Sub.: Computer Network & Security

PAGE NO.
DATE :

Unit IV: Transport Layer

§ Transport Layer:

- The Transport layer is layer 4 in the OSI Model, located between the Network Layer and the Application Layer.
- Its main purpose is to provide end-to-end / process-to-process / port-to-port communication.
- It ensures that data from an application on the sender host reaches the correct application on the receiver host.
- The transport layer relies on the Network Layer to deliver data to the correct host and then hands it over to the appropriate process.

§ Services:

(1) Process-to-Process Communication:

- The network layer delivers data to the host, but it does not ensure that the data reaches the correct application process.
- The transport layer provides this functionality using port numbers and socket addresses.

(2) Addressing - Port Numbers:

- A port number identifies a process on a host.
  - ↳ The client uses an ephemeral port number, which is temporary and usually greater than 1023.
  - ↳ The server uses a well-known port number, which is fixed and permanent.
- ICANN divides port numbers into three ranges:
  - ↳ **Well-known ports:** 0 to 1023, controlled by ICANN.
  - ↳ **Registered ports:** 1024 to 49151, can be registered to prevent duplication.
  - ↳ **Dynamic or private ports:** 49152 to 65535, temporary or private use.
- Ex: TFTP server uses port 69.
  SNMP uses ports 161 and 162.

(3) Socket Addresses:

- A socket is the combination of an IP address and a port number. It uniquely identifies a client or server process.
- Both the client and server need socket addresses to establish a transport-layer connection.

(Proof:- Tejal Rane)

(4) Encapsulation and Decapsulation:
- At the sender, the Transport Layer encapsulates application data by adding a transport header and passes the segment to the Network Layer.
- At the receiver, the Transport Layer decapsulates the segment, removes the header, and delivers data to the correct application process.

(5) Multiplexing and Demultiplexing:
- Multiplexing occurs at the sender when data from multiple processes is combined into a single transport stream.
- Demultiplexing occurs at the receiver when incoming segments are delivered to the correct application process.
- Socket addresses are used to ensure that each segment reaches the correct process.
Ex: Client processes P1 and P3 send requests to Server 1. Client process P2 sends a request to Server 2. The transport layer at the client multiplexes the segments, and the servers demultiplex them to the correct processes.

(6) Flow Control:
- Flow control ensures that the sender does not overwhelm the receiver.
- Delivery models:
  ↳ Pushing: The sender sends data as it is produced. Flow Control is required to prevent overflow.
  ↳ Pulling: The receiver requests data when it is ready. Flow control is not needed.
- Transport Layer flow control involves four entities: sender process, sender transport, receiver transport and receiver process. Buffers are used at both sender and receiver to temporarily store packets.
- Signals are sent between producer and consumer to stop or resume sending data when buffers are full or have vacancies.
- Ex: Using a single-slot buffer, the sender sends one packet and waits for acknowledgment before sending the next.

(7) Error Control:
- Error Control ensures reliable delivery over an unreliable network.
Functions of error control:
1. Detect and discard corrupted packets.
2. Track and resend lost packets.
3. Detect and discard duplicate packets.
4. Buffer out-of-order packets.
- Sequence number identify packets and help in resending lost packets, detecting duplicates, and reordering. The receiver sends acknowledgements for received packets, and the sender uses timers to resend packets if acknowledgments are not received.

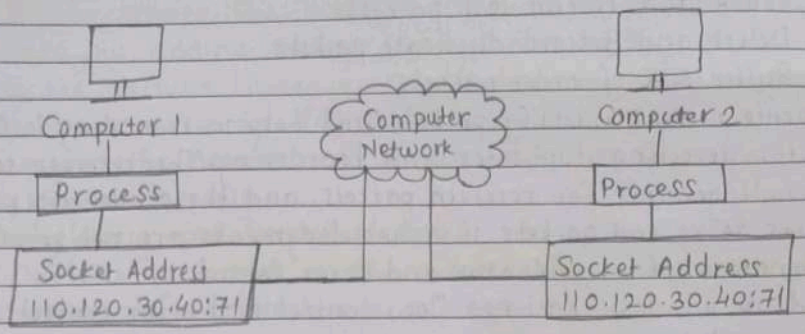
(8) Combination of Flow Control and Error Control:
- Sliding window combines flow control and error control efficiently.
- Sequence numbers are arranged in a circular buffer. The window represents buffer slots available for sending or receiving packets.
- The sender can send multiple packets without waiting for acknowledgment until the window is full. When acknowledgments are received, the window slides forward to allow more packets to be sent.
- Ex: Sequence numbers 0 to 15 with window size 7. Sent but unacknowledged packets are marked. The sliding window ensures continous transmission.

(9) Congestion Control:
- Congestion occurs when network load exceeds network capacity, causing packet loss, delay, and reduced throughput. It usually happens because routers and switches have limited buffer capacity.

§ Socket Programming:
- Socket: An endpoint for communication between two processes running on the same or different machines.
- Socket Address: The combination of an IP address and a port number that uniquely identifies a socket on the network (eg., 192.168.1.67:80). The colon (:) separates the IP and port.
- Purpose:
  ↳ Sockets allow two way communication between processes.
  ↳ Mostly used in client-server architecture.
  ↳ Socket programming teaches how to use socket APIs to

(Prof. Tejal Rane)

create communication between local and remote processes.



- Classes Used:

Service Type	Client/Server Classes
Connection-Oriented (TCP)	Socket, ServerSocket
Connectionless (UDP)	DatagramSocket, DatagramPacket

- Types of Socket Programming:

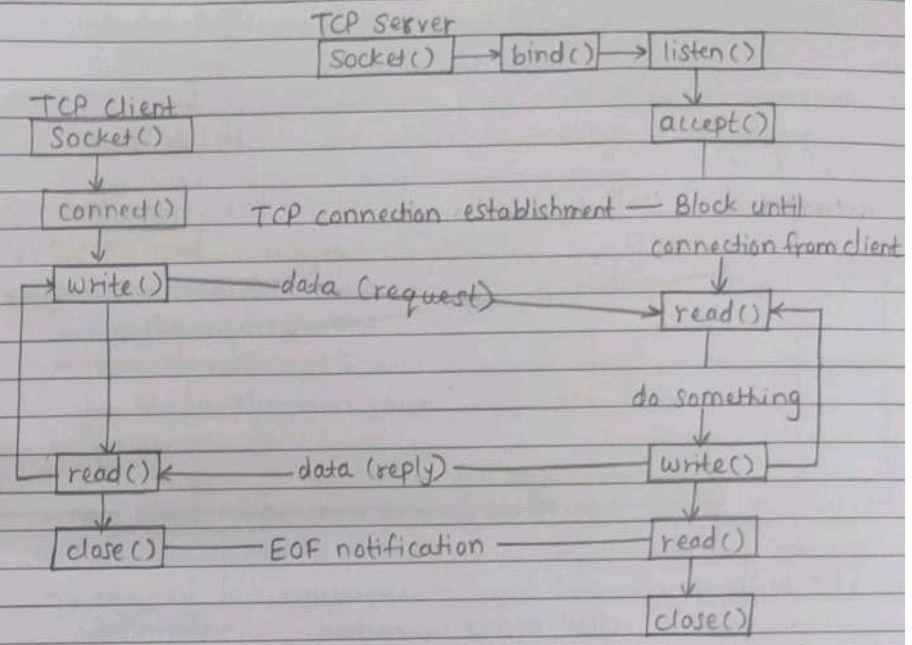
(1) Connection-Oriented (TCP):

↳ TCP: Transmission Control Protocol.

↳ Reliable and connection-oriented.

- Client-Side Steps & Functions:
  i. socket() → Create a socket endpoint
  ii. connect() → Establish connection to the server
  iii. read()/write() → Receive and send data
  iv. close() → Terminate the connection

- Server-Side Steps & Functions:
  i. socket() → Create socket endpoint.
  ii. bind() → Bind socket to IP address and port
  iii. listen() → Wait for client connections
  iv. accept() → Accept a client connection.
  v. read/write() → Receive and send data
  vi. close() → Terminate the connection.



