

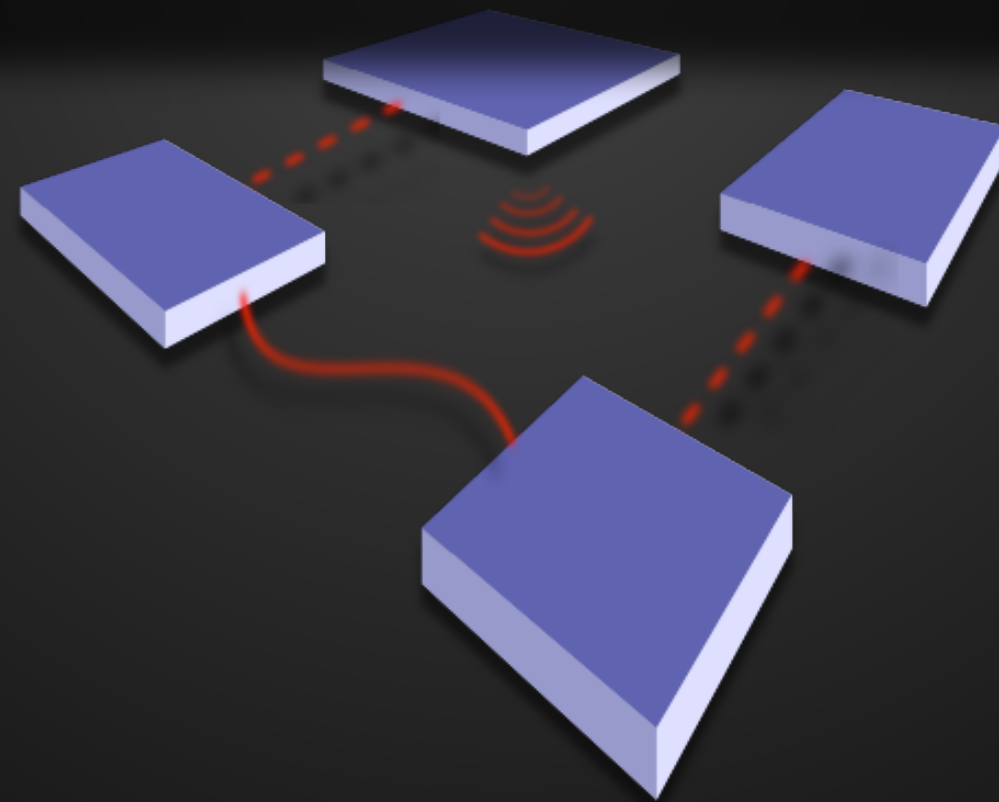
CS-435

spring semester 2020

Network Technology & Programming Laboratory

University of Crete
Computer Science Department

Stefanos Papadakis



CS-435

Lecture #5 preview

- The Transport Layer
 - User Datagram Protocol (UDP)
 - Transmission Control Protocol (TCP)

What does it do layer-wise?

- provide logical communication between different end systems application processes
- transport protocols instances run in end systems
- SOURCE:
 - breaks application messages into segments
 - passes down segments to the Network Layer
- DESTINATION:
 - reassembles segments into messages
 - passes messages up to Application layer

Transport / Network?

- network layer:
 - logical communication between hosts
- transport layer:
 - logical communication between processes

The Transport layer:

relies on & enhances the Network layer Services

The Internet transport layer: TCP & UDP

- services available:
 - **TCP**: Reliable in-order delivery with congestion control
 - flow control
 - connection setup
 - **UDP**: unreliable, unordered delivery
- services **not** available:
 - delay guarantees
 - bandwidth guarantees

The User Datagram Protocol

- Provides a “bare bones” Internet transport protocol
- “best effort” service, UDP segments may be:
 - lost
 - delivered out of order to application
- connectionless:
 - No connection setup before data transfer (no handshakes)
 - each UDP segment handled independently of others
- More control required by the application

The UDP

- often used for streaming multimedia applications
 - loss tolerant
 - rate & delay sensitive
- other UDP uses
 - DNS
 - SNMP
- reliable transfer over UDP: add reliability at application layer
 - application-specific error recovery!

The UDP header

0	15	16	31
Source Port Number(16 bits)		Destination Port Number(16 bits)	
Length(UDP Header + Data)16 bits		UDP Checksum(16 bits)	
Application Data (Message)			

- The checksum applies to the entire UDP segment
- The checksum field is optional. If it is not used, it is set to zero.
- NOTE: the IP checksum applies only to the IP header and not to the data field, which in this case consists of the UDP header and the user segment. Thus, if no checksum calculation is performed by UDP, then no check will be made on the user data at either the transport or network layer.

