TRANSPORT LAYER

Slides compiled by: Sanghamitra De

Transmission Control Protocol (TCP)

TCP

- TCP lies between the application layer and the network layer.
- TCP serves as the intermediary between the application programs and the network operations.

TCP Services

- **Process-to-Process Communication**
- **Stream Delivery Service**
  - ✓ ***Sending and Receiving Buffers*** *(*circular array of 1-byte locations*)*
  - ✓ ***Segments***
- **Full-Duplex Communication**
- **Multiplexing and Demultiplexing**
- **Connection-Oriented Service**
- **Reliable Service**

Table 15.1 *Well-known Ports used by TCP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day

Table 15.1 *Well-known Ports used by TCP (continued)*

Port	*Protocol*	*Description*
19	Chargen	Returns a string of characters
20 and 21	FTP	File Transfer Protocol (Data and Control)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol

Figure 15.2 *Stream delivery*

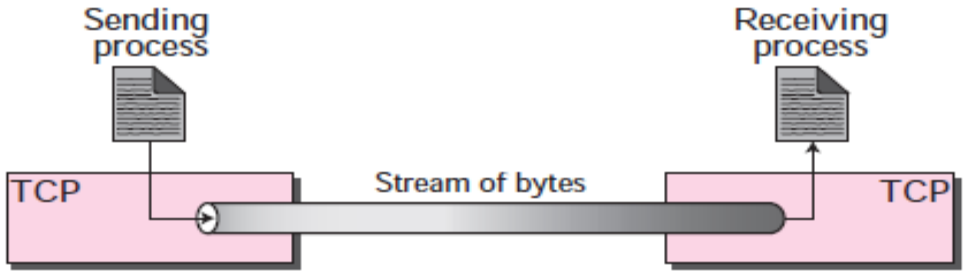


Figure 15.3 *Sending and receiving buffers*

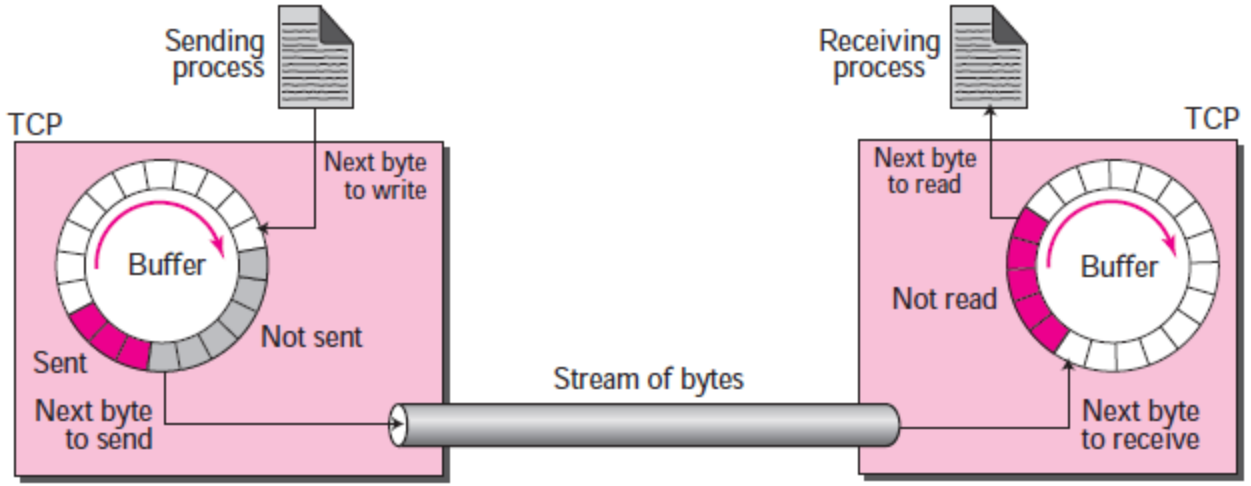
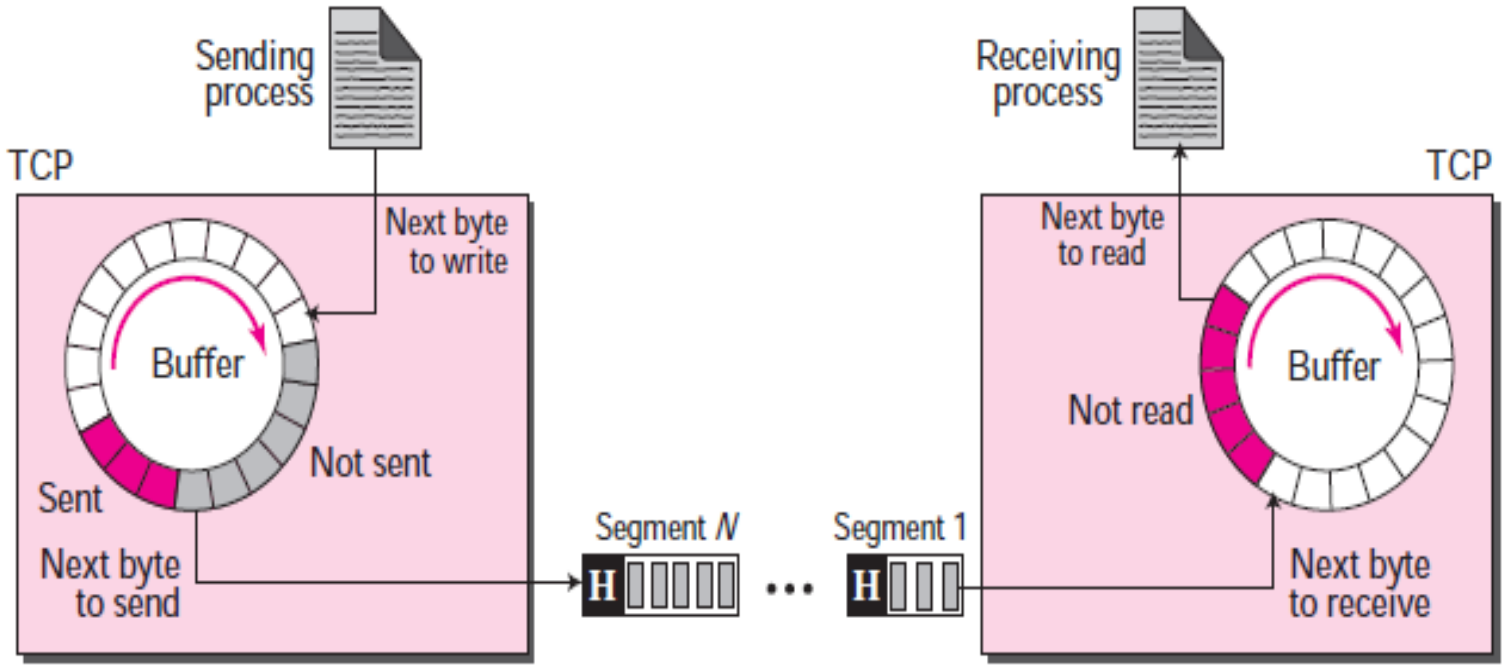


Figure 15.4 *TCP segments*

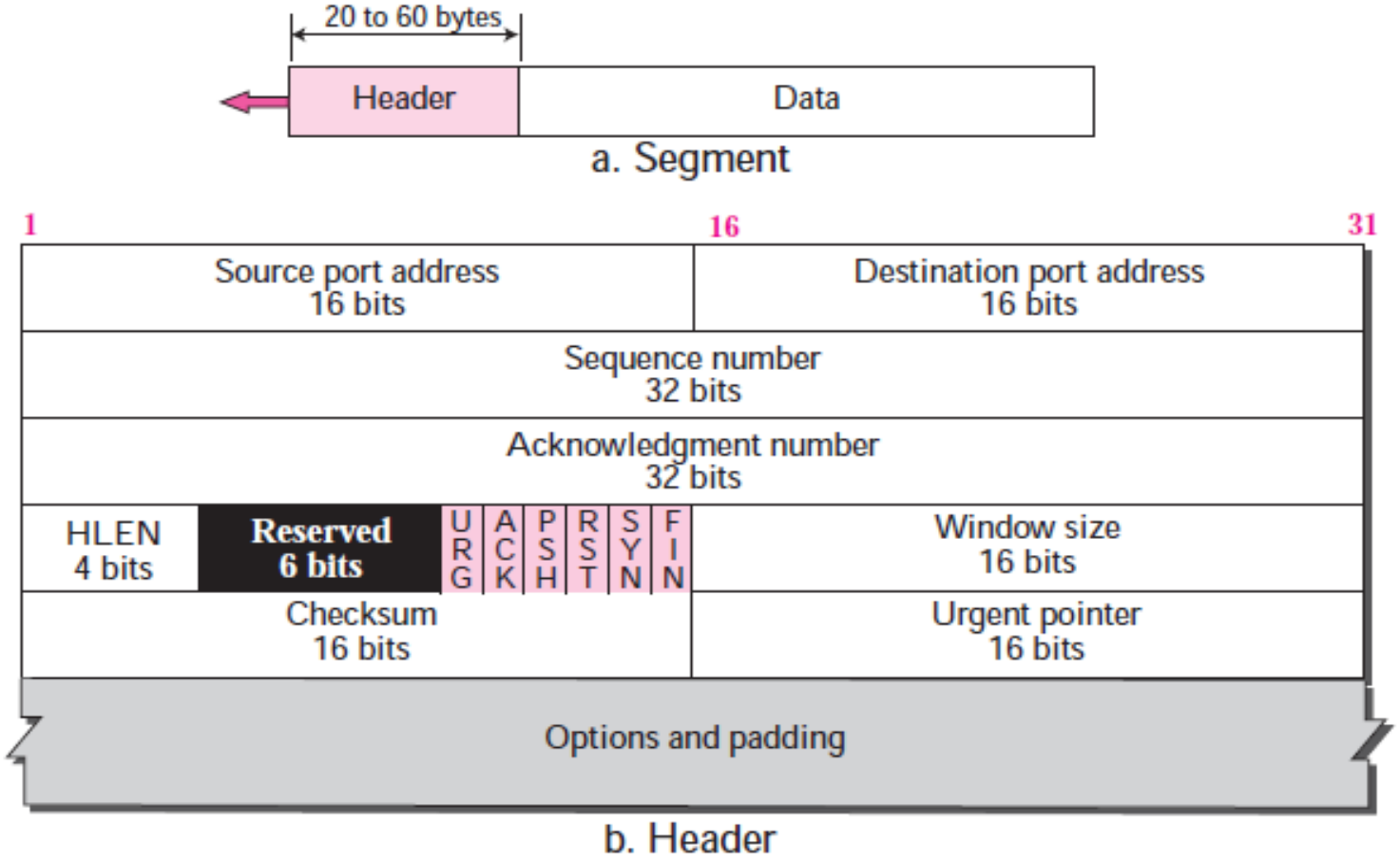


TCP Features

- **Numbering System**
  - ✓ ***Byte Number***
  - ✓ ***Sequence Number***
  - ✓ ***Acknowledgment Number***
- **Flow Control**
- **Error Control**
- **Congestion Control**

