Module 6
Transport Layer
UDP & TCP

BECE401L

Outline

Transport Layer:

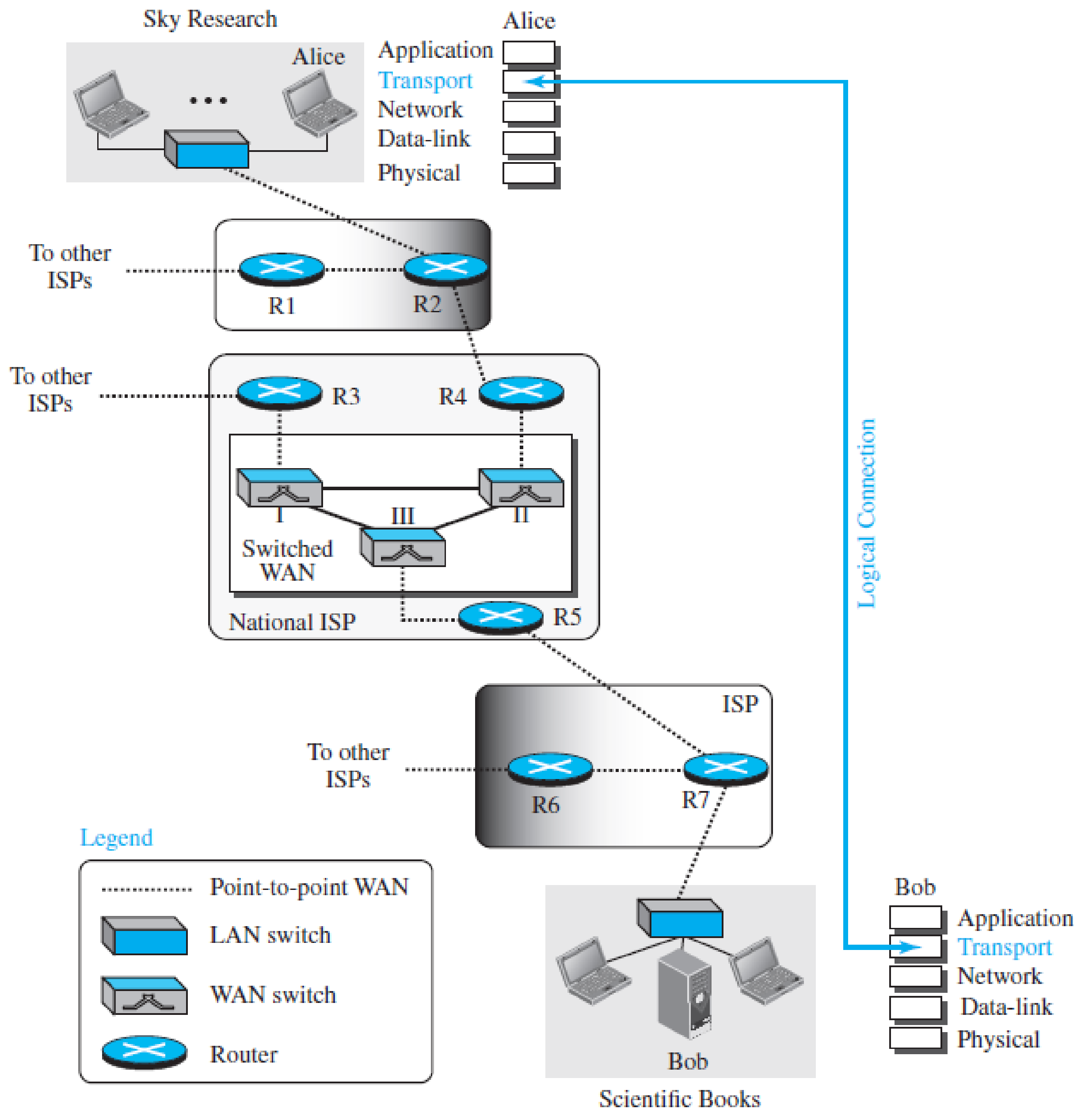
- Services
  - Process to Process Communication
  - Addressing of Port Numbers
  - ICANN Ranges
  - Encapsulation & Decapsulation
  - Flow, Error & Congestion Control
  - Multiplexing & Demultiplexing
- Connectionless & Connection Oriented Protocols

Transport Layer

Provides a process-to-process communication.

Services:

- Process to Process Communication
- Addressing of Port Numbers
- ICANN Ranges
- Encapsulation & Decapsulation
- Flow & Error Control
- Congestion Control

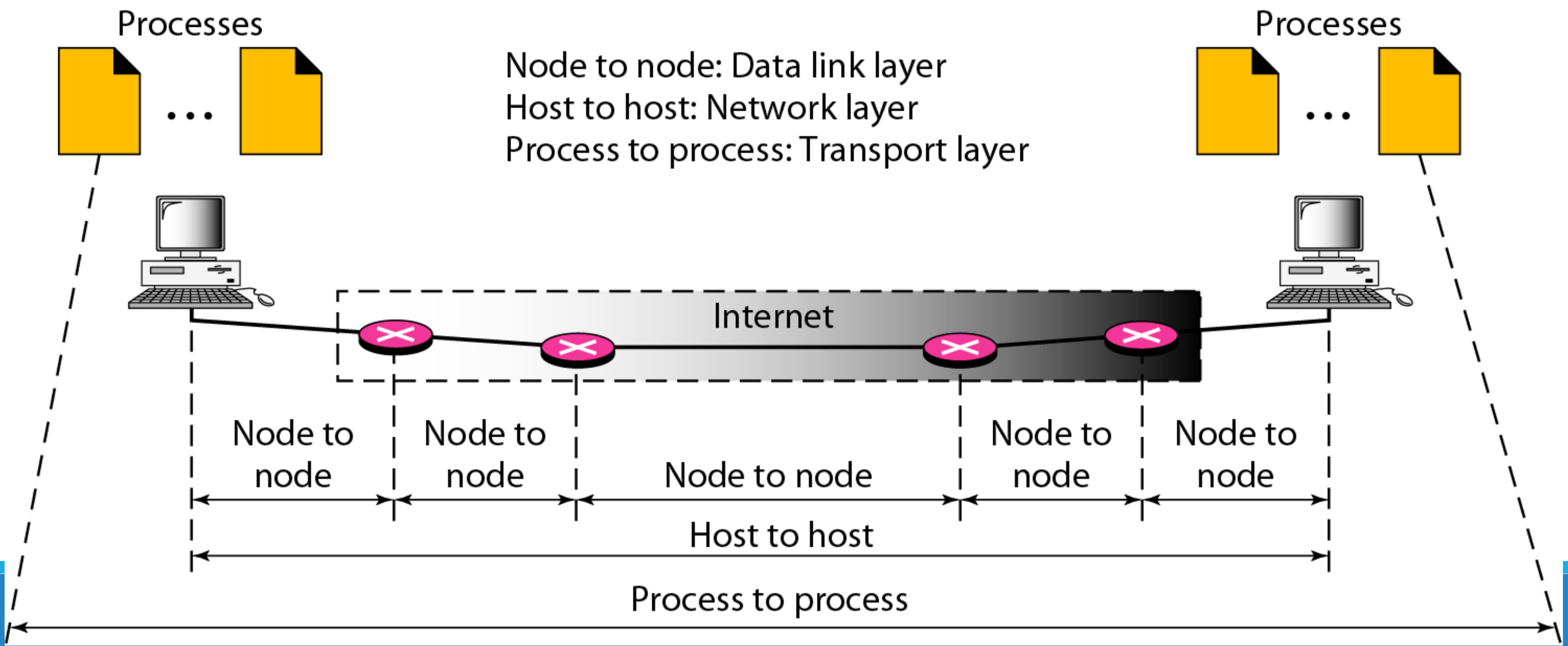


Transport Layer: Services

Process to Process Communication

A process is an application-layer entity that uses the services of the transport layer

- Network layer is responsible for host-to-host communication.
  - This is an incomplete delivery
- Transport-layer protocol is responsible for delivery of the message to the appropriate process.



Transport Layer: Services

Addressing: Port Numbers

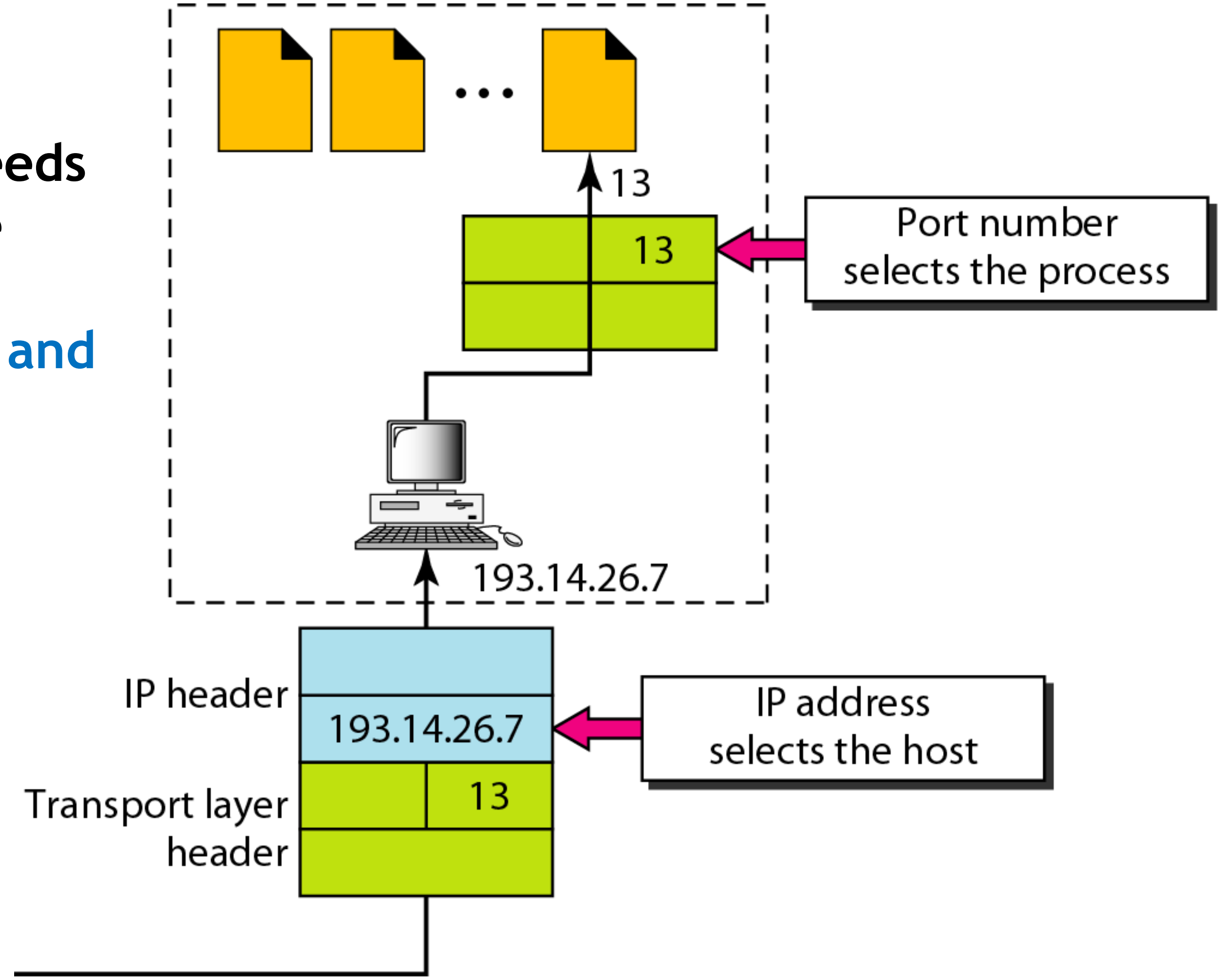
Client-server paradigm

- A process on the local host, called a client, needs services from a process usually on the remote host, called a server.