The TCP in a nutshell

- Connection-oriented
 - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- point-to-point
 - one sender, one receiver
- Byte-stream
 - app writes bytes
 - TCP sends segments
 - app reads bytes
- send & receive buffers
- pipelined - window size set by:
 - TCP congestion and
 - flow control
- Error control (retransmissions)

The TCP Segment

Source Port (16)			Destination Port (16)		
Sequence Number (32)					
Acknowledgement Number (32)					
Data offset	Reserved (6)	Flags (6)		Window (16)	
Checksum (16)			Urgent (16)		
Options and Padding					
Data (Varies)					

TCP the segment header fields

- **Source Port (16 bits):** Source TCP user.
- **Destination Port (16 bits):** Destination TCP user. eg. Telnet = 23; HTTP = 80.
- **Sequence Number (32 bits):** of first data octet in this segment except when the SYN flag is set.

Source Port (16)			Destination Port (16)		
Sequence Number (32)					
Acknowledgement Number (32)					
Data offset	Reserved (6)	Flags (6)		Window (16)	
Checksum (16)			Urgent (16)		
Options and Padding					
Options and Padding					

TCP the segment header fields

- **Acknowledgment Number (32 bits):** Contains the sequence number of the next data octet that the TCP entity expects to receive.
- **Data Offset (4 bits):** Number of 32-bit words in the header.
- **Reserved (6 bits):** Reserved for future use.

Source Port (16)			Destination Port (16)		
Sequence Number (32)					
Acknowledgement Number (32)					
Data offset	Reserved (6)	Flags (6)		Window (16)	
Checksum (16)			Urgent (16)		
Options and Padding					
Options and Padding					

TCP the segment header fields

- **Flags (6 bits):** SYN, FIN, RESET, PUSH, URG, ACK
- **Window (16 bits):** Flow control credit allocation, in octets.
- **Checksum (16 bits):** The ones complement of the ones complement sum of all the 16-bit words in the segment plus a pseudoheader.
- **Urgent Pointer (16 bits):** allows the receiver to know how much urgent data is coming.
- **Options (Variable):** eg. option that specifies the maximum segment size.

Source Port (16)			Destination Port (16)		
Sequence Number (32)					
Acknowledgement Number (32)					
Data offset	Reserved (6)	Flags (6)		Window (16)	
Checksum (16)			Urgent (16)		
Options and Padding					
Options and Padding					
Checksum (16)			Urgent (16)		

Sequence numbers in TCP

- Segments can have different length, hence sequence numbers are byte stream numbers
- Sequence number of one segment = byte stream number of first byte in segment
- ACK numbers = sequence number of next byte receiver is expecting
- Cumulative acknowledgements: when an ACK is received, all bytes with sequence number smaller than the ACK number have been correctly received.

TCP Connection Management

- TCP sender & receiver establish a “connection” before exchanging data segments
- initialize TCP variables:
 - Sequence Numbers
 - buffers,
 - flow control info (e.g. RcvWindow)

The 3-way handshake

Step 1

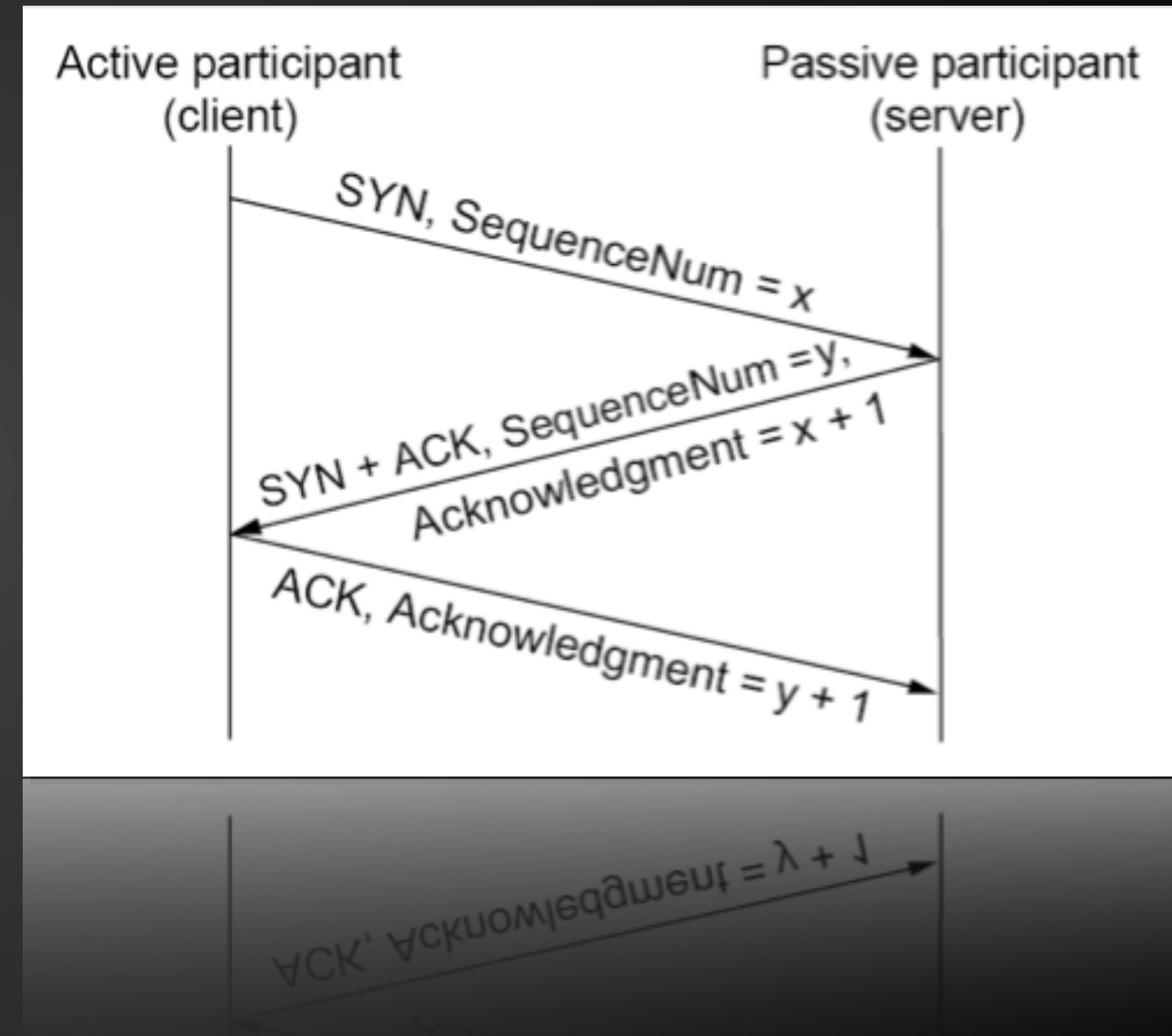
- client host sends TCP_SYN segment to server
 - Specifies initial sequenceNum
 - Carries no data