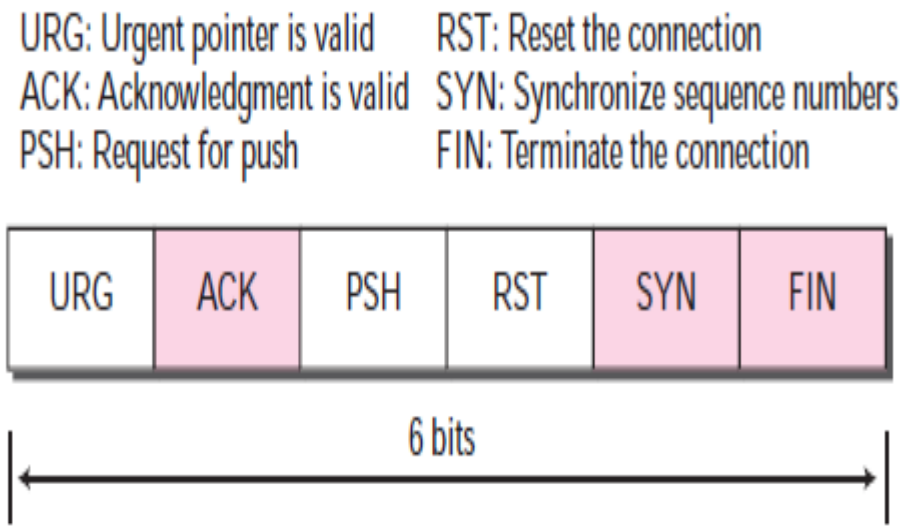
TCP Segment Format

Figure 15.5 *TCP segment format*



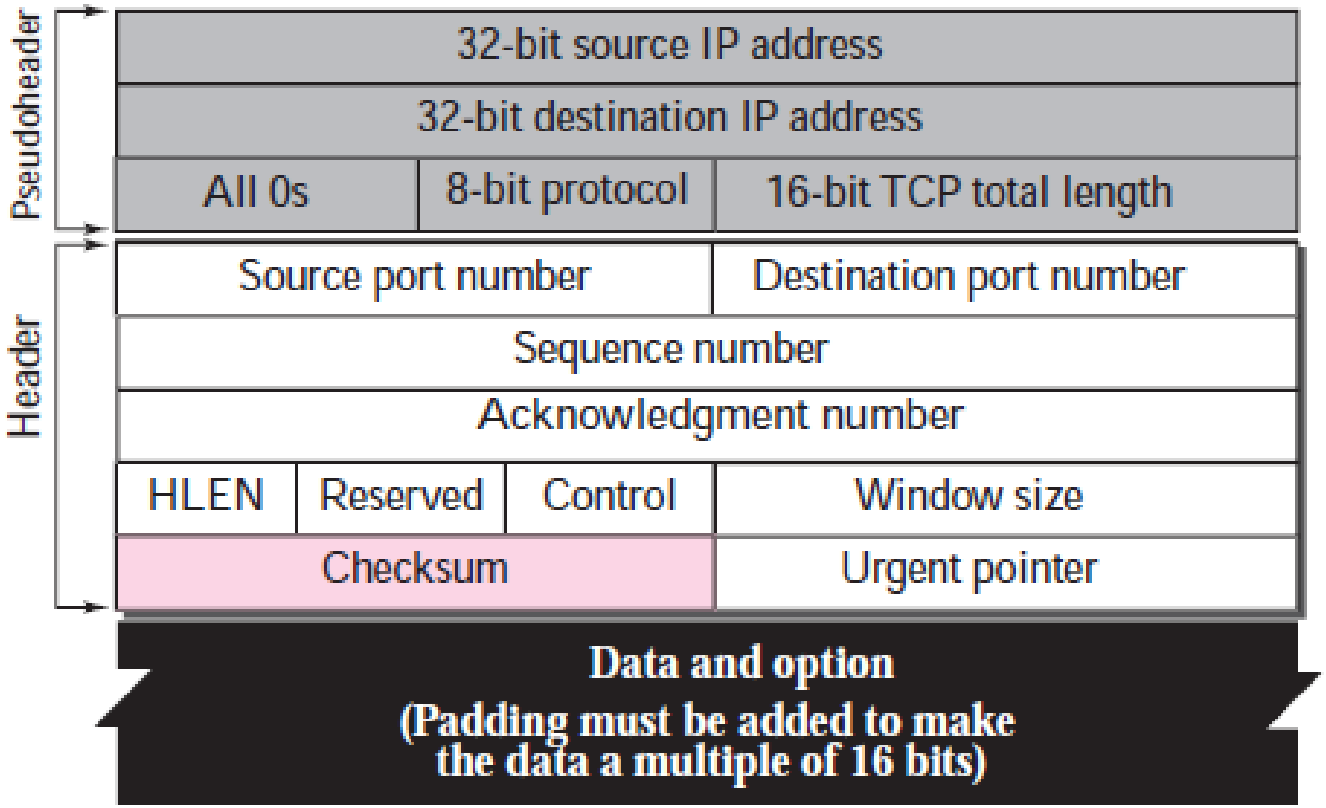
Control field in TCP Segment Format

Figure 15.6 *Control field*



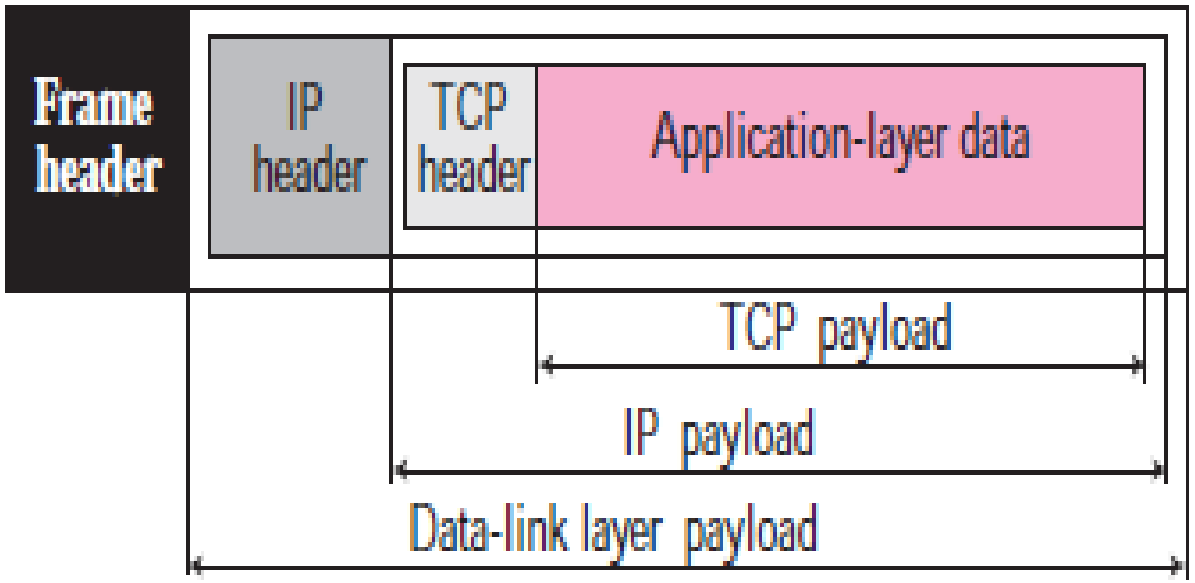
TCP Segment Format

Figure 15.7 *Pseudoheader added to the TCP datagram*



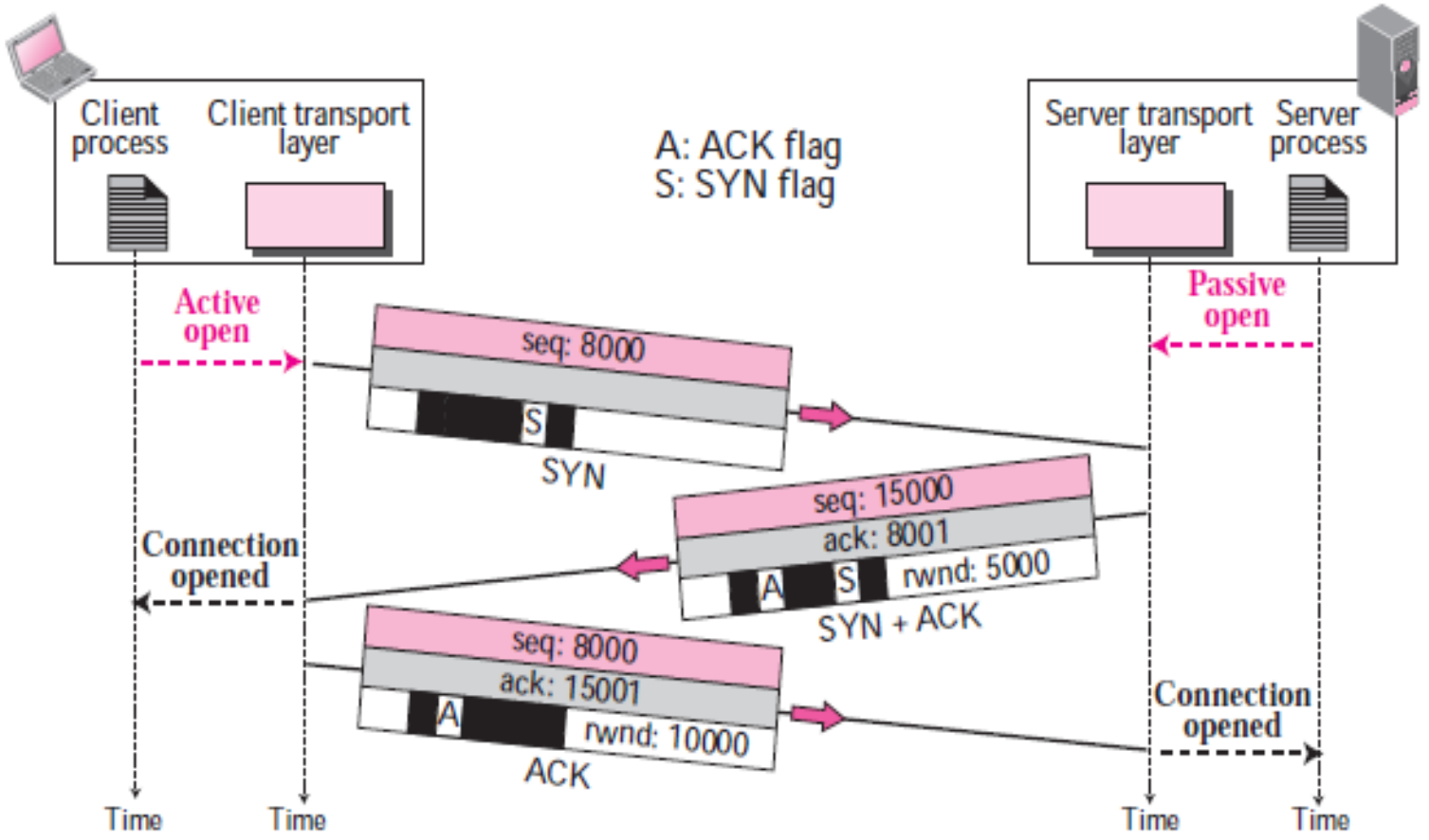
TCP Segment Encapsulation

Encapsulation



TCP Connection Establishment

Figure 15.9 *Connection establishment using three-way handshaking*

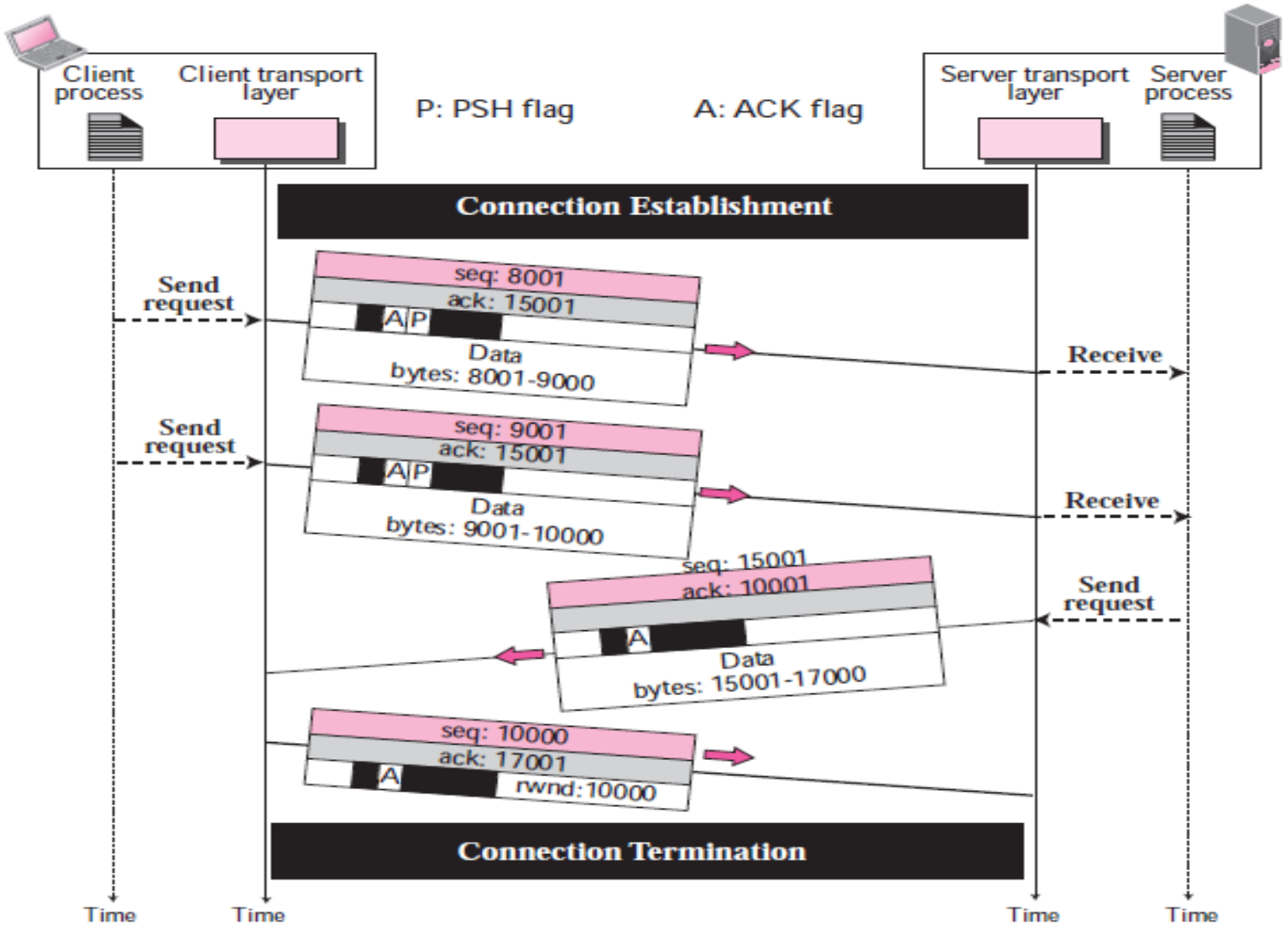


Uncommon Issues in 3-way handshake

- ***Simultaneous Open***
- ***SYN Flooding Attack***

TCP Data Transfer

Figure 15.10 *Data transfer*

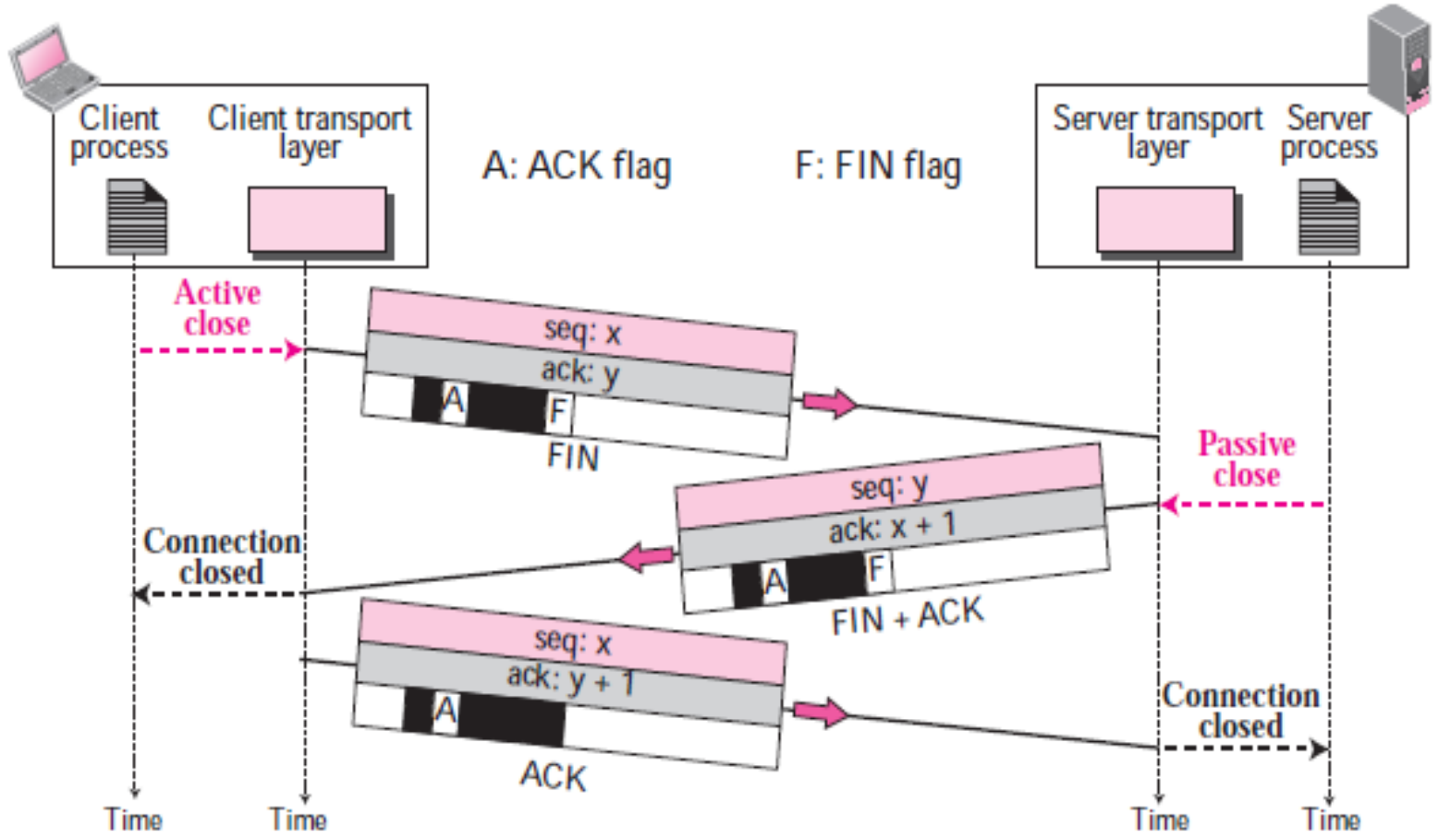


TCP Data Transfer

- ***Pushing Data***
- ***Urgent Data***

TCP Connection Termination

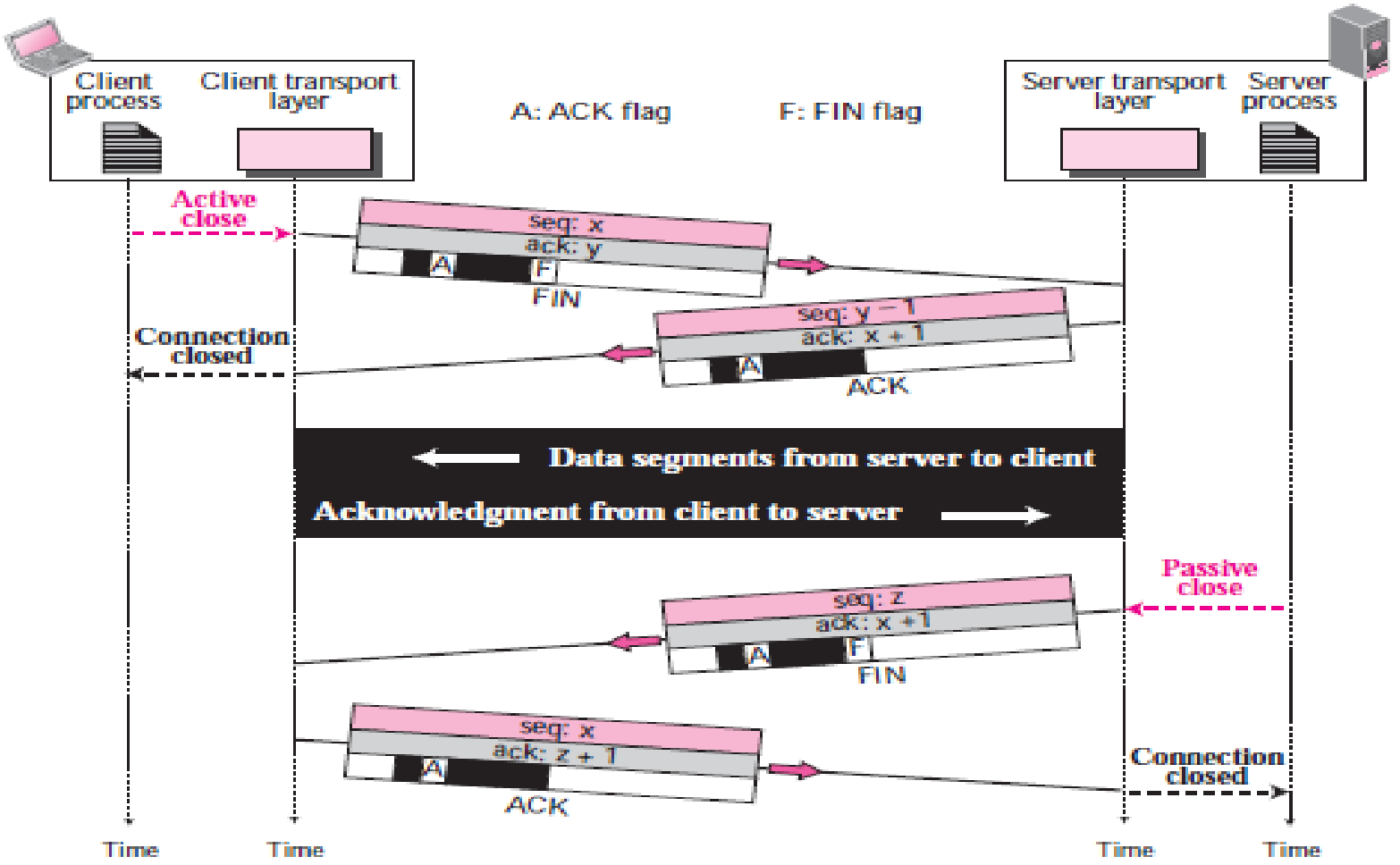
Figure 15.11 *Connection termination using three-way handshaking*



Half Close

- One end can stop sending data while still receiving data.
- This is called a half close.
- Either the server or the client can issue a half-close request.
- It can occur when the server needs all the data before processing can begin. E.g. sorting.

Figure 15.12 *Half-close*



Although the client has received sequence number y − 1 and is expecting y, the server sequence number is still y − 1. When the connection finally closes, the sequence number of the last ACK segment is still x, because no sequence numbers are consumed during data transfer in that direction.

Connection Reset

- Denying a Connection
  - ✓ TCP on one side has requested a connection to a non-existent port.
- Aborting a Connection
  - ✓ TCP may want to abort an existing connection due to an abnormal situation.
- Terminating an Idle Connection
  - ✓ TCP on one side may discover that the TCP on the other side has been idle for a long time.

TCP Message types

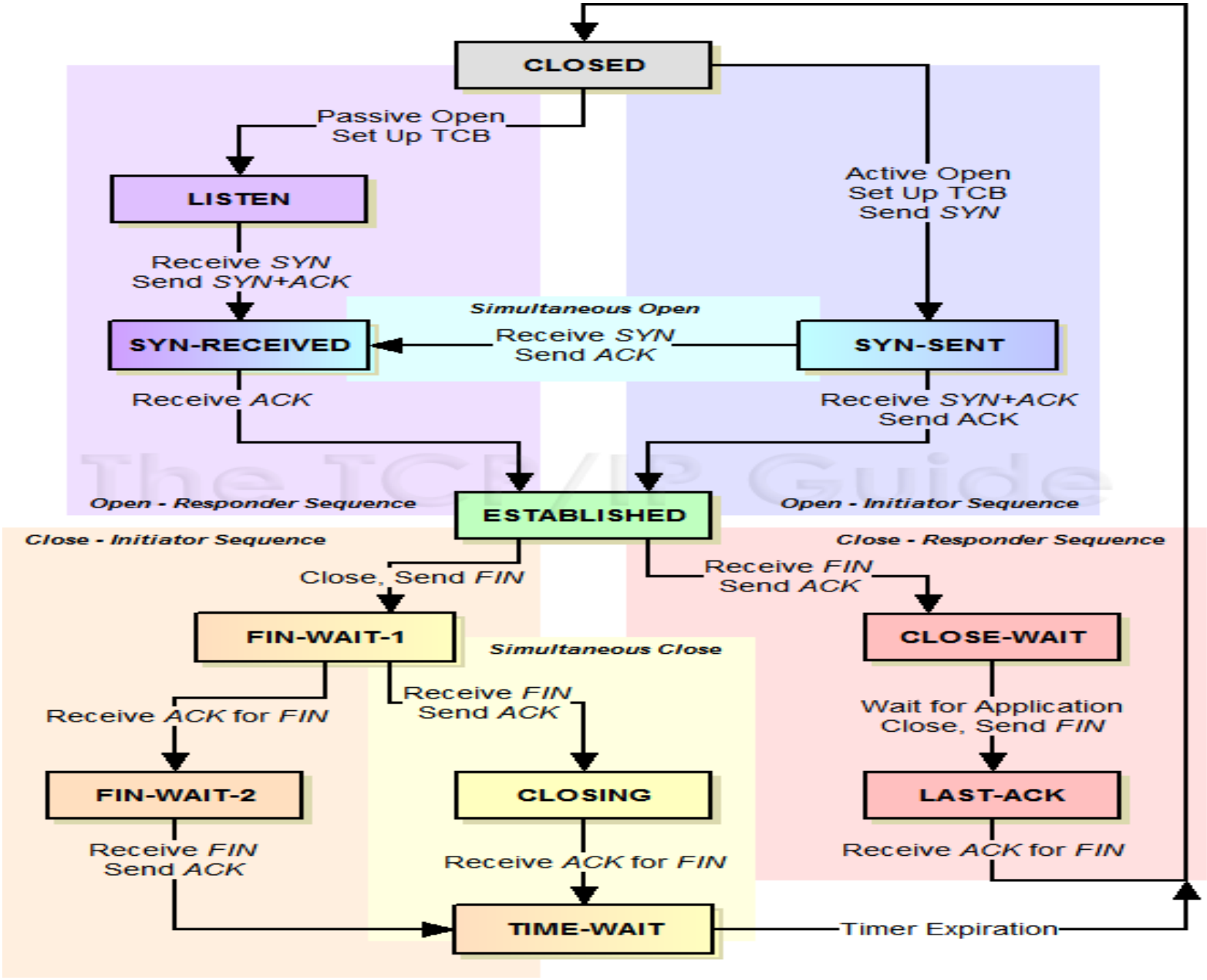
- ***SYN:*** A *synchronize* message, used to initiate and establish a connection. It is so named since one of its functions is to synchronizes sequence numbers between devices.

- ***FIN:*** A *finish* message, which is a TCP segment with the *FIN* bit set, indicating that a device wants to terminate the connection.

- ***ACK:*** An *acknowledgment*, indicating receipt of a message such as a *SYN* or a *FIN*.

TCP States & State Transition Diagram

States	State Description
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RECEIVED	SYN+ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
CLOSE-WAIT	First FIN received, ACK sent; waiting for application to close
LAST-ACK	Second FIN sent; waiting for ACK
FIN-WAIT-1	First FIN sent; waiting for ACK.
FIN-WAIT-2	ACK to first FIN received; waiting for second FIN
CLOSING	Both sides decided to close simultaneously
TIME-WAIT	Second FIN received, ACK sent; waiting for 2MSL time-out.