TCP Socket connection

(2) Connectionless (UDP):

↳ UDP: User Datagram Protocol

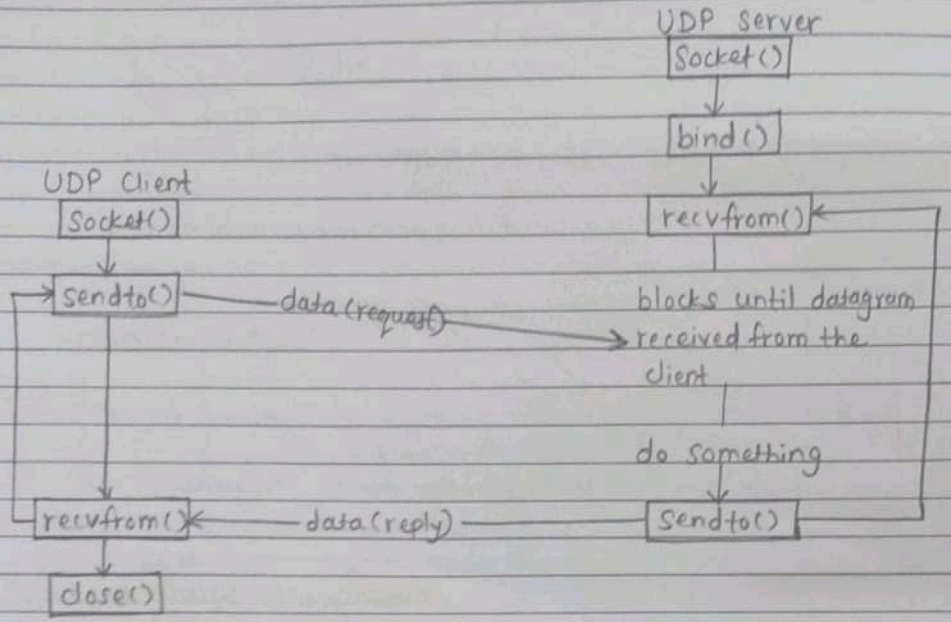
↳ Unreliable and connectionless.

- Client-Side Steps & Functions:
  i. socket() → Create Socket endpoint
  ii. sendto() → Send data to server
  iii. recvfrom() → Receive data from server.

- Server-Side Steps & Functions:
  i. socket() → Create socket endpoint
  ii. bind() → Bind socket to IP address and port
  iii. sendto() → Send data to client
  iv. recvfrom() → Receive data from client

* Note: No connection establishment (connect()) or termination (close()) is required in UDP.

(Prof. - Tejal Rane)



UDP socket connection.

- Socket Programming Interface Types:
  (1) Stream Sockets:
    ↳ Connection-oriented.
    ↳ Data arrives in order.

  (2) Datagram Sockets:
    ↳ Connectionless.
    ↳ Data may be lost or arrive out of order.

  (3) Raw Sockets:
    ↳ Bypass built-in protocol support (UDP/TCP)
    ↳ Used for custom low-level protocol development.

(Prof. - Tejal Rane)

§ Transport Layer Protocols:

(1) TCP (Transmission Control Protocol):
- Reliable and connection oriented.
- Ensures ordered delivery and error-free transmission.
- Used in applications where reliability is important (eg.: file transfer, email, web browsing)

(2) UDP (User Datagram Protocol):
- Unreliable and connectionless.
- Simple and efficient.
- Suitable for applications where error control is handled by the application layer (eg.: video streaming, DNS)

(3) SCTP (Stream Control Transmission Protocol):
- Combines some features of UDP and TCP.
- Reliable, supports multi-streaming and multi-homing.
- Designed for multimedia and telecommunication applications.

(4) RTP (Real-Time Transport Protocol):
- Designed for real-time multimedia transmission (audio, video).
- Works on top of UDP.
- Provides sequencing, timing, and payload type identification.
- Use cases: VoIP, video conferencing, live streaming.

§ Transmission Control Protocol (TCP):
- TCP is a connection-oriented and reliable transport layer protocol.
- It defines three phases:
  ↳ Connection establishment
  ↳ Data transfer.
  ↳ Connection termination
- Reliability is achieved using:
  ↳ Checksum (error detection).
  ↳ Retransmission of lost/corrupted packets
  ↳ Cumulative & selective acknowledgments.
  ↳ Timers

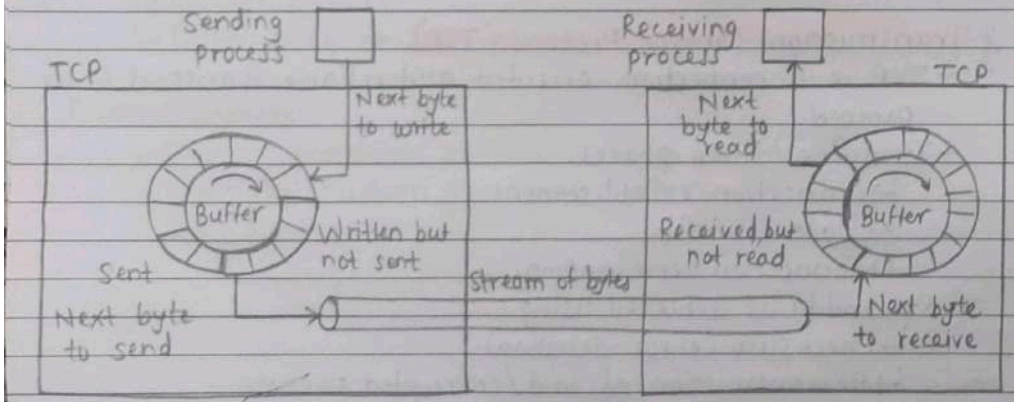
- TCP is the most widely used transport layer protocol in the internet

- **Services of TCP:**

(1) **Process-to-Process Communication:**
- Communication between processes, not just hosts
- Identified using port numbers
- Ex: HTTP → port 80, FTP → port 21.

(2) **Stream Delivery Service**
- TCP is stream-oriented.
- Data is delivered as a continous stream of bytes
- Acts like an imaginary tube between sender & receiver
- Application does not see message boundaries.

(3) **Sending & Receiving Buffers:**
- Required because sender and receiver may work at different speeds
- Sender Buffer:
  ↳ Stores data written by process but not yet sent.
  ↳ Keeps copies of sent but unacknowledged data.
- Receiver Buffer:
  ↳ Stores bytes received from network but not yet read by application.
- Buffers are usually implemented as circular arrays.



(4) **Segmentation:**
- TCP groups stream of bytes into segments.
- Each segment has a TCP header + data.
- Segments are encapsulated in IP datagrams.
- Segments may arrive out-of-order, lost or duplicated is handled by TCP.

(5) **Full Duplex Communication:**
- Data flows in both directions simultaneously.
- Each side maintains separate sending & receiving buffers.

(6) **Multiplexing and Demultiplexing:**
- Multiple processes can use TCP simultaneously.
- At the sender multiplexing is done.
- At the receiver demultiplexing is done using IP address, port number.

(7) **Connection-Oriented Service:**
- Communication occurs in three steps:
  - Connection Establishment
  - Data Transfer
  - Connection Termination.
- Connection is logical, not physical.
- Ensures in-order delivery of bytes.

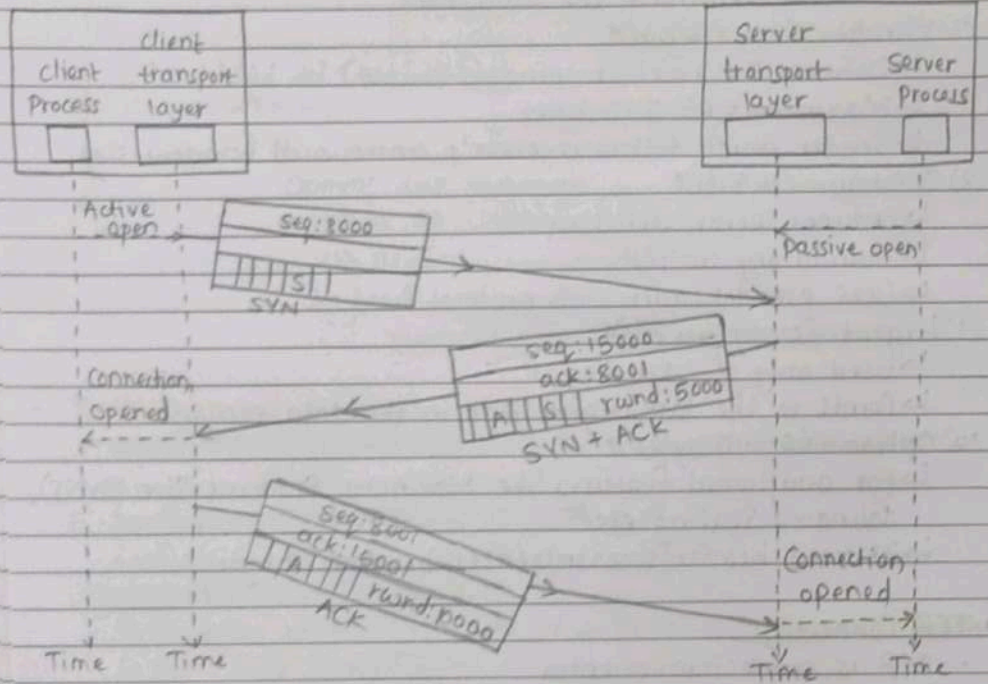
(8) **Reliable Service:**
- TCP ensures reliable communication using:
  - Acknowledgments (ACKs)
  - Retransmissions
  - Error control with checksum
- Receiver gets data correctly & in order.