Step 2

- server host receives SYN, replies with SYN-ACK segment
 - server allocates buffers
 - specifies server initial sequenceNum

Step 3

- client receives SYN-ACK, replies with ACK segment, which may contain data



Window Scaling Option

RFC 1323:

The three-byte Window Scale option may be sent in a SYN segment by a TCP.

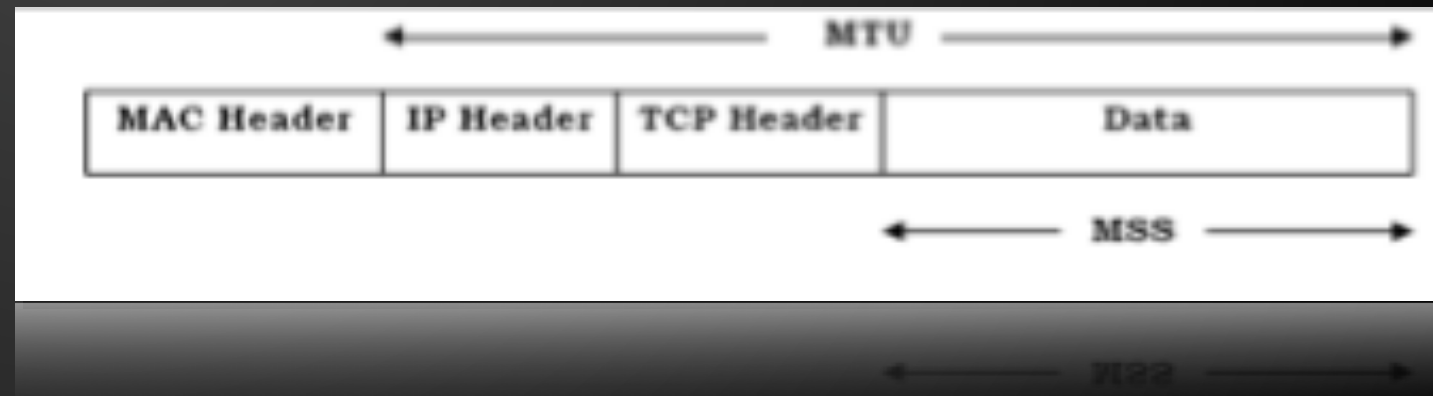
It has two purposes:

1. indicate that the TCP is prepared to do both send and receive window scaling, and
2. communicate a scale factor to be applied to its receive window.

The scale factor is limited to a power of two and encoded logarithmically, so it may be implemented by binary shift operations.

