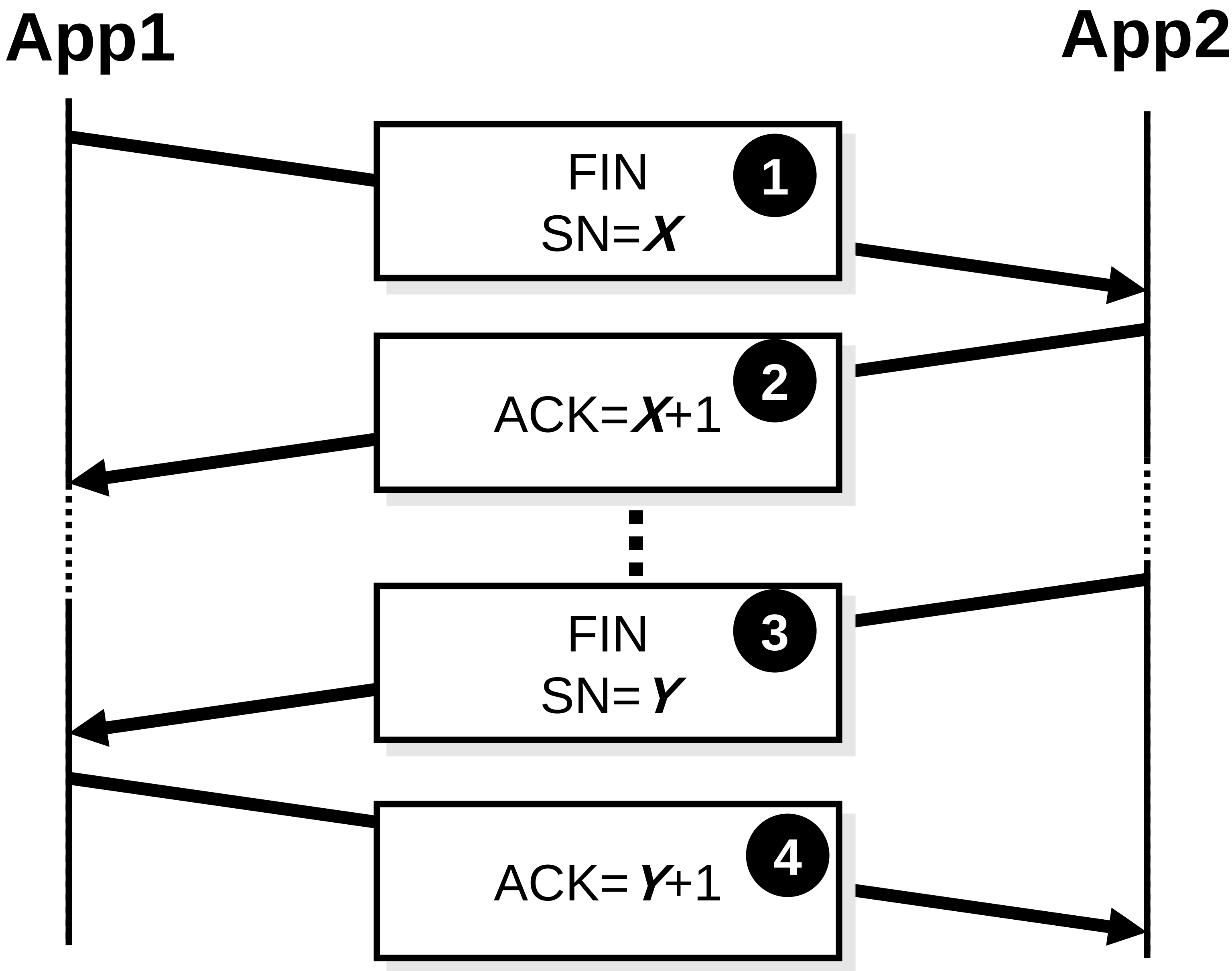
- Most TCP implementations will accept out-of-order segments (if there is room in the buffer).
- Once the missing segments arrive, a single ACK can be sent for the whole thing.
- Remember: IP delivers TCP segments, and IP is not reliable - IP datagrams can be lost or arrive out of order.

Termination

- The TCP layer can send a RST segment that terminates a connection if something is wrong.
- Usually the application tells TCP to terminate the connection politely with a FIN segment.

FIN

- Either end of the connection can initiate termination.
- A FIN is sent, which means the application is done sending data.
- The FIN is ACK’d.
- The other end must now send a FIN.
- That FIN must be ACK’d.


App1
App2
FIN
SN=X
1
2
ACK=X+1
FIN
SN=Y
3
ACK=Y+1
4

TCP Termination

1. App1: “I have no more data for you”.

2. App2: “OK, I understand you are done sending.”

   dramatic pause…

3. App2: “OK - Now I’m also done sending data”.

4. App1: “Goodbye, It’s been real pleasure talking to you ”

TCP TIME_WAIT

- Once a TCP connection has been terminated (the last ACK sent) there is some unfinished business:
  - What if the ACK is lost? The last FIN will be resent and it must be ACK'd.
  - What if there are lost or duplicated segments that finally reach the destination after a long delay?
- TCP hangs out for a while to handle these situations.