Operating systems today support both multiuser and multiprogramming environments.

For communication, we must define the local host, local process, remote host, and remote process.

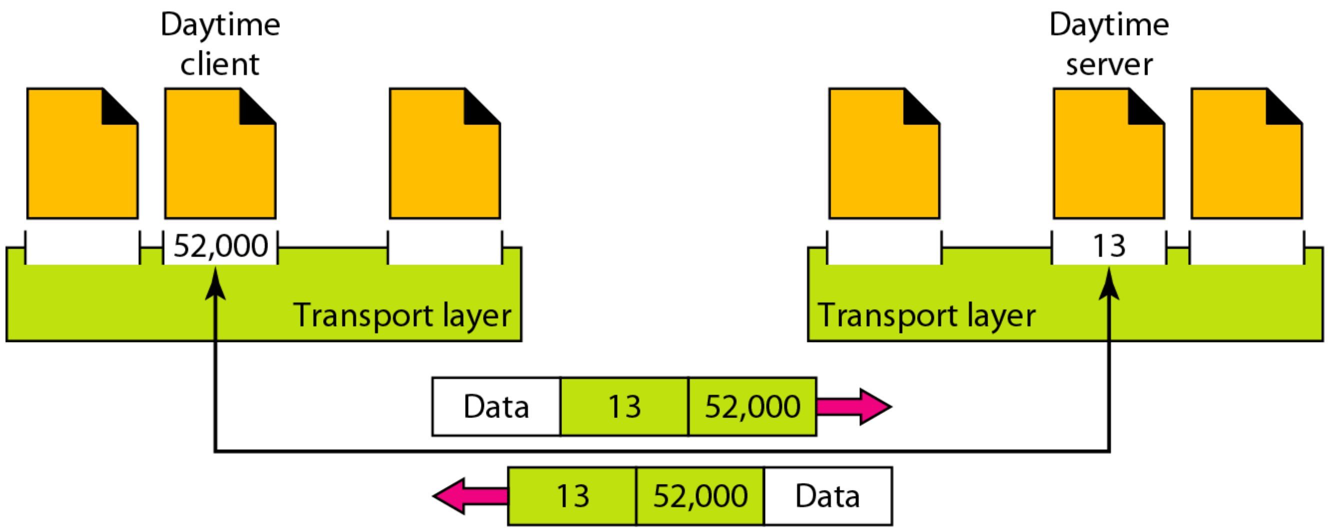
- The local host and the remote host are defined using IP addresses
- To define the processes, we need second identifiers, called *port numbers*.



Transport Layer: Services

Addressing: Port Numbers

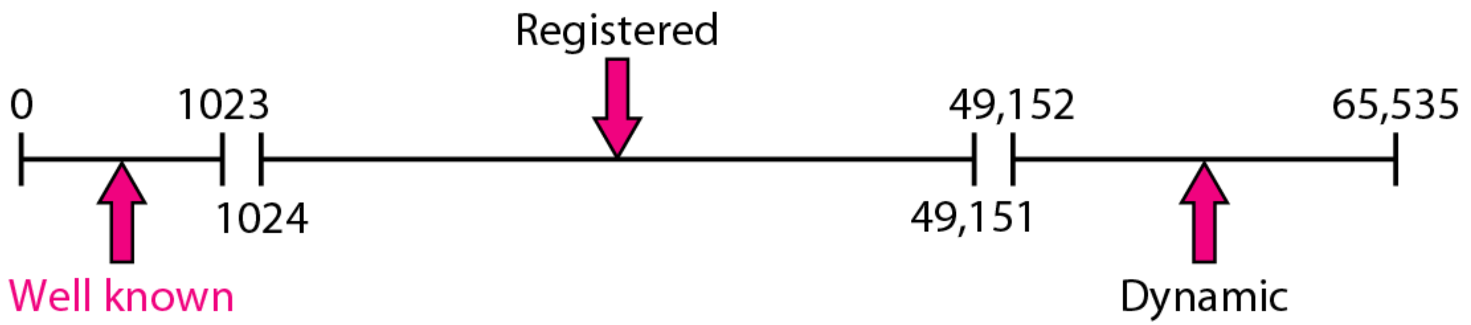
- TCP/IP protocol suite, the port numbers are integers between 0 and 65,535 (16 bits).
- Client program defines the ephemeral port number.
- Server process must also define itself with a port number.
  - Cannot be chosen randomly
- TCP/IP defines universal port numbers for servers; these are called well-known port numbers



Transport Layer: Services

Addressing: Port Numbers: ICANN Ranges

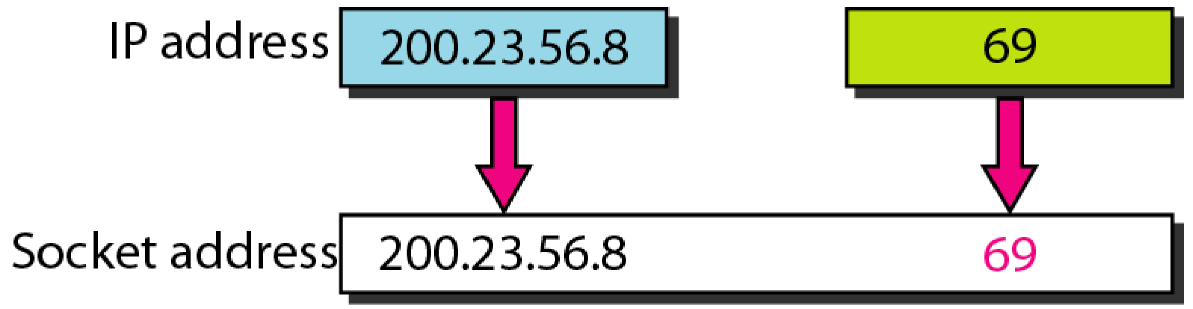
- **Well-known ports.**
  - The ports ranging from 0 to 1023 are assigned and controlled by ICANN..
- **Registered ports.**
  - The ports ranging from 1024 to 49,151 are not assigned or controlled by ICANN. They can only be registered with ICANN to prevent duplication.
- **Dynamic ports.**
  - The ports ranging from 49,152 to 65,535 are neither controlled nor registered.
  - They can be used as temporary or private port numbers.



Transport Layer: Services

Addressing: Socket Address

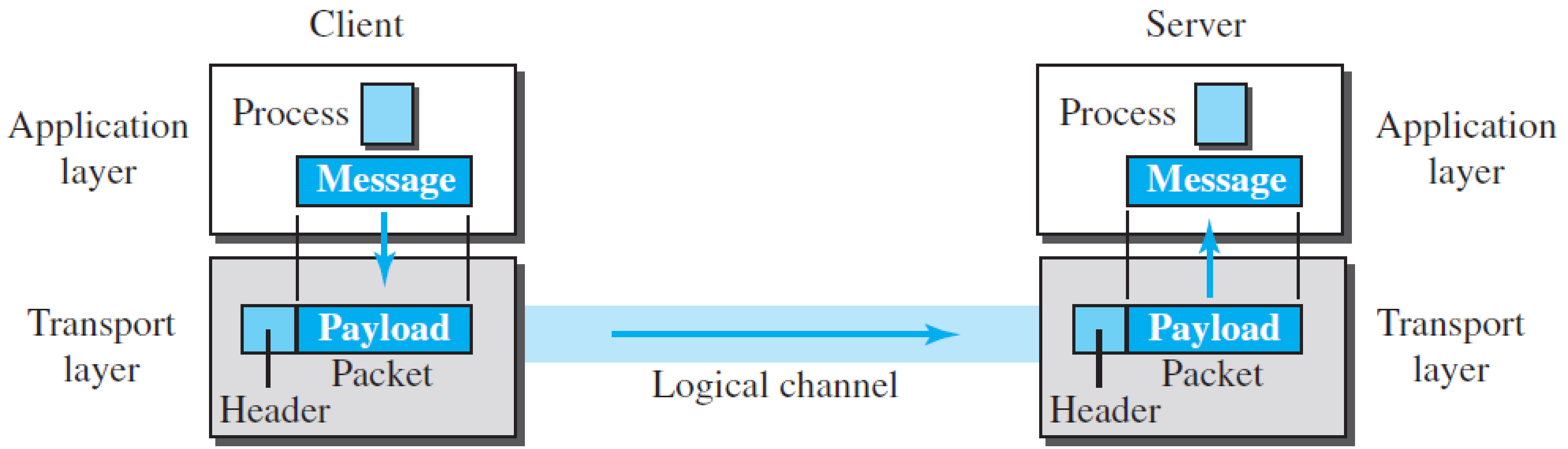
- A transport-layer protocol in the TCP suite **needs both the IP address and the port number, at each end, to make a connection**.
- The **combination of an IP address and a port number** is called a socket address.
- The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely



Transport Layer: Services

Encapsulation & Decapsulation

- Encapsulation happens at the sender site.
  - When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses and some other pieces of information, which depend on the transport-layer protocol.
  - The transport layer receives the data and adds the transport-layer header.
  - The packets at the transport layer in the Internet are called user datagrams, segments, or packets,
- Decapsulation happens at the receiver site



