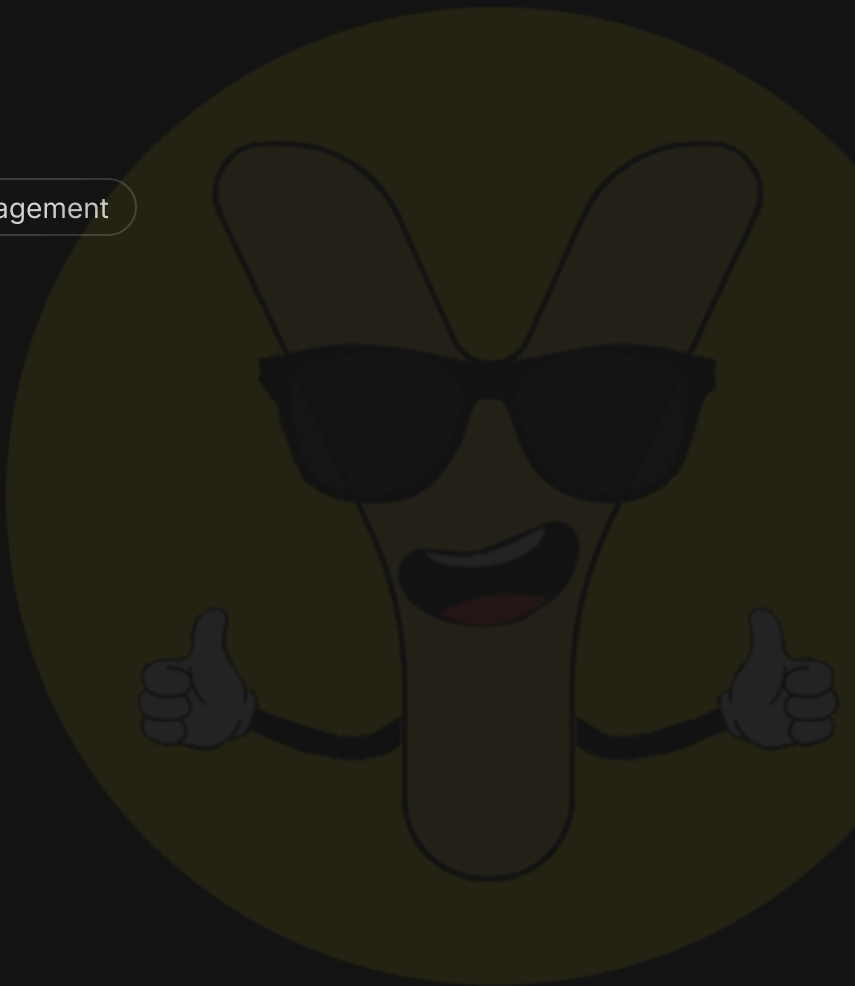
AKTU · 3RD YEAR · SEMESTER 6 · BCS603

Computer Networks

Unit 4 — Transport Layer

- Process-to-process delivery
- Transport layer protocols (UDP and TCP)
- Multiplexing
- Connection management
- Flow control and retransmission
- Window management
- TCP Congestion control
- Quality of service



DECODED BY

Chirag Jain

Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

Process-to-Process Delivery

PROCESS-TO-PROCESS DELIVERY

The mechanism by which the transport layer delivers data sent by a specific running application on the source host to the corresponding running application on the destination host.

- Network layer handles host-to-host delivery using IP addresses.
- Transport layer distinguishes individual applications on a host using 16-bit Port Numbers.
- A Socket Address is defined as the combination of an IP address and a Port Number: $\text{Socket} = (\text{IP Address}, \text{Port Number})$.
- A communication channel is uniquely identified by the 5-tuple: (Source IP, Source Port, Dest IP, Dest Port, Protocol).

Port Addressing Hierarchy

Port numbers are 16-bit integers ranging from 0 to 65535, divided into three categories by IANA.