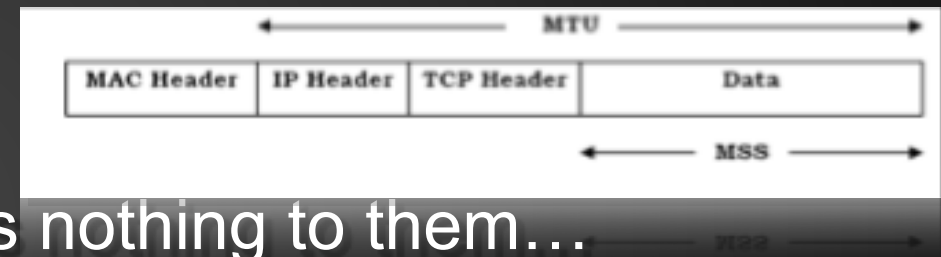
When is a TCP segment sent

- MSS (Maximum Segment Size):
 - includes **only the data** of the TCP segment
- **Receiver** indicates MSS it can accept at **connection setup**
- $MSS = MTU - (IP + TCP/UDP/ICMP \text{ hdr size}) = MTU - (20 + ?)$
- A segment sent when
 - an MSS fills, or if
 - TCP uses “no delay”

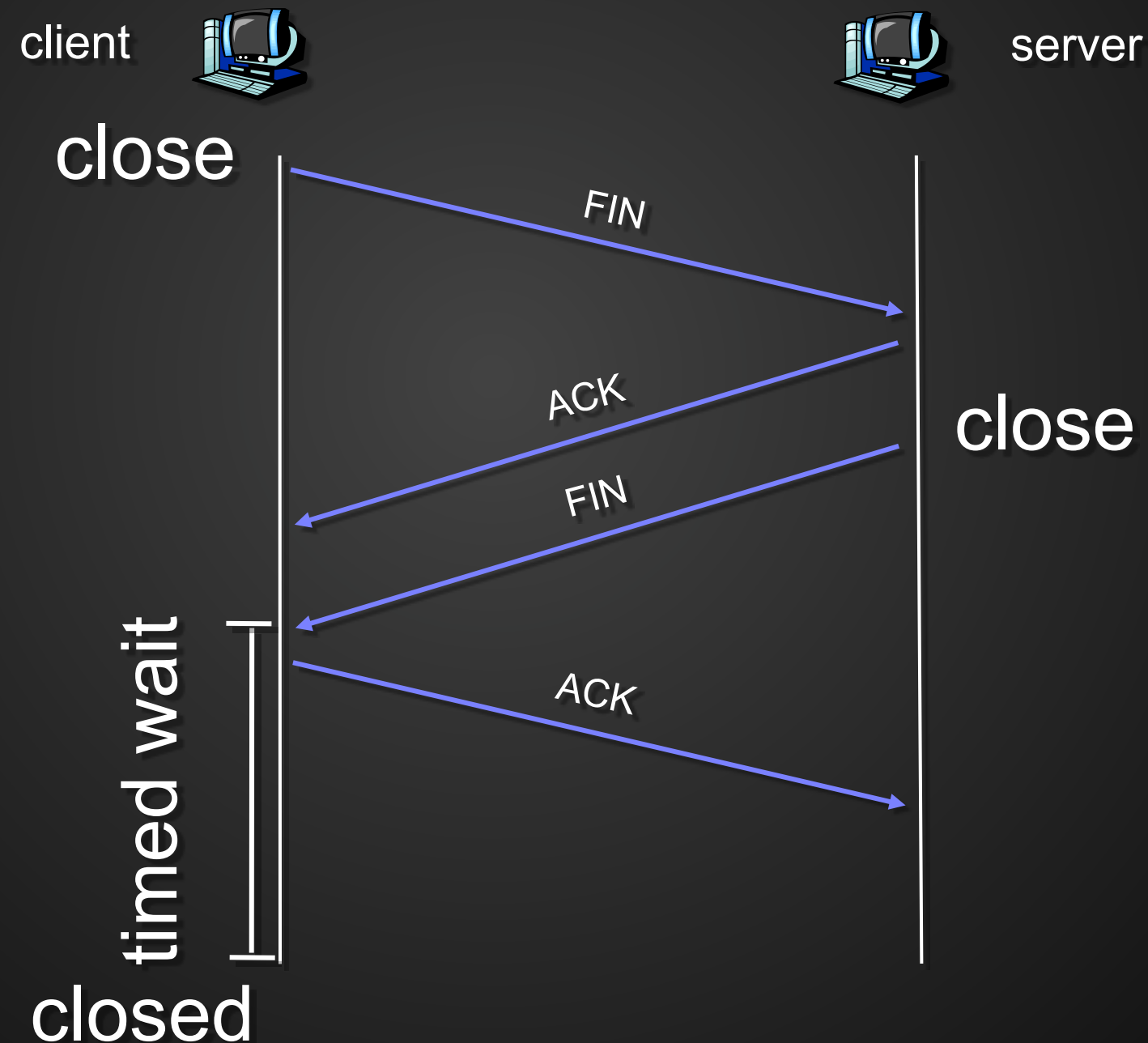


The TCP MSS

- **TCP** MSS should be smaller or equal to $MTU - 40$
- **BEWARE:** 40 bytes is typical (minimum) length of TCP + IP header
 - If no options are used
 - If other protocols used “40” means nothing to them...
- Ethernet MTU=1500 bytes
 - Ethernet header: 14 bytes, tail: 4 bytes
- IPv4 default/min value MSS=536, MTU=576 bytes (IPv6 MTU=1280)
 - Used when path MTU discovery not used