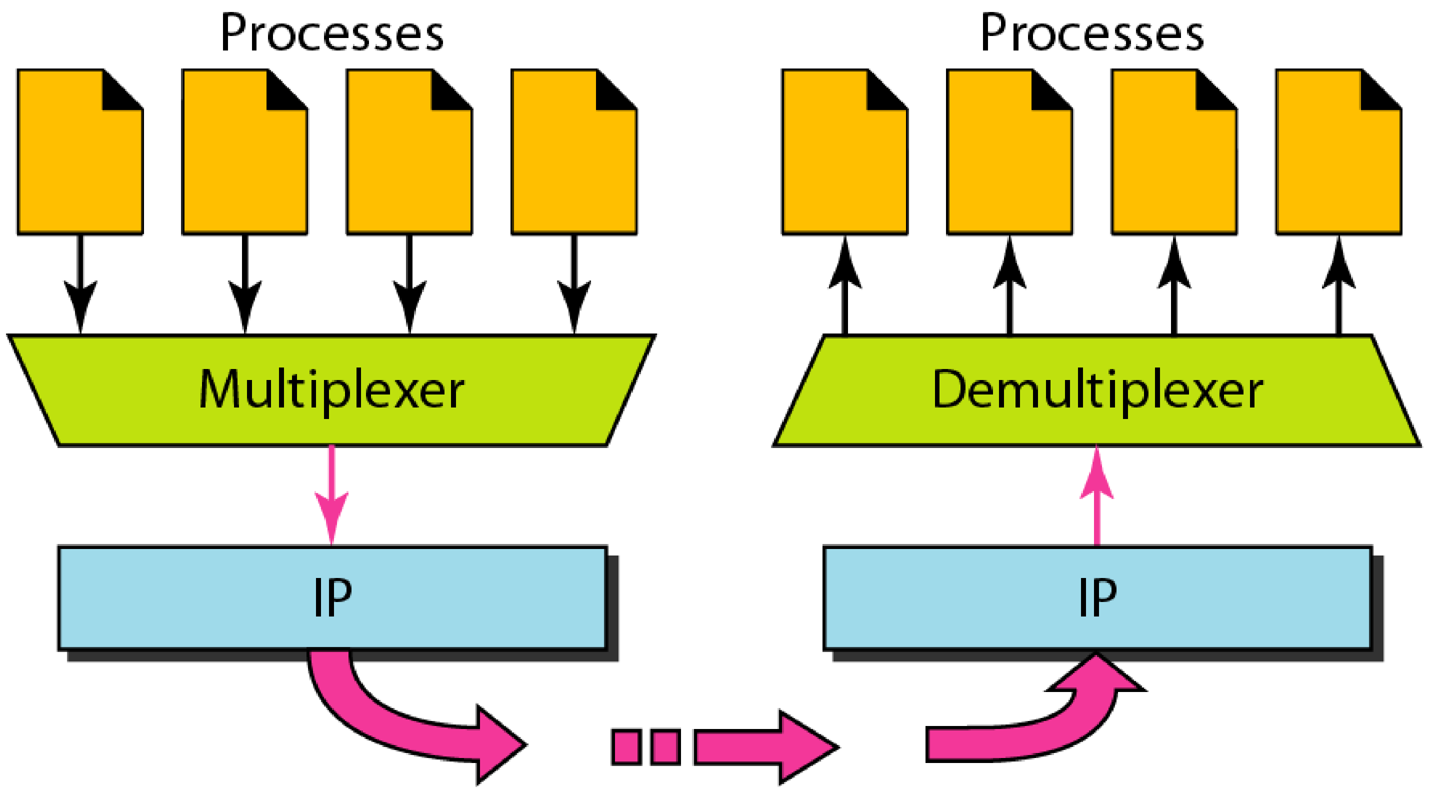
a. Encapsulation

b. Decapsulation

Transport Layer: Services

Multiplexing and Demultiplexing

Transport layer at the source performs multiplexing and at the destination performs demultiplexing

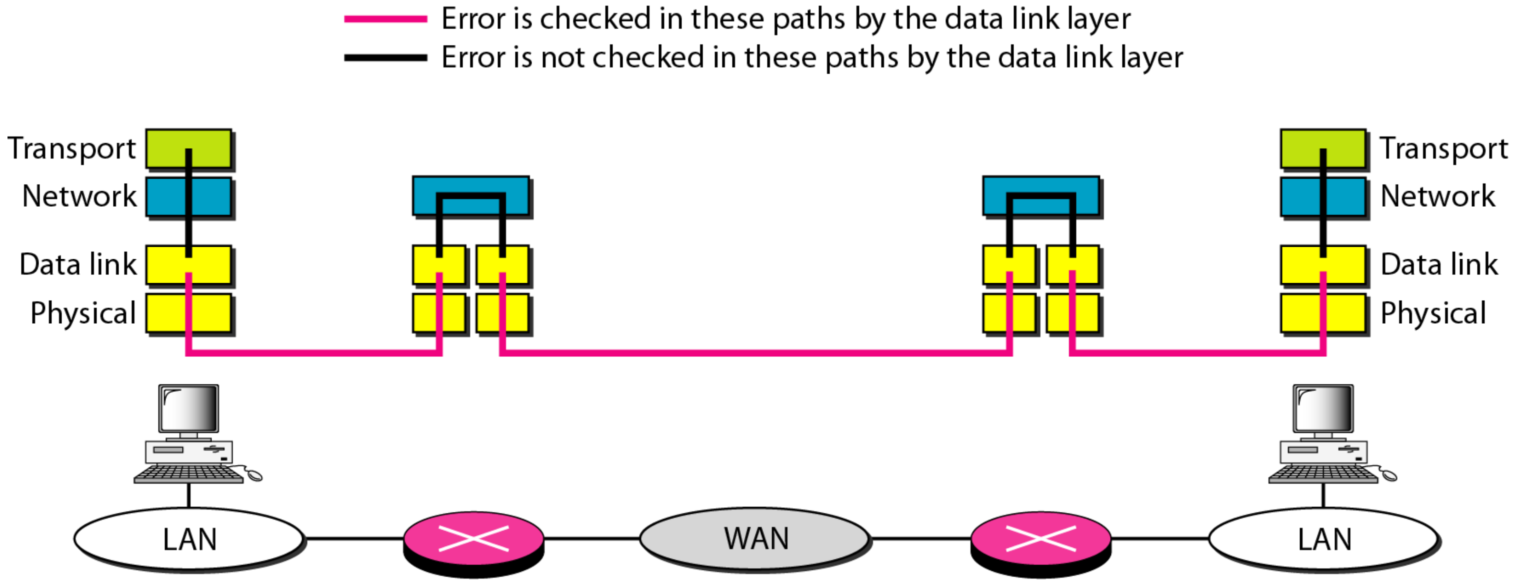


Transport Layer: Services

Error Control

Error control at the transport layer is responsible for

- **1.** Detecting and discarding corrupted packets.
- **2.** Keeping track of lost and discarded packets and resending them.
- **3.** Recognizing duplicate packets and discarding them.
- **4.** Buffering out-of-order packets until the missing packets arrive.


Error is checked in these paths by the data link layer
Error is not checked in these paths by the data link layer
Transport
Network
Data link
Physical
Transport
Network
Data link
Physical
LAN
WAN
LAN

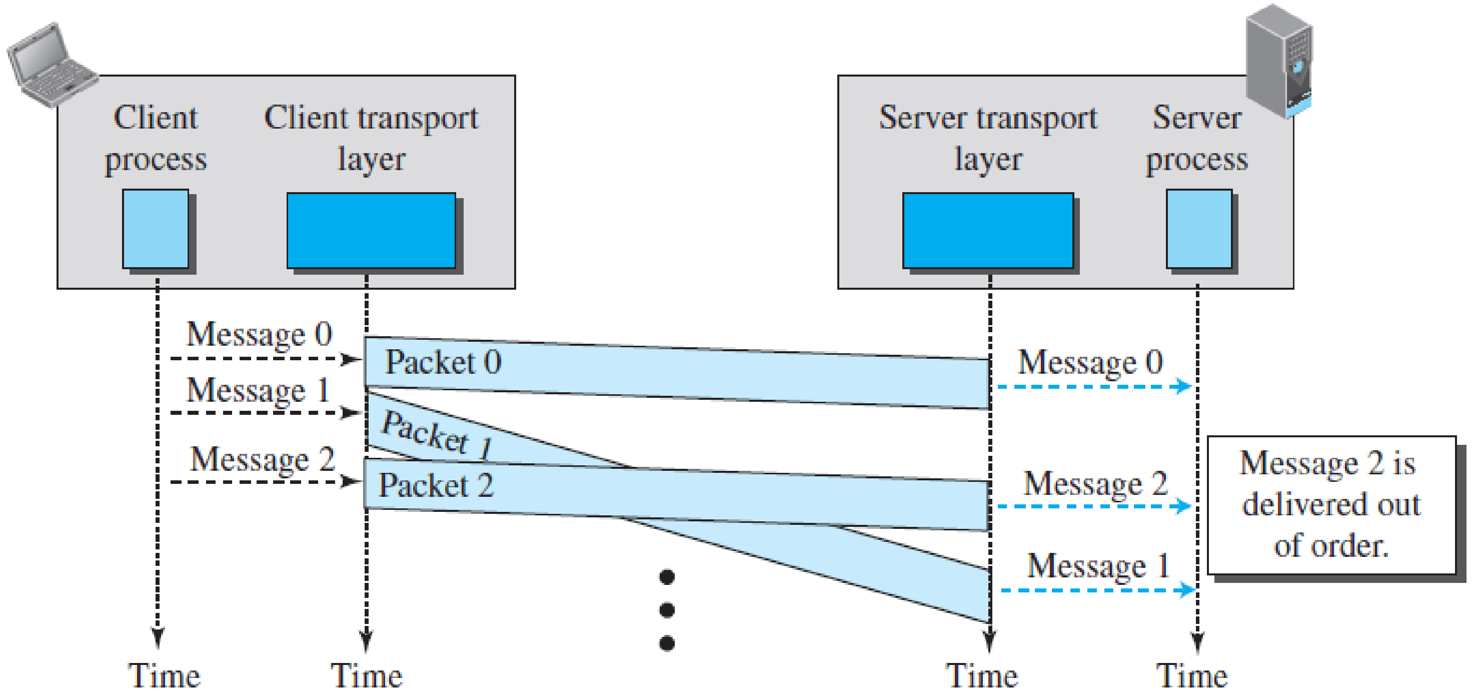
Transport Layer:

Connectionless & Connection Oriented Protocols

Connectionless Service

The source process needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver them one by one.

- Treats each chunk as a single unit without any relation between the chunks.
- Chunks are handed over to the connectionless transport protocol in order.
- The packets may arrive out of order at the destination and will be delivered out of order to the server process.
- No flow control, error control, or congestion control can be effectively implemented in a connectionless service.



Transport Layer:
Connectionless & Connection Oriented Protocols

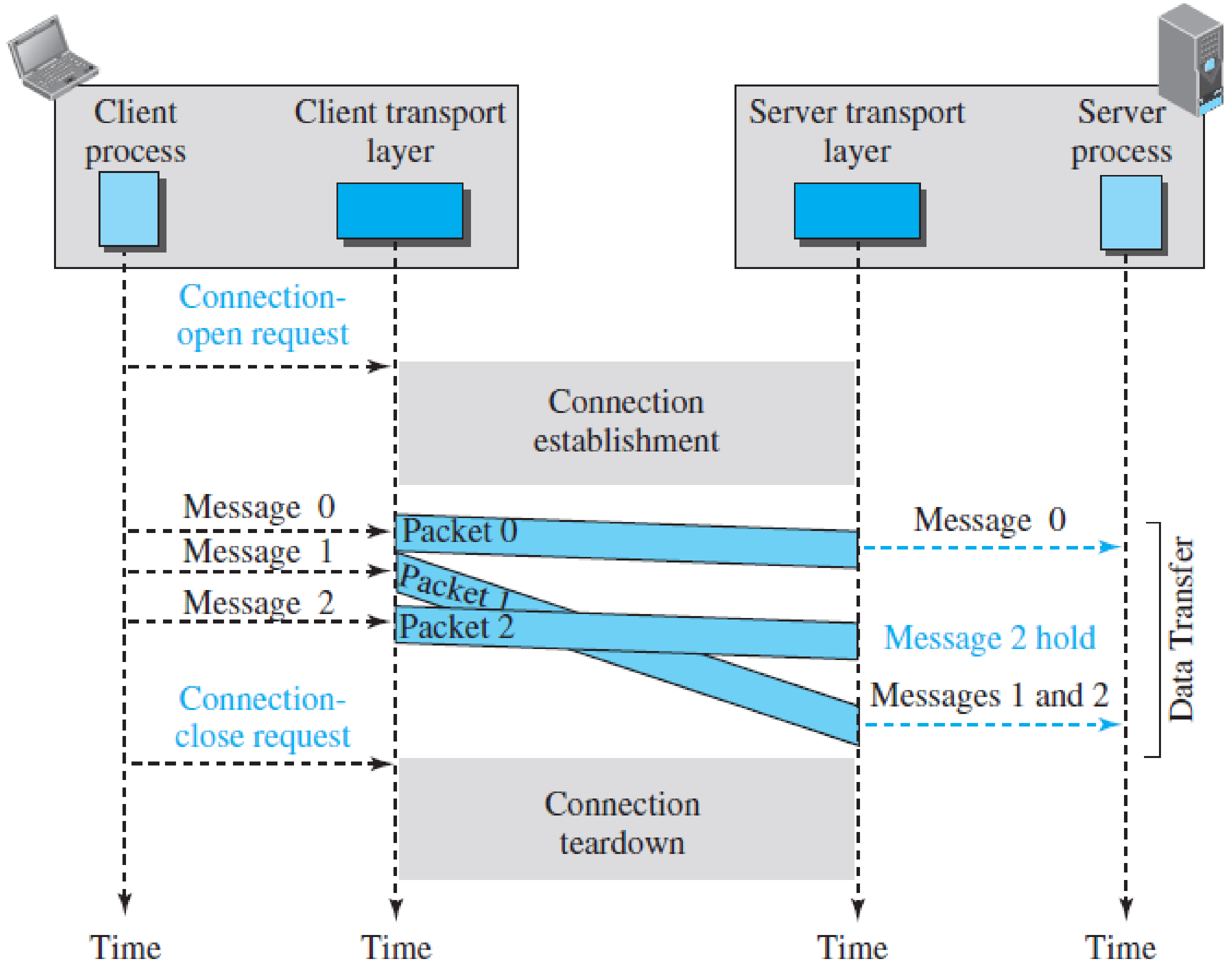
Connection-Oriented Service

The client and the server first need to establish a logical connection between themselves.