- Well-Known Ports (0 to 1023)
  - Assigned to standard server services (e.g., HTTP: 80, HTTPS: 443, FTP: 21, DNS: 53).
  - Require administrative/root privileges to bind on Unix-like operating systems.
- Registered Ports (1024 to 49151)
  - Not automatically assigned; registered by IANA to avoid duplication by third-party applications (e.g., MySQL: 3306).
- Dynamic / Ephemeral Ports (49152 to 65535)
  - Temporary client-side ports assigned dynamically by the operating system for a session.

User Datagram Protocol (UDP)

UDP is an unreliable, connectionless transport layer protocol defined in RFC 768.

- Connectionless service: Transmits packets immediately without setting up a connection or handshake.
- Unreliable delivery: No acknowledgments, retransmissions, or ordering guarantees; lost packets are dropped.
- No flow or congestion control: Sends data at the rate generated by the application layer.
- Lightweight overhead: UDP fixed header is only 8 bytes, compared to TCP minimum 20 bytes.
- Ideal for real-time systems, voice/video streaming (VoIP), DNS queries, and DHCP.

UDP Datagram Header Format

The UDP header consists of 4 fields of 2 bytes each, totalling 8 bytes (64 bits).

- Source Port (16 bits): Port number of the sending application process (optional in clients).
- Destination Port (16 bits): Port number of the receiving application process (mandatory).
- Total Length (16 bits): Total length of the UDP segment in bytes (Header + Payload); minimum value is 8.
- Checksum (16 bits): 16-bit one's complement sum covering pseudo-header, UDP header, and data (optional in IPv4, mandatory in IPv6).
- Pseudo-header contains Source IP, Destination IP, Protocol number (17), and UDP length to verify network-layer binding.

Transmission Control Protocol (TCP)

TRANSMISSION CONTROL PROTOCOL (TCP)

A connection-oriented, reliable, byte-stream transport protocol defined in RFC 793 that provides end-to-end error recovery and flow regulation.

- Stream-oriented: Application sends bytes as an unstructured stream, grouped by TCP into segments.
- Full-duplex service: Data can flow bidirectionally between both endpoints simultaneously.
- Reliable delivery: Uses sequence numbers, positive acknowledgments, and retransmission timers.
- Flow and Congestion control: Regulates send rate using sliding window and congestion window sizes.

TCP Segment Header Architecture

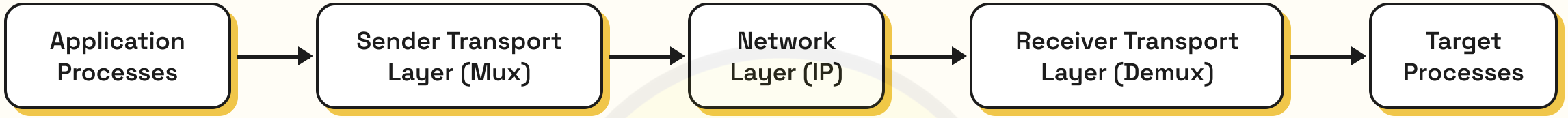
The TCP header contains 20 to 60 bytes, depending on optional fields.

- Sequence Number (32 bits): Byte index of the first data byte in this segment within the byte stream.
- Acknowledgment Number (32 bits): Cumulative byte index expected next from the sender; valid if ACK bit is set.
- Header Length (4 bits): Number of 4-byte words in the header (value 5 to 15, yielding 20 to 60 bytes).
- Control Flags (6 bits): URG (urgent), ACK (valid ack), PSH (push data), RST (reset connection), SYN (synchronize seq), FIN (terminate).
- Receive Window / rwnd (16 bits): Number of bytes the receiver buffer can accept for flow control.
- Checksum (16 bits) and Urgent Pointer (16 bits): Error check and offset to urgent data.

