TCP: Congestion Control

Required reading:
Kurose 3.6, 3.7

EECS 3214, Winter 2020
Instructor: N. Vlajic

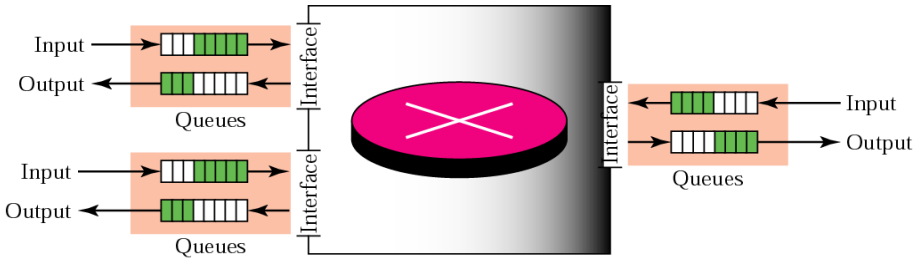
Congestion Control

Network Congestion Causes – occurs if **network load** (number of packets sent to network) is greater than **network capacity** (number of packets network can handle)

- too many sources sending too much data too fast for network to handle

Network Congestion Manifestations

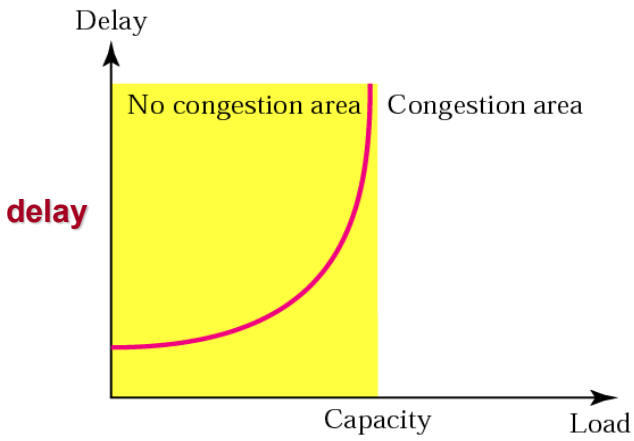
1) long packet delays due to long queueing delays in router buffers
2) lost packets (decreased throughput) due to buffer overflow at routers

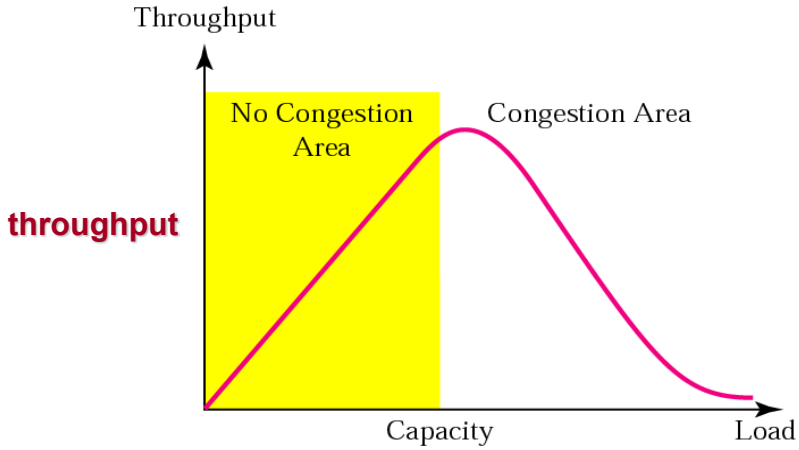


Congestion Control – set of mechanisms and techniques to control network congestion and keep overall traffic load below network capacity

Congestion Control (cont.)

Example **[network / congestion performance measures]**





How can you explain the exponential increase in *delay* as *load* approaches *capacity* and the exponential decrease in *throughput* as *load* exceeds *capacity*?

Network load has a "positive feedback" effect on network delay increase and network throughput decrease.