The data exchange can only happen after the connection establishment.

After data exchange, the connection needs to be torn down

Flow control, error control, and congestion control can be implemented.



Outline

Transport Layer Protocols

- UDP
  - Services
  - Applications
- TCP
  - Services
  - Segment Format
  - A TCP Connection

User Datagram Protocol

User Datagram Protocol (UDP)

- A connectionless, unreliable transport protocol.
- Provide process-to-process communication.
- UDP is used for its simplicity efficiency in applications where error control can be provided by the application-layer process.

A very simple protocol using a minimum of overhead.

UDP is Preferred When:

- If a process wants to send a small message and does not care much about reliability.

Sending a small message using UDP takes much less interaction between the sender and receiver than using TCP.

UDP Services

Process-to-Process Communication

- Provides process-to-process communication using **socket addresses** (combination of IP addresses and port numbers).

Connectionless Services

- Each user datagram sent by UDP is an **independent datagram**.
- User datagrams are not numbered.

Flow Control

- There **is no flow control**

Error Control

- There **is no error control**

Checksum

- UDP checksum calculation includes three sections:
  - *A pseudo-header*
  - *UDP header, and*
  - *The data coming from the application layer.*

UDP Services

Congestion Control

- There **is no congestion control**

Encapsulation and Decapsulation

Queuing

- Queues are associated with ports.
- At the client site, when a process starts,
  - It requests a port number from the operating system.
  - Some implementations create both an incoming and an outgoing queue associated with each process.
  - Other implementations create only an incoming queue associated with each process.

Multiplexing and Demultiplexing

- In a host running a TCP/IP protocol suite
  - There is only one UDP but possibly several processes that may want to use the services of UDP.
  - To handle this situation, UDP multiplexes and demultiplexes.

UDP Applications

UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control.

UDP is suitable for a process with internal flow- and error-control mechanisms.

- For example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.

UDP is a suitable transport protocol for multicasting.

- Multicasting capability is embedded in the UDP software but not in the TCP software.

UDP is used for management processes such as SNMP.

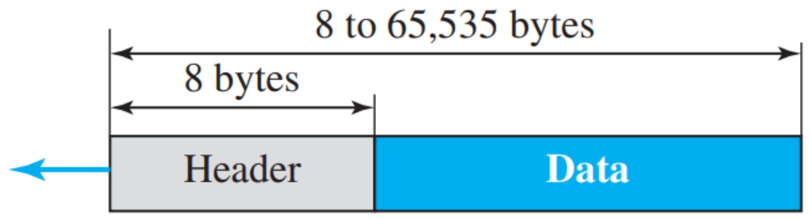
UDP is used for some route updating protocols such as Routing Information Protocol

UDP is normally used for interactive real-time applications that cannot tolerate uneven delay between sections of a received message.

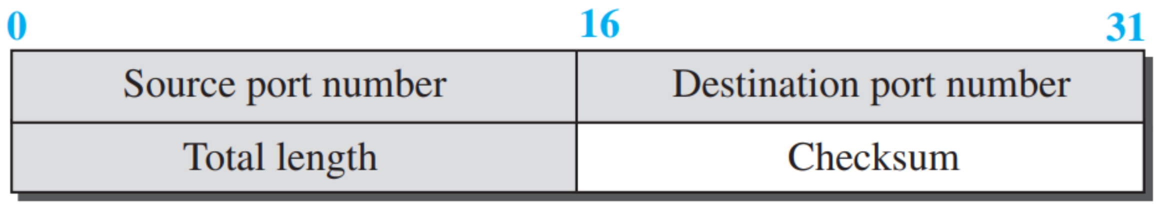
User Datagram

UDP packets, called user datagrams, have a fixed-size header of 8 bytes made of four fields, each of 2 bytes (16 bits).

- Figure shows the format of a user datagram.
- The first two fields define the source and destination port numbers.
- The third field defines the total length of the user datagram, header plus data.
- The 16 bits can define a total length of 0 to 65,535 bytes.
- However, the total length needs to be less because a UDP user datagram is stored in an IP datagram with the total length of 65,535 bytes



a. UDP user datagram



b. Header format

Transmission Control Protocol

A connection oriented, reliable protocol

- Defines connection establishment,
- Data transfer, and
- Connection teardown phases.

Uses a combination of GBN and SR protocols to provide reliability.

- Uses checksum (for error detection),
- Retransmission of lost or corrupted packets,
- Cumulative and selective acknowledgments, and
- Timers

Most common transport-layer protocol in the Internet

TCP: Services

Stream Delivery Services

TCP, unlike **UDP, is a stream-oriented protocol**.

- UDP adds its own header to each of these messages and delivers it to IP for transmission.
- Each message from the process is called a *user datagram*.

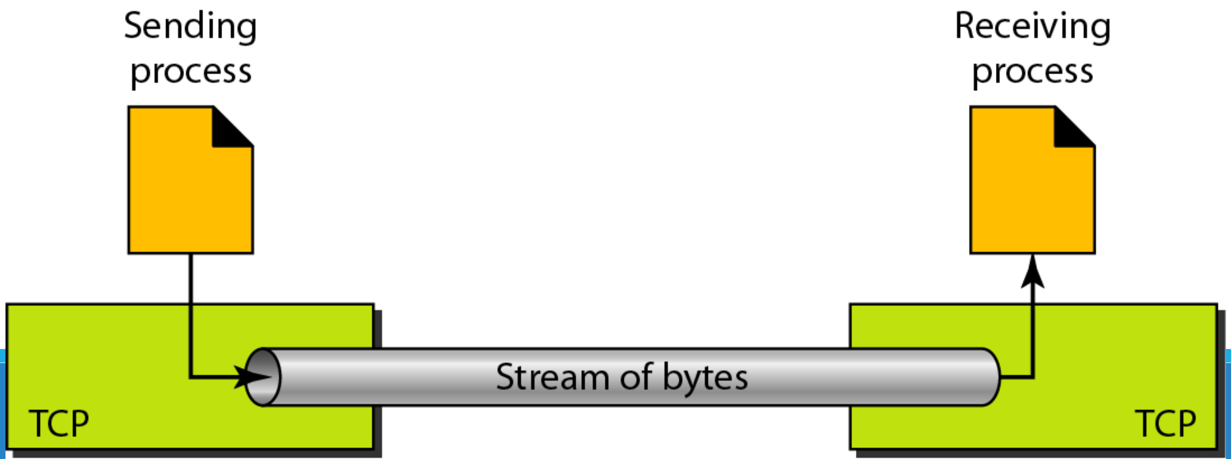
Neither IP nor UDP recognizes any relationship between the datagrams.

TCP, allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes.

- Creates an environment in which the two processes seem to be connected by an **imaginary "tube"** that carries their bytes across the Internet.

Sending process produces (writes to) the stream

Receiving process consumes (reads from) it.



TCP: Services

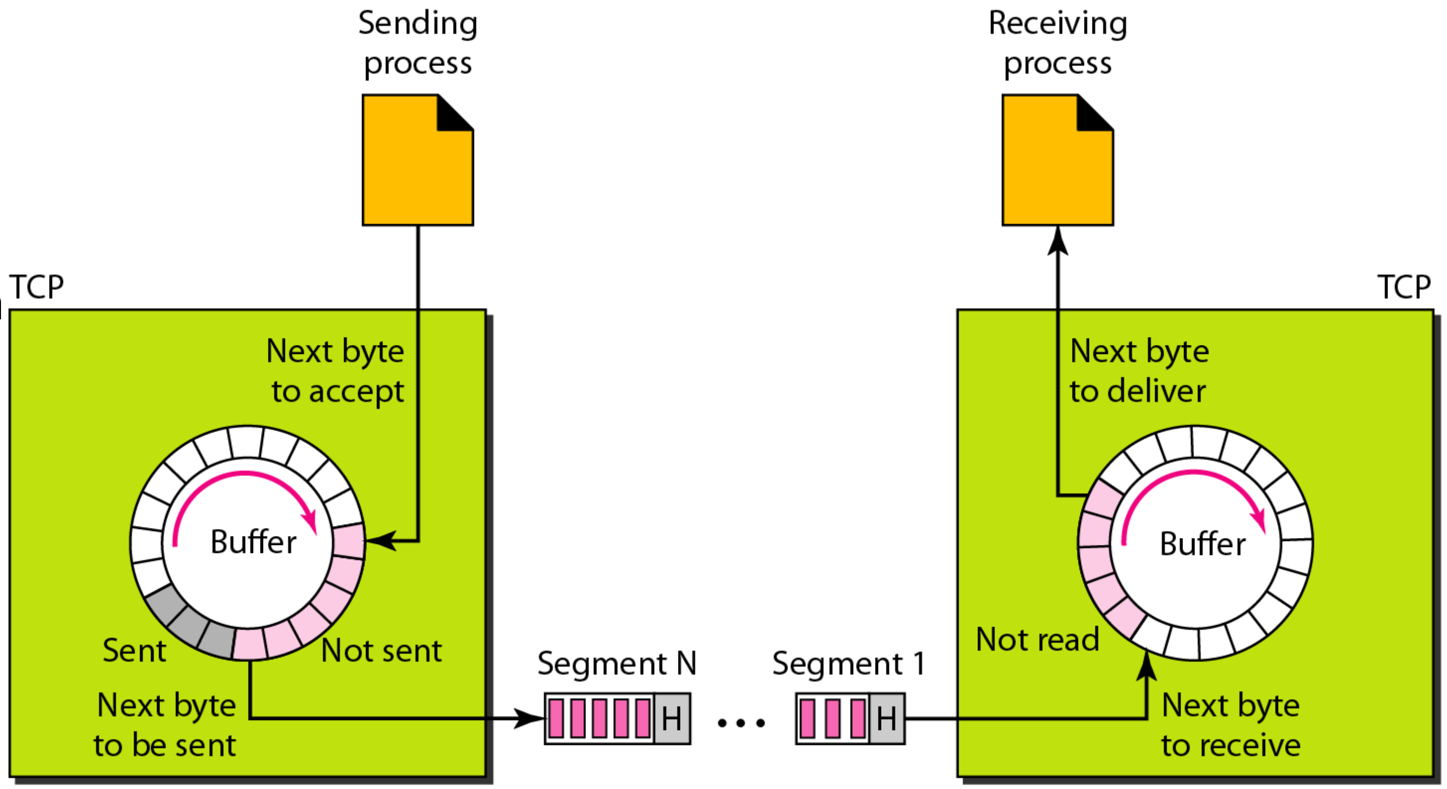
TCP Segments

Buffering *handles the disparity between the speed of the producing and consuming processes.*