Comparison: TCP vs UDP

Parameter	TCP	UDP
Connection	Connection-oriented (3-way handshake)	Connectionless (No handshake)
Reliability	Reliable (ACK + Retransmission)	Unreliable (Best-effort)
Header Overhead	20 to 60 bytes	Fixed 8 bytes
Ordering	Maintains sequence order	Packets can arrive out of order
Flow & Congestion	Supported (Sliding Window, AIMD)	Not supported
Speed	Slower due to overhead and handshakes	Fast and lightweight
Applications	HTTP, HTTPS, FTP, SSH, SMTP	DNS, DHCP, VoIP, Video Streaming

Multiplexing and Demultiplexing



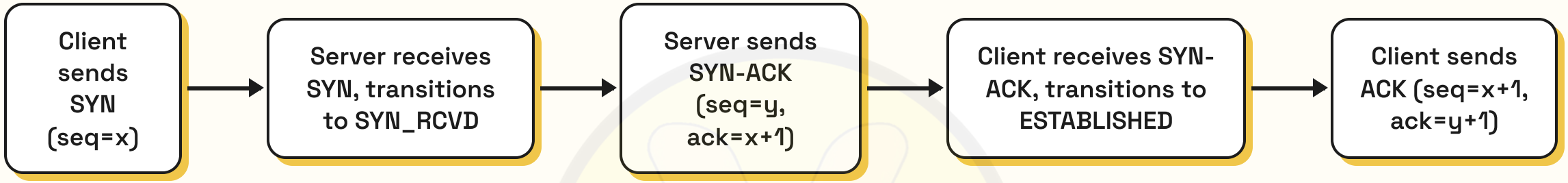
Multiplexing encapsulates process data with port headers; Demultiplexing uses ports to deliver data to correct sockets.

Multiplexing and Demultiplexing Mechanics

Transport layer multiplexes multiple application streams onto a single IP channel and demultiplexes them at destination.

- Multiplexing at Sender
  - Gathers data from multiple socket buffers.
  - Appends transport header containing Source Port and Destination Port to form segments.
- Connectionless Demultiplexing (UDP)
  - Receiver routes packet to socket based only on Destination Port Number.
  - Two packets with different source IPs but same destination port reach the exact same socket.
- Connection-Oriented Demultiplexing (TCP)
  - Receiver routes segment using the 4-tuple: (Source IP, Source Port, Dest IP, Dest Port).
  - Allows multiple concurrent connections to the same server port (e.g., port 80) into separate dedicated sockets.

TCP Connection Establishment: 3-Way Handshake



Three-way handshake synchronizes Initial Sequence Numbers (ISNs) and allocates buffer resources.

TCP Connection Termination

TCP terminates connections independently in both directions using a 4-step handshake.

- Step 1 (Client FIN): Client sends FIN segment to close transmission from its side; enters FIN_WAIT_1.
- Step 2 (Server ACK): Server sends ACK confirming receipt; client enters FIN_WAIT_2. Server enters CLOSE_WAIT state.
- Step 3 (Server FIN): Server finishes sending pending data, then transmits FIN segment; enters LAST_ACK.
- Step 4 (Client ACK): Client sends ACK for server's FIN, enters TIME_WAIT state, then closes after timer expires.
- TIME_WAIT purpose: Held for $2 \times \mathrm{MSL}$ (Maximum Segment Lifetime, typical 2-4 min) to ensure final ACK was received and old duplicate packets expire in the network.

Flow Control: Sliding Window

Flow control prevents a fast sender from overwhelming a slow receiver's buffer space.

- Receiver Advertised Window ($rwnd$): Receiver sends available buffer capacity inside the window field of every ACK.
- Usable Window Formula: $\text{Usable Window} = rwnd - (\text{LastByteSent} - \text{LastByteAcked})$.
- Byte-oriented window: Window slides forward as cumulative acknowledgments arrive from receiver.
- Zero Window Notification: If receiver buffer fills completely, it advertises $rwnd = 0$, pausing the sender.
- Persistence Timer: When $rwnd = 0$, sender starts persistence timer to periodically send 1-byte probe packets, preventing deadlocks if window updates are lost.

Retransmission and RTT Estimation

TCP ensures reliability by retransmitting dropped segments using adaptive timeout calculation.