- **TCP Segment:**
- A TCP packet is called a segment
- Each segment consists of a header (20-60 bytes) + application data.
- Minimum header = 20 bytes (no options)
- Maximum header = 60 bytes (with options)

(Prof. Tejal Rane)

- TCP communication involves three main phases:
  (1) Connection Establishment
  (2) Data Transfer
  (3) Connection Termination

(1) **Connection Establishment**:
↳ TCP is full-duplex, meaning both client and server can send and receive data simultaneously.
↳ Before data transfer, both sides must initialize communication and get approval.
↳ This is acheived through the Three-Way Handshake process



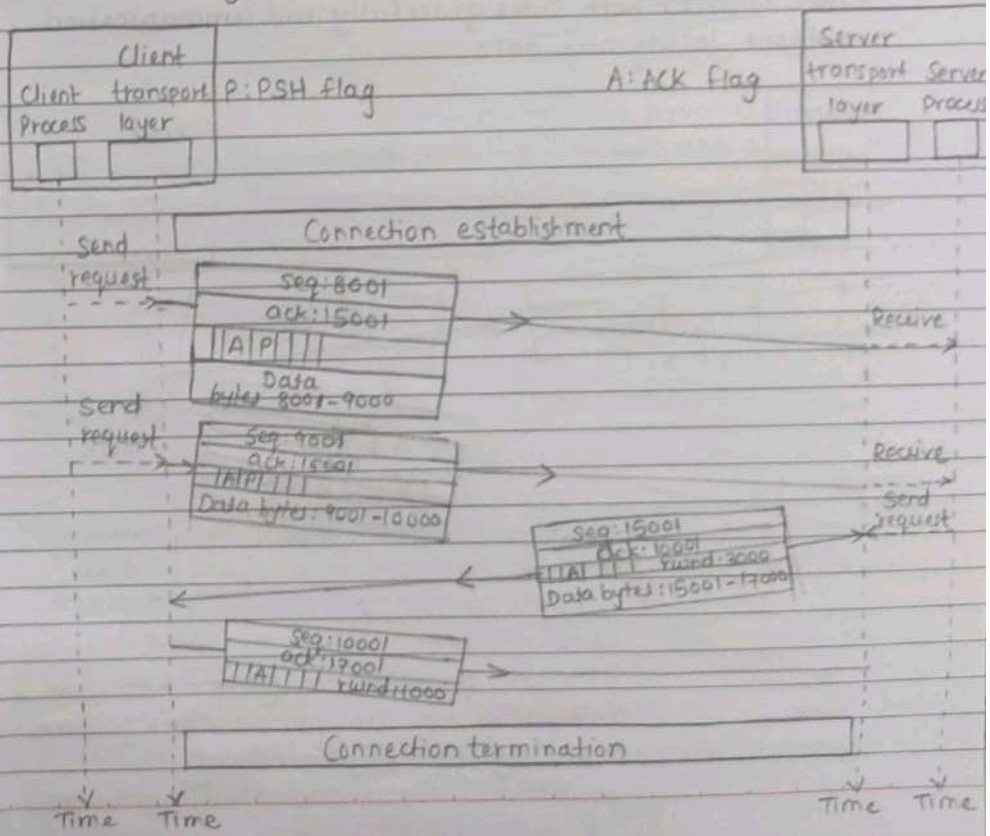
Connection establishment using three way handshaking

↳ Steps:
(i) **SYN**: The client sends a SYN segment (synchronization) with an initial sequence number (ISN).
(ii) **SYN + ACK**: The server replies with a SYN+ACK segment, which both acknowledges the client's SYN and provides the server's ISN.

(Proof - Tejal Rane)

(iii) **ACK**: The client sends an ACK segment to confirm the server's SYN.
After these three steps, the connection is established.

(2) **Data Transfer**: Once connected, bidirectional communication begins:
↳ Both client and server can send data + acknowledgments (piggybacking)
↳ Ex: The client sends multiple segments, and the server replies with its own data segments.
↳ PSH (Push Flag): Used when data must be delivered immediately without waiting for more bytes to fill the buffer.
↳ Urgent Data:
→ TCP allows urgent bytes to be send for immediate processing
→ An Urgent Pointer indicates where the urgent data ends.



(3) TCP Connection Termination:

TCP connections can be closed in different ways. The most common methods are Three-Way Handshake, Four-Way (Half-Close) or using a Reset (RST) flag.

(a) Three-Way Handshake Termination:

↳ This is a standard method of closing a TCP connection.

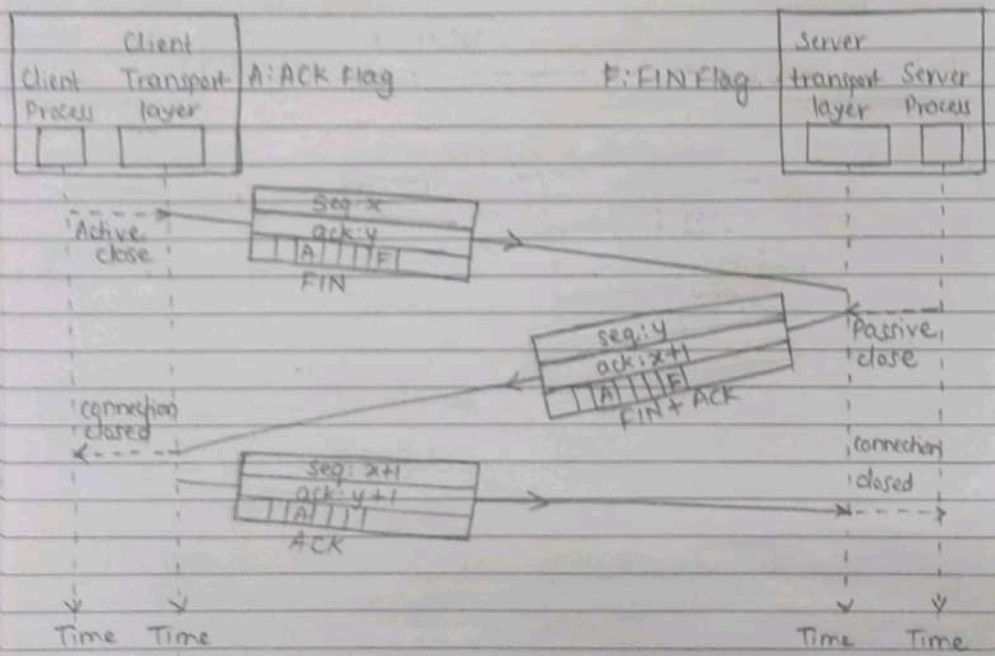
↳ Steps involved:

(i) Client → Server: Sends a **FIN** segment (may include the last data). It consumes 1 sequence number.

(ii) Server → Client: Responds with **FIN + ACK** to acknowledge the client's request and signal its own closure.

(iii) Client → Server: Sends a final **ACK** segment to confirm termination (No sequence number consumed if it carries no data).

↳ This ensures both sides gracefully end communication without losing any data.



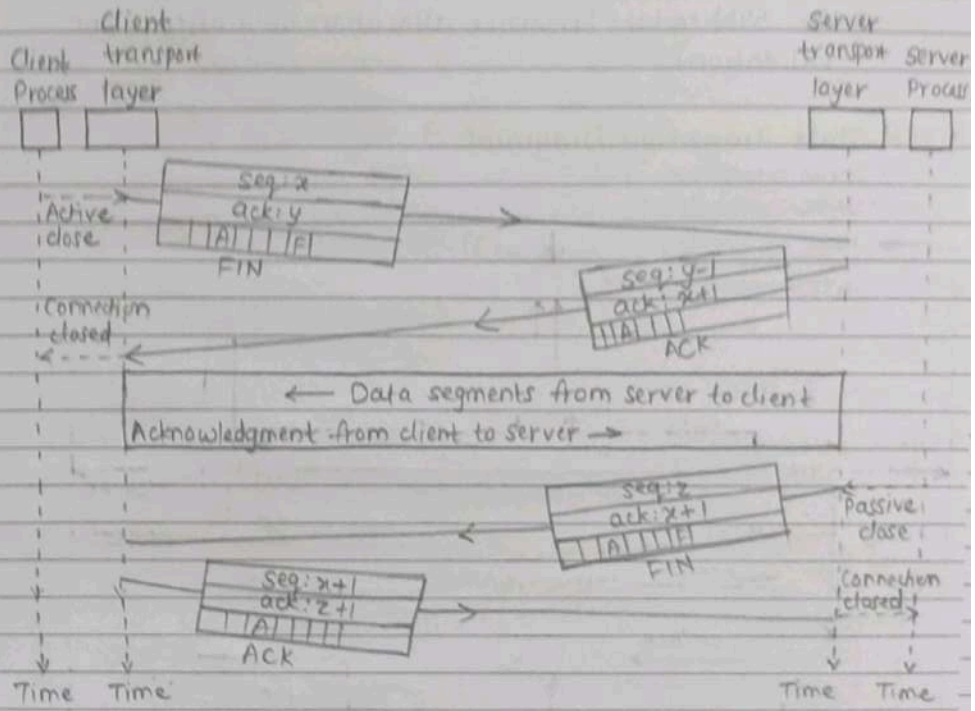
Connection termination using three-way handshake

(b) Half-Close (Four-Way Termination):

↳ In this method, one side stops sending data but continues to receive from the other side.