CLOSED
Passive Open
Set Up TCB
Active Open
Set Up TCB
Send SYN
LISTEN
Receive SYN
Send SYN+ACK
Simultaneous Open
SYN-RECEIVED
Receive SYN
Send ACK
SYN-SENT
Receive ACK
Receive SYN+ACK
Send ACK
Open - Responder Sequence
Open - Initiator Sequence
ESTABLISHED
Close - Initiator Sequence
Close - Responder Sequence
Close, Send FIN
Receive FIN
Send ACK
FIN-WAIT-1
Simultaneous Close
CLOSE-WAIT
Receive ACK for FIN
Receive FIN
Send ACK
Wait for Application
Close, Send FIN
FIN-WAIT-2
CLOSING
LAST-ACK
Receive FIN
Send ACK
Receive ACK for FIN
Receive ACK for FIN
TIME-WAIT
Timer Expiration

Flow Control in TCP

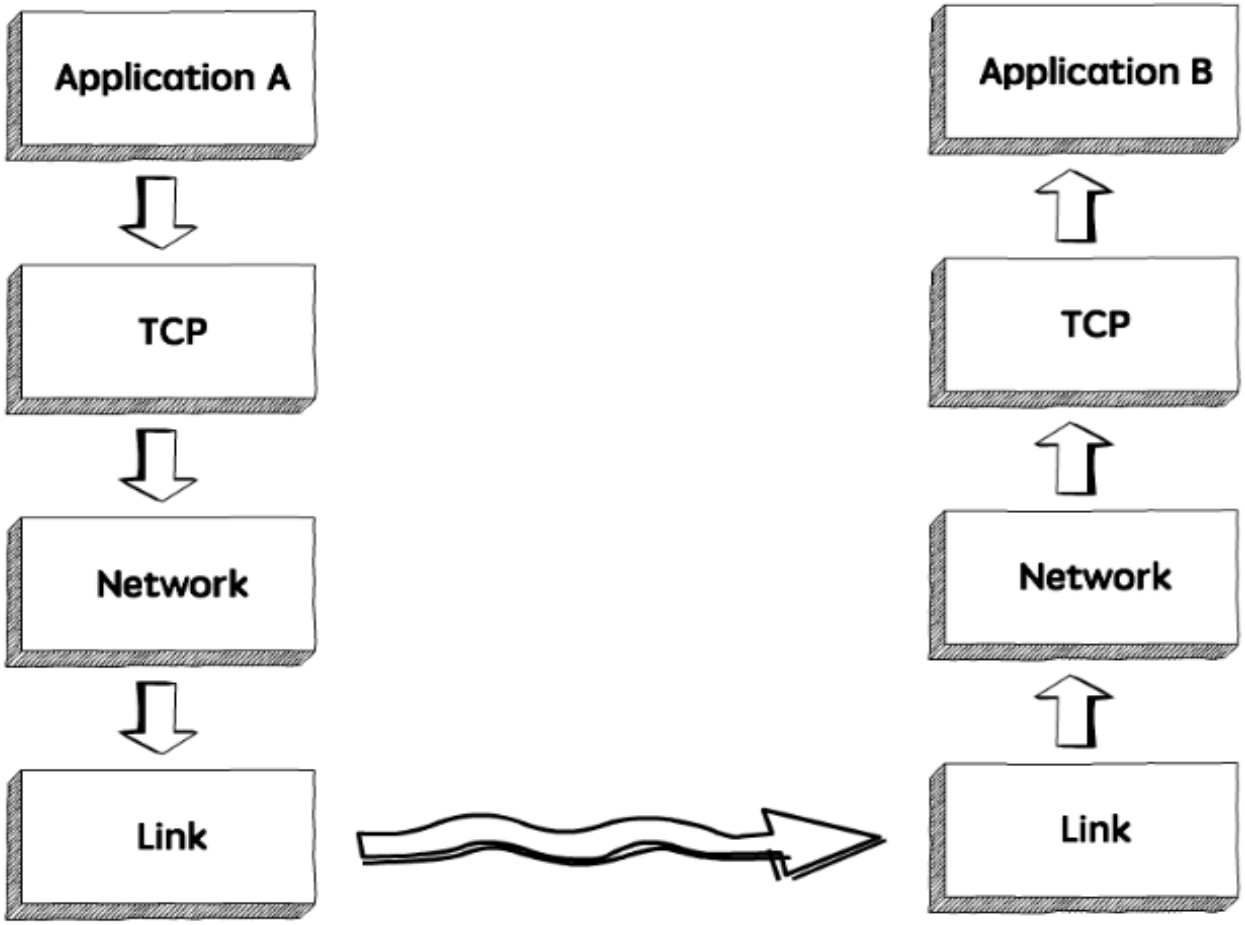
- TCP's flow control is a mechanism to ensure the sender is not overwhelming the receiver with more data than it can handle.
- A sliding window is used to make transmission more efficient as well as to control the flow of data so that the destination does not become overwhelmed with data.
- TCP sliding windows are byte-oriented & is of variable size.
- *Receive Window (rwnd)*, is the spare room in the receive buffer.
- *Congestion Window* (cwnd) is a TCP state variable that limits the amount of data the TCP can send into the network before receiving an ACK.
- With every ACK message the receiver advertises its current receive window.

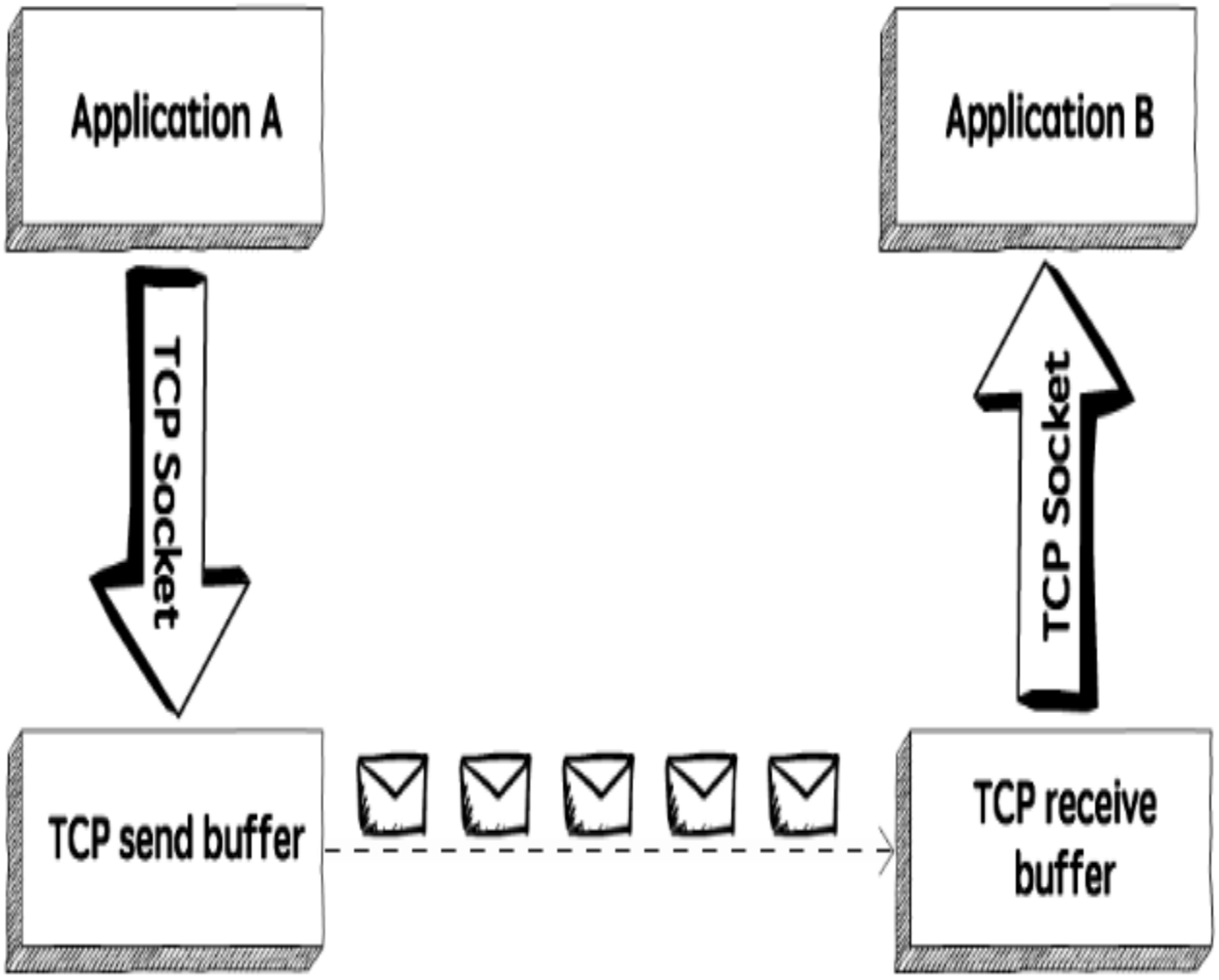
Flow Control in TCP

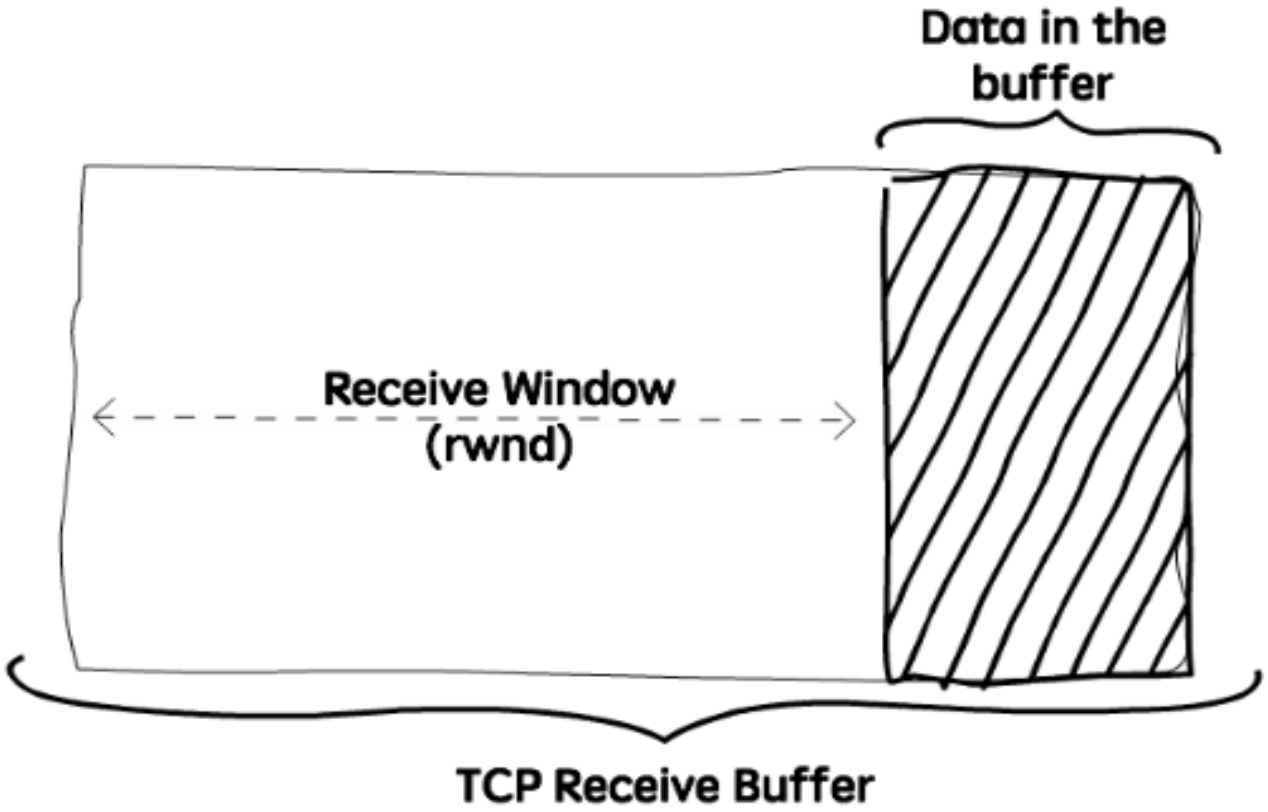
- The size of the window at one end is determined by the lesser of two values: *receiver window (rwnd)* or *congestion window (cwnd)*.

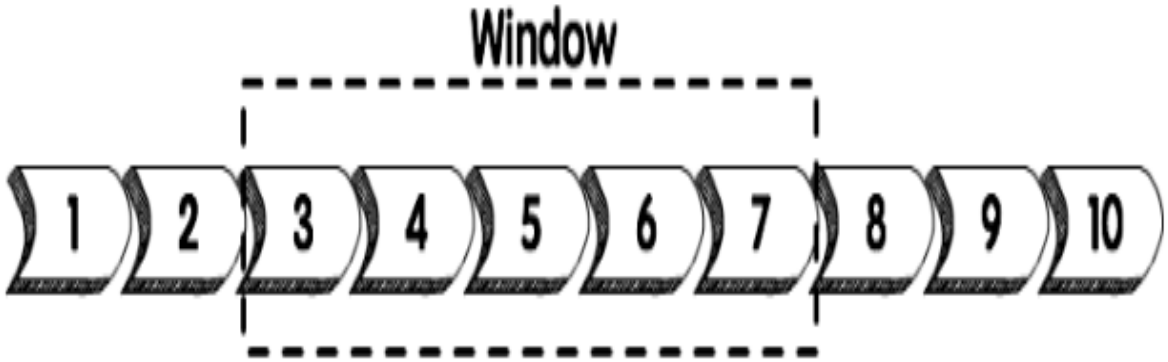
  window size= min(rwnd, cwnd)

- Sliding window in TCP spans a portion of the buffer containing bytes received from the process.
- Bytes inside the window are the bytes that can be in transit & can be sent without worrying about acknowledgment.
- The imaginary window has two walls: left and right.
- Window activities: *opened*, *closed*, or *shrunk*.

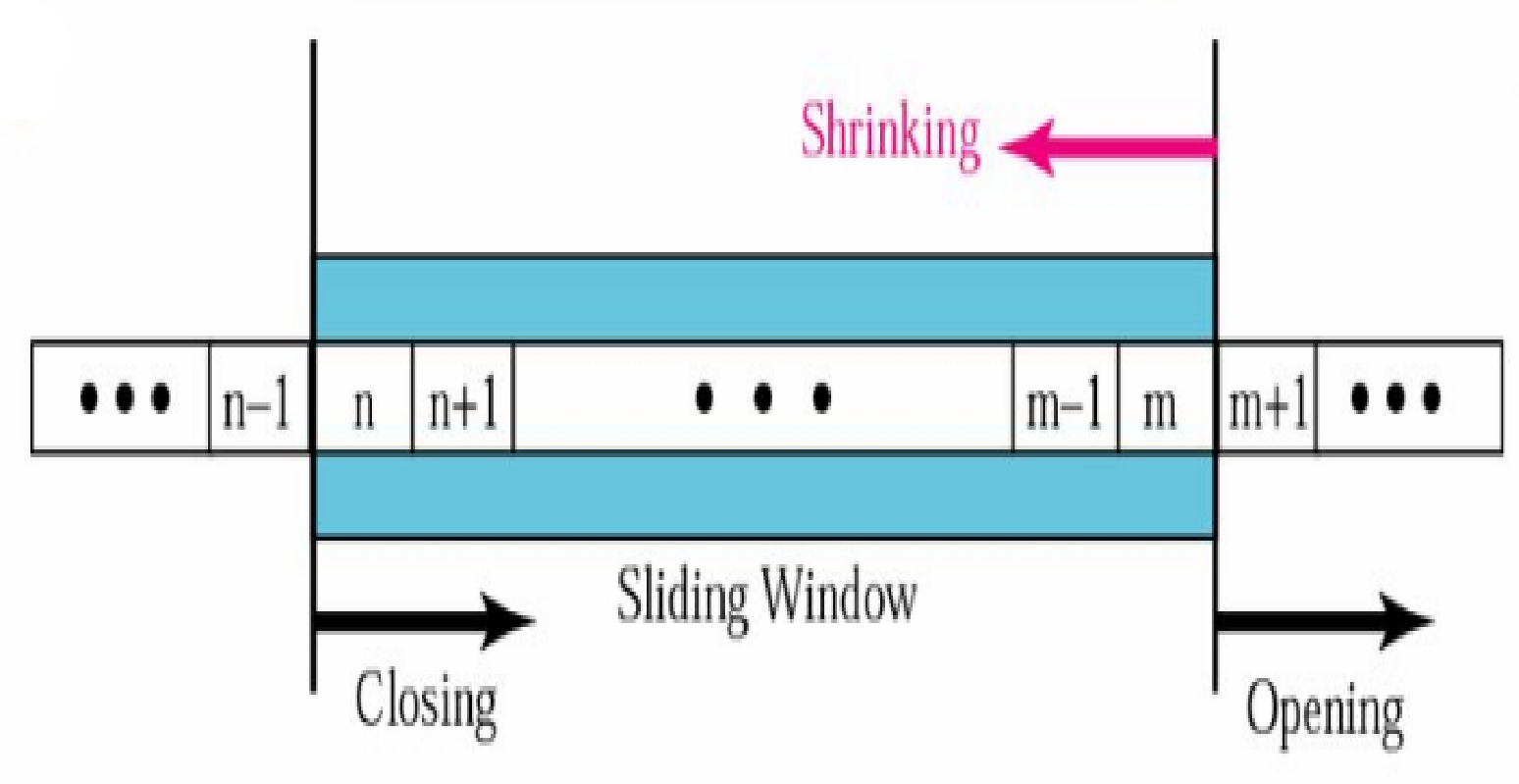

Application A
TCP
Network
Link
Application B
TCP
Network
Link

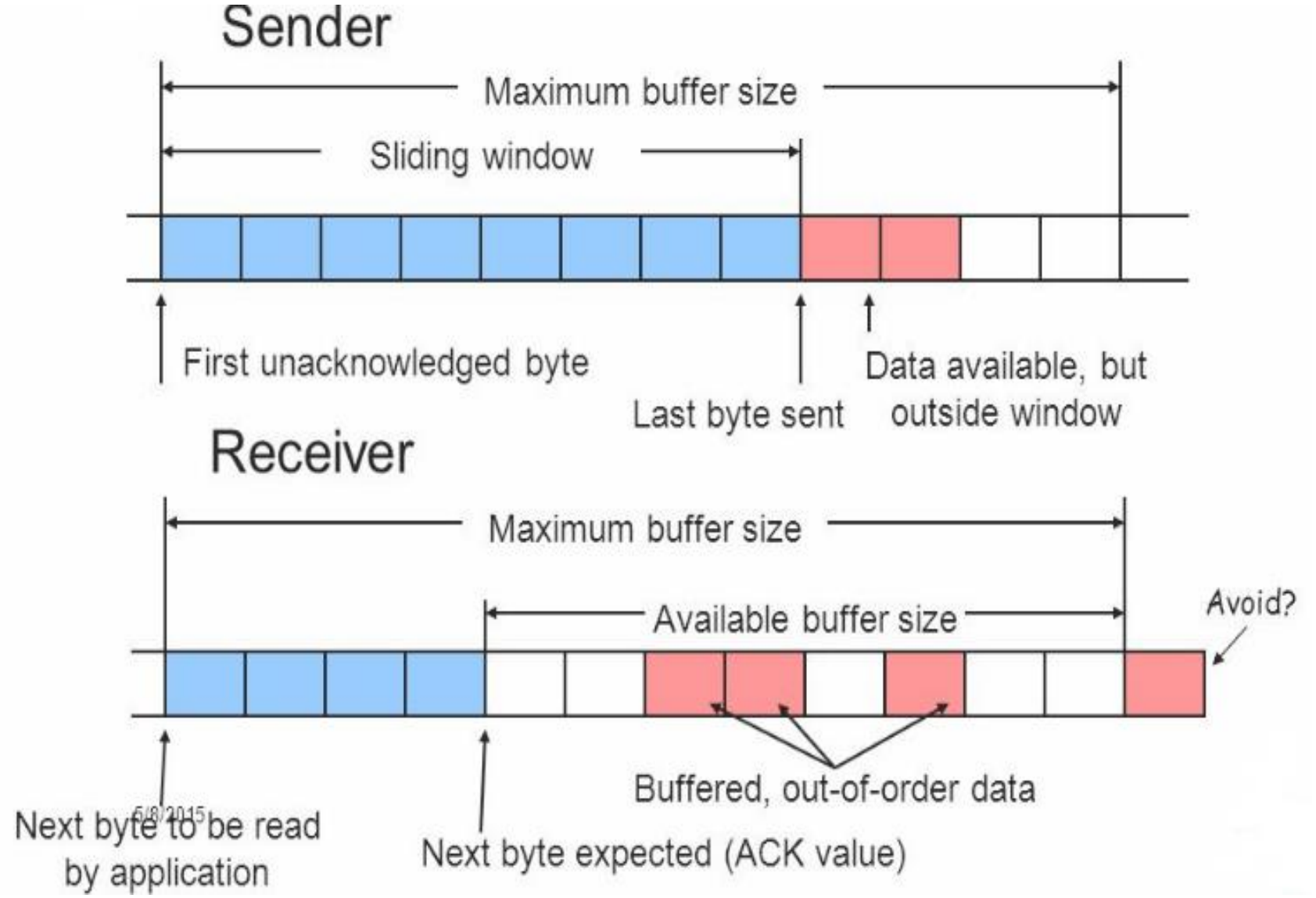

Application A
Application B
TCP Socket
TCP Socket
TCP send buffer
TCP receive buffer


Data in the buffer
Receive Window (rwnd)
TCP Receive Buffer



Example of a sliding window. As soon as packet 3 is acked, we can slide the window to the right and send the packet 8.