- Retransmission Timeout (RTO): Set based on dynamically estimated Round Trip Time (RTT).
- Smoothed RTT (SRTT): $SRTT = (1 - \alpha) \times SRTT + \alpha \times RTT_{\text{sample}}$, where standard $\alpha = 0.125$.
- RTT Variation (RTTVAR): $RTTVAR = (1 - \beta) \times RTTVAR + \beta \times |SRTT - RTT_{\text{sample}}|$, where $\beta = 0.25$.
- Timeout Computation: $RTO = SRTT + 4 \times RTTVAR$.
- Karn's Algorithm: Do not update RTT estimates using retransmitted segments; double RTO after every timeout (exponential backoff).

Window Management & Silly Window Syndrome

Silly Window Syndrome (SWS) occurs when data is exchanged in extremely small segments, wasting bandwidth.

- Receiver-Side SWS
  - Caused by receiver advertising tiny window openings (e.g., 1 byte) as soon as buffer space frees up.
  - Clark's Solution: Prevent receiver from advertising window updates until buffer can accept either Maximum Segment Size (MSS) or half buffer capacity.
- Sender-Side SWS
  - Caused by application generating data slowly (1 byte at a time) and sending tiny packets immediately.
  - Nagle's Algorithm: If unacknowledged data is outstanding, buffer incoming data until a full MSS is ready to transmit or ACK is received.

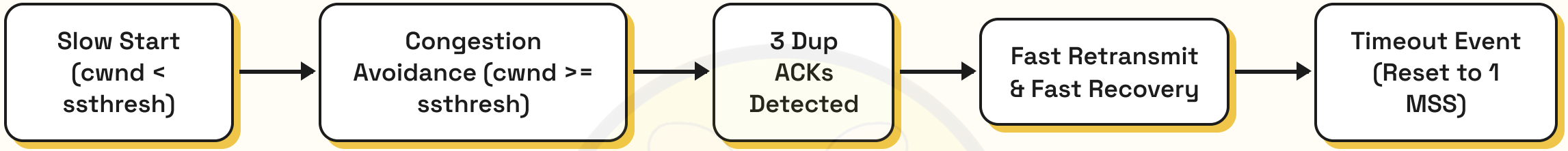
TCP Congestion Control: Fundamentals

CONGESTION CONTROL

Algorithms used by the sender to infer and adapt to network resource exhaustion (router queue overflows) between endpoints.

- Difference: Flow control protects the receiving host; Congestion control protects intermediate network routers.
- Congestion Window ($cwnd$): Sender-side state variable reflecting network capacity.
- Effective Transmission Window: Sender limits in-flight bytes to $W = \min(cwnd, rwnd)$.
- Loss indicates congestion: Packet loss (timeout or duplicate ACKs) is treated as evidence of network buffer drops.

TCP Congestion Control States



Transition model of standard TCP congestion control (Tahoe / Reno).

TCP Congestion Control Algorithms

Standard TCP implements four intertwined congestion-handling algorithms.

- Slow Start Phase
  - Starts with $cwnd = 1\ \text{MSS}$. For every ACK received, $cwnd = cwnd + 1\ \text{MSS}$ (doubles every RTT).
  - Continues until $cwnd \geq ssthresh$ (slow start threshold) or loss occurs.
- Congestion Avoidance (Additive Increase)
  - When $cwnd \geq ssthresh$, $cwnd$ increments by $1\ \text{MSS}$ per RTT: $cwnd = cwnd + \frac{\text{MSS}}{cwnd}$ per ACK.
  - Linear growth probes for maximum available bandwidth safely.
- Multiplicative Decrease (Loss Response)
  - On Timeout: Set $ssthresh = \frac{cwnd}{2}$, reset $cwnd = 1\ \text{MSS}$, enter Slow Start.
  - On 3 Duplicate ACKs (TCP Reno): Set $ssthresh = \frac{cwnd}{2}$, $cwnd = ssthresh + 3\ \text{MSS}$, enter Fast Recovery without resetting to 1.

