

# UNIT 5

## The Transport Layer

(7 Hours) — Computer Networking (CACS303)

---

### 5.1 Functions of the Transport Layer

The Transport Layer sits between the Application Layer and the Network Layer in both the OSI and TCP/IP models. Its central role is to provide end-to-end, process-to-process delivery of an entire message, taking a message from an application, breaking it into transmittable segments, and handing them to the network layer for transport across possibly many intermediate routers, then reassembling them correctly at the destination.

- **Process-to-process delivery:** while the network layer delivers packets host-to-host, the transport layer extends delivery to the correct process (application) running on that host, using port numbers.
- **Segmentation and reassembly:** breaks a long message from the application into smaller segments, each numbered so that they can be reassembled correctly at the receiver, regardless of the order in which they arrive.
- **Connection control:** may be connectionless (each segment travels independently, e.g. UDP) or connection-oriented (a logical connection is established before data transfer, e.g. TCP).
- **Flow control:** ensures a fast sender does not overwhelm a slow receiver, typically using a sliding-window mechanism, done end-to-end rather than link by link.
- **Error control:** ensures the entire message arrives at the receiving transport layer without error (undamaged, without duplication, without loss), typically achieved through retransmission.
- **Congestion control:** prevents too much data from being injected into the network, which could cause routers or links to become overloaded.

#### Key Takeaways — 5.1

- ✓ The Transport Layer provides process-to-process (end-to-end) delivery, unlike the Network Layer's host-to-host delivery.
- ✓ It performs segmentation/reassembly, connection control, flow control, error control, and congestion control.
- ✓ It can be connectionless (UDP) or connection-oriented (TCP) depending on the protocol.

## 5.2 Elements of Transport Protocols

Although transport protocols resemble data link layer protocols in many respects (both must handle error and flow control, for example), important differences arise because the transport layer operates over a network layer that may lose, reorder, or delay packets unpredictably. The following elements are common to most transport protocols.

### 5.2.1 Addressing

Transport-layer addressing uses port numbers (also called Transport Service Access Points, or TSAPs) to identify the specific application process at each end of a connection. A transport-layer address is therefore the combination of an IP address and a port number, together called a socket. A connection is uniquely identified by the pair of sockets at each end: (source IP, source port, destination IP, destination port).

- Well-known ports (0–1023): reserved for standard services, e.g. HTTP (80), FTP (21), DNS (53).
- Registered ports (1024–49151): assigned to specific applications by IANA but not as tightly controlled.
- Dynamic/ephemeral ports (49152–65535): temporarily assigned by the operating system to client processes for the duration of a connection.

### 5.2.2 Establishing and Releasing a Connection

For connection-oriented transport protocols such as TCP, a connection must be explicitly opened before data can flow and explicitly closed afterward, so that both sides agree on initial sequence numbers and resources are properly allocated and released.

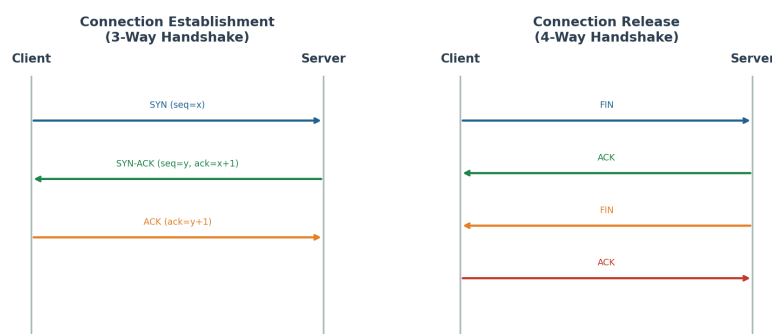


Fig. 5.1 — Connection establishment via the three-way handshake, and connection release via the four-way handshake.

Establishment (three-way handshake): the client sends a SYN segment proposing an initial sequence number; the server replies with a SYN-ACK segment acknowledging the client's sequence number and proposing its own; the client completes the handshake with an ACK

segment. Only after this exchange do both sides consider the connection open.

Release (four-way handshake): because a TCP connection is full-duplex, each direction must be closed independently. One side sends a FIN when it has no more data to send; the other side acknowledges it. The connection is not fully closed until both directions have exchanged a FIN and its corresponding ACK.