↳ It is useful when all data from one side must be processed before closing.

↳ Ex: A sorting application where the client sends all data first, then waits for sorted results from the server.



Half-close

(c) Connection Reset (Abrupt Termination):

↳ Sometimes, a connection must be ended immediately.

↳ TCP uses the RST (Reset) flag for:

→ Denying a connection request

→ Aborting an ongoing connection.

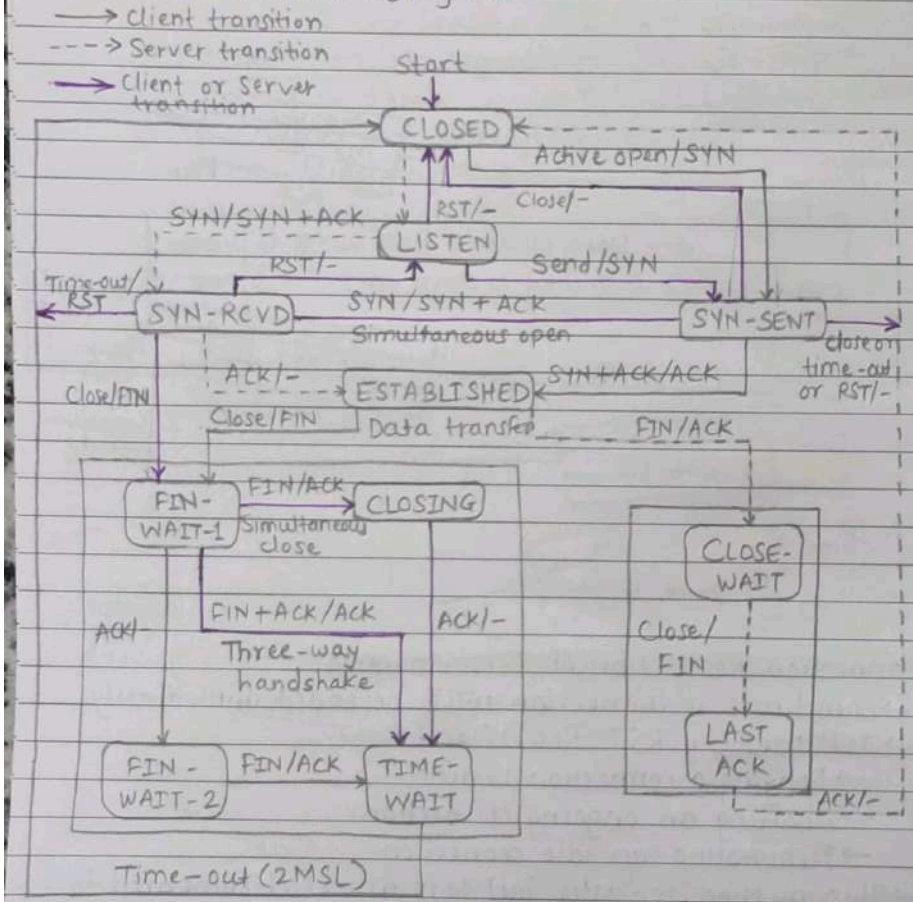
→ Terminating an idle connection.

↳ This method is faster but less graceful compared to handshake-based termination.

(Prof. Tejal Rane)

* Security Issue - SYN Flooding Attack:
  - Attacker sends multiple fake SYN requests with forged IP addresses.
  - Server allocates resources but never gets final ACK.
  - Leads to Denial of Service (DoS).
  - Solutions:
    - Limit connection requests.
    - Filter suspicious IPs.
    - Use SYN cookies (resource allocation only after client validation).

• TCP State Transition Diagram:



(Prof. Tejal Rane)

TCP behaviour during connection establishment, data transfer, and termination is specified as a Finite State Machine (FSM). The state transition diagram shows how a TCP client and server move between states based on events.

- Client-Side Transition (Active Open):
  1. Active Open → Client issues request. TCP sends SYN, enters SYN-SENT.
  2. On receiving SYN+ACK, client replies with ACK → goes to ESTABLISHED.
  3. Data Transfer occurs in both directions.
  4. Active Close → Client sends FIN, enters FIN-WAIT-1.
     ↳ After ACK → goes to FIN-WAIT-2
     ↳ On receiving FIN → sends ACK, enters TIME-WAIT.
     ↳ Remains in TIME-WAIT for 2MSL (Maximum Segment Lifetime), then goes to CLOSED.

- Server-Side Transition (Passive Open):
  1. Passive Open → Server TCP waits in LISTEN state.
  2. On receiving SYN → replies with SYN+ACK, enters SYN-RCVD.
  3. On receiving ACK → enters ESTABLISHED.
  4. Data Transfer continues until client closes.
  5. On receiving FIN from client → server sends ACK, enters CLOSE-WAIT.
  6. When application issues Passive Close → server sends FIN, enters LAST-ACK.
  7. On receiving client's final ACK → moves to CLOSED.

• Flow Control in TCP:
Flow Control ensures that a fast sender does not overwhelm a slow receiver.
  - TCP separates flow control from error control.
  - Works using sliding window mechanism (Receive Window rwnd)
  - Receive Window (rwnd):
    ↳ Receiver advertises how much buffer space it has (rwnd).
    ↳ Sender adjusts its send window based on rwnd.

↳ Closing: Window shrinks when more data arrives.
↳ Opening: Window expands when receiver's process consumes data.
↳ Receive window never shrinks.
↳ Send window can shrink (but discouraged, except in shutdown)

- Flow Control Feedback:
  ↳ Feedback travels from receiver to sender via acknowledgments.
  ↳ Sender adjusts send window size according to rwnd.
- Window Behaviour:
  - Send Window:
    ↳ Closes (left wall moves right) when ACKs arrive.
    ↳ Opens (right wall moves right) when rwnd allows more sending.
    ↳ Shrinks if receiver advertises smaller rwnd.
  - Receive Window:
    ↳ Closes when new data arrive.
    ↳ Opens when application consumes data.
    ↳ Never shrinks (except temporary shutdown).
- Window Shutdown:
  ↳ Receiver can advertise rwnd = 0 → temporarily stop sender.
  ↳ Sender stops sending, but may probe with 1-byte segments (to avoid deadlock).

- Silly Window Syndrome (SWS):
  ↳ Problem: Small, inefficient segments transmitted (eg: 1-byte payload with 40+ bytes header)
  ↳ Causes:
    → Sender sends small chunks (application produces data slowly)
    → Receiver consumes data slowly and advertises very small rwnd.
  ↳ Solutions:
    → Sender side (Nagle's Algorithm):
      - Send the first small segment immediately.
      - Then, accumulate data until either:
        - An ACK is received, or
        - Enough data collected to fill a maximum-size segment.
    → Receiver side (Clark's solution):
      - Advertise rwnd = 0 until:
        - Enough space for maximum segment size, OR
        - At least half the buffer is free.
      - Or use delayed ACK (≤ 500 ms)

- Error Control in TCP:
  TCP ensures reliable, in-order, error-free delivery.

* Tools Used:
  1. Checksum (16-bits): Detects corruption.
  2. Acknowledgments (ACKs): Confirms receipts.
  3. Timers (Timeouts): Ensures lost segments are retransmitted.

* Types of Acknowledgments
  ↳ Cumulative ACK (default):
    → ACK number = next expected byte.
    → Confirms receipt of all bytes before it.
    → Simple but does not tell about lost / delayed segments.
  ↳ Selective ACK (SACK, optional):
    → Reports specific out-of-order blocks received.
    → Helps sender retransmit only missing segments.

* ACK Generation Rules:
  1. ACK is piggybacked with data (if possible).
  2. If only one segment received then delay ACK (≤ 500ms)
  3. If two in-order unacknowledged segments then send ACK immediately.
  4. If out-of-order segment arrives then send ACK for expected byte (duplicate ACK).
  5. When missing segment arrives then ACK it immediately.
  6. Duplicate segment then discard & ACK next expected byte.

* Retransmission:
  ↳ Timeout (RTO-based): If timer expires then resend earliest unACKed segment.
  ↳ Fast Retransmit (3 Duplicate ACKs): Resend missing segment immediately (before timeout).