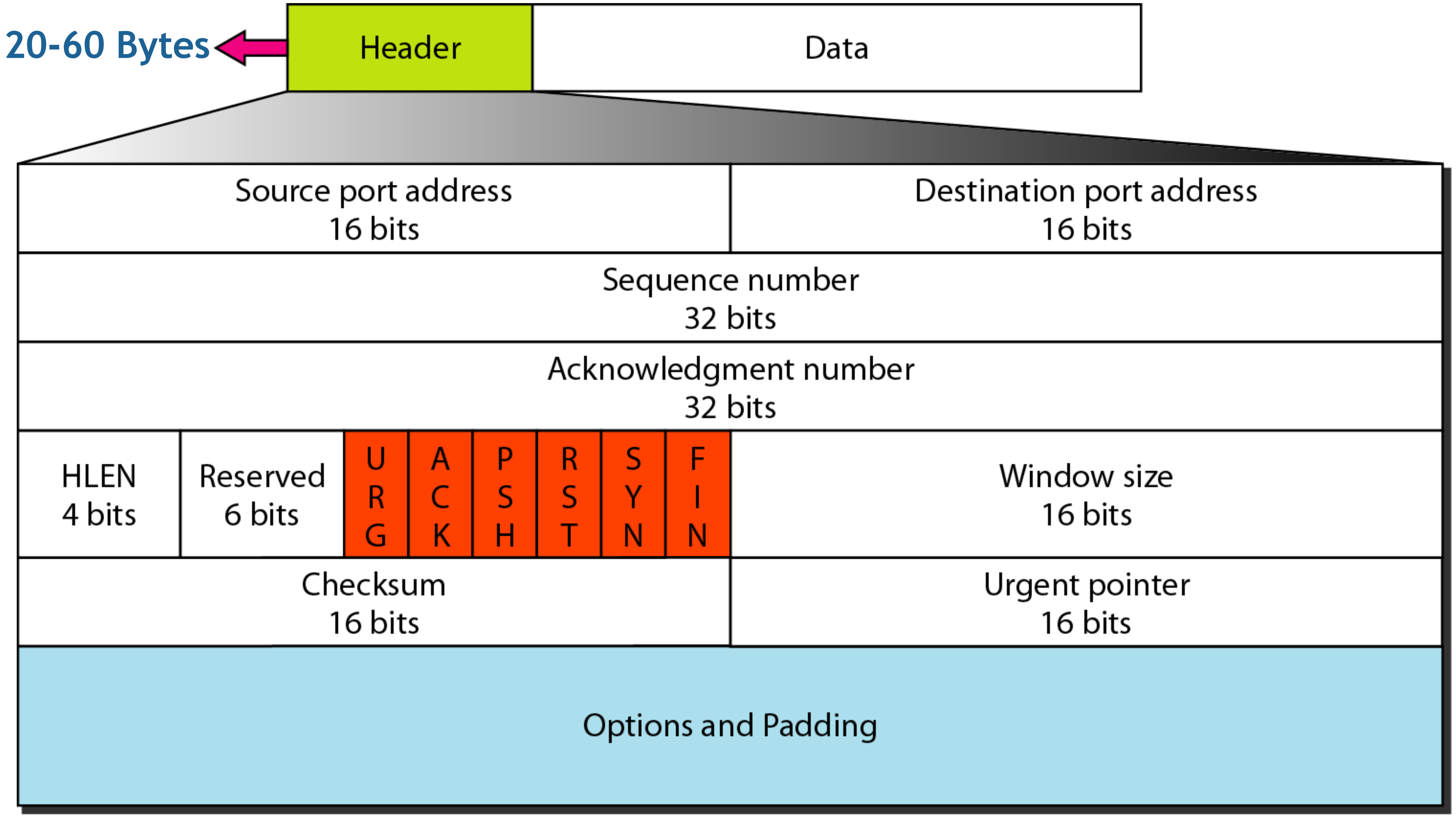
Network layer, as a service provider for TCP, needs to send data in packets, **not as a stream of bytes.**

At the transport layer, **TCP groups a number of bytes together into a packet called a *segment*.**

Segments are encapsulated in an IP datagram and transmitted



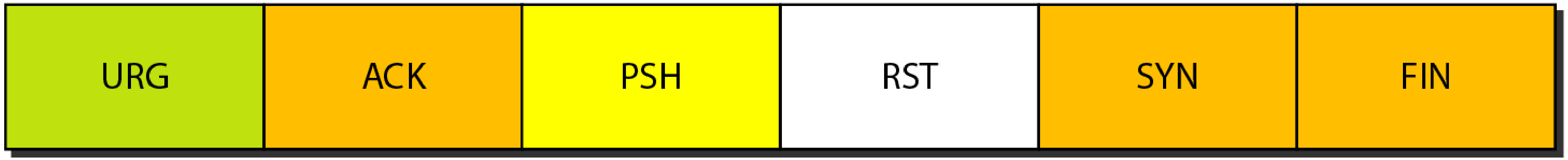
TCP: Segment Format



TCP: Segment Format

Control Field

URG: Urgent pointer is valid	RST: Reset the connection
ACK: Acknowledgment is valid	SYN: Synchronize sequence numbers
PSH: Request for push	FIN: Terminate the connection



A TCP Connection

Connection Establishment using 3 Way Handshaking

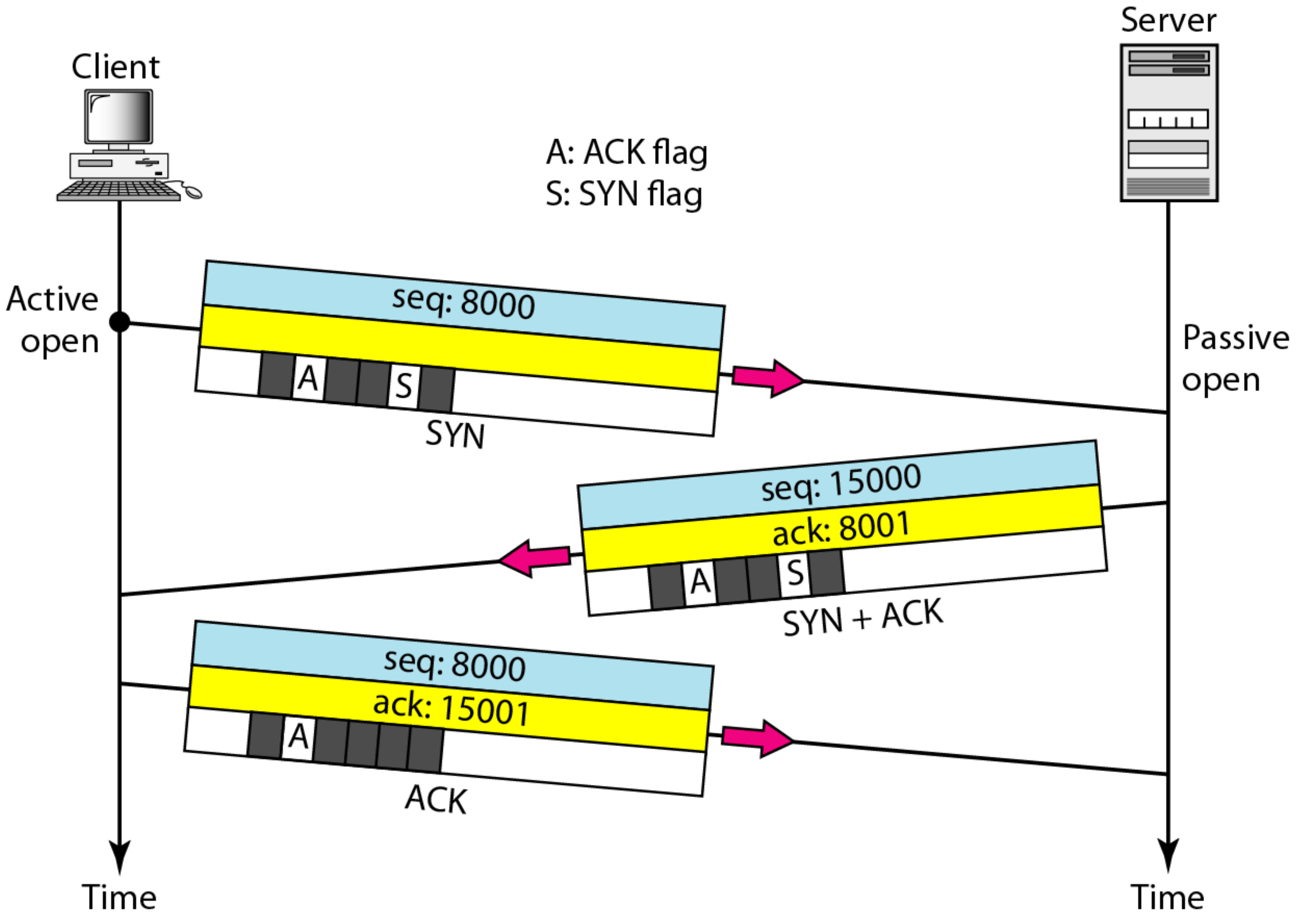
Example: Client Server Connection.

- *Passive open:* Starts with the server. The server program tells its TCP that it is ready to accept a connection.
- *Active Open:* Client program issues a request.

A SYN segment cannot carry data, but it consumes one sequence number.

SYN 1 ACK segment cannot carry data, but it does consume one sequence number.

An ACK segment, if carrying no data, consumes no sequence number.



TCP Congestion Control

TCP congestion control refers to the mechanism that prevents congestion from happening or removes it after congestion takes place.

When congestion takes place in the network, **TCP handles it by reducing the size of the sender's window.**

The window size of the sender is determined by the following two factors:

- **Receiver window size (rwnd)**
  - The size of the send window is controlled by the receiver using the value of rwnd,
  - which is advertised in each segment traveling in the opposite direction
- **Congestion window size (cwnd)**
  - Size is controlled by the congestion situation in the network

TCP Congestion Control

Receiver Window Size

- It shows how much data can a receiver receive in bytes without giving any acknowledgment.
- Things to remember for receiver window size:
  - The ***sender should not send data greater than that of the size of receiver window***.
  - If the **data sent is greater** than that of the size of the receiver's window, **then it causes retransmission of TCP due to the dropping of TCP segment.**
  - Hence **sender should always send data that is less than or equal to the size of the receiver's window.**
  - TCP header is used for sending the **window size** of the receiver to the sender.

Congestion Window

- It is the state of TCP that **limits the amount of data to be sent by the sender into the network even before receiving the acknowledgment**.
- Following are the things to remember for the congestion window:
  - To calculate the size of the congestion window, different variants of TCP and methods are used.
  - Only the *sender knows the congestion window and its size and it is not sent over the link or network*.

Sender window size = Minimum (Receiver window size, Congestion window size)

TCP Congestion Detection

TCP sender uses the occurrence of two events as signs of congestion in the network:

- Time-out and
- Receiving three duplicate ACKs.