3-Way Handshake Issue (SYN Flooding Attack)

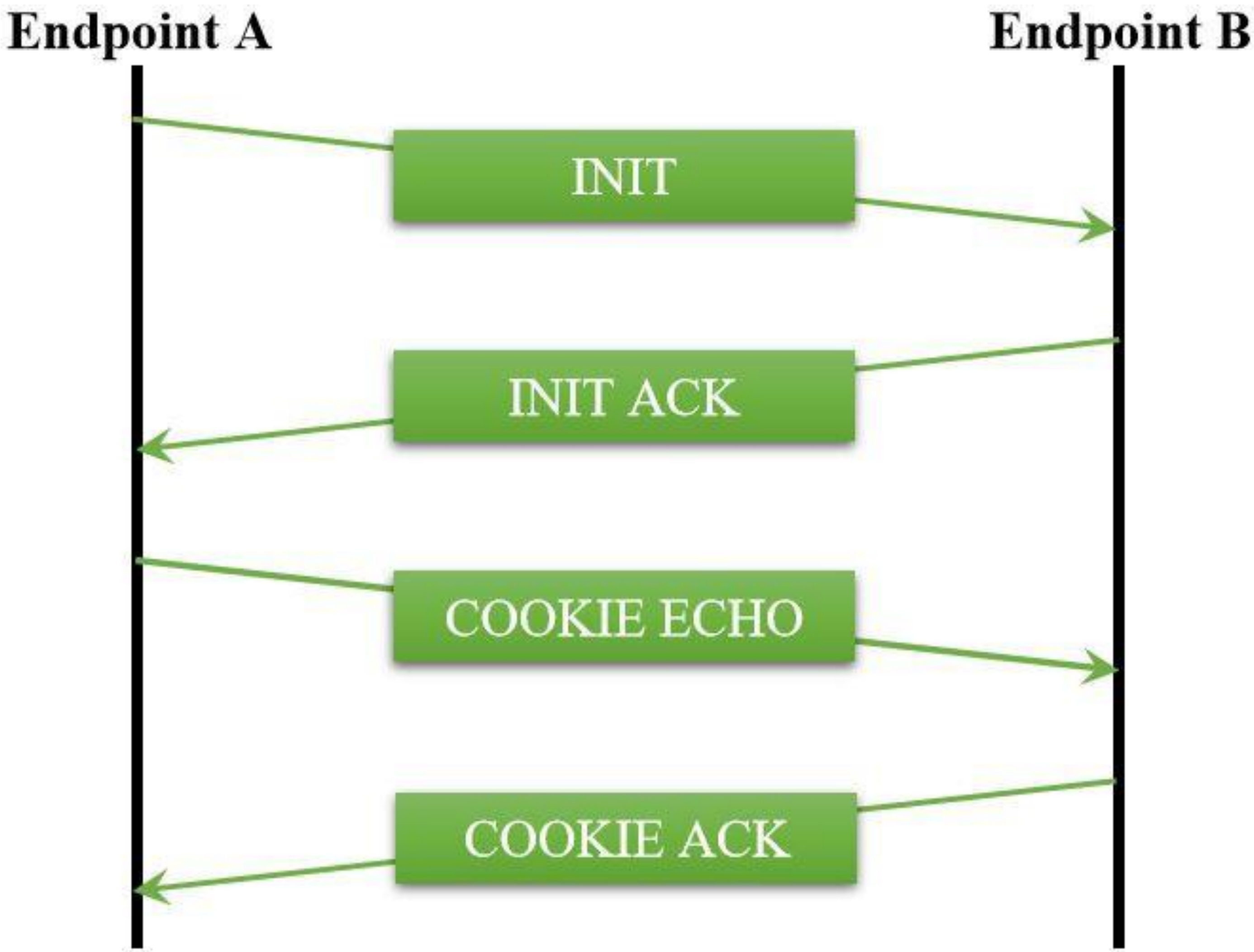
A **SYN flood** is a **Denial of Service (DoS)** attack that targets TCP's **three-way handshake**.

- When the server receives a **SYN**, it immediately allocates resources (like memory for a connection control block) before verifying if the client really exists.
- Attackers can send thousands of fake **SYNs** with **spoofed (Fake) IPs** → server fills up its connection queue, Hence, real clients can't connect.
- **That's the SYN flood attack.**

SCTP (Stream Control Transmission Protocol)

- It is used for **reliable message-based communication** between internet applications.
- Data is sent in **streams**. Each stream has its own sequence numbering (S-SN), so blocking in one stream does not delay others.
- Safer connection setup than TCP (prevents SYN flood attacks).
- Mainly used in Telecom, VoIP, WebRTC, reliable signaling.
- It uses 4-way handshake.

4-way Handshake



4-way Handshake (Cont.)

1. **INIT** (client → server)
   - Client requests to start an association (connection).
2. **INIT-ACK** (server → client)
   - Server replies with an INIT-ACK that includes a cookie (special token).
   - Server does not allocate any resources yet, it just sends back the cookie.

4-way Handshake (Cont.)

3. **COOKIE-ECHO** (client → server)
   - Client returns the cookie back to the server, proving it received the INIT-ACK.
4. **COOKIE-ACK** (server → client)
   - Server verifies the cookie (using a secret key).
   - If valid → connection (association) established.
   - Only now does the server allocate memory and resources.

TCP vs SCTP

Feature	TCP	SCTP
Handshake type	3-way (SYN, SYN-ACK, ACK)	4-way (INIT, INIT-ACK, COOKIE-ECHO, COOKIE-ACK)
When server allocates resources	After receiving SYN	After validating COOKIE-ECHO
SYN flood resistance	Vulnerable	Resistant (stateless handshake)
Authentication of client before resource allocation	None	Uses cookie to confirm client validity

Test Questions

- Why is a 3-way handshake necessary?
- Who sends the first FIN - the server or the client?
- Once the connection is established, what is the difference between the operation of the server's TCP layer and the client's TCP layer?