Connection Teardown

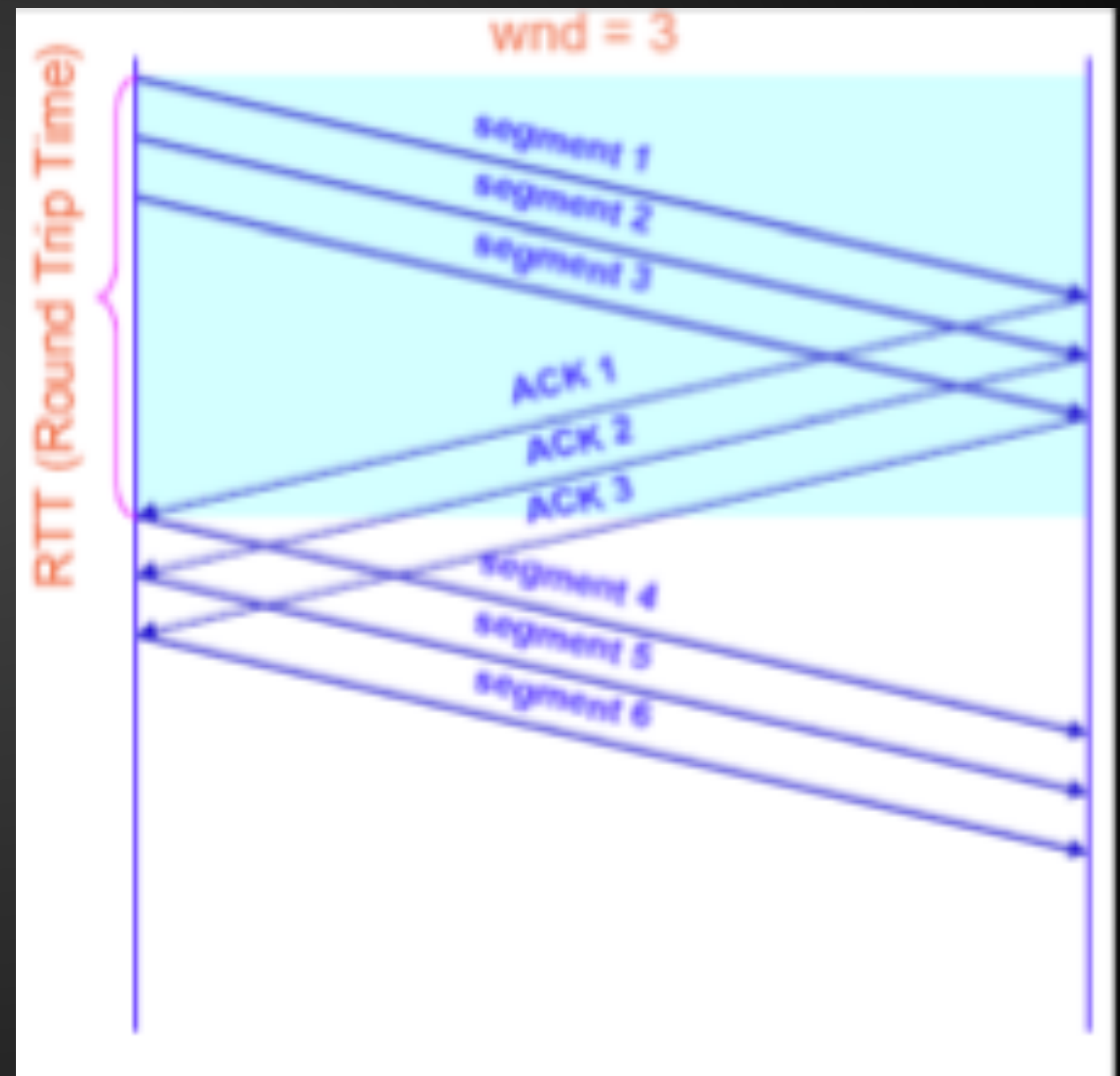


Sliding Window

- Three purposes
 - Reliable, In-order data delivery
 - Flow control
 - Congestion control

Window Size & throughput

- Sliding-window based flow control:
- Higher window \rightarrow higher throughput
 - $\text{Throughput} = \text{window} / \text{RTT}$
 - Need to worry about sequence number wrapping
- **Remember:** window size controls throughput