Time-out.

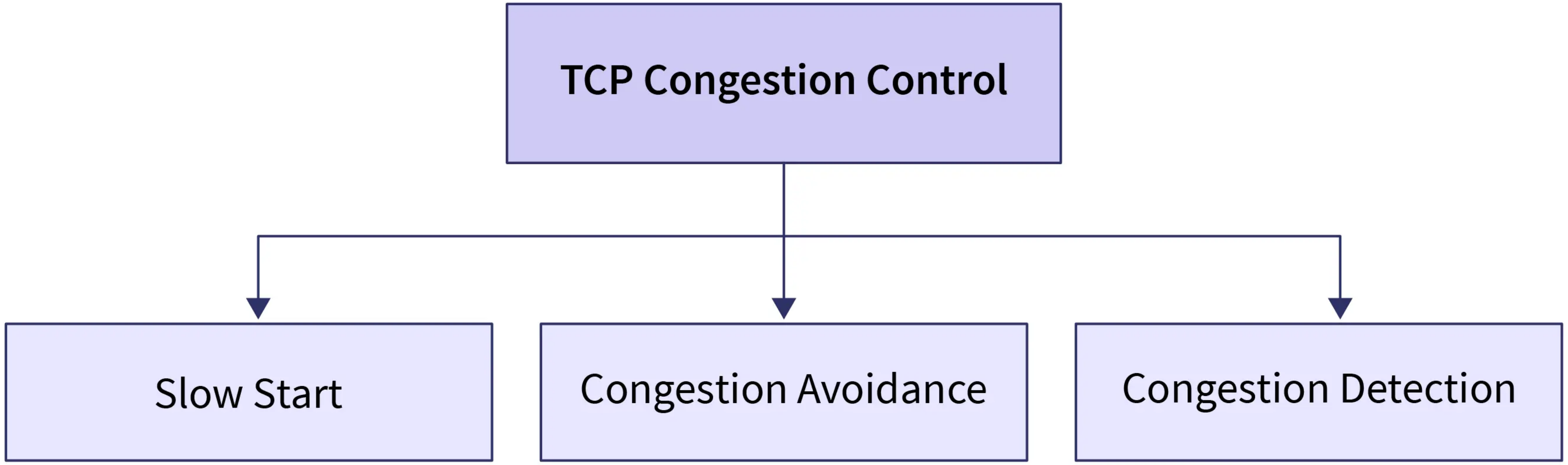
- If a TCP sender **does not receive an ACK for a segment** or a **group of segments** before the **time-out occurs**,
- it assumes that **the corresponding segment or segments are lost** and the loss is due to congestion.

Receiving of Three Duplicate ACKs

- Four ACKs with the same acknowledgment number.
- Recall that when a **TCP receiver sends a duplicate ACK**, it is the sign that a **segment has been delayed**, but sending three duplicate ACKs is the sign of a missing segment, which can be due to **congestion in the network**.
- When a **receiver sends three duplicate ACKs**, it means that **one segment is missing**, but three segments have been received.
- The network is **either slightly congested or has recovered from the congestion**.

TCP Congestion Control

Approaches for Congestion Control:

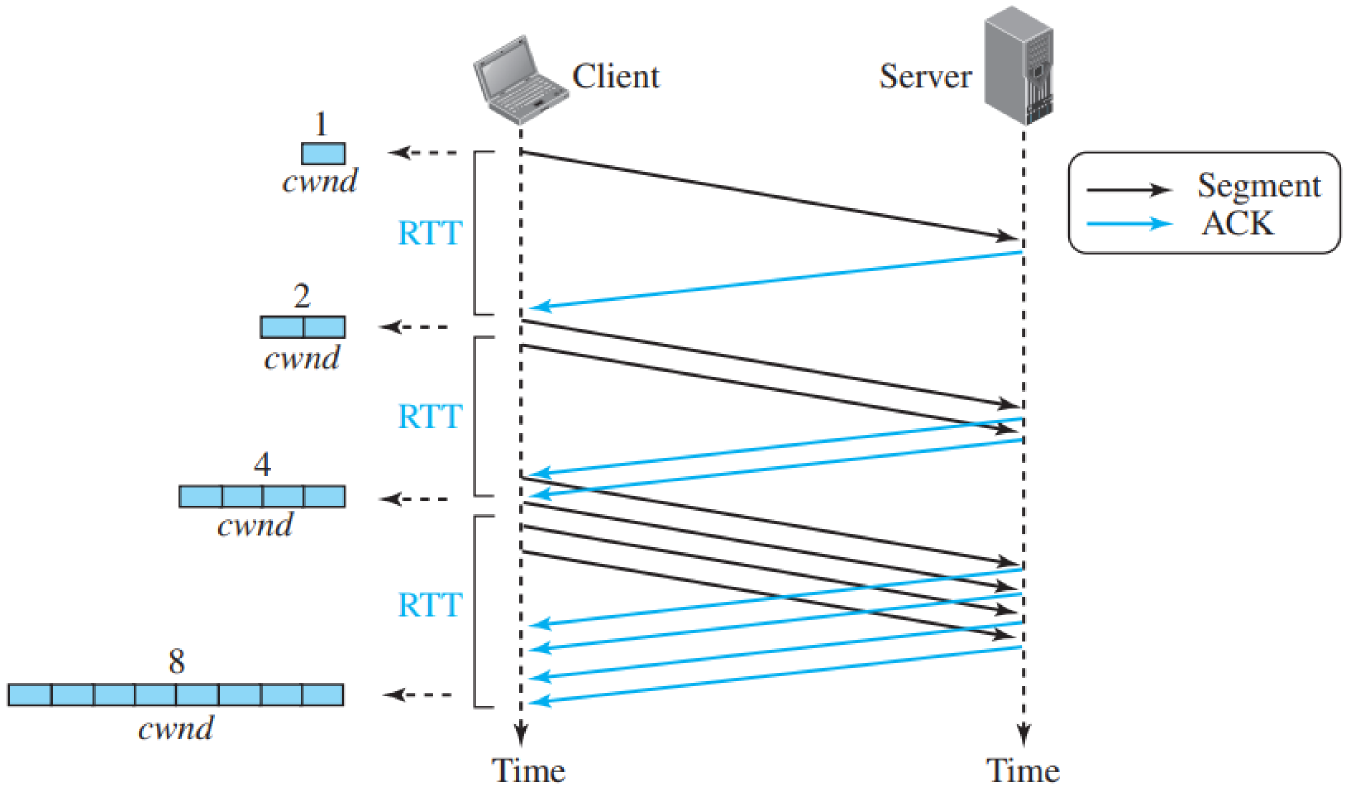


TCP Congestion Control

Slow Start:

Based on the idea that the **size of the congestion window (cwnd)** starts with **one maximum segment size (MSS)**, but it **increases one MSS each time an acknowledgment arrives**.

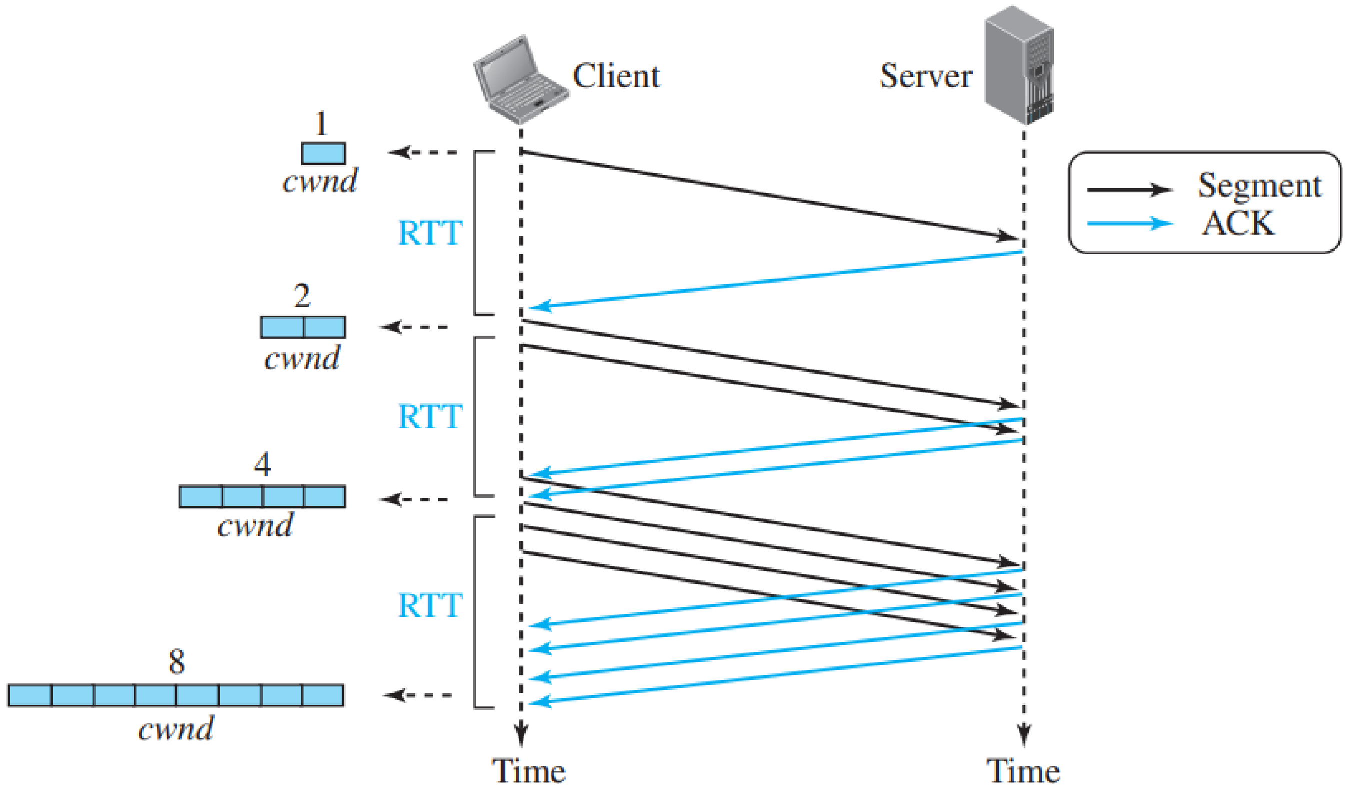
- The name of this algorithm is misleading; the algorithm **starts slowly, but grows exponentially.**
- In Figure.
- We assume that **rwnd is much larger than cwnd**, so that the sender window size always equals cwnd.
- We also assume that **each segment is of the same size and carries MSS bytes**.
- For simplicity, we also ignore the delayed-ACK policy and assume that each segment is acknowledged individually.