*Note: A three-way handshake (rather than a simple two-message exchange) is necessary because both the network and transport layers may duplicate or delay old segments; the extra step lets both sides confirm that stale segments from a previous connection are not mistaken for a new one.*

### 5.2.3 Flow Control and Buffering

Flow control prevents a fast sender from overwhelming a slow receiver's ability to process incoming data. Most transport protocols use a sliding-window mechanism, in which the receiver advertises how much buffer space it currently has available (the receive window), and the sender limits the amount of unacknowledged data in flight to that window size. Both sender and receiver typically maintain buffers to temporarily hold data — the sender's buffer holds data that has been sent but not yet acknowledged (in case retransmission is needed), while the receiver's buffer holds data that has arrived but not yet been consumed by the application.

### 5.2.4 Error Control

Error control ensures that data is delivered reliably: without loss, duplication, or corruption. Transport protocols typically use sequence numbers, checksums, acknowledgments (ACKs), and timers with retransmission to detect and recover from lost or corrupted segments. If an ACK is not received within a timeout period, the sender assumes the segment was lost and retransmits it.

### 5.2.5 Multiplexing and Demultiplexing

Because a single host may run many network applications simultaneously, but typically has only one (or a few) network-layer connections, the transport layer must combine data from multiple application processes for transmission over the network (multiplexing) and, at the receiving end, deliver incoming segments to the correct process (demultiplexing) — both are accomplished using port numbers.

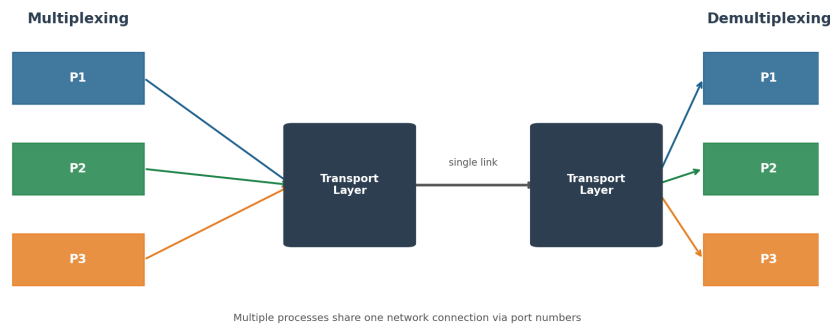


Fig. 5.2 — Multiplexing at the sender combines multiple processes onto one link; demultiplexing at the receiver delivers segments to the correct process using port numbers.

## 5.2.6 Crash Recovery

If a transport-layer host crashes and later restarts, the protocol must define how the connection state is recovered so that data is neither permanently lost nor duplicated. This is difficult because, after a crash, a host has no memory of the sequence numbers or acknowledgments it had negotiated. Approaches typically depend on whether the client or the server crashes, and to what extent the application layer (rather than the transport layer alone) must take responsibility for detecting and recovering lost work, since no combination of client and server recovery strategies can guarantee correct recovery in every case.

### Key Takeaways — 5.2

- ✓ Ports (combined with IP addresses into sockets) provide transport-layer addressing for processes.
- ✓ Connections are established with a three-way handshake and released with a four-way handshake in TCP.
- ✓ Flow control uses sliding windows and buffering; error control uses sequence numbers, checksums, ACKs, and timers.
- ✓ Multiplexing/demultiplexing lets many application processes share the network layer via port numbers.
- ✓ Crash recovery is inherently difficult since a restarted host loses its prior connection state.

## 5.3 User Datagram Protocol (UDP)

UDP is a simple, connectionless, unreliable transport-layer protocol defined in RFC 768. It adds only minimal functionality on top of the network layer — essentially just process-to-process addressing and an optional checksum — and does not provide flow control, error recovery, or guaranteed delivery.

### 5.3.1 User Datagram

The unit of data at the UDP layer is called a user datagram, consisting of an 8-byte header followed by the data.

| Field            | Size    | Description                                                                   |
|------------------|---------|-------------------------------------------------------------------------------|
| Source Port      | 16 bits | Port number of the sending process (optional; can be all zeros if not needed) |
| Destination Port | 16 bits | Port number of the receiving process                                          |
| Length           | 16 bits | Total length of the UDP datagram (header + data), in bytes                    |
| Checksum         | 16 bits | Optional error-detection value covering header, data, and a pseudo-header     |

### 5.3.2 UDP Operations

- **Connectionless service:** each user datagram is an independent unit, sent without any handshake; datagrams for the same destination may take different paths and can arrive out of order, be duplicated, or be lost entirely.
- **No flow or error control:** UDP does not manage the rate of data flow and will not retransmit lost datagrams; the checksum, if used, only detects corruption — it does not trigger any automatic recovery.
- **Encapsulation and decapsulation:** to send data, UDP receives it from the application, encapsulates it in a user datagram, and passes it to the network layer; the reverse (decapsulation) happens at the receiver.
- **Queuing:** at both sender and receiver, UDP maintains queues associated with each port so that multiple processes can send/receive independently.