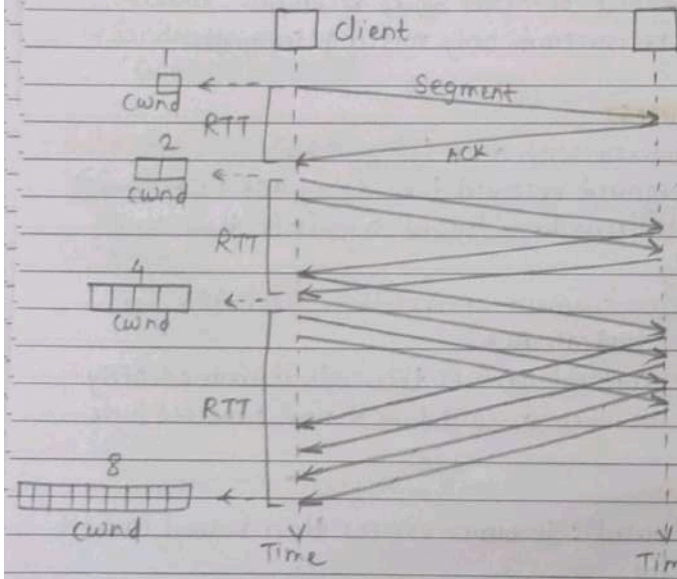
(Prof. Tejal Rane)

* **Handling Out-of-Order Segments:**
  ↳ Receiver buffers them (not delivered to application).
  ↳ Data always delivered in-order to process.

• **Congestion Control in TCP:**
TCP must handle congestion in the network to avoid segment loss and network collapse. It uses a congestion window (cwnd) along with the receiver window (rwnd) to control the number of unacknowledged bytes in transit.
Actual Window Size = min(rwnd, cwnd)

1. **Congestion Detection:**
TCP assumes congestion in the network using two events:
(i) Time-out: strong congestion (no ACK received in time).
(ii) Three duplicate ACKs: weak congestion (a segment is missing, but others are delivered).

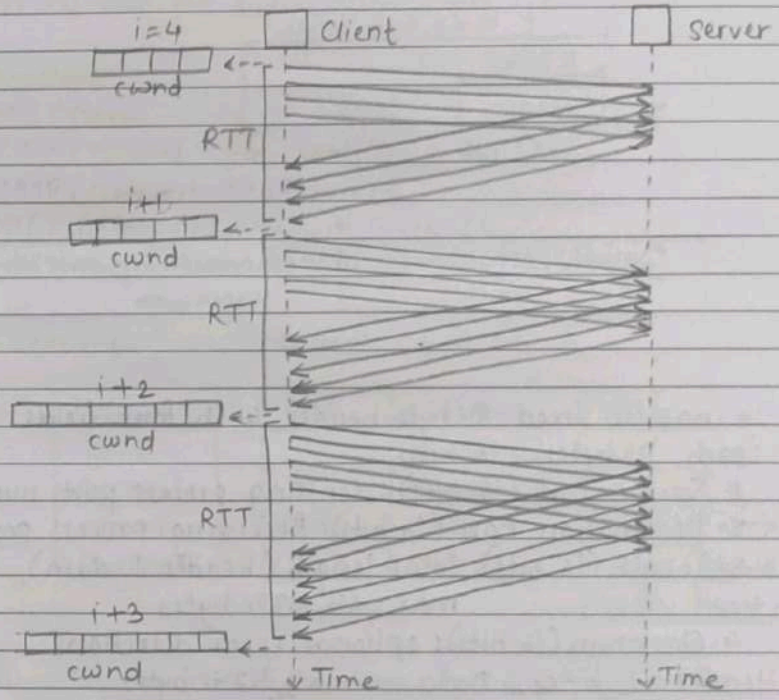
2. **Congestion Control Policies:** TCP uses three main algorithms:
(a) **Slow Start (Exponential Increase):**



Slow start, exponential increase

↳ Initial cwnd = 1 MSS
↳ For each ACK cwnd = cwnd + 1 MSS
↳ Growth is exponential until a threshold (ssthresh) is reached.
↳ When cwnd ≥ ssthresh then move to Congestion Avoidance.

(b) **Congestion Avoidance (Additive Increase):**
↳ When cwnd ≥ ssthresh, cwnd increases linearly.
↳ For each RTT, cwnd = cwnd + 1 MSS.
↳ Growth is additive.



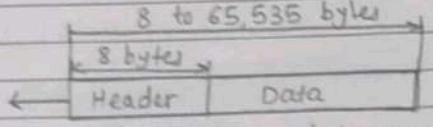
Congestion avoidance, additive increase

(c) **Fast Recovery:**
↳ Triggered when 3 duplicate ACKs are received.
↳ cwnd is adjusted to ssthresh + 3MSS.
↳ cwnd then grows additively until a new ACK arrives.
↳ Helps in quick recovery without reducing cwnd to 1.

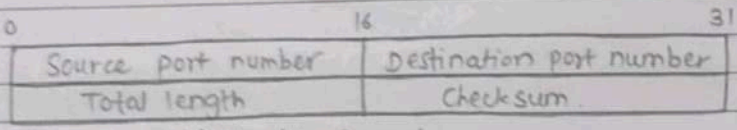
(Prof. - Tejal Rane)

User Datagram Protocol (UDP):

- UDP is a connectionless, unreliable transport protocol.
- It provides process-to-process communication (using port numbers) instead of only host-to-host communication.
- UDP adds very little to IP i.e. only ports, length and checksum.
- It is simple, fast, and has low overhead, but does not guarantee delivery, order, or error recovery.
- User Datagram Packet Format:
  - ↳ A UDP packet is called a User Datagram.



(a) UDP user datagram



(b) Header Format

↳ It has a fixed 8-byte header with four fields (each 2 bytes = 16 bits):

1. Source Port (16 bits): Sending process port number.
2. Destination Port (16 bits): Receiving process port number.
3. Length (16 bits): total length (header + data), max = 65,535 bytes.
4. Checksum (16 bits): optional error detection.

↳ Header = 8 bytes ; Data = 0 - 65,527 bytes.

- UDP Services:

1. Process-to-process Communication: using socket addresses (IP + port).
2. Connectionless Service: no connection setup / termination; each datagram is independent.
3. No Flow Control: Receiver may overflow; handled by application.
4. Error Control: Only checksum; If error detected then packet is discarded silently.
5. Checksum: calculated using pseudoheader + header + data. Detects corruption, but no retransmission.
6. No Congestion Control: assumes data is small & sporadic.

- Features:

1. Simple, low overhead.
2. Independent packets so less delay.
3. Suitable for small, real-time or multicast traffic.
4. Less reliable but much faster than TCP.

- Applications:

1. DNS (Domain Name System).
2. TFTP (Trivial File Transfer Protocol).
3. SNMP (Network Management).
4. RIP (Routing Information Protocol).
5. Real-time audio/video (VoIP, Skype, streaming).

- Well known Port Numbers:

Port	Protocol
7	Echo
9	Discard
11	Users
13	Daytime

Port	Protocol
17	Quote
19	Chargen
53	DNS
67	DHCP

Port	Protocol
69	TFTP
111	RPC
123	NTP
161	SNMP server
162	SNMP client

Q. The following is the content of a UDP header in hexadecimal format

CB84000D001C001C

a. What is the source port number?
b. What is the destination port number?
c. What is the total length of the user datagram?
d. What is the length of the data?
e. Is the packet directed from a client to a server or vice versa?

(Prof. - Tejal Rane)

f. What is the client process?

Soln:

Given UDP Header (in Hexadecimal):

CB84 000D 001C 001C

Step 1 – Divide into 4 fields (16 bits each):

Field	Hex Value	Decimal Value
Source Port	CB84	52100
Destn Port	000D	13
Length	001C	28 bytes
Checksum	001C	–

Step 2 – Find Data Length:

UDP length = 28 bytes

Header Length = 8 bytes

Data Length = 28 – 8 = 20 bytes

Step 3 – Determine direction

Destination Port = 13 → Well-known port.

Direction: Client → Server.

Step 4 – Identify Client Process.

Port 13 corresponds to the Daytime Service Process.

✱ Note:

For identifying client or server:

- If destination port is well known → Client to Server.
- If source port is well known → Server to Client.

§ Stream Control Transmission Protocol (SCTP):

- SCTP is a transport layer protocol designed by IETF as an alternative to TCP and UDP.
- It combines features of both:
  - ↳ Reliability of TCP.
  - ↳ Message orientation of UDP.
- It is mainly used in applications like telephony signaling (SS7 over IP), multimedia, and Internet communications.