Worked Example: Congestion Window Evolution

A TCP Reno connection has ssthresh = 16 MSS. Starting at cwnd = 1 MSS in Slow Start, what is the value of cwnd during the first 6 RTTs assuming no packet drops?

1. RTT 1: cwnd starts at 1 MSS. On 1 ACK, cwnd doubles to 2 MSS.
2. RTT 2: cwnd = 2 MSS. After 2 ACKs, cwnd doubles to 4 MSS.
3. RTT 3: cwnd = 4 MSS. After 4 ACKs, cwnd doubles to 8 MSS.
4. RTT 4: cwnd = 8 MSS. After 8 ACKs, cwnd doubles to 16 MSS (equals ssthresh).
5. RTT 5: cwnd reaches ssthresh (16 MSS). Enters Congestion Avoidance (linear growth). cwnd increases by 1 MSS per RTT: cwnd = 17 MSS.
6. RTT 6: Remains in Congestion Avoidance. cwnd increases by 1 MSS: cwnd = 18 MSS.

At RTT 6, cwnd = 18 MSS (transitioned from exponential growth to linear growth at RTT 4).

Quality of Service (QoS) Parameters

QoS refers to the capability of a network to provide better service and guarantees to selected network traffic.

- Throughput: Rate at which data is successfully delivered across the channel (bits/second).
- Delay (Latency): Time taken by a packet to travel from source to destination (transmission + propagation + queuing delay).
- Jitter: Variation in packet arrival delays. High jitter degrades real-time applications like voice/video calls.
- Packet Loss Rate: Fraction of transmitted packets dropped due to buffer overflow or transmission errors.
- Reliability: Guarantee that the delivered data stream is error-free, intact, and delivered in correct sequence.

Traffic Shaping: Leaky Bucket vs Token Bucket

Criteria	Leaky Bucket	Token Bucket
Output Rate	Constant, rigid output rate	Variable output rate up to token capacity
Burst Handling	Discards or buffers bursts; smooths strictly	Permits bursts if sufficient tokens exist
Token Generation	No tokens; packets leak out directly	Tokens generated periodically at rate r
Buffer Storage	Holds packets directly in queue	Stores tokens (up to max capacity b)
Packet Loss	Occurs when packet queue overflows	No loss of packets unless data buffer overflows
Implementation	Strict packet queueing model	Counter-based (simple token accumulation)

QoS Architectures: IntServ vs DiffServ

Two primary architectural frameworks developed by IETF to provide end-to-end QoS guarantees.

- Integrated Services (IntServ / RFC 1633)
  - Per-flow resource reservation model using RSVP (Resource Reservation Protocol).
  - Every intermediate router maintains connection state for every individual flow.
  - Provides strict quantitative guarantees (guaranteed service, controlled load).
  - Suffers from severe scalability limitations in large core internet backbones.
- Differentiated Services (DiffServ / RFC 2475)
  - Class-based traffic prioritization model without per-flow state storage.
  - Packets marked at edge routers using 6-bit DSCP (Differentiated Services Code Point) in IPv4 TOS / IPv6 TC byte.
  - Core routers apply Per-Hop Behaviors (PHB): Expedited Forwarding (EF) or Assured Forwarding (AF).
  - Highly scalable and widely deployed across enterprise and carrier backbones.

Exam Corner

- **Q1** Differentiate between TCP and UDP with respect to header formats, reliability, and suitable applications.
- **Q2** Explain the TCP three-way handshake for connection establishment and four-way termination with clear timing diagrams.
- **Q3** Define Silly Window Syndrome. Explain the sender-side (Nagle's algorithm) and receiver-side (Clark's solution) remedies.
- **Q4** Explain TCP congestion control mechanism with Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery phases.
- **Q5** Derive and explain how TCP estimates Round Trip Time (RTT) and calculates Retransmission Timeout (RTO).
- **Q6** Compare and contrast the Leaky Bucket and Token Bucket algorithms for traffic shaping with diagrams.
- **Q7** Write short notes on: (a) Port Numbers and Sockets, (b) IntServ vs DiffServ QoS models.

Unit 4 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 5 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