Shrinking
n-1
n
n+1
m-1
m
m+1
Sliding Window
Closing
Opening


Sender
Maximum buffer size
Sliding window
First unacknowledged byte
Last byte sent
Data available, but outside window
Receiver
Maximum buffer size
Available buffer size
Avoid?
Buffered, out-of-order data
Next byte to be read by application
Next byte expected (ACK value)

Error Control in TCP

- TCP is a reliable transport layer protocol
- TCP provides reliability using error control
- Mechanisms:
  - ✓ Checksum
  - ✓ Acknowledgment
  - ✓ Time-out
- Detects corrupt segments, lost segments, out-of-order segments, and duplicated segments. Also corrects errors after detection

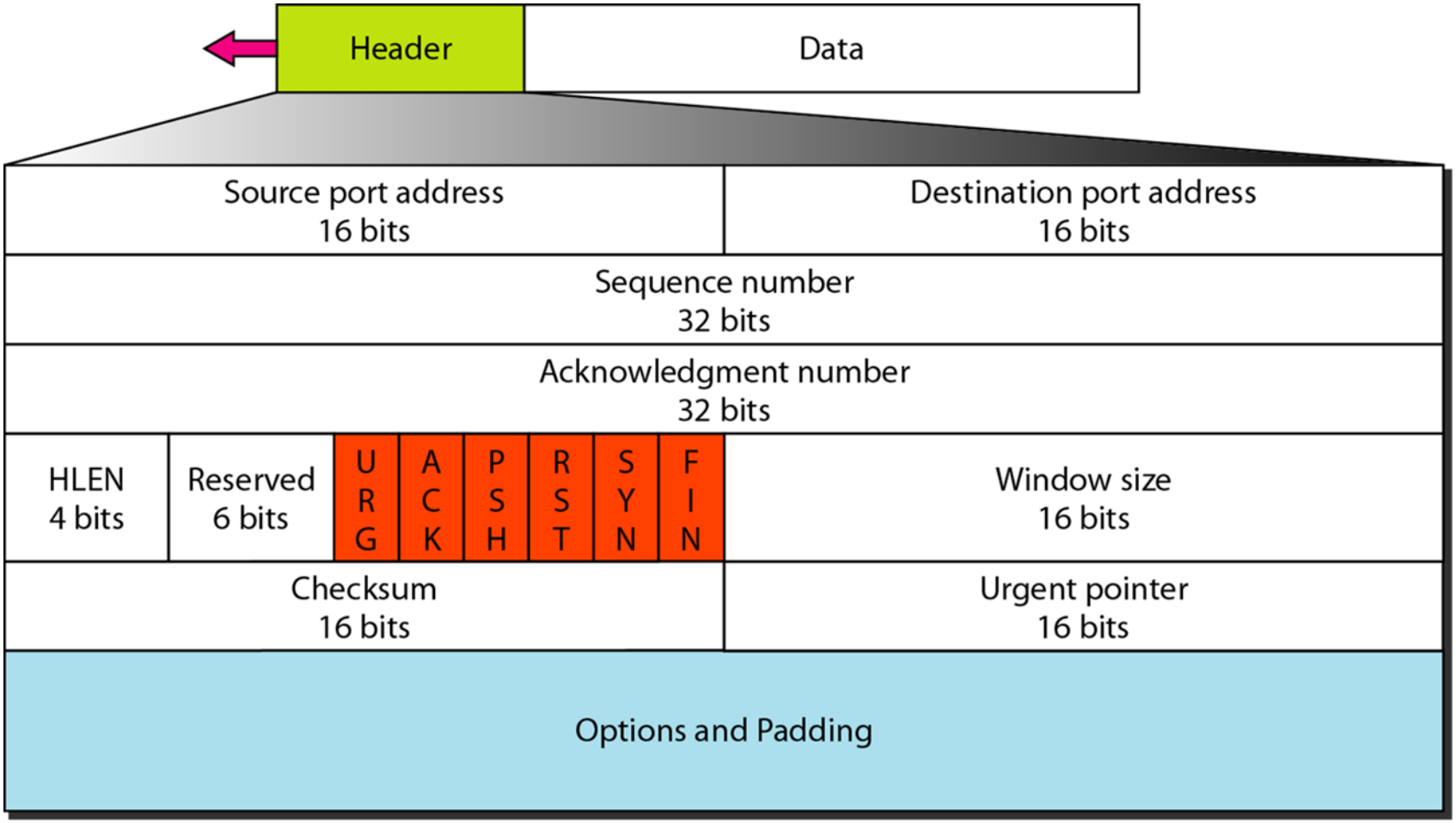
Error Control in TCP

Error Control Purpose in TCP:

- Detect corrupt segments
- Detect lost segments
- Detect out-of-order segments
- Detect duplicate segments

TCP can also correct errors after they are detected.

TCP segment format



Checksum

- TCP uses a 16-bit checksum that is mandatory in every segment.
- Checksum field in each segment required to check for a corrupted segment
- If segment is corrupted, it is discarded by the destination TCP and is considered as lost.
- TCP checksums are calculated over the entire segment, both the header and the data.
- The ENTIRE segment is divided into 16-bit pieces and add up all of them.

Acknowledgment

- TCP uses acknowledgments to confirm the receipt of data segments.
- Control segments that carry no data but consume a sequence number are also acknowledged.
- ACK segments are never acknowledged.

NOTES

- The bytes of data being transferred in each connection are numbered by TCP.
- The numbering starts with a randomly generated number.
- The value in the sequence number field of a segment defines the number of the first data byte contained in that segment.
- The value of the acknowlegement field in a segment defines the number of the next byte a party expects to receive.
- The acknowledgment number is cumulative.

Sequence Numbers

Segment 1	→	**Sequence Number: 10,001 (range: 10,001 to 11,000)**
Segment 2	→	**Sequence Number: 11,001 (range: 11,001 to 12,000)**
Segment 3	→	**Sequence Number: 12,001 (range: 12,001 to 13,000)**
Segment 4	→	**Sequence Number: 13,001 (range: 13,001 to 14,000)**
Segment 5	→	**Sequence Number: 14,001 (range: 14,001 to 15,000)**

NOTES

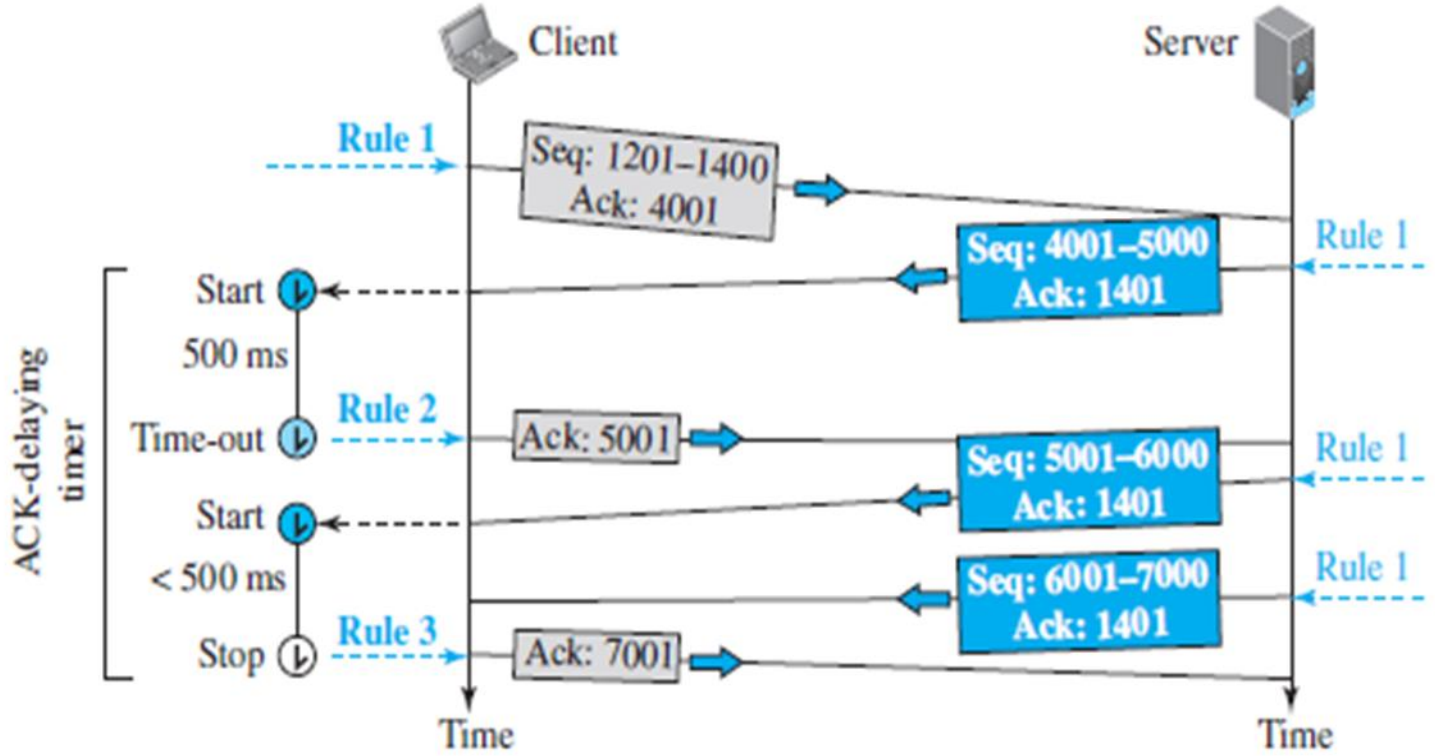
- A SYN segment cannot carry data, but it consumes one sequence number.
- A SYN + ACK segment cannot carry data, but does consume one sequence number.
- An ACK segment, if carrying no data, consumes no sequence number.
- The FIN segment consumes one sequence number if it does not carry data.
- The FIN + ACK segment consumes one sequence number if it does not carry data.

Some scenarios

- Normal Operation
- Lost Segment
- Fast Retransmission

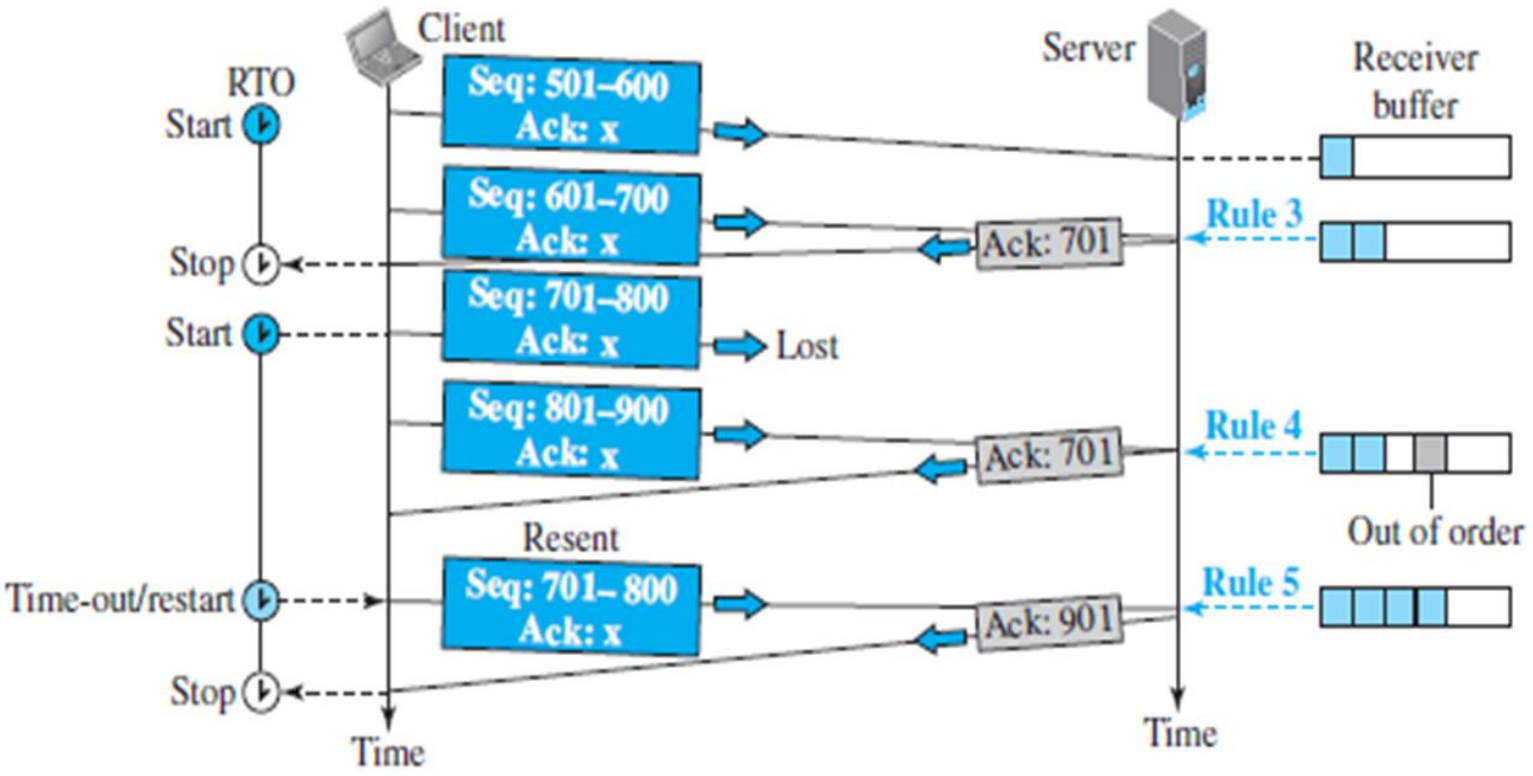
Scenario 1: Normal Operation

Figure *Normal operation*



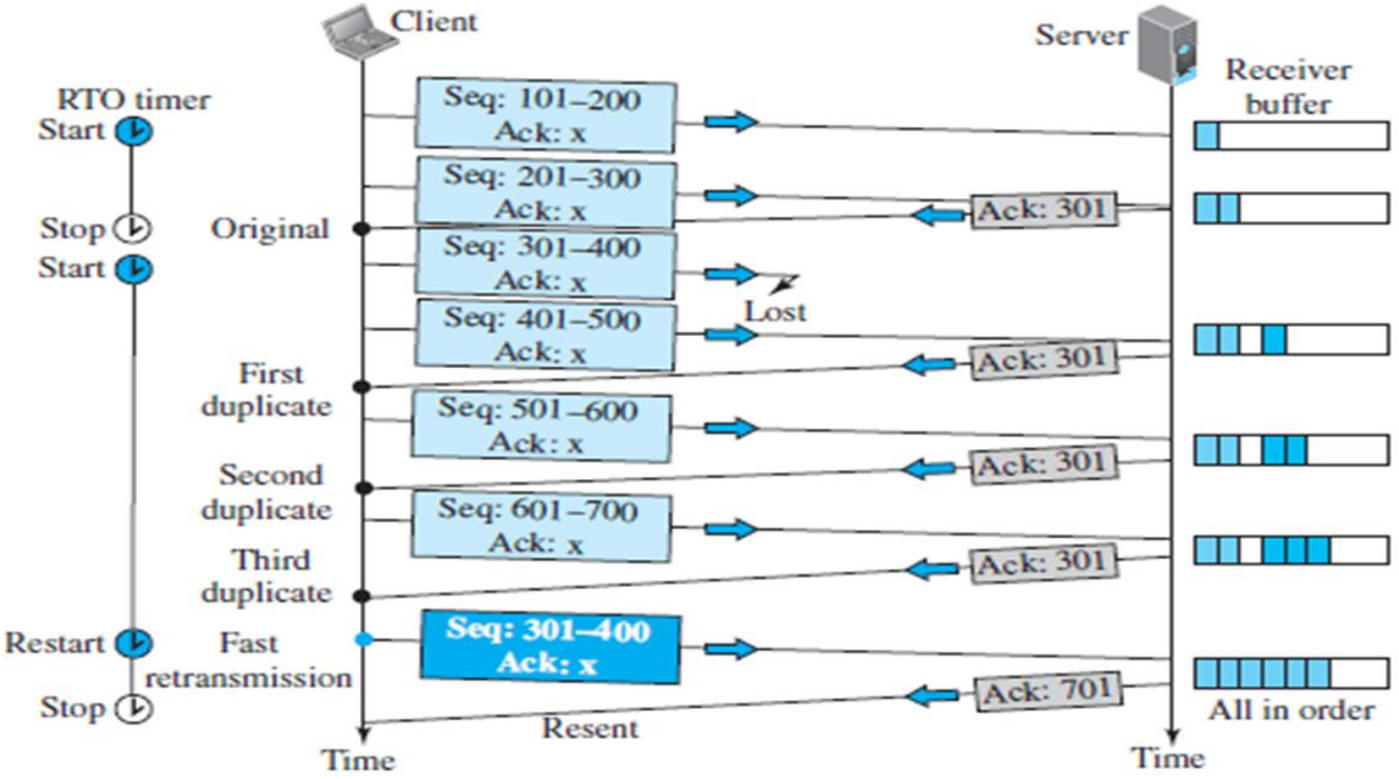
Scenario 2: Lost Segment

Figure *Lost segment*



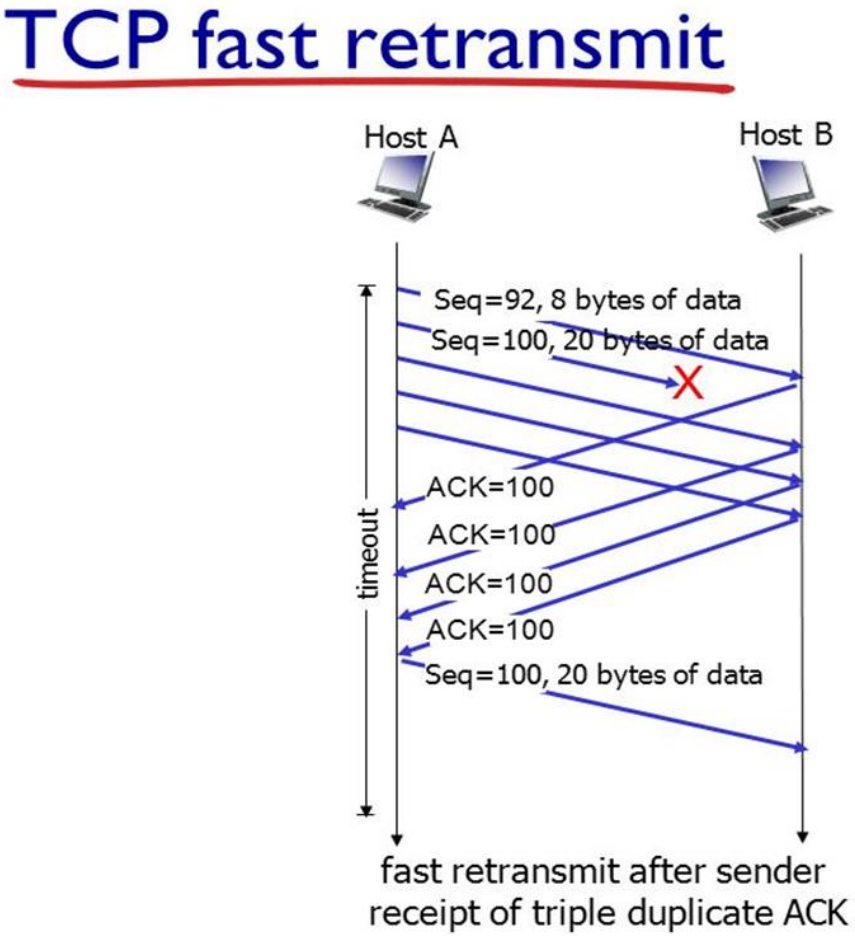
Scenario 3: Fast Retransmission

Figure *Fast retransmission*



The receiver TCP delivers only ordered data to the process.