TCP Congestion Control

Slow Start:

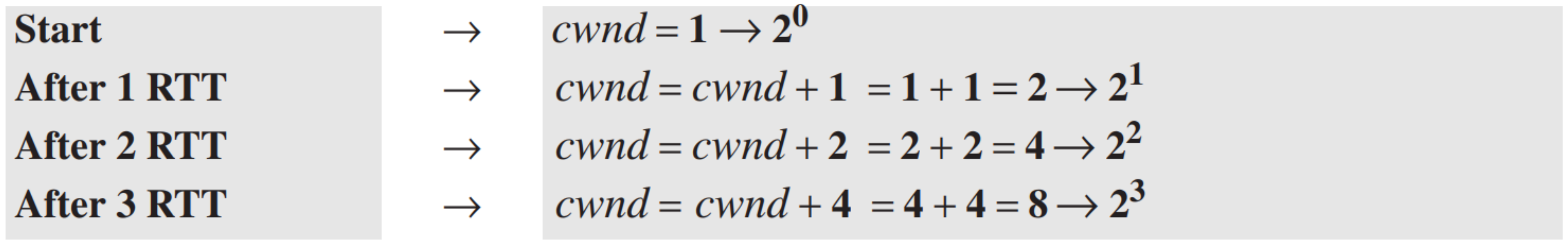
- The sender starts with cwnd = 1.
  - Means that the **sender can send only one segment.**
- After the first ACK arrives, **the acknowledged segment is purged from the window,**
  - Means there is now one empty segment slot in the window.
- The size of the congestion window is also increased by 1 because the arrival of the acknowledgment is a good sign that there is no congestion in the network.
- The size of the window is now 2.
- After sending two segments and receiving two individual acknowledgments for them, the size of the congestion window now becomes 4, and so on



TCP Congestion Control

Slow Start:

- The size of the congestion window in this algorithm is a function of the number of ACKs arrived and can be determined as follows.
- If an ACK arrives, cwnd = cwnd + 1.
- If we look at the size of the cwnd in terms of round-trip times (RTTs), we find that the growth rate is exponential in terms of each round trip time, which is a very aggressive approach:

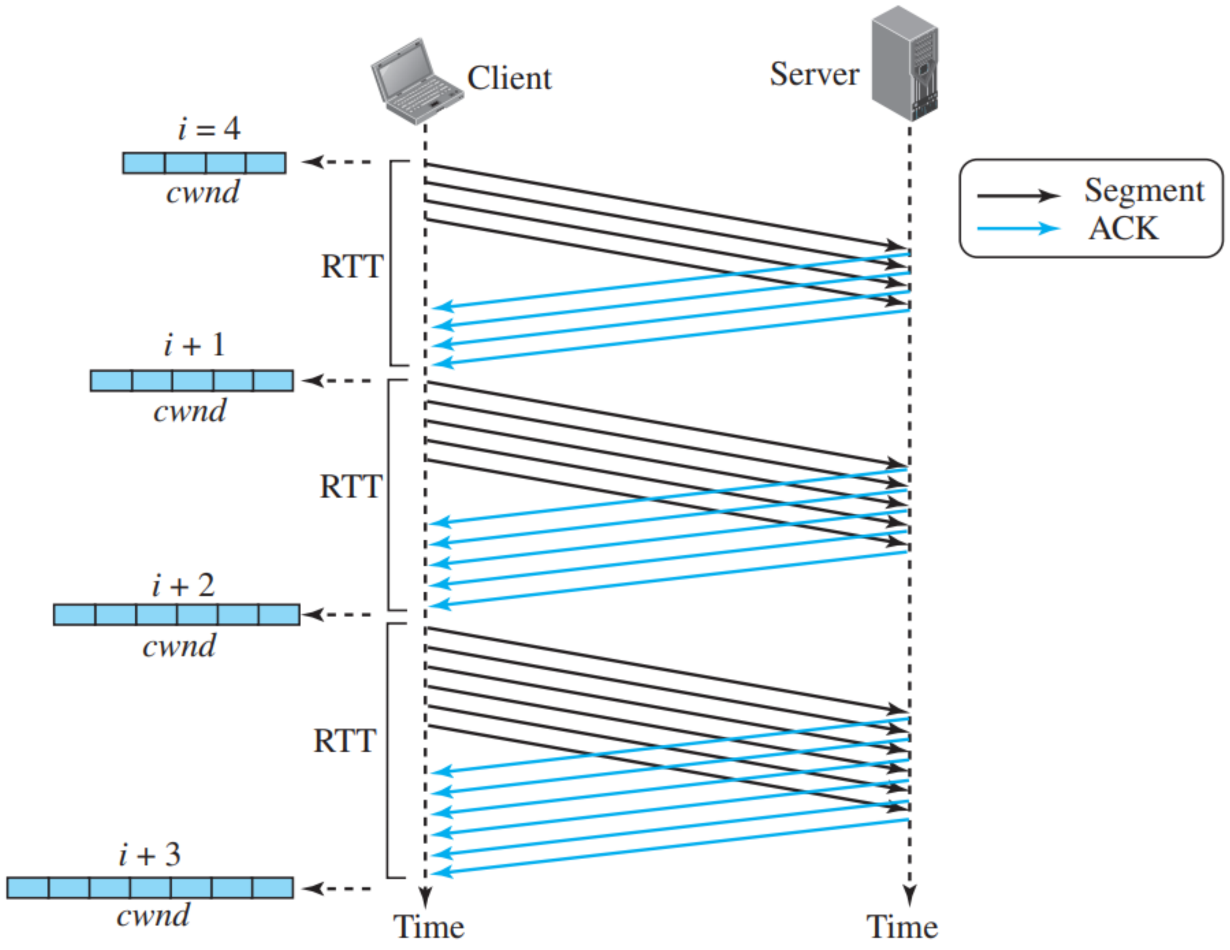


Start	$\rightarrow$	$cwnd = 1 \rightarrow 2^0$
After 1 RTT	$\rightarrow$	$cwnd = cwnd + 1 = 1 + 1 = 2 \rightarrow 2^1$
After 2 RTT	$\rightarrow$	$cwnd = cwnd + 2 = 2 + 2 = 4 \rightarrow 2^2$
After 3 RTT	$\rightarrow$	$cwnd = cwnd + 4 = 4 + 4 = 8 \rightarrow 2^3$

TCP Congestion Control

Congestion Avoidance:

- If we continue with the slow-start algorithm, the size of the congestion window increases exponentially.
- To avoid congestion before it happens, we must slow down this exponential growth.
- TCP defines another algorithm called **congestion avoidance, which increases the cwnd additively instead of exponentially**.
- ***When the size of the congestion window reaches the slow-start threshold in the case where cwnd = i,***
  - The slow-start phase stops and the additive phase begins.
  - In this algorithm, each time the whole "window" of segments is acknowledged, the **size of the congestion window is increased by one**.
  - A window is the number of segments transmitted during RTT.



TCP Congestion Control

Congestion Avoidance:

- The sender starts with cwnd = 4.
  - This means that the sender can send only four segments.
- After **four ACKs arrive**, the acknowledged segments are purged from the window, which means there is now one extra empty segment slot in the window.
  - The size of the congestion window is also increased by 1.
  - The size of window is now 5.
- After **sending five segments** and receiving five acknowledgments for them, the size of the congestion window now becomes 6, and so on.

- In other words, the **size of the congestion window in this algorithm** is also a **function of the number of ACKs that have arrived** and can be determined as follows:
  - If an ACK arrives, **cwnd = cwnd + (1/cwnd).**
  - The size of the window increases only 1/cwnd portion of MSS (in bytes).
  - In other words, all segments in the previous window should be acknowledged to increase the window 1 MSS bytes.
- If we look at the size of the cwnd in terms of round-trip times (RTTs),
  - We find that the **growth rate is linear in terms of each round-trip time**, which is much more conservative than the slow-start approach.

TCP Congestion Control

Congestion Avoidance:

Start	$\rightarrow$	$cwnd = i$
After 1 RTT	$\rightarrow$	$cwnd = i + 1$
After 2 RTT	$\rightarrow$	$cwnd = i + 2$
After 3 RTT	$\rightarrow$	$cwnd = i + 3$

TCP Congestion Control

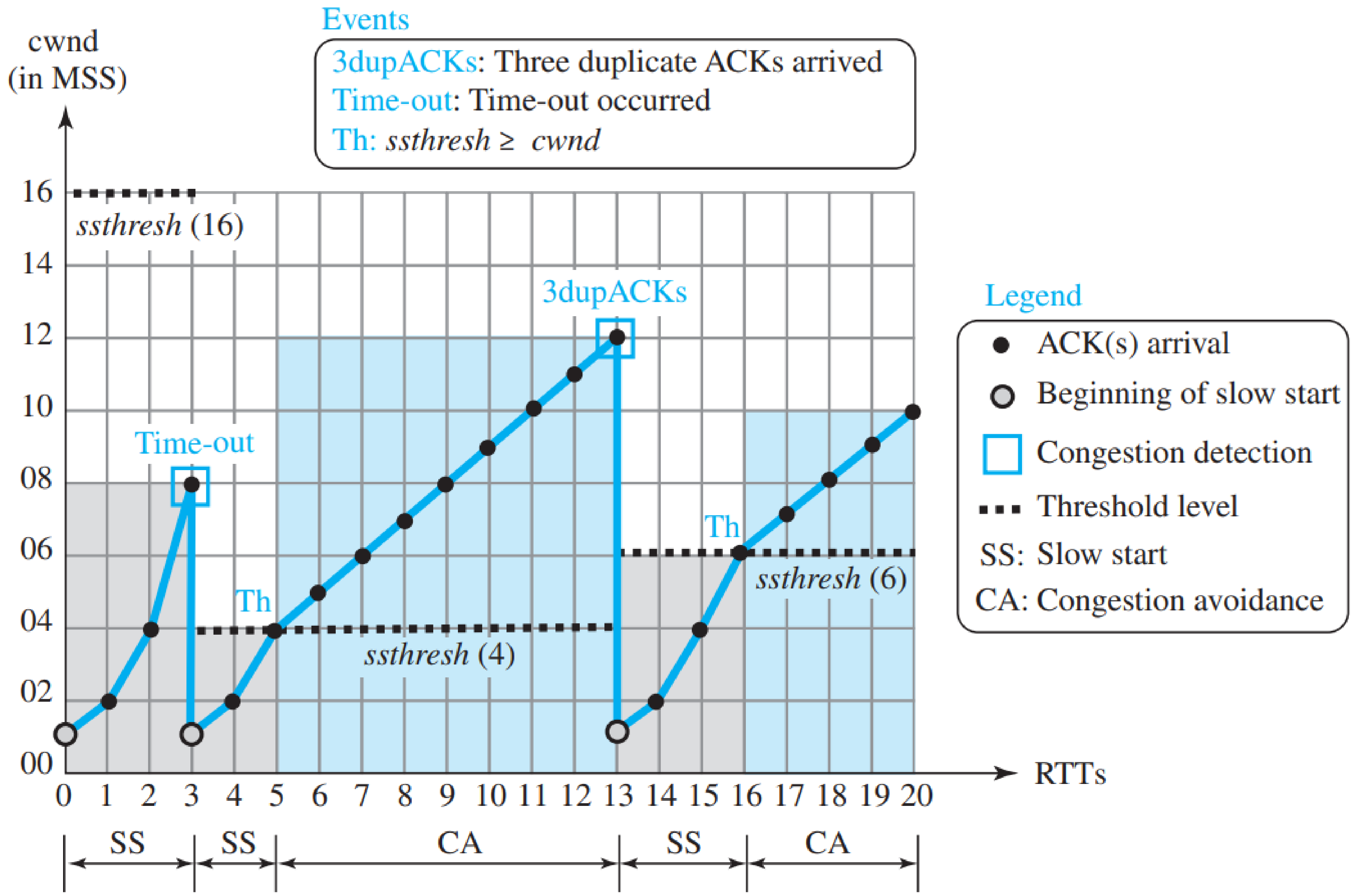
Fast Recovery:

- The fast-recovery algorithm is optional in TCP.
- The old version of TCP did not use it, but the new versions try to use it.
- It starts when three duplicate ACKs arrive, which is interpreted as light congestion in the network.
- Like congestion avoidance, this algorithm is also an additive increase, but it increases the size of the congestion window when a duplicate ACK arrives (after the three duplicate ACKs that trigger the use of this algorithm).
- We can say If a duplicate ACK arrives, cwnd = cwnd + (1 / cwnd).