- Services:

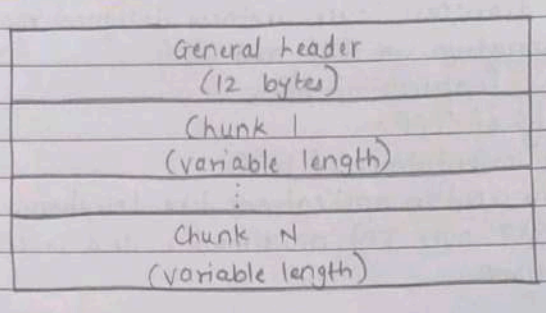
1. Process-to-Process Communication: Provides communication between two processes like TCP & UDP.
2. Full-Duplex Communication: Data can flow in both directions simultaneously.
3. Connection-Oriented Service: The connection in SCTP is called an Association.
4. Reliable Service: Provides reliable data transfer using acknowledgments, retransmissions, and checksums.
5. Multistreaming: Multiple independent streams can exist within a single association; loss in one stream does not block others.
6. Multihoming: Each endpoint can use multiple IP addresses for fault tolerance and path redundancy.

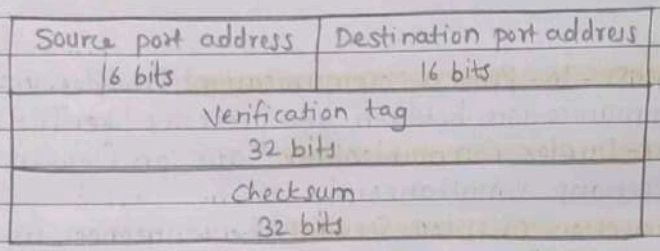
- Features:

1. TSN (Transmission Sequence Number): Each data chunk is assigned a unique TSN.
2. Stream Identifier (SI): Identifies the stream within an association.
3. Stream Sequence Number (SSN): Maintains order within a stream.
4. Packet Structure: Contains one or more chunks (control or data).
5. Message Oriented: Preserves message boundaries.

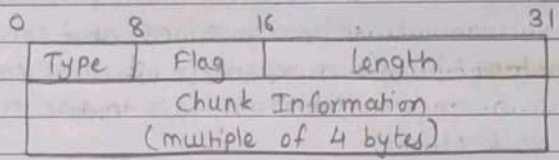
(Prof – Tejal Rane)

- SCTP Packet Format:





General header



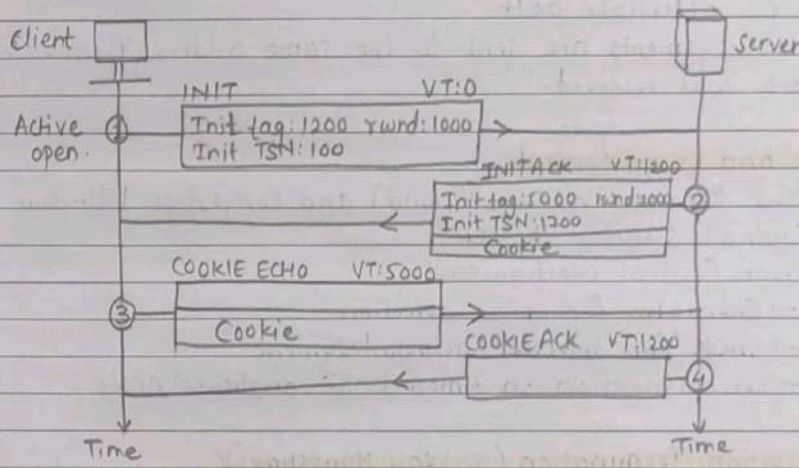
Common layout of a chunk.

- Header Field:
  1. Source Port Number (16 bits): Port number of sender
  2. Destination Port Number (16 bits): Port number of receiver.
  3. Verification Tag (32 bits): Unique value identifying association.
  4. Checksum (32 bits): Detects transmission errors.

- Chunk Format:
  1. Type (8 bits): Identifies type (Control / Data)
  2. Flags (8 bits) Special control bits
  3. Length (16 bits): Total length of the chunk.
  4. Data (Variable): Contains actual control information or user data.

- Types of Chunks:
  1. Control Chunks: INIT, INIT-ACK, COOKIE-ECHO, COOKIE-ACK, SHUTDOWN, etc.
  2. Data chunks: Carry user data along with TSN, Stream ID, and SSN.

- SCTP Association:
- Association Establishment (4-Way Handshake):
SCTP uses a four-step handshake to establish an association:



1. INIT (Client to Server): Client sends INIT chunk requesting association.
2. INIT-ACK (Server to Client): Server replies with INIT-ACK and includes a cookie.
3. COOKIE-ECHO (Client to Server): Client returns the cookie to prove authenticity.
4. COOKIE-ACK (Server to Client): Server verifies and finalizes the association.

This 4-way handshake provides protection against SYN flooding attacks and prevents half-open connections.

- Data Transfer:
↳ SCTP supports multistreaming, allowing multiple independent

(Prof. Tejal Rane)

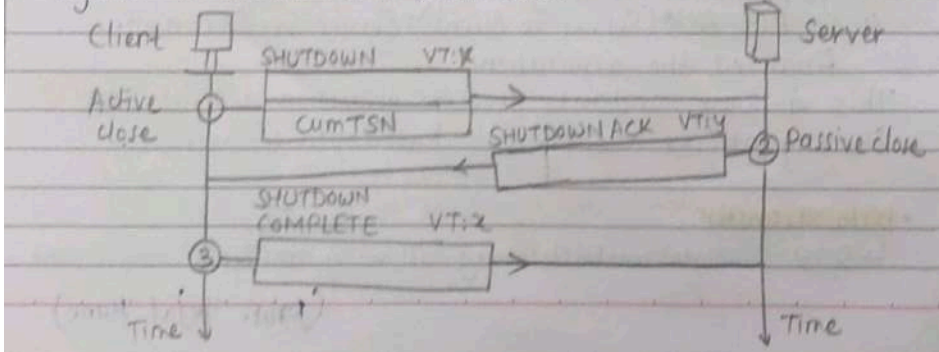
message streams in a single association.
↳ Supports ordered and unordered delivery.
↳ Uses SACK (Selective Acknowledgement) to confirm received chunks.
↳ Ensures message boundaries are preserved.

- **Multihoming:**
  ↳ Each endpoint can have multiple IP addresses.
  ↳ One address acts as primary, others as backup paths.
  ↳ If the primary path fails, SCTP automatically switches to an alternate path.
  ↳ Acknowledgments are sent to the same address from which data was received.

- **Flow and Error Control:**
  ↳ Uses Receiver Window (rwnd) and Congestion Window (cwnd) similar to TCP.
  ↳ Error Control Mechanism:
    → Checksum for error detection.
    → SACK for selective acknowledgment.
    → Retransmission on timeout or duplicate ACKs.

- **Association Termination (3-Way Handshake):**
  SCTP terminates an association in three steps:
  1. **SHUTDOWN**: Endpoint indicates no more data to send.
  2. **SHUTDOWN-ACK**: Peer acknowledges shutdown.
  3. **SHUTDOWN-COMPLETE**: Completes the termination.

SCTP does not support half-close; both directions close together to ensure clean termination.



§ **Real-time Transport Protocol (RTP):**
- RTP is used for real-time multimedia traffic such as voice, video and streaming.
- It does not provide delivery functions like routing, error recovery, or port assignment.
- Instead, it works with UDP, providing end-to-end delivery of real-time data.
- RTP is conceptually located between UDP and the application layer.

- **RTP Packet Format:**
An RTP packet consists of a fixed header (12 bytes), optional fields, and a payload (application data).

Ver	P	X	Contr. count	M	Payload type	Sequence number
Timestamp						
Synchronization source (SSRC) identifier						
Contributing source (CSRC) identifier (1)						
⋮						
Contributing source (CSRC) identifier (N)						
Extension header						

- **Version (Ver)**: 2 bits, current version = 2
- **Padding (P)**: 1 bit, indicates padding at the end (used in encryption)
- **Extension (X)**: 1 bit, signals an optional header.
- **Contributing Source Count (CC)**: 4 bits, number of contributing sources (max 15)
- **Marker (M)**: 1 bit, marks end of a frame.
- **Payload Type (PT)**: 7 bits, indicates payload format (audio, video, etc.)
- **Sequence Number**: 16 bits, helps detect packet loss/reordering.
- **Timestamp**: 32 bits, used for synchronization of media streams.

(Prof. Tejal Rane)

- SSRC (Synchronization Source ID): 32 bits, unique identifier for the main source.
- CSRC (Contributing Source IDs): identifies up to 16 contributing sources.

- Operation of RTP:
  ↳ Application data is encapsulated into RTP packets
  ↳ RTP packets are then encapsulated into UDP datagrams and transmitted via IP.
  ↳ Port Numbers: RTP generally uses an even-numbered ports

§ Congestion Control:
Congestion control refers to techniques that prevent or remove congestion in a network.
Two categories:
(1) Open-loop congestion control (Prevention): Applied before congestion happens
(2) Closed-loop congestion control (Removal): Applied after congestion has occurred.