Fast Retransmission



Why does TCP wait for three duplicate ACK before fast retransmit?

- Since TCP does not know whether a duplicate ACK is caused by a lost segment or just a reordering of segments, it waits for a small number of duplicate ACKs to be received. It is assumed that if there is just a reordering of the segments, there will be only one or two duplicate ACKs before the reordered segment is processed, which will then generate a new ACK. If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost. TCP then performs a retransmission of what appears to be the missing segment, without waiting for a retransmission timer to expire.

Retransmission & Time Out

- In modern implementations, a retransmission occurs if the retransmission timer expires or three duplicate ACK segments have arrived.
- No retransmission timer is set for an ACK segment.
- Data may arrive out of order and be temporarily stored by the receiving TCP, but TCP guarantees that no out-of-order segment is delivered to the process.

Congestion Control in TCP

Congestion Window

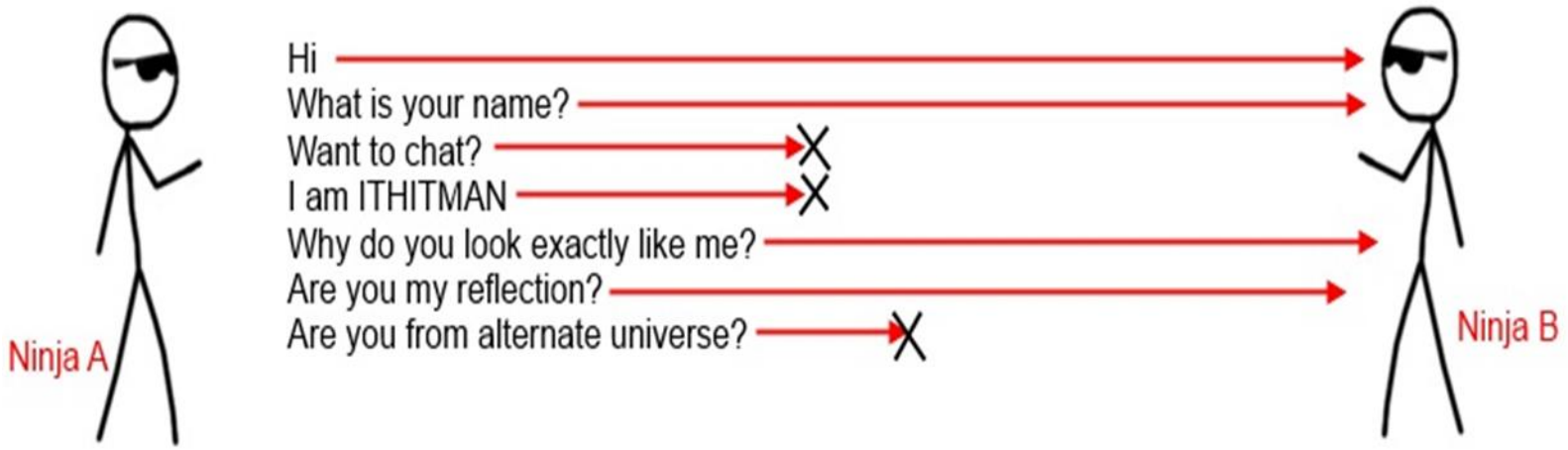
- Sender Window size = minimum(rwnd,cwnd)
- Where, rwnd = receiver-advertised window size & cwnd = congestion window size

Congestion Policy

TCP Policy for congestion handling:

- Slow Start, Exponential Increase
  - Slow start threshold (ssthresh)
- Congestion Avoidance, Additive increase
- Congestion Detection, Multiplicative decrease

Slow Start, Exponential Increase

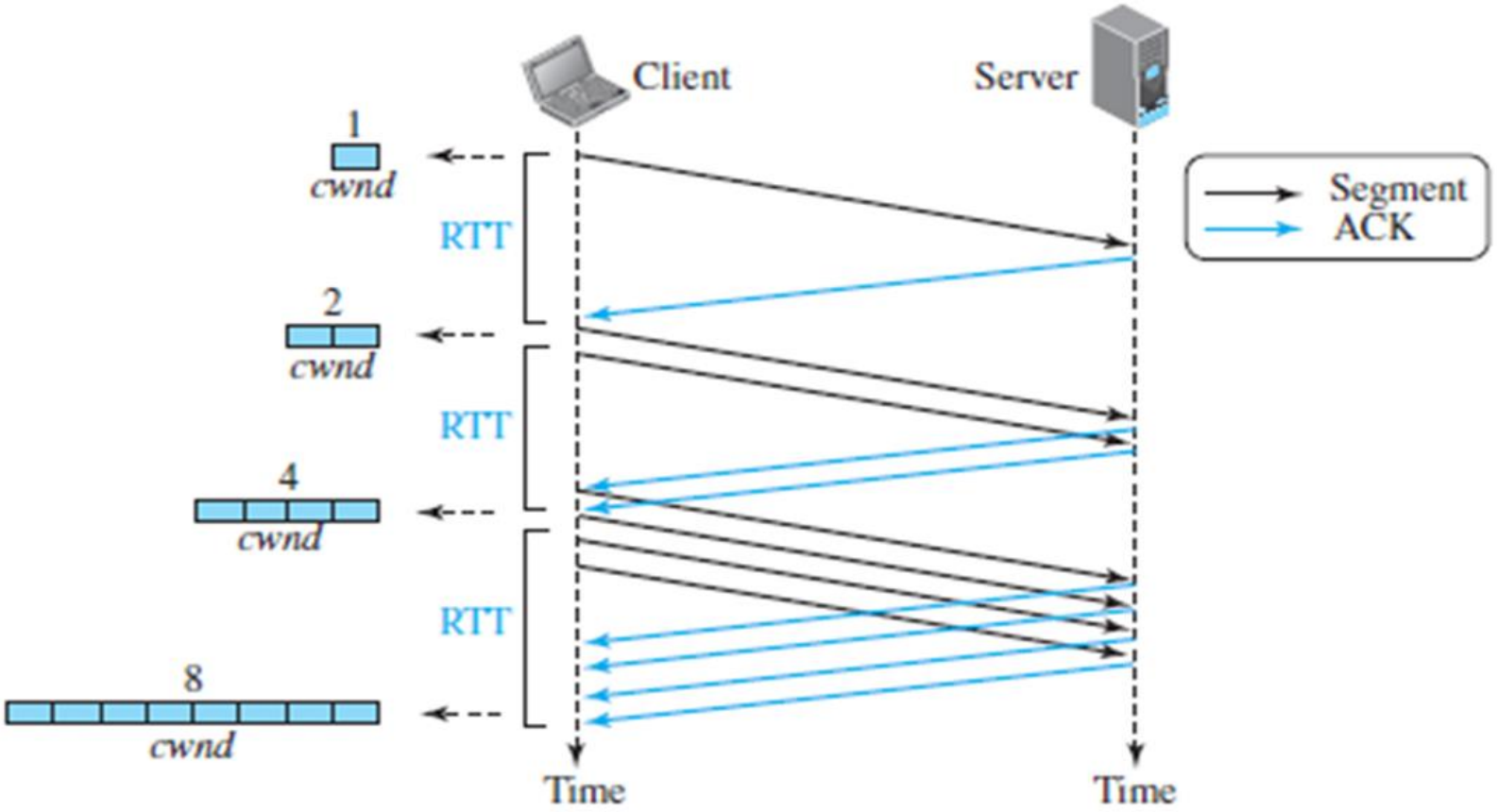


Too many sentences and words are being thrown in the air and the receiver (Ninja B) is bombarded with traffic. Ninja A needs to slow down his roll ....

Basic algorithm of the TCP Slow Start

- In the slow-start algorithm, size of the congestion window (cwnd) starts with one maximum segment size (MSS).
- Size of the congestion window increases exponentially until it reaches a threshold.
- Growth rate of the congestion window is *exponential* in terms of each round trip time (RTT).

Figure *Slow start, exponential increase*

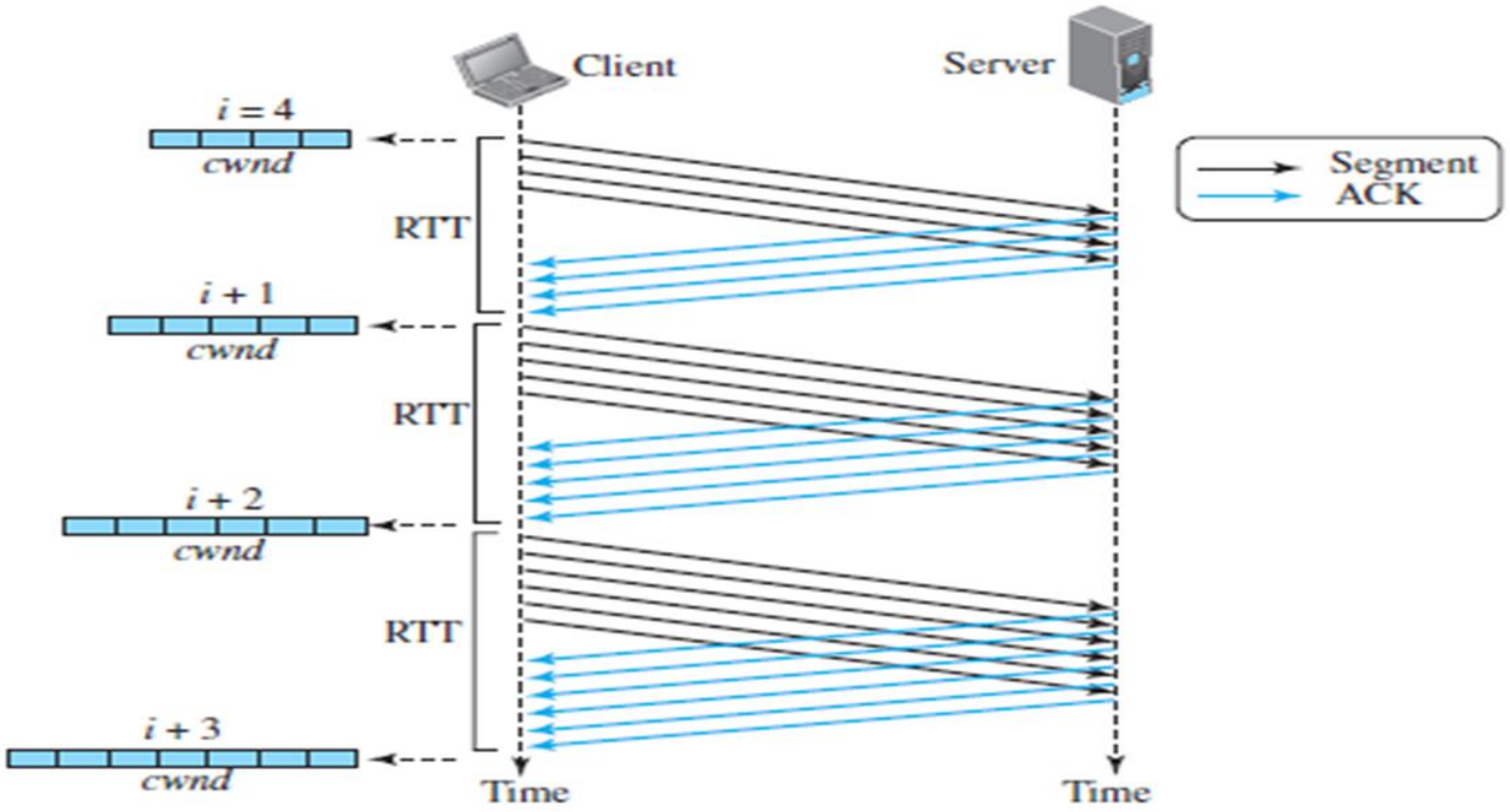


Start	→	$cwnd = 1 \rightarrow 2^0$
After 1 RTT	→	$cwnd = cwnd + 1 = 1 + 1 = 2 \rightarrow 2^1$
After 2 RTT	→	$cwnd = cwnd + 2 = 2 + 2 = 4 \rightarrow 2^2$
After 3 RTT	→	$cwnd = cwnd + 4 = 4 + 4 = 8 \rightarrow 2^3$

TCP Congestion Avoidance

- *Congestion Avoidance* uses a linear growth function **(additive increase).**
- Once **slow start threshold (ssthresh)** is reached, **cwnd** is increased by at most one segment per RTT.
- **cwnd window** continues to open with this **linear rate** until a congestion event is detected.
- When congestion is detected (through timeout and/or duplicate ACKs), **ssthresh** is set to half the cwnd
- On congestion detection, data rate is reduced in order to let the network recover.

Figure *Congestion avoidance, additive increase*



Start	→	$cwnd = i$
After 1 RTT	→	$cwnd = i + 1$
After 2 RTT	→	$cwnd = i + 2$
After 3 RTT	→	$cwnd = i + 3$

TCP Congestion Detection

- In a long TCP connection, ignoring slow-start states and short exponential growth during fast recovery, the TCP congestion window is **cwnd = cwnd + (1 / cwnd)** when an ACK arrives (congestion avoidance), and **cwnd = cwnd / 2** when congestion is detected, as though SS does not exist and the length of FR is reduced to zero. The first is called **additive increase; the second is called multiplicative decrease**.

- If **detection** is **by time-out,** a **new slow-start phase** starts.
- If **detection** is **by 3 ACKs,** a **new congestion avoidance phase** starts.

Saw-tooth pattern of the changing congestion window size

Figure 15.37 *Congestion example*

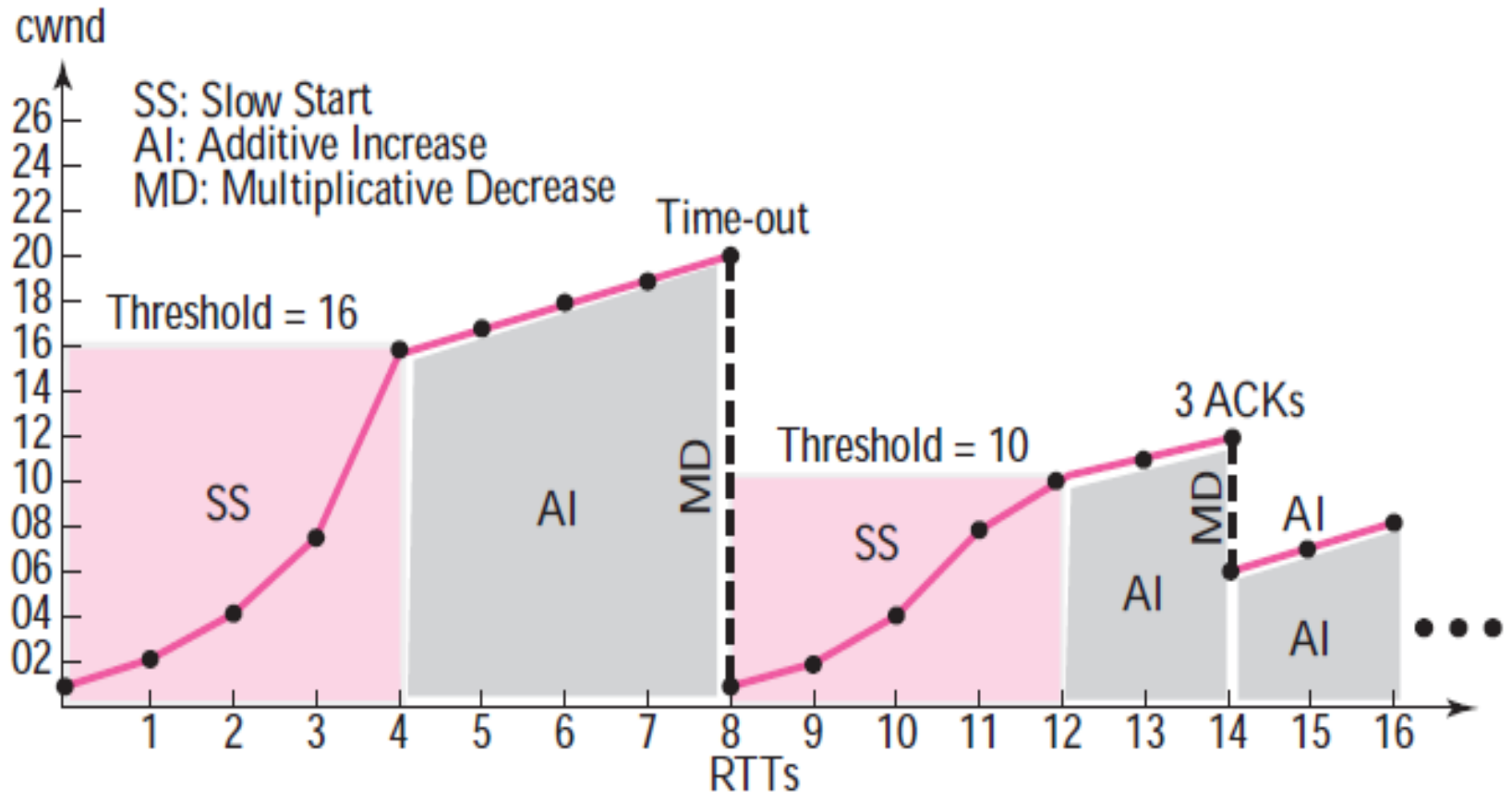
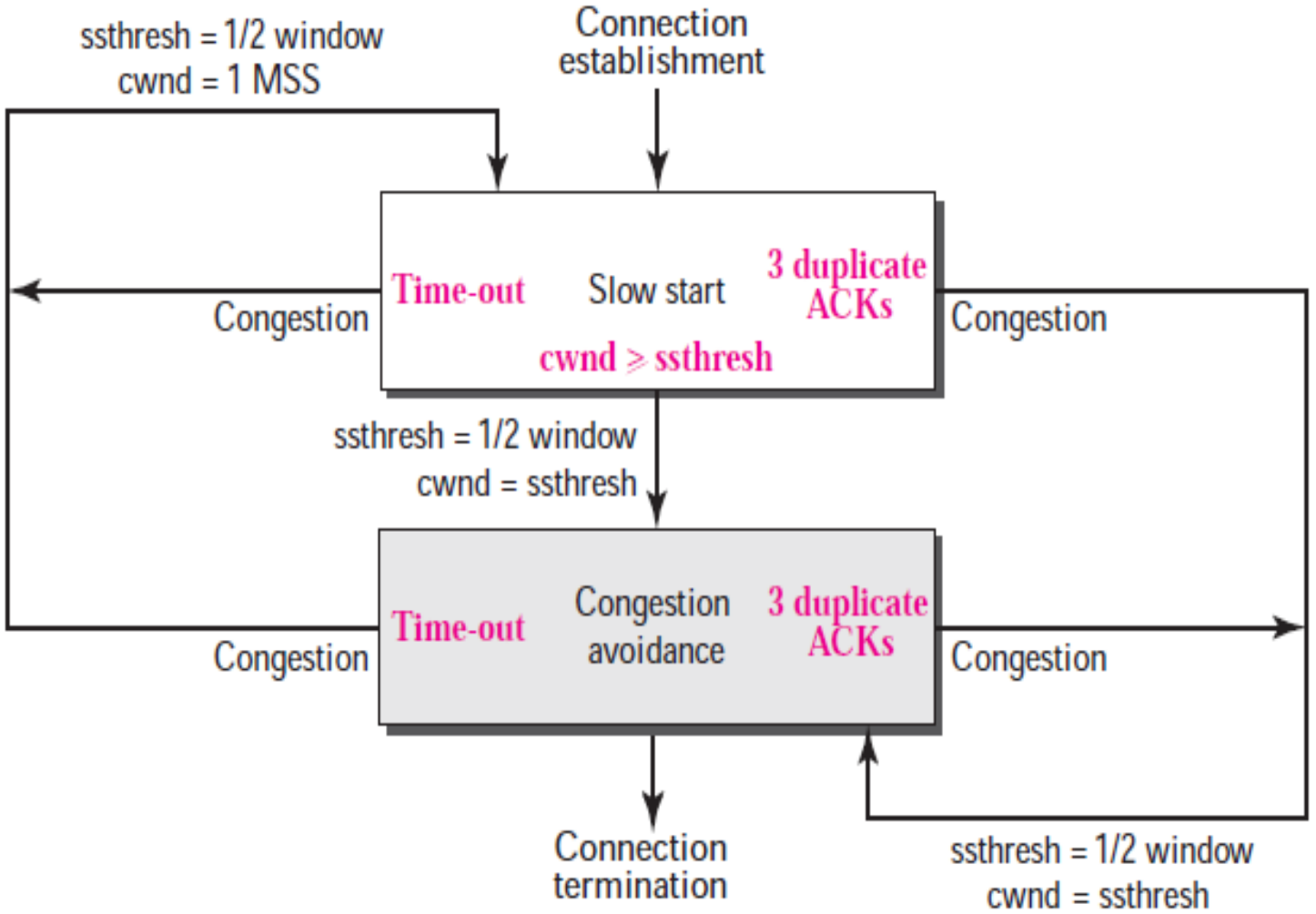
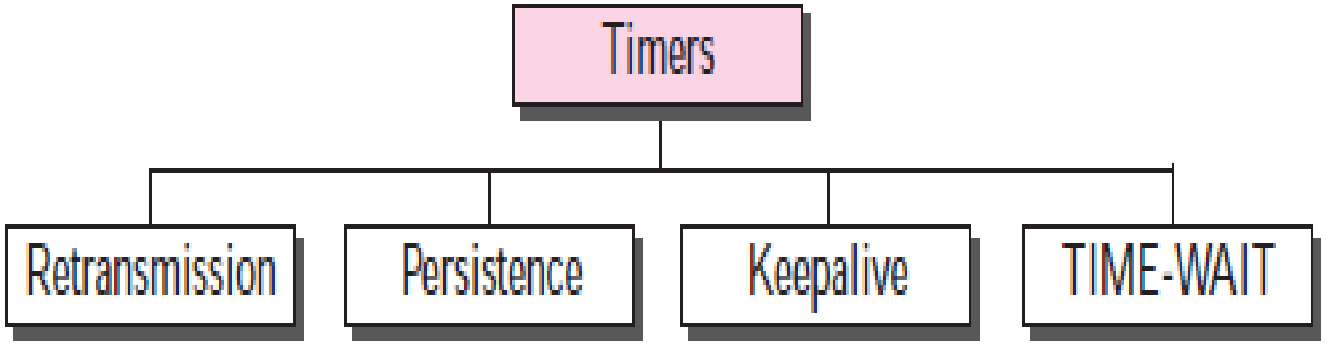