### 5.3.3 Uses of UDP

- Applications that need a lightweight transaction-oriented protocol, such as DNS queries.
- Applications with their own, more specialized reliability or flow-control mechanisms built at the application layer (e.g. TFTP).

- Real-time multimedia applications (voice/video over IP, streaming), where occasional loss is tolerable but retransmission delay is not.
- Multicast and broadcast applications, since UDP is well suited to one-to-many communication.
- Simple request-response protocols such as SNMP, and routing update protocols such as RIP.

### 5.3.4 Remote Procedure Call (RPC)

RPC is a protocol/mechanism that allows a program on one machine to call a procedure (function) that executes on another machine, as if it were a local call, hiding the underlying network communication from the programmer. Because RPC exchanges are often short request-response transactions, they are frequently built on top of UDP for efficiency, with the application itself handling retransmission of a request if no response arrives within a timeout.

- The client makes a normal-looking procedure call; a client stub packages (marshals) the parameters into a message.
- The message is transmitted (typically via UDP) to the remote server.
- A server stub on the remote machine unpacks (unmarshals) the parameters, calls the actual procedure, and sends the result back the same way.

#### Key Takeaways — 5.3

- ✓ UDP is connectionless and unreliable, with only an 8-byte header (source port, destination port, length, checksum).
- ✓ It provides no flow control or automatic retransmission, making it lightweight but not guaranteed.
- ✓ UDP suits DNS, real-time multimedia, multicast/broadcast, and simple request-response protocols like RPC and SNMP.

## 5.4 Principles of Reliable Data Transfer

Building a protocol that reliably transfers data over an inherently unreliable channel (one that may corrupt, lose, or reorder bits) is one of the fundamental problems in networking. Reliable data transfer (RDT) protocols are developed incrementally, adding mechanisms to handle progressively more difficult channel conditions.

### 5.4.1 Building a Reliable Data Transfer Protocol

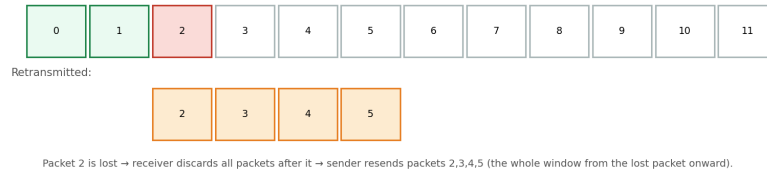
- Reliable transfer over a perfectly reliable channel: the simplest case — no bit errors, no loss — so the sender simply sends and the receiver simply receives, with no feedback needed.
- Reliable transfer over a channel with bit errors: requires error detection (via checksums), receiver feedback (ACK/NAK messages), and retransmission of corrupted packets. A stop-and-wait protocol is used: the sender sends one packet and waits for an ACK/NAK before sending the next.
- Handling duplicate packets: since ACK/NAK messages can themselves be corrupted or lost, sequence numbers must be added to packets so that the receiver can detect whether a newly arrived packet is a retransmission of one it has already received.
- Reliable transfer over a lossy channel: in addition to bit errors, packets (or their ACKs) may be lost entirely. A countdown timer is introduced at the sender: if no ACK arrives within a timeout, the sender assumes loss and retransmits.

*Note: This progression — from a perfect channel, to a channel with only bit errors, to a channel with both bit errors and loss — is the standard way reliable data transfer protocols (RDT 1.0, 2.0, 2.1, 2.2, 3.0) are built up step by step.*