(1) Open-loop Congestion Control (Prevention):
Handled by source/destination using the following policies:
- Retransmission Policy: Efficient retransmission with proper timers prevents unnecessary packet duplication
- Window Policy: Selective Repeat is better than Go-Back-N since it resends only lost packets, avoiding extra traffic.
- Acknowledgment Policy: Receiver may delay or combine ACKs to reduce load on the network.
- Discarding Policy: Routers may drop less-sensitive packets (eg, audio) to reduce congestion without major quality loss.
- Admission Policy: In virtual-circuit networks, routers check resources before admitting new flows to avoid future congestion.

(prof. Tejal Rane)

(2) Closed-loop Congestion Control (Removal):
Applied when congestion has already happened Mechanisms include:
- Backpressure: Congested node tells upstream nodes to slow down; propagates back to the source.
- Choke Packet: Router directly informs the source (eg, ICMP Source Quench).
- Implicit Signaling: Source infers congestion from delays or missing ACKs
- Explicit Signalling: Routers set bits inside packets
  - Backward signalling: Warns source to slow down.
  - Forward Signalling: Warns destination, which can delay ACKs to slow sender.

§ Quality of Service (QoS):
- Quality of Service (QoS) refers to the set of techniques that ensure a flow (sequence of packets) achieves the required performance in a network. It focuses on how well the network can handle different types of traffic (eg, email vs. video conferencing).

- Flow Characteristics:
1. Reliability: Probability of delivering packets without loss
   ↳ Important for: Email, file transfer, Internet browsing.
   ↳ Less critical for: Telephony, audio conferencing.

2. Delay: Time taken for a packet to travel from source to destination.
   ↳ Sensitive applications: Telephony, video conferencing, remote login.
   ↳ Less sensitive: File transfer, email.

3. Jitter: Variation in packet delay.
   ↳ Low jitter: Packets arrive with uniform delay is acceptable for audio/video.
   ↳ High jitter: Packets arrive irregularly causes disturbance in multimedia.

4. **Bandwidth**: Amount of data that can be transmitted per second.
   ↳ High requirement: Video conferencing (Mbps range)
   ↳ Low requirement: Email (few kbps)

- **Techniques to improve QoS:**

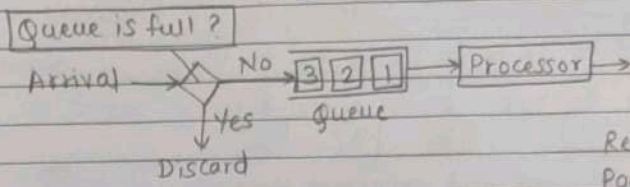
1. **Scheduling**: Determines order of packet processing.
   - **FIFO (First-In-First-Out) Queuing:**
     ↳ Packets are processed in the order they arrive.
     ↳ If arrival rate > processing rate then queue fills and new packets are discarded.
     ↳ No differentiation between packets (all treated equally)
     ↳ Suitable for simple networks but causes:
       → Delay and jitter.
       → Unfairness for delay-sensitive traffic (eg: voice/video)
     ↳ FIFO is default queuing in the Internet.

   - **Priority Queuing (PQ):**
     ↳ Packets are classified by priority (based on ToS field in IPv4, Traffic class in IPv6, or port numbers)
     ↳ Each class has its own queue.
     ↳ Router always processes higher priority queue first.
     ↳ Improves QoS for delay-sensitive traffic (voice/video)
     ↳ If high-priority traffic is continous then lower priority packets suffer starvation.
     Ex: high-priority queue: Voice packets
         low-priority queue: Email packets.

   - **Weighted Fair Queuing (WFQ):**
     ↳ Improvement over priority queuing.
     ↳ Packets classified into different queues with weights assigned.
     ↳ Router serves each queue round-robin order based on weight.
     Ex: If weights = 3:2:1 then router processes 3 packets from Q1, 2 from Q2, 1 from Q3 repeatedly.

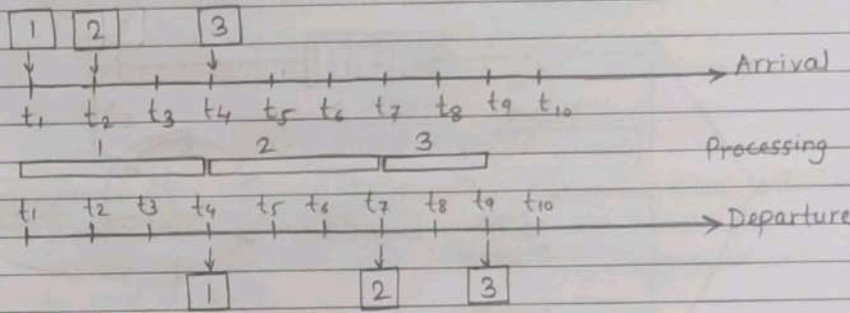
(Prof. Tejal Rane)

↳ Advantages:
- Provides fair bandwidth distribution.
- Reduces starvation.
- Ensures QoS differentiation.



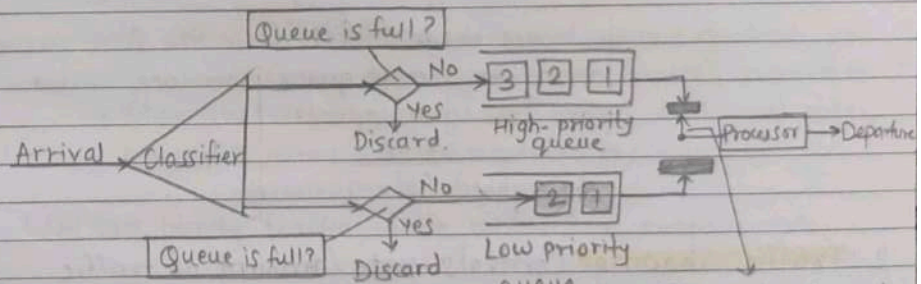
a. Processing in the router

Required processing time
Packet 1: three time units
Packet 2: three time units
Packet 3: two time units.



b. Arrival and departure time

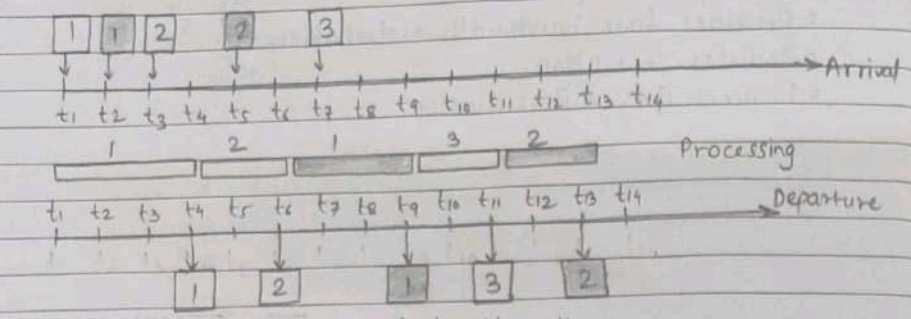
FIFO queue.



Low priority packet
High priority packet

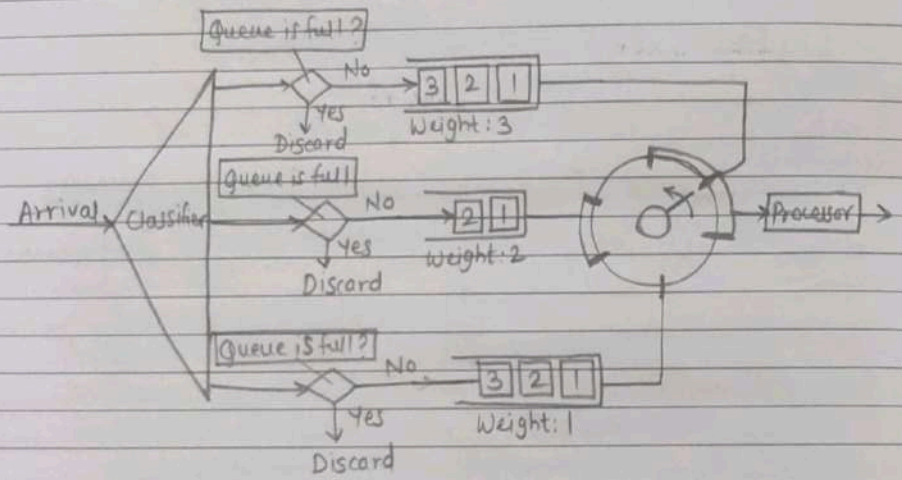
(a) Processing in router

The switch turns to the other queue when the current one is empty.



b. Arrival and departure time

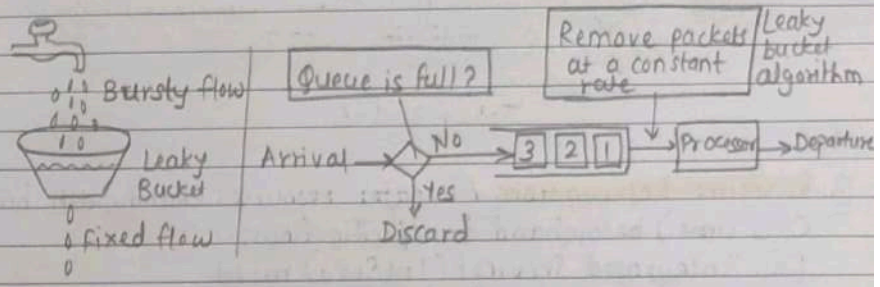
Priority queuing



The turning switch selects three packets from the first queue, then two packets from the second queue, then one packet from the third queue. The cycle repeats.