Figure 15.36 *TCP congestion policy summary*



TCP Timers

Figure 15.38 *TCP timers*



Retransmission Timer

- Used to retransmit lost segments
- Retransmission time-out (RTO) calculation:
  - ✓ Calculating measured RTT (RTT_M)
    - In TCP, there can be only one RTT measurement in progress at any time.
    - Measured RTT is likely to change for each round trip
  - ✓ Smoothed RTT (RTT_S), a weighted average of RTTM and the previous RTT_S is used
  - ✓ Most implementations use RTT deviation, called RTT_D, based on the RTT_S and RTT_M
- Value of RTO is based on the smoothed round-trip time and its deviation.

Formulas used in RTO calculation

Initially	→	No value
After first measurement	→	$RTT_S = RTT_M$
After each measurement	→	$RTT_S = (1 - \alpha)\ RTT_S + \alpha \times RTT_M$

Initially	→	No value
After first measurement	→	$RTT_D = RTT_M / 2$
After each measurement	→	$RTT_D = (1 - \beta)\ RTT_D + \beta \times \lvert RTT_S - RTT_M \rvert$

Original	→	Initial value
After any measurement	→	$RTO = RTT_S + 4 \times RTT$

✓Value of α is set to 1/8.
✓New RTT_S is calculated as 7/8 of the old RTT_S and 1/8 of the current RTT_M.
✓Value of β is set to 1/4.
TCP does not consider the RTT of a retransmitted segment in its calculation of a new RTO.

Persistence Timer

- When receiving TCP announces a window size zero, sending TCP stops transmitting segments until receiving TCP sends an ACK segment announcing a nonzero window size.
- This ACK segment can be lost- DEADLOCK!!
- ACK segments are not acknowledged nor retransmitted in TCP.
- No retransmission timer for a segment containing only an acknowledgment.
- **Solution**: PERSISTENCE TIMER at sending TCP side.
- If persistence timer goes off, send ***probe*** segment.
- Response: resend acknowledgement.
- No response continue sending probe segments, doubling value of persistence timer (threshold=60s).
- Beyond threshold sender sends one probe segment every 60 s until the window is reopened.

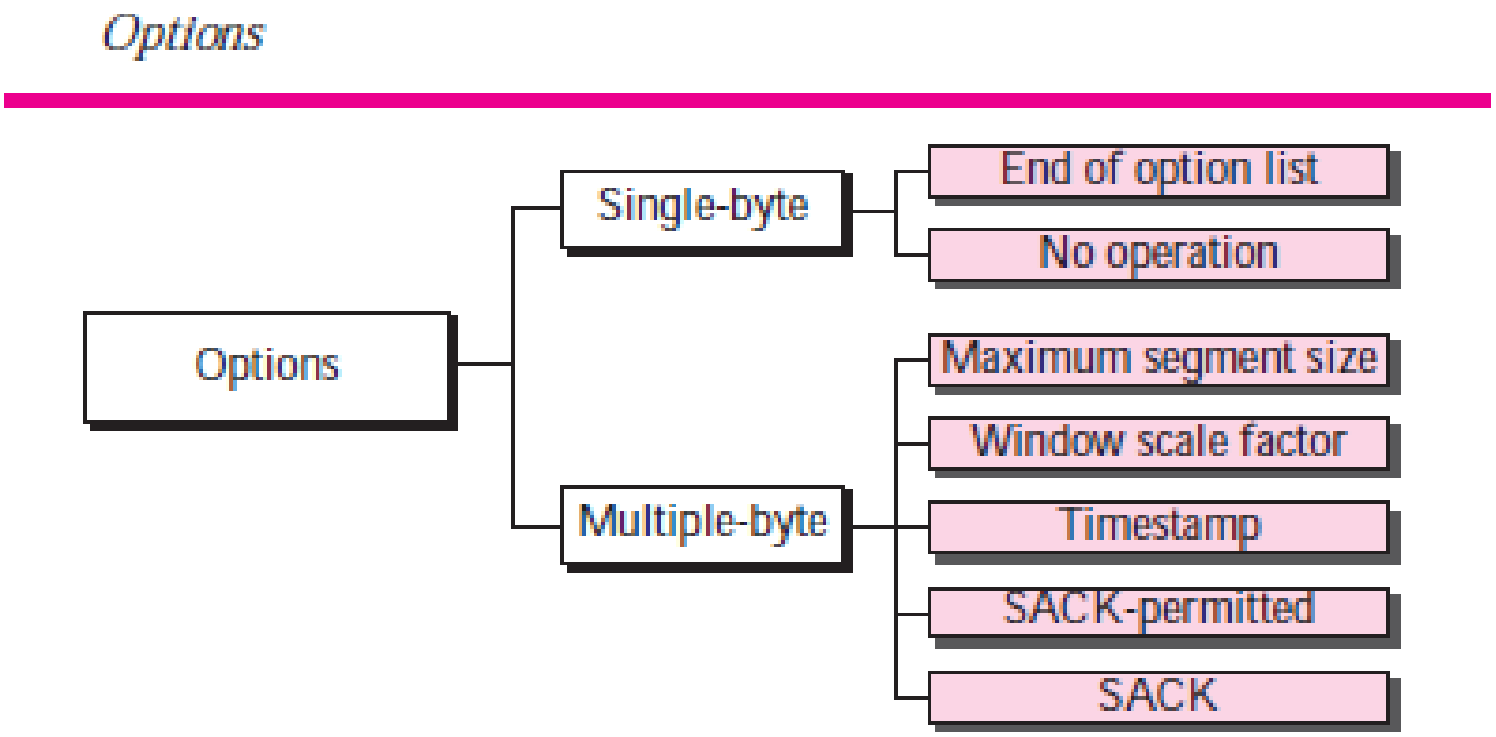
Keepalive Timer

- Keepalive timer is used to prevent a long idle connection between two TCPs.
- Each time server hears from a client, it resets this timer.
- Time-out is usually 2 hours. If server does not hear from the client after 2 hours, it sends a probe segment.
- If there is no response after 10 probes, each of which is 75s apart, it assumes client is down and terminates connection.

TIME_WAIT Timer

- The TIME_WAIT (2MSL) timer is used during connection termination (Refer TCP State Transition Diagram).
- Purpose of TIME_WAIT is to allow the networking to distinguish packets that arrive as belong to the 'old, existing' connection from a new one.
- The recommendation is to set the TIME_WAIT timer to twice the Maximum Segment Lifetime (MSL).

TCP Options


Options
Options
Single-byte
End of option list
No operation
Multiple-byte
Maximum segment size
Window scale factor
Timestamp
SACK-permitted
SACK

TCP Options

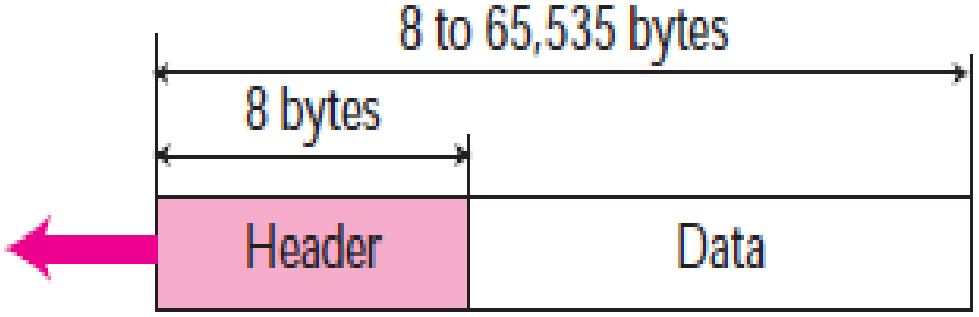
- End of Option (EOP) - used for padding at the end of the option section.
- Maximum Segment Size (MSS) - defines size of the biggest unit of data that can be received by the destination of the TCP segment. Value can be 0 to 65,535 bytes.
- Window Scale Factor - defines the size of the sliding window. Window can range from 0 to 65,535 bytes.
- Timestamp - measures the round-trip time and prevents wraparound sequence numbers.
- SACK permitted - used only during connection establishment. Host sending SYN segment adds this option to show that it can support SACK option.
- SACK - used during data transfer only if both ends agree. Option includes a list for blocks arriving out of order (which can be selectively acknowledged).

User Datagram Protocol

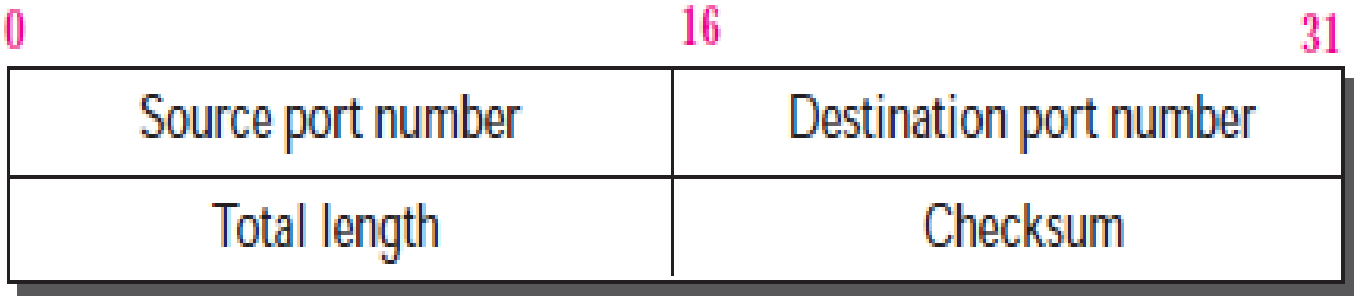
User Datagram Protocol (UDP)

- UDP is a **connectionless, unreliable transport protocol**.
- UDP uses port numbers to accomplish process-to-process communication.
- UDP has no flow control mechanism.
- UDP has no acknowledgment for received packets.
- If UDP detects an error in the received packet, it drops it.
- UDP packets/user datagrams have a fixed-size header of 8 bytes.

User datagram format



a. UDP user datagram



b. Header format

UDP Header

- **Source port number** used by the process running on the source host.
- 16 bits long, range from 0 to 65,535.
- For client port number is an ephemeral port number.
- For server port number is a well-known port number.
- **Destination port number** used by the process running on the destination host.
- 16 bits long.
- Ephemeral and Well-known port numbers used.
- **Length** is 16-bit field defines total length of the user datagram, header plus data.
- 16 bits can define a total length of 0 to 65,535 bytes.
- User datagram is encapsulated in an IP datagram

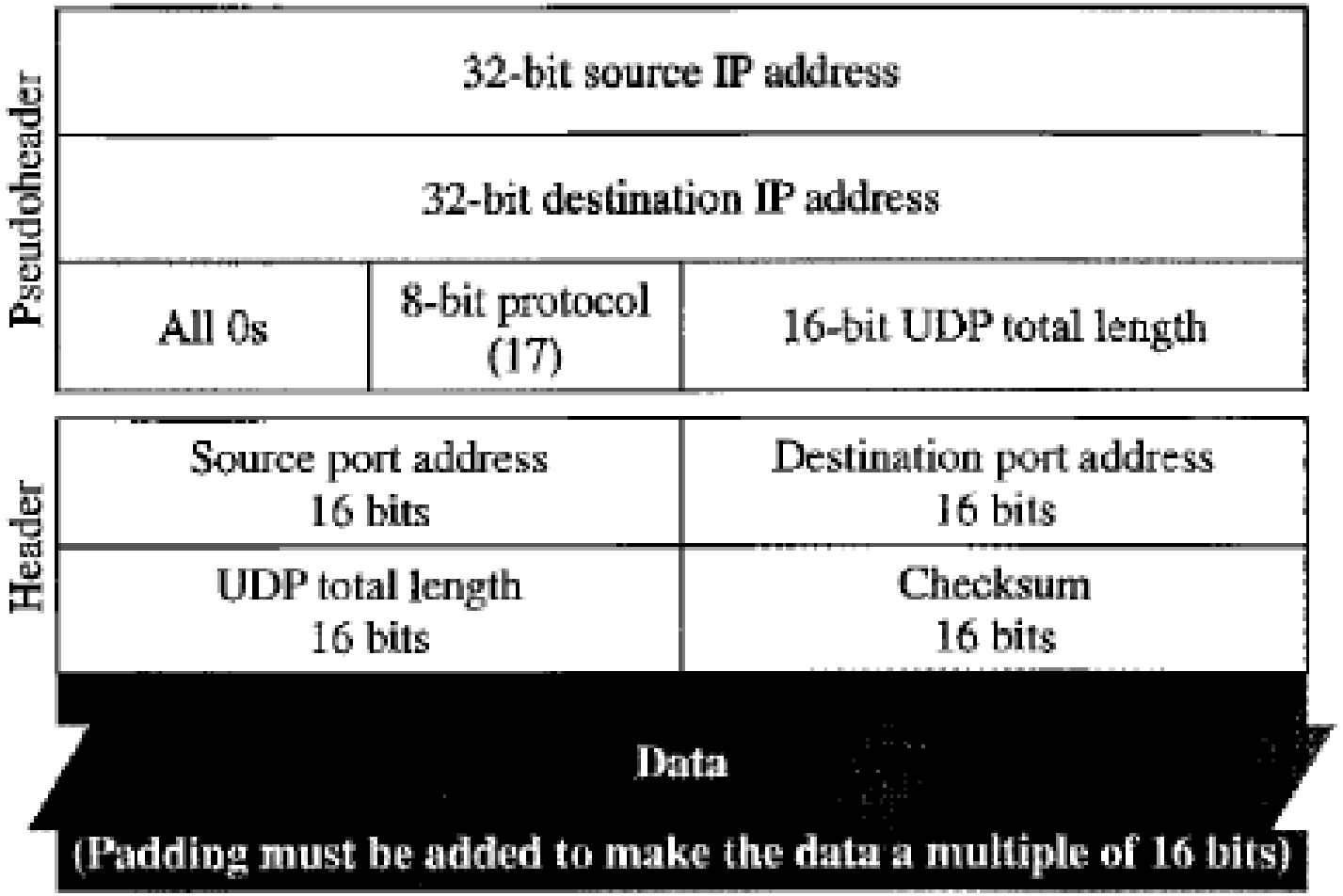
UDP length = IP length − IP header's length

- **Checksum** - used to detect errors over the entire user datagram.
- Checksum inclusion is optional.

Checksum

- UDP checksum calculation is different from the one for IP.
- Checksum includes three sections: pseudoheader, UDP header, and data from application layer.
- **Pseudoheader is part of the header of the IP packet in which the user datagram** is to be encapsulated with some fields filled with 0s.
- If the checksum does not have the pseudoheader, a user datagram may arrive safe and sound.
- If IP header is corrupted, it may be delivered to the wrong host.
- Protocol field is added to ensure that the packet belongs to UDP, and not to TCP.
- Value of protocol field for UDP is 17.
- If value of protocol field is changed during transmission, checksum calculation at the receiver will detect it and UDP drops the packet.
- It is not delivered to the wrong protocol.

Pseudoheader for checksum calculation



UDP operation

- Connectionless Services

  processes sending short messages, messages less than 65,507 bytes can use UDP

- Flow Control

  No flow control. So, no window mechanism. Process using UDP should provide for this service.

- Error Control

  No error control mechanism in UDP except for checksum.

- Congestion Control

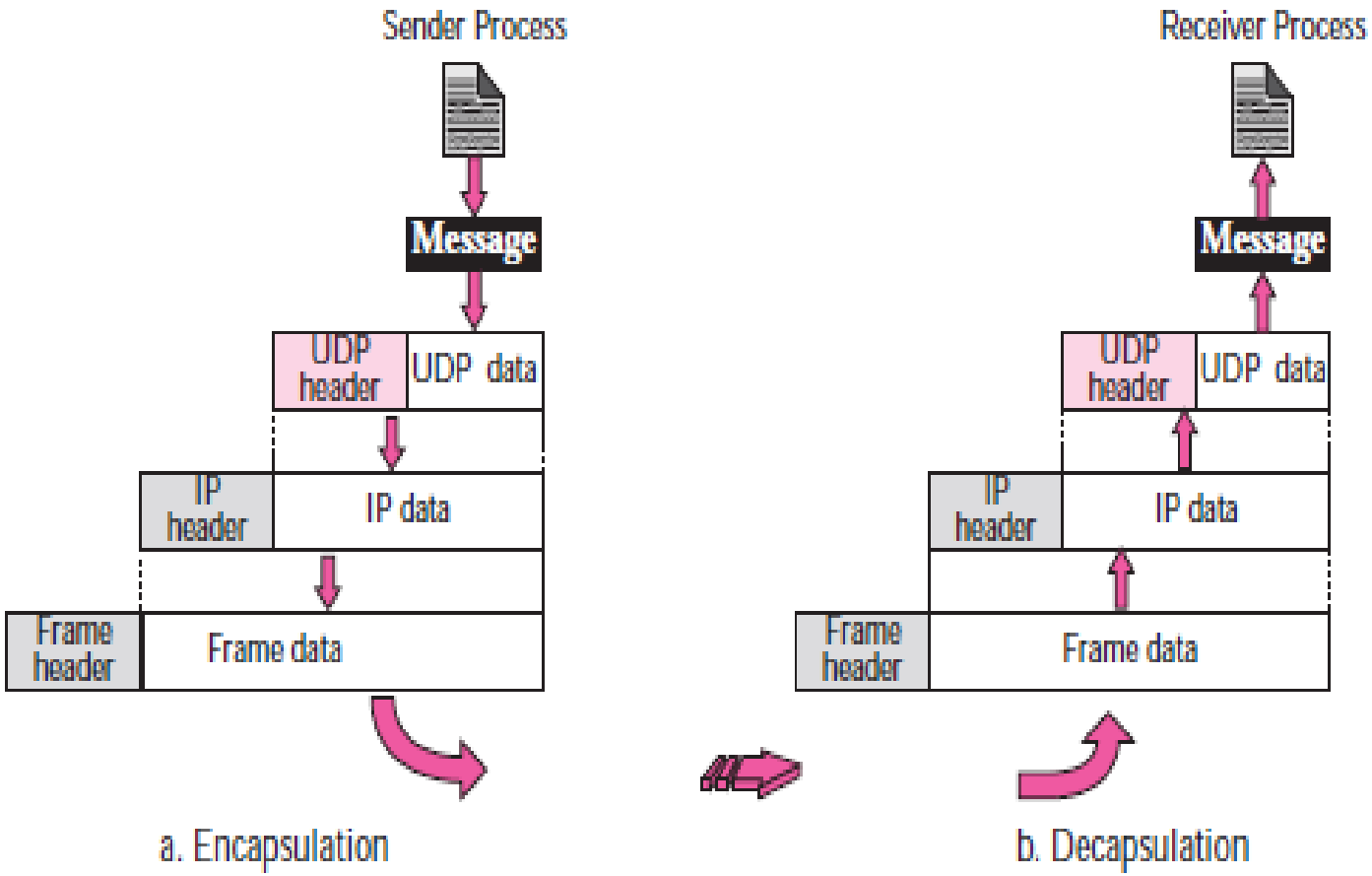
  No congestion control. UDP assumption of small & sporadic packets so network congestion improbable.