### 5.4.2 Pipelined Reliable Data Transfer Protocols

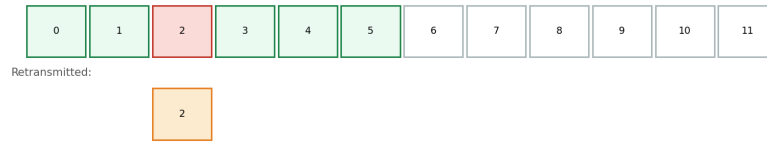
The stop-and-wait approach described above is simple but highly inefficient: the sender is idle while waiting for each acknowledgment, so link utilization can be extremely poor, especially over links with a large bandwidth-delay product. Pipelining solves this by allowing the sender to transmit multiple packets without waiting for each one to be acknowledged individually; the sender maintains a range of permissible, not-yet-acknowledged sequence numbers called a sliding window. Pipelining requires: (1) a larger range of sequence numbers since multiple in-transit packets must be uniquely identified, and (2) buffering at the sender and/or receiver to hold packets that are in flight or have arrived out of order. Two general approaches for handling loss and error recovery in a pipelined protocol are Go-Back-N and Selective Repeat.

#### Go-Back-N (GBN)



Packet 2 is lost → receiver discards all packets after it → sender resends packets 2,3,4,5 (the whole window from the lost packet onward).

#### Selective Repeat (SR)



Packet 2 is lost → receiver buffers out-of-order packets 3,4,5 → sender resends only packet 2.

Fig. 5.3 — Go-Back-N discards and resends the entire window after a lost packet; Selective Repeat buffers out-of-order packets and resends only the lost one.

### 5.4.3 Go-Back-N (GBN)

In Go-Back-N, the sender may have up to N unacknowledged packets in the pipeline at a time (the window size N). The receiver is very simple: it keeps only the next expected sequence number, discards any out-of-order packet (even if correctly received), and sends a cumulative acknowledgment for the highest in-order sequence number received so far. If the sender's timer expires for the oldest unacknowledged packet, it retransmits all packets in the current window, not just the one that was lost — hence the name "go back N."

- Advantage: very simple receiver logic, minimal receiver buffering.
- Disadvantage: a single lost packet can force retransmission of many correctly received packets, wasting bandwidth, especially with large windows.

### 5.4.4 Selective Repeat (SR)

Selective Repeat improves efficiency by having the sender retransmit only the specific packet(s) that are lost or corrupted, not the entire window. This requires the receiver to individually acknowledge each correctly received packet and to buffer out-of-order packets until any gaps are filled, at which point it delivers the in-order run to the application. The sender maintains an individual timer for each unacknowledged packet.

- Advantage: much more efficient use of bandwidth since only lost packets are retransmitted.
- Disadvantage: more complex — requires per-packet timers, larger receiver buffers, and careful handling of the sequence-number range to avoid ambiguity between old and new packets covered by the same window.

| Feature                | Go-Back-N                                 | Selective Repeat                      |
|------------------------|-------------------------------------------|---------------------------------------|
| Receiver buffering     | None (out-of-order packets discarded)     | Buffers out-of-order packets          |
| Acknowledgment type    | Cumulative ACK                            | Individual (selective) ACK per packet |
| Retransmission on loss | Entire window from the lost packet onward | Only the specific lost packet         |
| Timers                 | Single timer (oldest unacked packet)      | Per-packet timer                      |
| Complexity             | Simple                                    | More complex                          |
| Bandwidth efficiency   | Lower (redundant retransmissions)         | Higher                                |

### Key Takeaways — 5.4

- ✓ Reliable data transfer is built incrementally: handling bit errors first, then packet loss, using ACKs/NAKs, sequence numbers, and timers.
- ✓ Pipelining (sending multiple packets before waiting for ACKs) is essential for good link utilization; it uses a sliding window.
- ✓ Go-Back-N resends the whole window after a loss (simple receiver); Selective Repeat resends only the lost packet (efficient but complex).

## 5.5 Transmission Control Protocol (TCP)

TCP is the Internet's primary connection-oriented, reliable, byte-stream transport protocol, defined in RFC 793 (and updated by later RFCs). It combines many of the mechanisms described above — connection establishment/release, flow control, error control, and congestion control — into a single, widely deployed protocol.

### 5.5.1 TCP Services

- Process-to-process communication: using port numbers to deliver data to the correct application.
- Stream delivery service: TCP treats data from the application as a continuous stream of bytes rather than discrete messages, and delivers that same stream, in order, to the receiving application.
- Full-duplex service: data can flow in both directions simultaneously over a single connection.
- Connection-oriented service: a logical connection is established (via the three-way handshake) before data transfer and properly released afterward.
- Reliability: guarantees that all data sent is received correctly, in order, and without duplication, using sequence numbers, acknowledgments, checksums, and retransmission.