TCP Congestion Window

Taco TCP

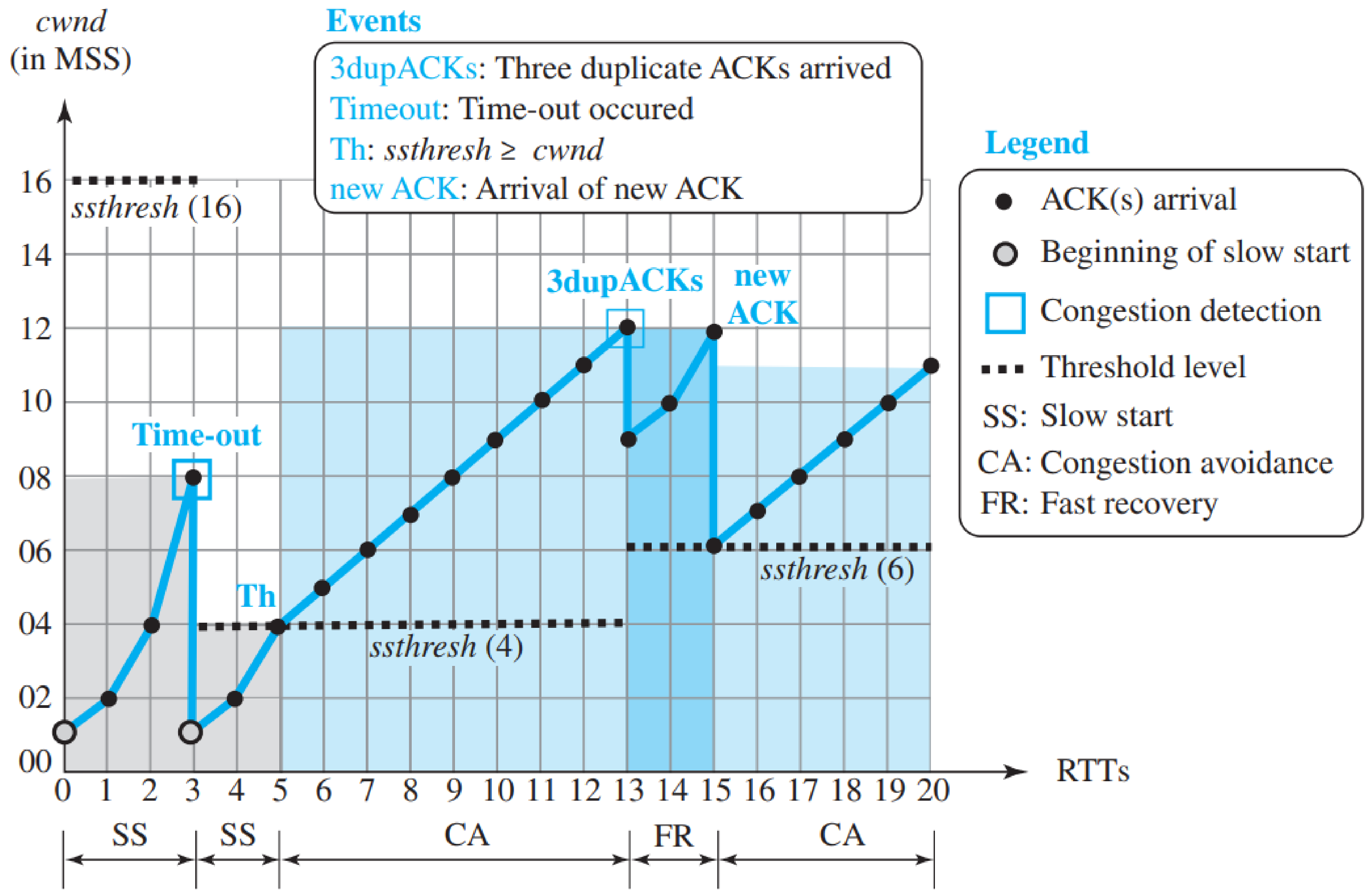
The early TCP, known as Taho TCP, used only two different algorithms in their congestion policy: slow start and congestion avoidace



TCP Congestion Window

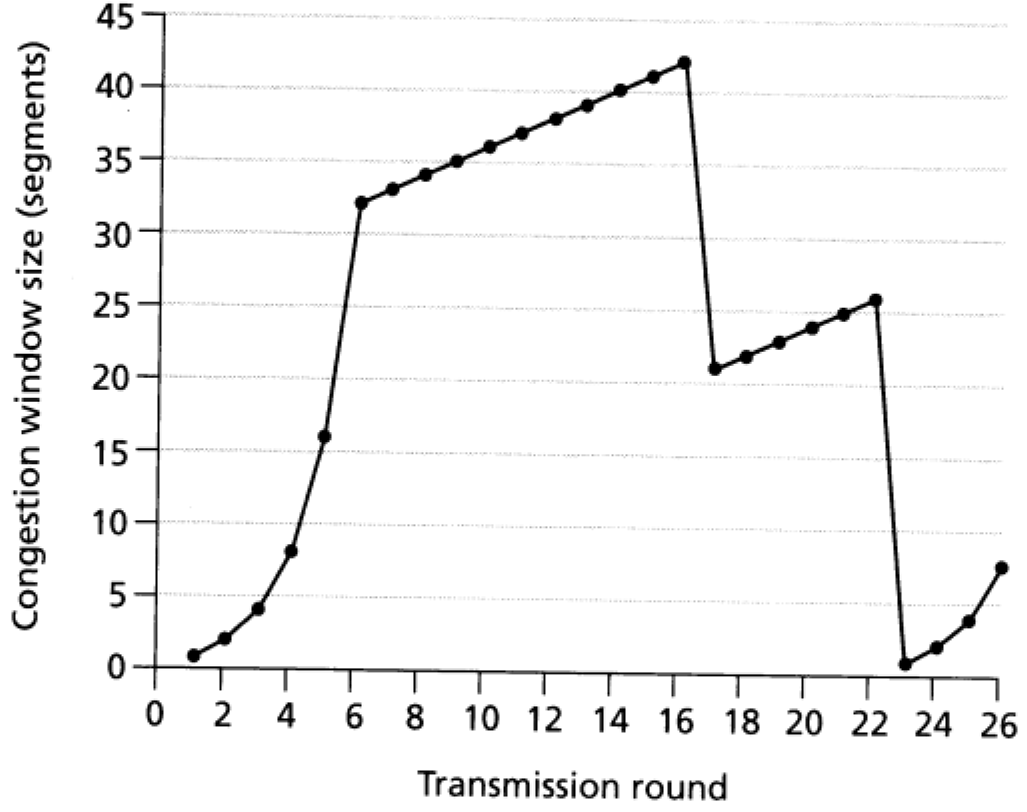
Reno TCP

A newer version of TCP, called Reno TCP, added a new state to the congestion-control FSM, called **the fast-recovery state**



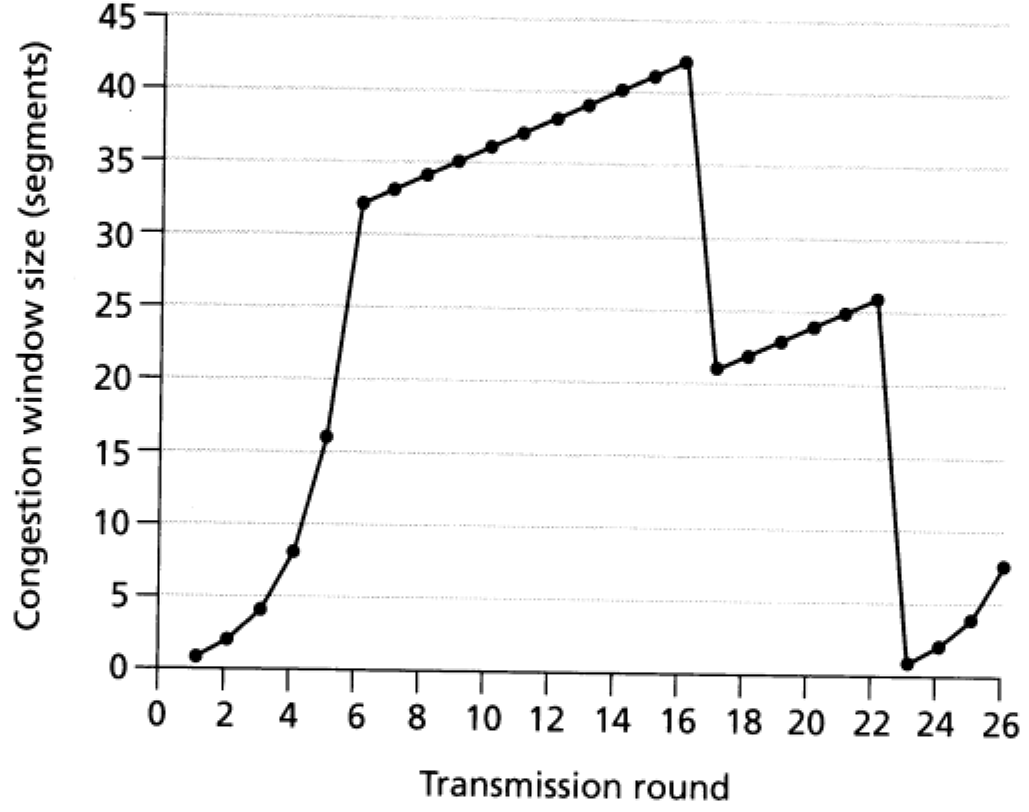
TCP Congestion Window

- Identify time intervals where TCP slow-start is operating.
    - TCP slowstart is operating in the intervals [1,6] and [23,26]
- Identify time intervals where TCP congestion-avoidance is operating
    - TCP congestion avoidance is operating in the intervals [6,16] and [17,22]
- After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout event?
    - After the 16th transmission round, packet loss is recognized by a triple duplicate ACK.
    - If there was a timeout, the congestion window size would have dropped to 1.
- After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout event?
    - After the 22nd transmission round, segment loss is detected due to timeout, and hence the congestion window size is set to 1 (remember that this means that TCP can send up to 1 MSS).



TCP Congestion Window

- What is the *ssthreshold* value at the first transmission round?
  - The threshold is initially 32, since it is at this window size that slow-start stops and congestion avoidance begins.
- What is the *ssthreshold* value at the 18th transmission round?
  - The threshold is set to half the value of the congestion window when packet loss is detected.
  - When loss is detected during transmission round 16, the congestion windows size is 42.
  - Hence the threshold is 21 during the 18th transmission round.
- What is the *ssthreshold* value at the 24th transmission round?
  - The threshold is set to half the value of the congestion window when packet loss is detected. When loss is detected during transmission round 22, the congestion windows size is 26. Hence the threshold is 13 during the 24th transmission round.
- What will be the values of *cwind* and *ssthreshold* if packet loss is detected after the 26th round by receipt of triple duplicate ACKs?
  - Congestion window and threshold will be set to half the current value of the congestion window (8) when loss occurred. Thus new values of the threshold and window will be 4.



Reference

Forouzan, A. Behrouz. ***Data Communications & Networking***. 5th Edition. Tata McGraw-Hill Education.

Chapter 24 Transport Layer Protocols

Topic: 24.1, 24.2, 24.3