Weighted fair queueing

2. Traffic Shaping: Controls rate / amount of traffic.
   - Leaky Bucket:
     - ↳ Acts like a bucket leaking water at a constant rate.
     - ↳ Working:
       - → Input: variable / bursty traffic.
       - → Output: fixed, constant rate.
       - → If bucket is full then new packets discarded.
     - ↳ Smoothens bursty traffic, controls congestion.
     - ↳ Algorithm
       1. Initialize counter = n bits/bytes per clock tick.
       2. Send packets while counter ≥ packet size then decrement counter.
       3. Reset counter each clock tick.

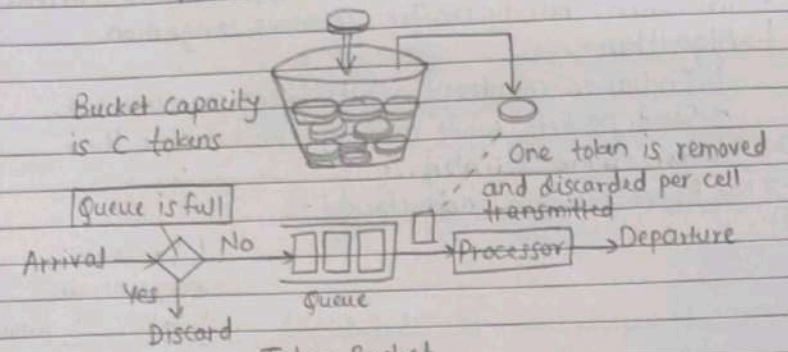


Leaky Bucket Algorithm

   - Token Bucket Algorithm:
     - ↳ Credits an idle host with tokens that can be used later to send packets.
     - ↳ Working:
       - → Tokens generated at rate r tokens/sec.
       - → Bucket capacity = C tokens.
       - → 1 token = permission to send 1 packet (or byte)
       - → If idle then tokens accumulate up to capacity.
       - → If bucket empty then packets must wait (cannot send)
       - → Max packets in time t: $rxt + C$
       - → Max avg rate: $\frac{rxt+C}{t}$ packets/sec.
     - ↳ Allows bursty traffic while controlling average rate.
   - Combined Leaky + Token Bucket:
     - ↳ Token Bucket first allows bursts if tokens available.
     - ↳ Leaky Bucket next shapes final output to steady rate.
     - ↳ Provides both flexibility and smoothness.

(Prof. Tejal Rane)

Tokens added at the rate of r per second, tokens are discarded if bucket is full



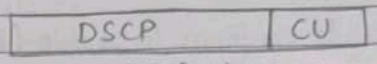
Token Bucket

3. **Resource Reservation**: Allocates resources (bandwidth, buffer, CPU time) beforehand for specific flows.
Ex.: Integrated Services (IntServ) model.

4. **Admission Control**: Router/switch decides whether to accept a new flow by checking available resources and commitments.

8 **Differentiated Services (DiffServ)**:
- DiffServ was introduced by IETF to overcome the shortcoming of Integrated Services (IntServ).
- It is a class-based QoS model designed for IP.
- Two key changes were made:
  1. Processing at the edge, not core: Core routers need not store per-flow information.
  2. Per-class service instead of per-flow: Packets are routed based on class of service, solving service-type limitations.

- **DS field**:
  ↳ Each packet in DiffServ has a DS (Differentiated Services) field, set at the network boundary (by host or edge router).
  ↳ Replaces:
    → TOS field in IPv4.
    → Class field in IPv6



DS field

  ↳ Structure:
    → **DSCP (6 bits)**: Differentiated Services Code Point → specifies Per-Hop Behaviour (PHB)
    → **CU (2 bits)**: Currently unused.
  ↳ Routers use DSCP as an index to look up the forwarding behaviour in their table.

- **Per-Hop Behaviors (PHB)**:
DiffServ defines how each router/node treats packets:
  1. **Default PHB (DE)**:
     Same as best-effort delivery (compatible with TOS).
  2. **Expedited Forwarding (EF)**:
     ↳ Provides low loss, low latency, and guaranteed bandwidth.
     ↳ Works like a virtual leased line (good for voice/video).
  3. **Assured Forwarding (AF)**:
     ↳ Delivers packets with high assurance if within profile.
     ↳ If profile is exceeded, excess packets may be discarded.

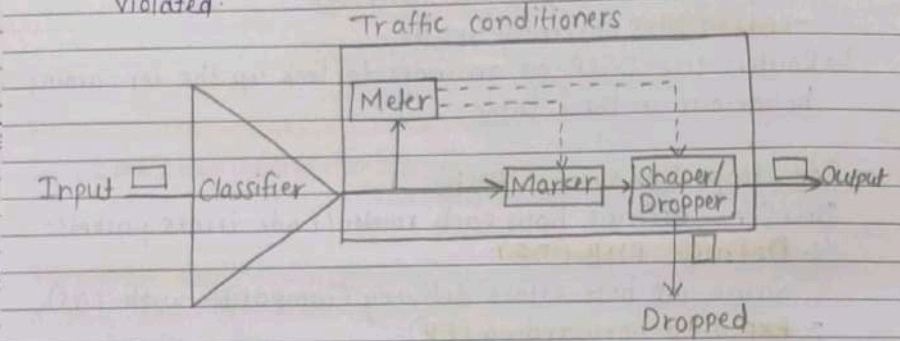
- **Traffic Conditioner**: Used at the network edge to enforce traffic profiles.
  1. **Meter**:
     ↳ Checks if incoming flow matches the traffic profile (eg. token bucket).
     ↳ Sends results to other components.
  2. **Marker**:
     ↳ Assigns DSCP value to packets.

(Prof. Tejal Rane)

↳ Can down-mark (reduce priority) if flow exceeds profile.
↳ Cannot up-mark a packet.

3. Shaper:
↳ Reshapes traffic to conform to profile (buffers and delays packets if needed).

4. Dropper
↳ Works like a shaper without buffer.
↳ Directly discards packets if profile is sever violated.



Traffic-conditioners

§ TCP and UDP for Wireless Networks:

Wireless networks have unique characteristics such as high bit error rate, variable latency, and temporary disconnections. These affect how transport layer protocols like TCP and UDP perform when used over wireless links.

– TCP over Wireless Networks:

a. **Reliable but Sensitive to Errors:**
TCP is a reliable connection-oriented protocol. It assumes that packet loss is due to congestion. However, in wireless networks, losses mostly occur due to signal fading, interference, or mobility, not congestion.
This makes TCP reduces its transmission rate unnecessarily, decreasing performance.

b. **Congestion Control Problem:**
TCP uses mechanisms like slow start, congestion avoidance, and fast retransmit, which work well in wired networks. In wireless networks, these mechanisms misinterpret random losses as congestion, leading to reduced throughput.

c. **Solutions/Enhancements:**
i. Split TCP Connection: Separate wired and wireless parts to isolate wireless errors.
ii. Snoop protocol: Base station caches packets and performs local retransmissions.
iii. TCP Westwood: Estimates bandwidth to handle wireless losses more efficiently.
iv. Link-layer retransmission: Reduces packet loss seen by TCP.

– UDP over Wireless Networks:

a. **Connectionless and Lightweight:**
UDP is a connectionless and unreliable protocol. It does not perform error recovery, retransmission, or congestion control.
Hence UDP performs better than TCP in wireless environments where packet loss is common.

b. **Used for Real-Time Applications:**
Since UDP has low overhead and delay, it is ideal for applications such as voice over IP (VoIP), video streaming, and online gaming, where occasional loss is acceptable but delay is not.

c. **No Congestion Control:**
UDP does not reduce its sending rate during wireless losses. While this ensures continuity, it can lead to network congestion if not managed properly by the application layer.

(Prof. Tejal Rane)