### 5.5.2 TCP Features

- Numbering system: every byte of data is assigned a sequence number; segments carry the sequence number of their first byte, allowing correct reassembly and detection of missing data.
- Sliding-window flow control: the receiver advertises a window size indicating how many additional bytes it can currently accept, and the sender limits unacknowledged, in-flight data accordingly.
- Cumulative acknowledgment with selective ACK (SACK) option: the basic acknowledgment number field is cumulative, but the optional SACK extension allows the receiver to also report specific non-contiguous blocks of data it has received.
- Error control: uses checksums, sequence numbers, acknowledgments, and retransmission timers to detect and recover from corrupted, lost, or duplicated segments.
- Congestion control: dynamically adjusts the sending rate based on perceived network congestion (covered in Section 5.6).

### 5.5.3 TCP Segment Header

A TCP segment consists of a header (20 bytes minimum, up to 60 bytes with options) followed by application data.

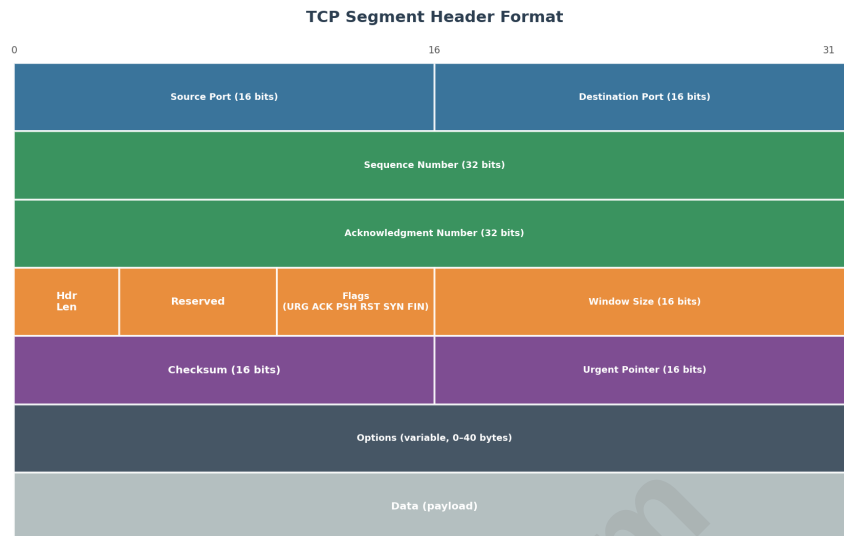


Fig. 5.4 — The TCP segment header format, showing all standard fields.

| Field                     | Size             | Purpose                                                                                |
|---------------------------|------------------|----------------------------------------------------------------------------------------|
| Source / Destination Port | 16 + 16 bits     | Identify the sending and receiving application processes                               |
| Sequence Number           | 32 bits          | Sequence number of the first byte of data in this segment                              |
| Acknowledgment Number     | 32 bits          | Next byte number the receiver expects (cumulative ACK)                                 |
| Header Length (HLEN)      | 4 bits           | Length of the TCP header in 32-bit words (accounts for options)                        |
| Flags (control bits)      | 6 (or more) bits | URG, ACK, PSH, RST, SYN, FIN — control connection state and segment handling           |
| Window Size               | 16 bits          | Number of bytes the sender of this segment is willing to receive (flow control)        |
| Checksum                  | 16 bits          | Error-detection value covering header, data, and a pseudo-header                       |
| Urgent Pointer            | 16 bits          | Valid only if URG flag is set; points to urgent data within the segment                |
| Options                   | 0–40 bytes       | Optional features such as Maximum Segment Size (MSS), Window Scaling, SACK, Timestamps |

- URG: urgent pointer field is valid.
- ACK: acknowledgment number field is valid (set on almost all segments after the initial SYN).

- PSH: requests that data be pushed to the receiving application immediately rather than buffered.
- RST: resets the connection, typically due to an error.
- SYN: synchronizes sequence numbers; used to initiate a connection.
- FIN: indicates the sender has finished sending data and wishes to terminate the connection.