- Encapsulation & Decapsulation

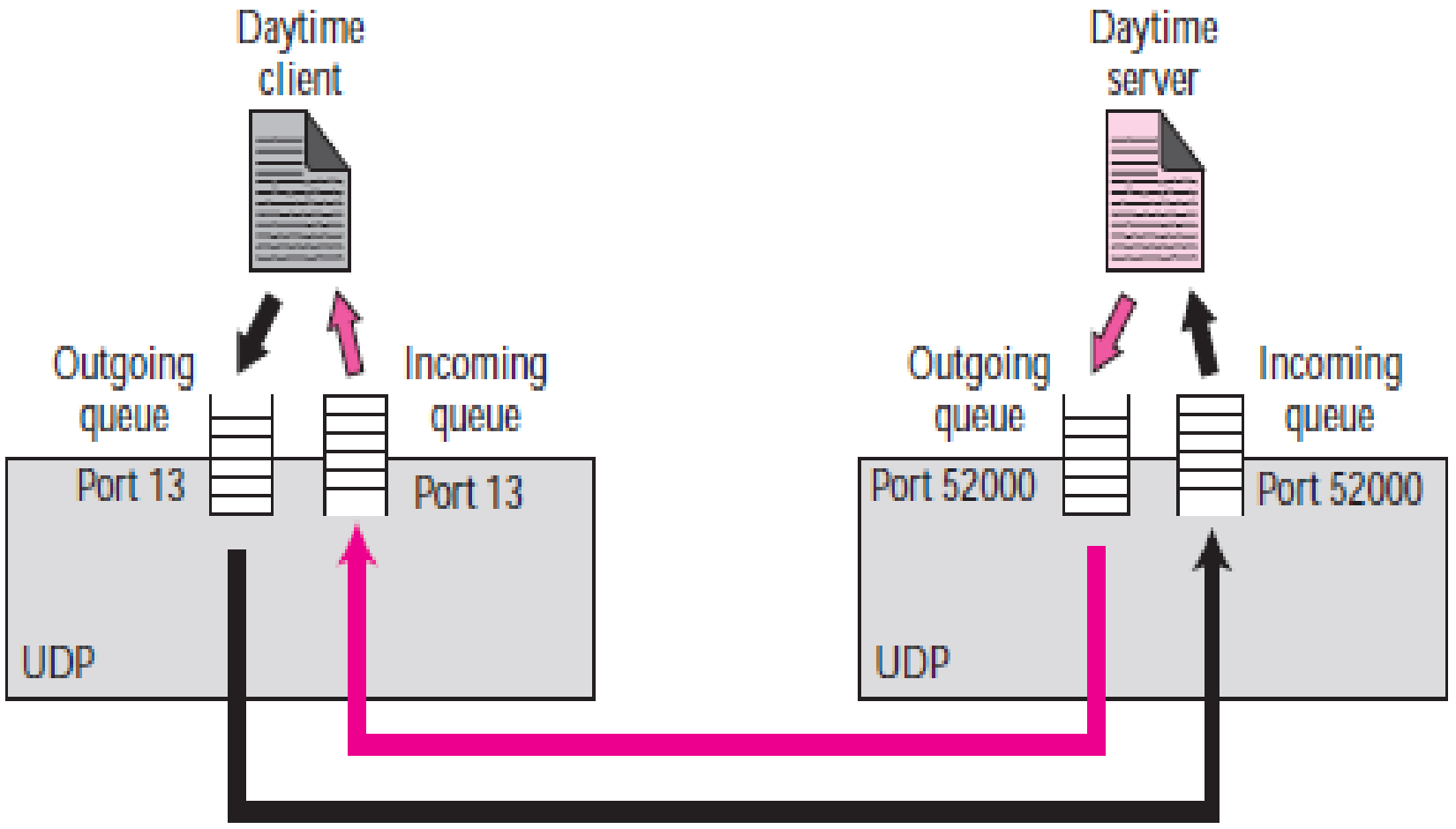
  process passes message to UDP along with a pair of socket addresses & length of data

- Queuing

Encapsulation and decapsulation



Queues in UDP

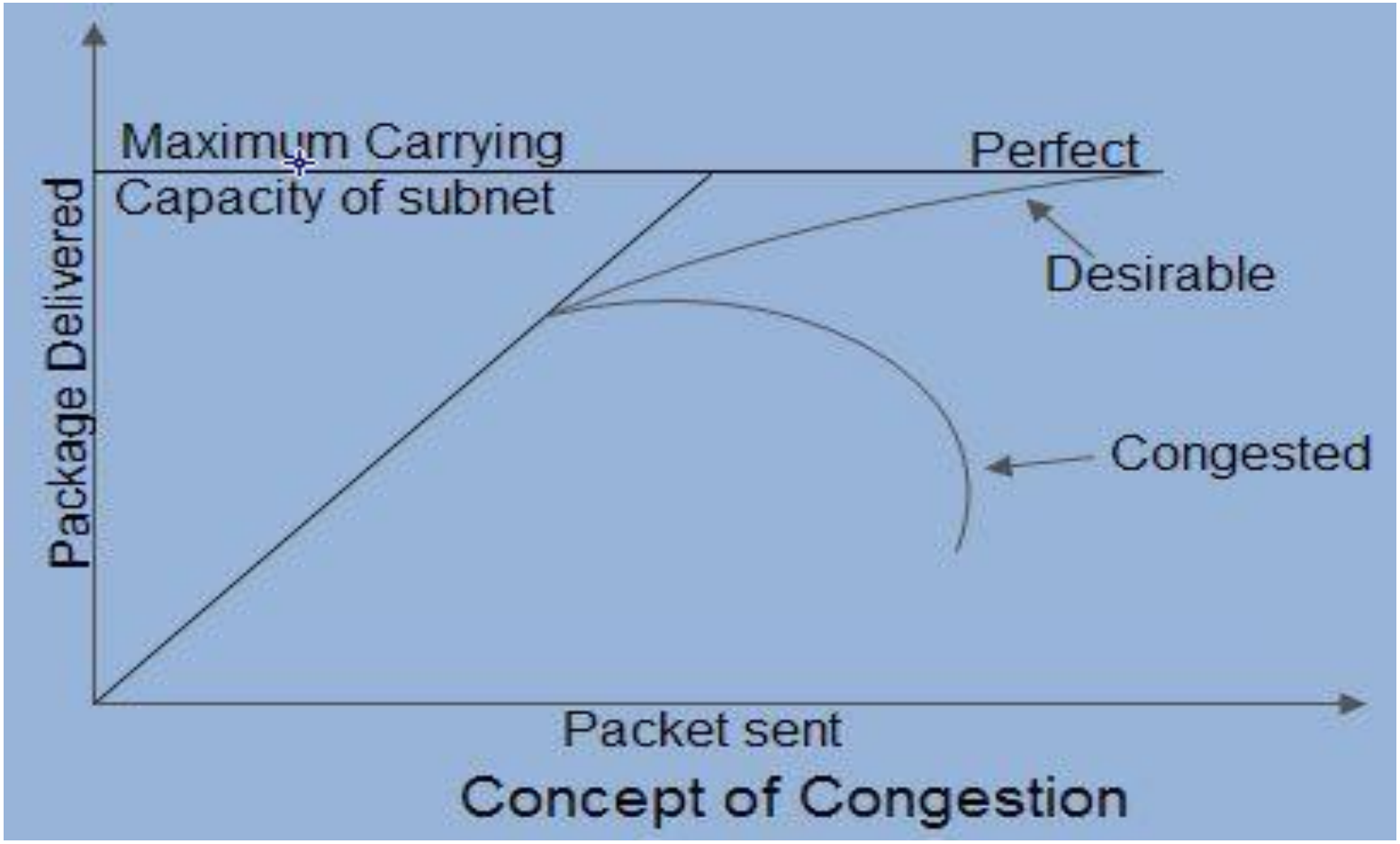


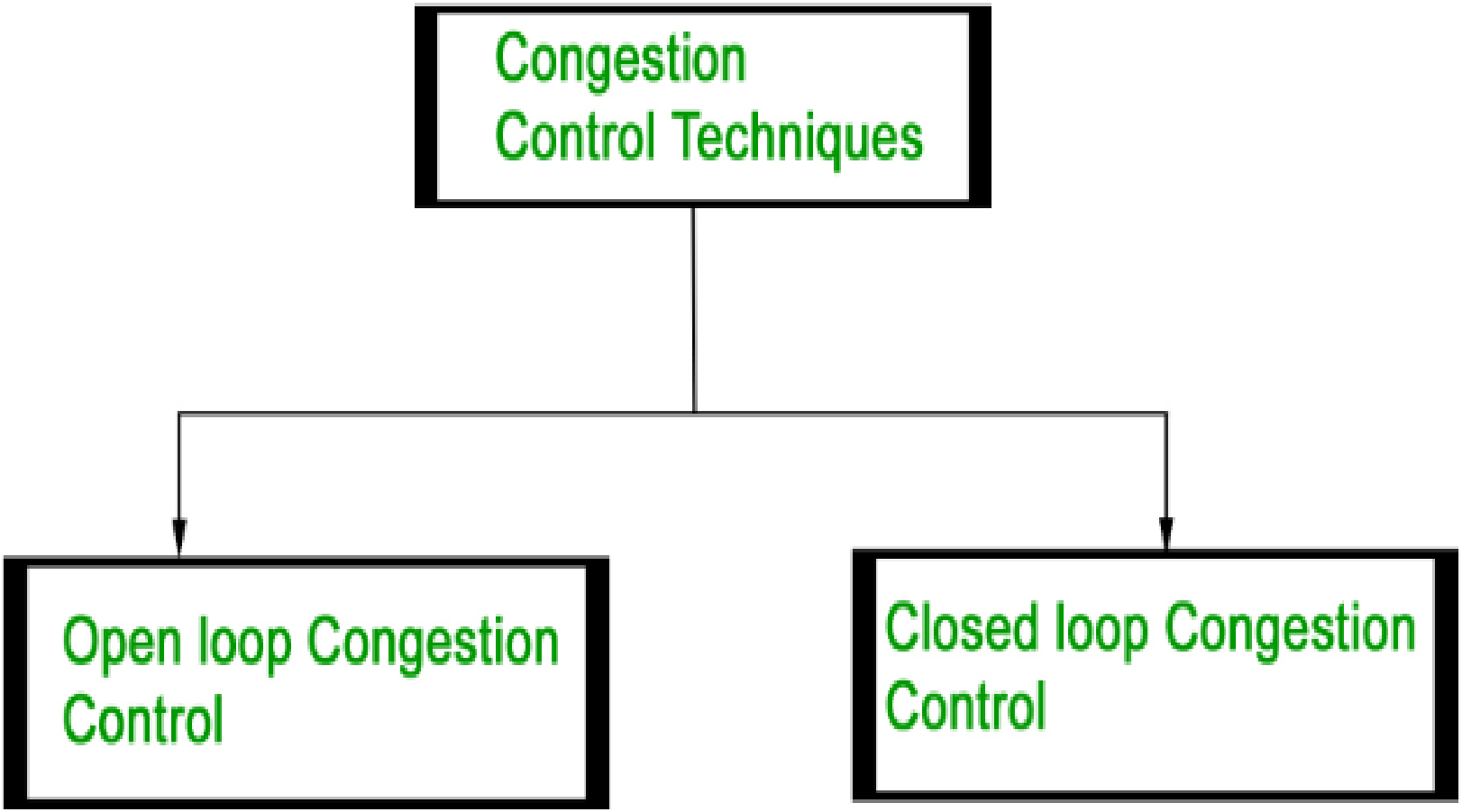
Uses of UDP

- Suitable for processes that requires simple request-response communication with little concern for flow and error control.
- Suitable for processes with internal flow and error-control mechanisms.
- 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 (RIP).
- UDP is normally used for real-time applications that cannot tolerate uneven delay between sections of a received message.

Congestion Control: Open Loop, Closed Loop choke packets

- Congestion is a situation in Communication Networks in which too many packets are present in a part of the subnet, performance degrades.
- Congestion in a network may occur when the load on the network (i.e. the number of packets sent to the network) is greater than the capacity of the network (i.e. the number of packets a network can handle.)


Maximum Carrying
Capacity of subnet
Perfect
Desirable
Congested
Package Delivered
Packet sent
Concept of Congestion

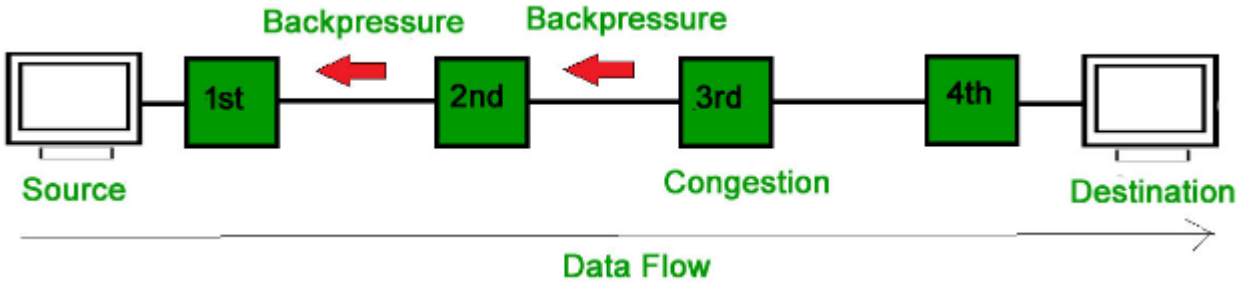

Congestion
Control Techniques
Open loop Congestion
Control
Closed loop Congestion
Control

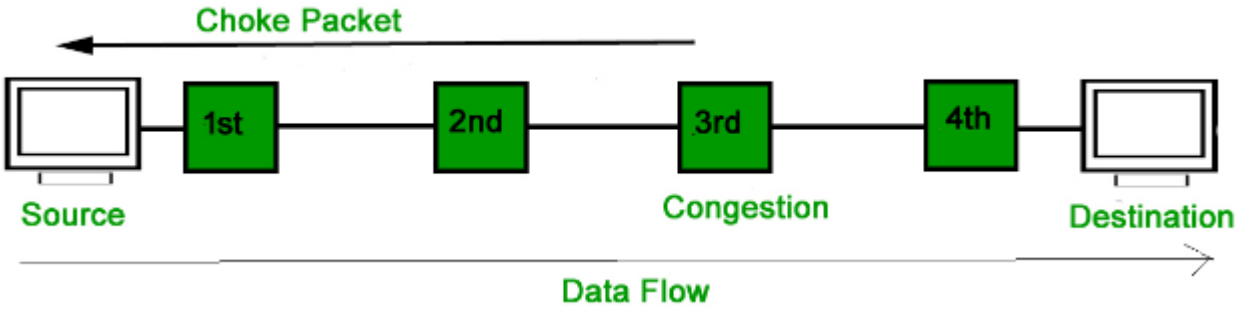
Open Loop Congestion Control

- Congestion control policies are applied to prevent congestion before it happens.
- Congestion control is handled either by the source or the destination.
- Policies adopted by open loop congestion control:
  - ✓ Retransmission Policy
  - ✓ Window Policy
  - ✓ Acknowledgment Policy
  - ✓ Discarding Policy
  - ✓ Admission Policy

Closed Loop Congestion Control

- Such methods used to treat or alleviate congestion after it happens.
- Several techniques are used by different protocols- some of them are:
  - ✓ Backpressure
  - ✓ Choke Packet Technique
  - ✓ Implicit Signaling
  - ✓ Explicit Signaling
    - Forward Signaling
    - Backward Signaling


Backpressure
Backpressure
1st
2nd
3rd
4th
Source
Congestion
Destination
Data Flow

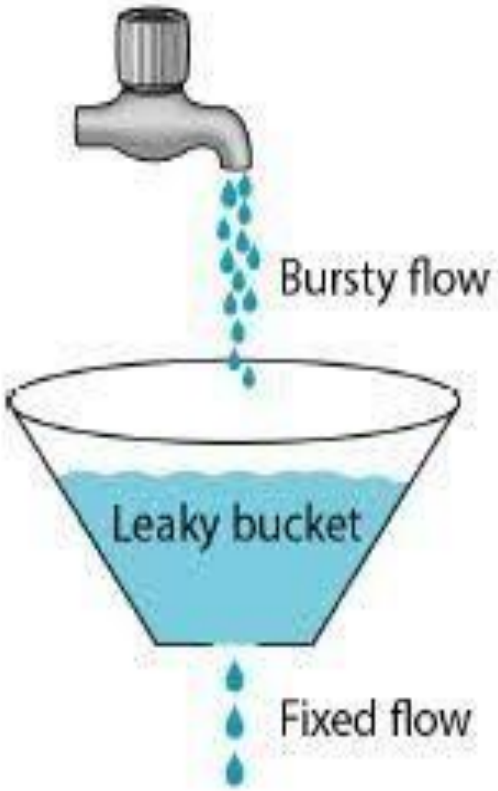

Choke Packet
1st
2nd
3rd
4th
Source
Congestion
Destination
Data Flow

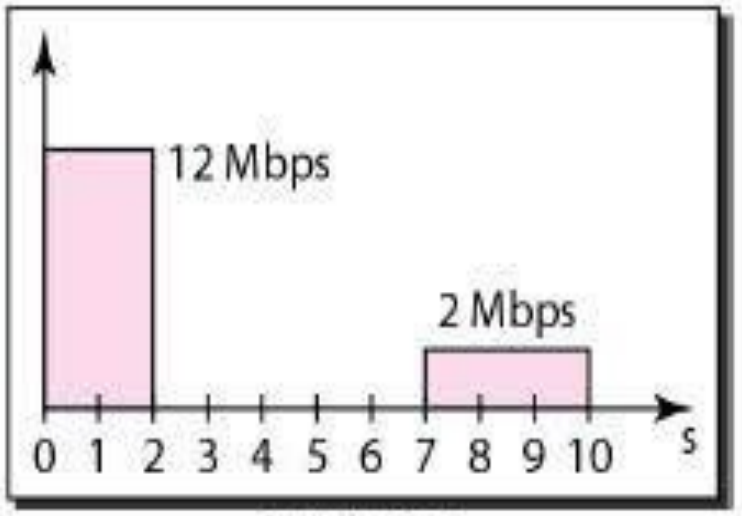
Techniques to improve QoS: Congestion control algorithms

- Leaky Bucket Algorithm
- Token bucket Algorithm

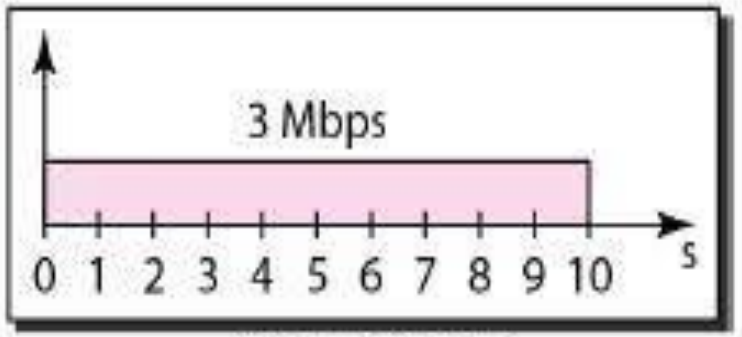
Leaky Bucket Algorithm

- It is a traffic shaping mechanism that controls the amount and the rate of the traffic sent to the network.
- A leaky bucket algorithm shapes bursty traffic into fixed rate traffic by averaging the data rate.
- Imagine a bucket with a small hole at the bottom.
- The rate at which the water is poured into the bucket is not fixed and can vary but it leaks from the bucket at a constant rate.
- As long as water is present in bucket, the rate at which the water leaks does not depend on the rate at which the water is input to the bucket.
- Also, when bucket is full, any additional water that enters into the bucket spills over the sides and is lost.