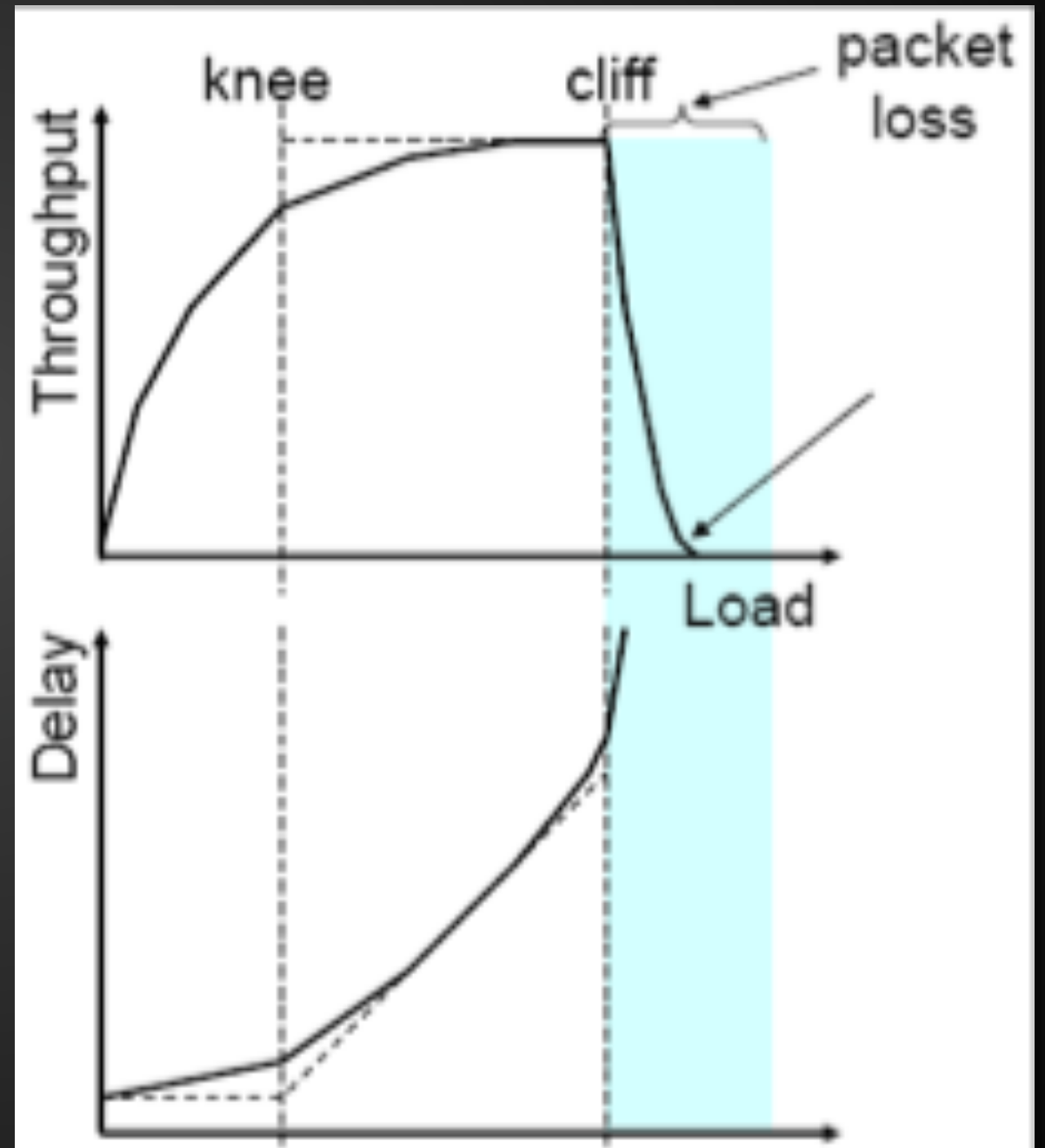


Congestion Collapse

- How might this happen?
 - assume network is congested (a router drops packets)
 - you learn the receiver didn't get the packet
 - either by ACK or Timeout
- what do you do?
 - retransmit packet
- still receiver didn't get the packet (because it is dropped again)
 - retransmit again
 - ... and so on ...
- and now assume that everyone is doing the same!
 - Network will become more and more congested
 - with duplicate packets rather than new packets!

Manifestation

- Knee (point after which):
 - throughput increases very slow
 - delay increases fast
- Cliff (point after which):
 - throughput starts to decrease very fast towards zero (congestion collapse)
 - delay approaches infinity



Solutions?

- Increase buffer size
 - Why is this not THE solution??
- Slow down
 - If you know that your packets are not delivered because of network congestion
- Questions:
 - How do you detect network congestion?
 - In what fashion to slow down?

Solutions?