### **Key Takeaways — 5.5**

- ✓ TCP provides connection-oriented, reliable, full-duplex, byte-stream delivery between processes.
- ✓ Its numbering system, sliding-window flow control, cumulative/selective acknowledgment, and retransmission timers together provide reliability.
- ✓ The TCP header carries port numbers, sequence/acknowledgment numbers, control flags, window size, checksum, and options.

bcalink.com

## 5.6 Principles of Congestion Control

Congestion occurs when the demand placed on network resources (routers, links) exceeds their capacity, leading to increased delay and packet loss. Unlike flow control, which protects a single slow receiver, congestion control protects the network as a whole from being overwhelmed by the combined traffic of many senders.

### 5.6.1 Causes and Costs of Congestion

- Too many senders pushing data into the network faster than links or routers can forward it, causing router queues (buffers) to fill and overflow.
- Packets that are dropped due to full buffers must be retransmitted, which adds even more load to an already congested network — a potentially destructive feedback loop.
- Increased queuing delay as buffers fill, degrading performance for all flows sharing the congested link.

### 5.6.2 Approaches to Congestion Control

- End-to-end congestion control: the network layer provides no explicit feedback about congestion; end systems (e.g. TCP senders) must infer congestion indirectly, typically from observed packet loss or increased delay.
- Network-assisted congestion control: routers provide explicit feedback to the end systems about congestion in the network, for example through Explicit Congestion Notification (ECN) bits marked in packet headers.

### 5.6.3 TCP Congestion Control (AIMD)

TCP implements end-to-end congestion control using a congestion window (cwnd) that limits how much unacknowledged data the sender may have in transit, in addition to the receiver's flow-control window. The overall strategy follows Additive Increase, Multiplicative Decrease (AIMD): increase the sending rate slowly while no congestion is detected, but cut it sharply as soon as congestion (packet loss) is detected. TCP congestion control operates in distinct phases:

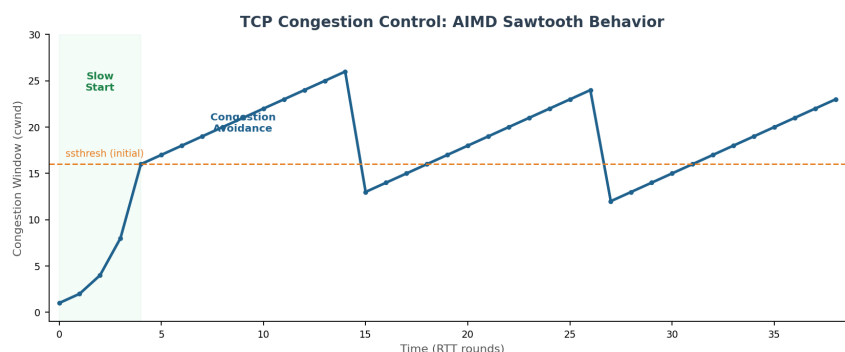


Fig. 5.5 — The characteristic “sawtooth” pattern of TCP’s congestion window over time, showing slow start, congestion avoidance, and multiplicative decrease after loss.

- Slow start: cwnd begins small (e.g. 1 MSS) and doubles every round-trip time (exponential growth) until it reaches a threshold called ssthresh, or until a loss event occurs.
- Congestion avoidance: once cwnd exceeds ssthresh, growth becomes linear — cwnd increases by roughly one MSS per round-trip time (additive increase) — to probe for available bandwidth more cautiously.
- Multiplicative decrease: when a loss event is detected (via timeout or duplicate ACKs), ssthresh is set to half the current cwnd, and cwnd is reduced sharply — to 1 MSS after a timeout (re-entering slow start), or to the new ssthresh after fast recovery from triple duplicate ACKs.
- Fast retransmit / fast recovery: upon receiving three duplicate ACKs (strongly suggesting a single lost segment rather than serious congestion), the sender retransmits the missing segment immediately without waiting for a timeout, and reduces cwnd less drastically than after a full timeout.

*Note: The resulting graph of cwnd over time has a characteristic “sawtooth” shape: it climbs (first exponentially, then linearly) until a loss is detected, then drops sharply, and the cycle repeats — this behavior is what gives AIMD its name and its self-correcting, fairness-promoting property among competing flows.*

### Key Takeaways — 5.6

- ✓ Congestion control protects the network from being overwhelmed by combined traffic from many senders, distinct from receiver-focused flow control.
- ✓ TCP uses AIMD: slow start (exponential growth) followed by congestion avoidance (linear growth), with a sharp multiplicative decrease after loss.
- ✓ Fast retransmit/fast recovery reacts to triple duplicate ACKs more gently than a full timeout, avoiding an unnecessary return to slow start.

---

— End of Unit 5 —