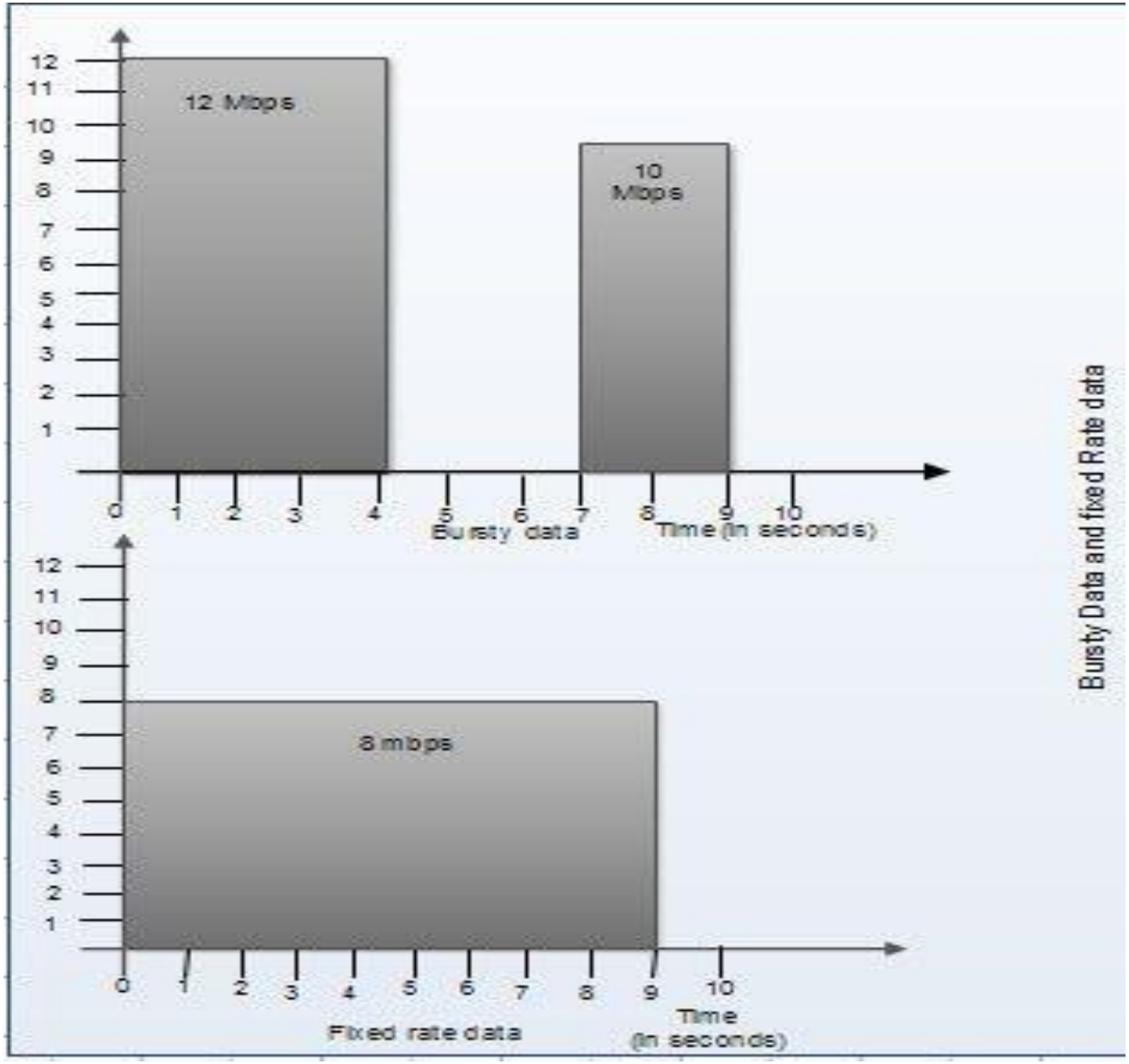
Bursty data



Fixed-rate data

Leaky Bucket Algorithm - example

- Consider data is coming from the source at variable speeds. Suppose that a source sends data at 12 Mbps for 4 seconds.
- Then there is no data for 3 seconds.
- The source again transmits data at a rate of 10 Mbps for 2 seconds.
- So, in a time span of 9 seconds, 68 Mb data has been transmitted.
- If a leaky bucket algorithm is used, the data flow will be 8 Mbps for 9 seconds.
- So constant flow is maintained.

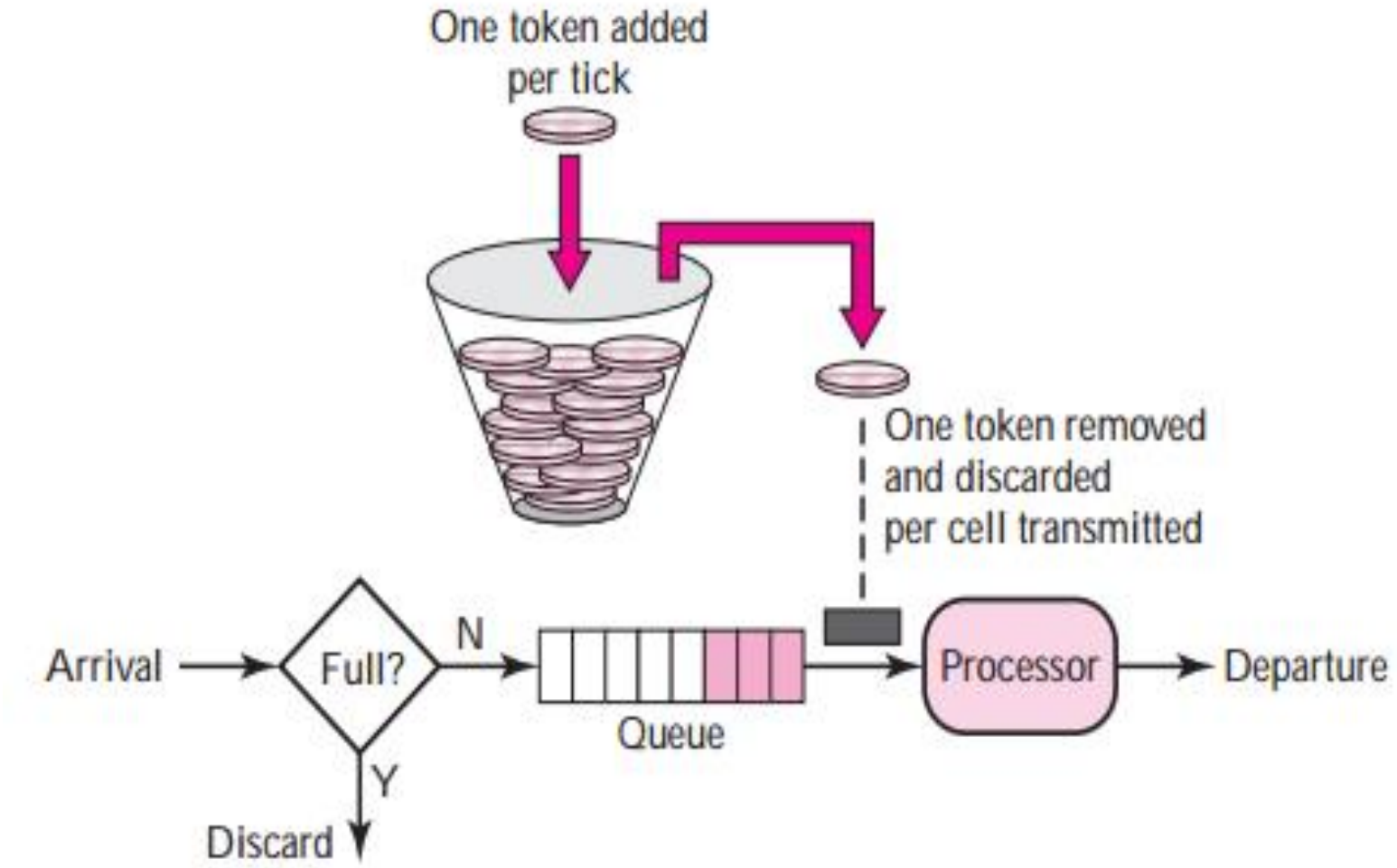

12 Mbps
10 Mbps
Bursty data
Time (in seconds)
8 mbps
Fixed rate data
Time (in seconds)
Bursty Data and fixed Rate data

Leaky Bucket - demerits

- The leaky bucket algorithm allows only an average (constant) rate of data flow.
- It cannot deal with bursty data.
- A leaky bucket algorithm does not consider the idle time of the host.
- If the host was idle for 10 seconds and now it is willing to sent data at a very high speed for another 10 seconds, the total data transmission will be divided into 20 seconds and average data rate will be maintained.
- The host is having no advantage of sitting idle for 10 seconds.

Token bucket Algorithm

- A token bucket algorithm allows bursty data transfers.
- A token bucket algorithm is a modification of leaky bucket in which leaky bucket contains tokens.
- In this algorithm, a token(s) are generated at every clock tick.
- For a packet to be transmitted, system must remove token(s) from the bucket.
- So, a token bucket algorithm allows idle hosts to accumulate credit for the future in form of tokens.
- For example, if a system generates 100 tokens in one clock tick and the host is idle for 100 ticks.
- The bucket will contain 10,000 tokens.
- Now, if host wants to send bursty data, it can consume all 10,000 tokens at once for sending 10,000 cells or bytes.
- So a host can send bursty data as long as bucket is not empty.


One token added
per tick
One token removed
and discarded
per cell transmitted
Arrival
Full?
N
Y
Discard
Queue
Processor
Departure

Quality of Service (QoS) & Traffic Management

- Traffic Management concerned with delivering QoS to specific packet flows.
- Mechanisms used to manage flows to control load applied to various switches & routers.
- Setting of priorities & scheduling mechanisms at routers, switches and multiplexers to provide differentiated treatment for packets.
- Traditional Internet gives single class of best-effort service
  - ✓ Even though ToS bits were included in the original IP header
- All packets treated the same

Significant factors in providing QoS

- Application performance
- Bandwidth required to provide performance
- Complexity/cost of required mechanisms

Providing better service

- Routing or Forwarding
- Scheduling or Dropping
- Relative or Absolute

Relative QoS

- Priority scheduling
  - ✓ Favored packets get lower delay and lower drop rate
- Priority dropping
  - ✓ All sent packets get same average delay

Differentiated Services (DiffServ)

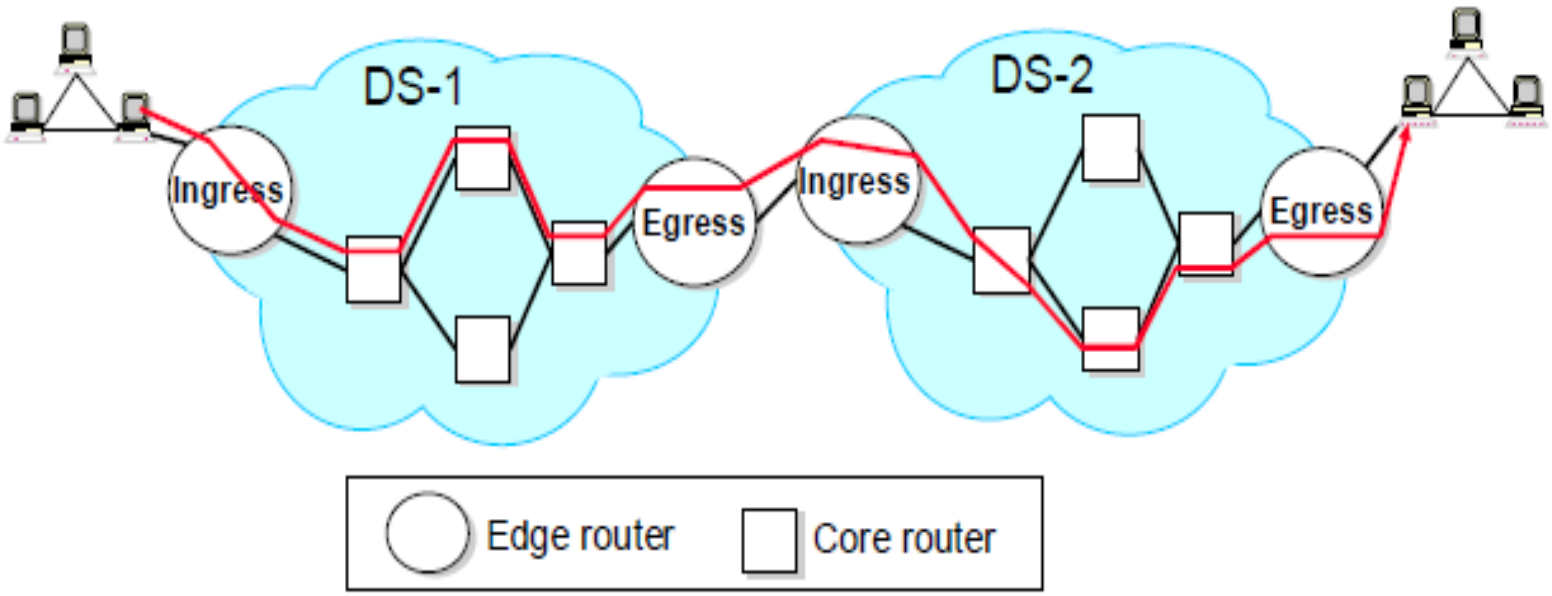
- Goal: offer different levels of service
  - ✓ Organized around domains
  - ✓ Edge and core routers

- Edge routers
  - ✓ Sort packets into classes (based on variety of factors)
  - ✓ Police/shape traffic
  - ✓ Set bits (DSCP- differentiated services code point) in packet header

- Core routers
  - ✓ Handle packet (PHB = per-hop behaviors) based on DSCP

- A group of routers that implement common, administratively defined DiffServ policies are referred to as a *DiffServ domain*.

DiffServ Architecture



Provide low-latency to critical network traffic such as voice or streaming media while providing simple best-effort service to non-critical services such as web traffic or file transfers.

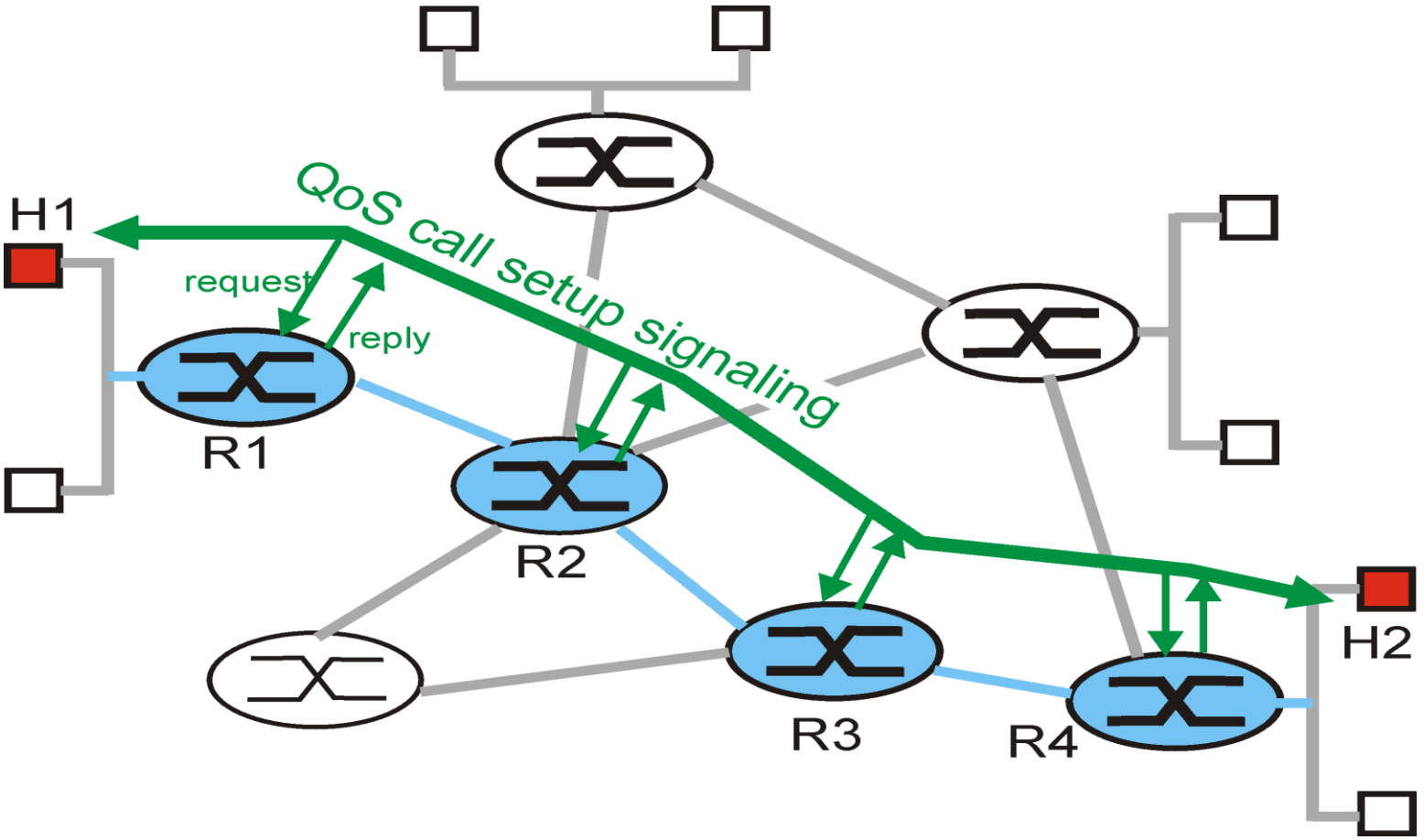
Integrated Services or IntServ

- Specifies a fine-grained QoS system, which is often contrasted with DiffServ's coarse-grained control system.
- Requires admission control for each flow
  - ✓ Per-flow, not per traffic class
  - ✓ E.g., reservations
- Every router in the system implements IntServ, every application that requires some kind of guarantees has to make an individual reservation.
- Flow Specs describe what the reservation is for, while RSVP is the underlying mechanism to signal it across the network.

Flow Specs

- What does the traffic look like?
  - ✓ Done in the Traffic SPECification part or TSPEC.
  - ✓ TSPECs include token bucket algorithm parameters.
- What guarantees does it need?
  - ✓ Done in the service Request SPECification part or RSPEC.
  - ✓ RSPECs specify what requirements there are for the flow
    - 'Controlled Load' setting for lightly loaded network
    - 'Guaranteed' setting for an absolutely bounded service

Integrated Services or IntServ



Issues for IntServ

- Routers between the sender and listener have to decide if they can support the reservation being requested.
- If not, they send a reject message to let the listener know about it.
- Once they accept the reservation they have to carry the traffic.
- All routers along the traffic path must support it.
- Many states must be stored in each router.
- Supported only in small scale.
- For scaled up network size it is difficult to keep track of all the reservations.

END

- http://www.tcpipguide.com/free/t_TCPOperationalOverviewandtheTCPFiniteStateMachineF-2.htm
- https://www.brianstorti.com/tcp-flow-control/