- Detect when network approaches/reaches knee point
 - Stay there
- How do you get there?
- What if you overshoot? (i.e. go over knee point)
- Possible solutions:
 - Increase window size until you notice congestion
 - Decrease window size if network congested

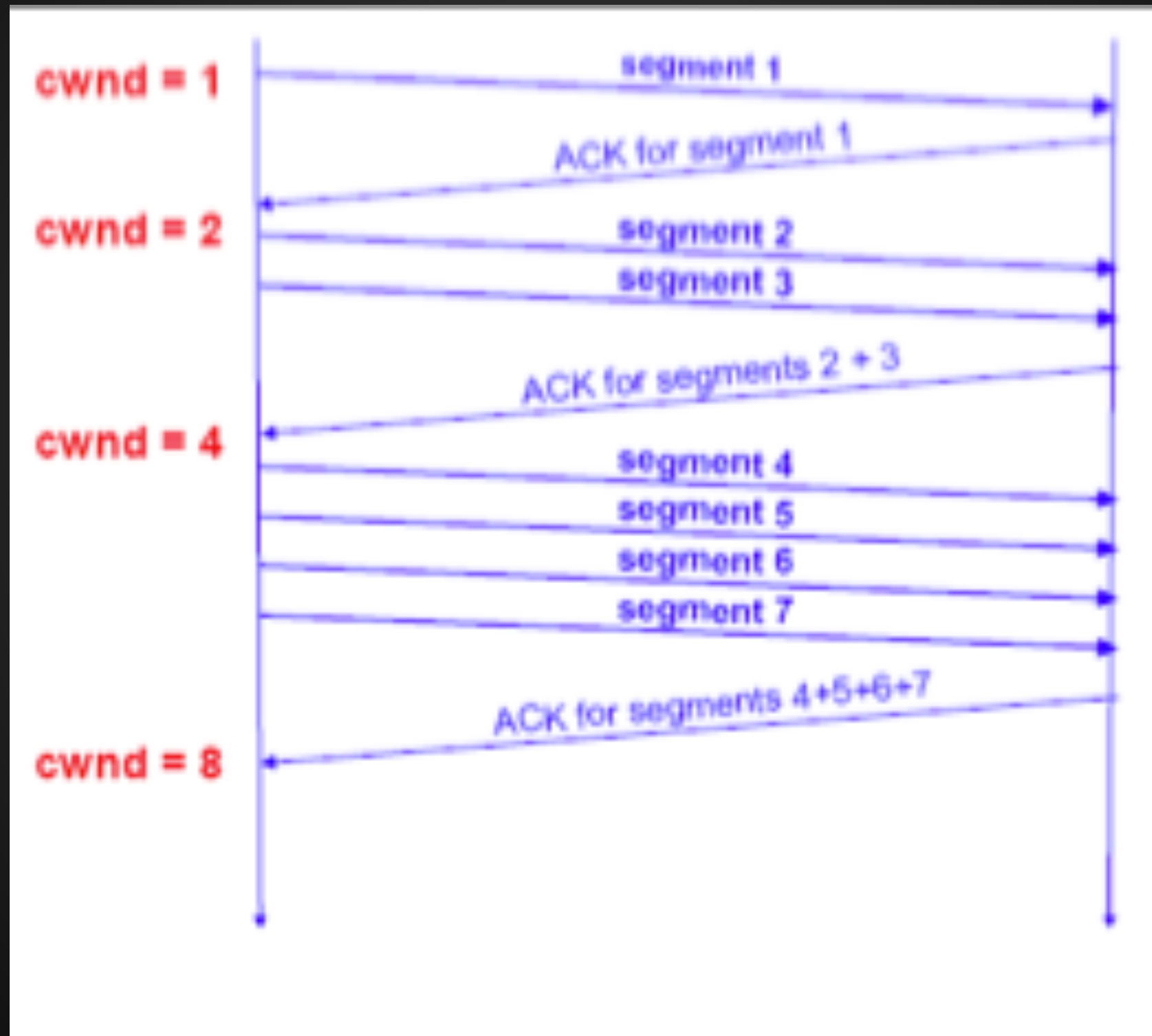
TCP: Slow Start

- Goal: discover congestion quickly
 - How?
 - Quickly increase cwnd until network congested to get a rough estimate of the optimal of cwnd
- How do we know when network is congested?
- Packet loss (TCP)
 - Over the cliff → DO congestion control
- Congestion notification
 - Over the knee but before the cliff → congestion avoidance
- How do we know a packet is lost?

TCP: Slow Start

- Whenever starting traffic on a new connection, -OR-
- whenever increasing traffic after congestion was experienced:
- Set $\text{cwnd} = 1$
- Each time a **segment** is **received** increment cwnd by one ($\text{cwnd}++$).
- Does Slow Start increment slowly?

TCP: Slow Start



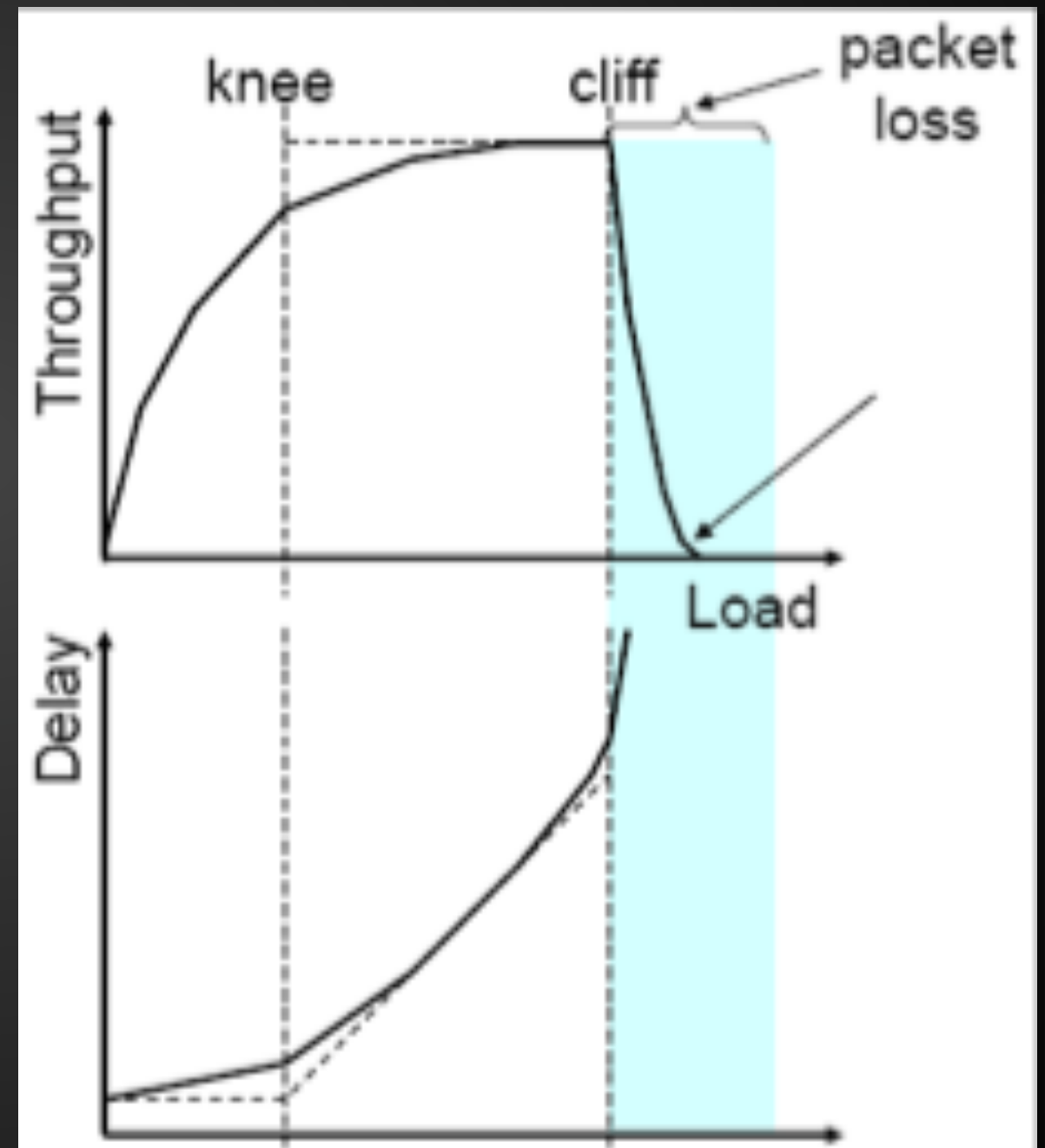
TCP: Slow Start

- TCP slows down the increase of cwnd when
 $\text{cwnd} \geq \text{ssthresh}$
- Slow-start threshold (ssthresh) is used to determine whether the slow-start or congestion avoidance algorithm is used to control data transmission

TCP: CA

Goal:

- maintain operating point at the left of the cliff:



TCP: CA

- How?
 - **Additive increase:** starting from the rough estimate, slowly increase cwnd to probe for additional available bandwidth
 - **Multiplicative decrease:** cut congestion window size aggressively if packet loss detected

TCP: CA

- Slowing down “Slow Start”:
 - If $\text{cwnd} > \text{ssthresh}$ then
 - each time a segment is acknowledged increment cwnd by $1/\text{cwnd}$ ($\text{cwnd} += 1/\text{cwnd}$).
 - Effectively cwnd is increased by one only if all segments have been acknowledged
(approx. by one in one RTT)

Fast Retransmit

- if segment lost TCP slow to retransmit
- **fast retransmit**
 - if receive 3 ACKs for same segment then immediately retransmit since likely lost
- **fast recovery**
 - lost segment means congestion
 - halve window then increase linearly
 - avoids slow-start

Slow start, CA, Fast Retransmit, Fast Recovery