1) **When a packet is delayed, the source, not receiving the acknowledgment, retransmits the packet, which makes the queues longer and delays (i.e. congestion) worse!**

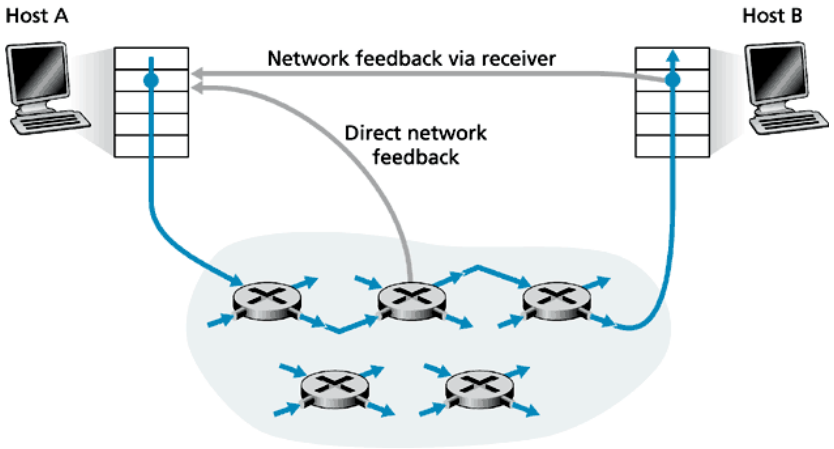
2) **When the load exceeds the capacity, the queues become full and the routers have to discard some packets. More discarded packets ⇒ more retransmitted packets ⇒ less room for regular (non-retransmitted) packets in queues ⇒ lower throughput**

Congestion Control (cont.)

Approaches Towards Congestion Control

1) **network assisted congestion control – also known as explicit congestion control**

- **network-layer components (routers) provide explicit feedback to end systems about congestion state in the network – feedback could be sent in either direction:**
  - **(a) from a network router directly back to the sender**
  - **(b) a network router marks a field in a packet flowing from the sender to the receiver; upon receipt of the marked field the receiver notifies the sender …**
- **example: ICMP Source Quench - not mandatory**



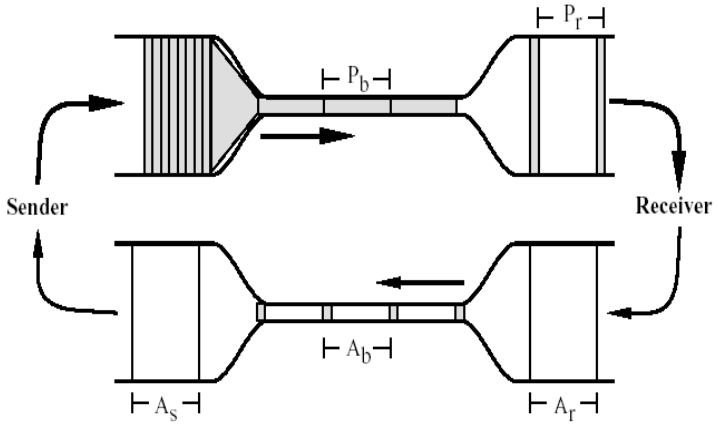
Congestion Control (cont.)

Approaches Towards Congestion Control (cont.)

2) **end-to-end congestion control – also known as implicit congestion control**

- **network-layer provides NO explicit support for congestion-control purposes**
- **end system infers congestion based on observed network behavior (e.g. packet loss, delay, etc.)**
- **example: TCP window-based congestion control**

  (e.g. TCP packet loss is taken as an indication of network congestion and TCP decreases its sending window size accordingly)



TCP must use end-to-end congestion control, since the IP layer provides no explicit feedback to the end systems regarding network congestion.

TCP Congestion Control

Causes of Packet Loss in TCP/IP Networks

What should sender do?!

1) receiver is overwhelmed with data and drops packets, or receiver sends new RcvWindow – flow control problem
   - solution: sender should decrease sending rate

2) a router on the path between sender and receiver is congested and drops packets – congestion control problem
   - solution: sender should decrease sending rate

Slow receiver and slow network require same response by the sender!!!

In TCP, the sender's window size should be controlled not only by the receive window size (RcvWindow) but also by congestion in the network.

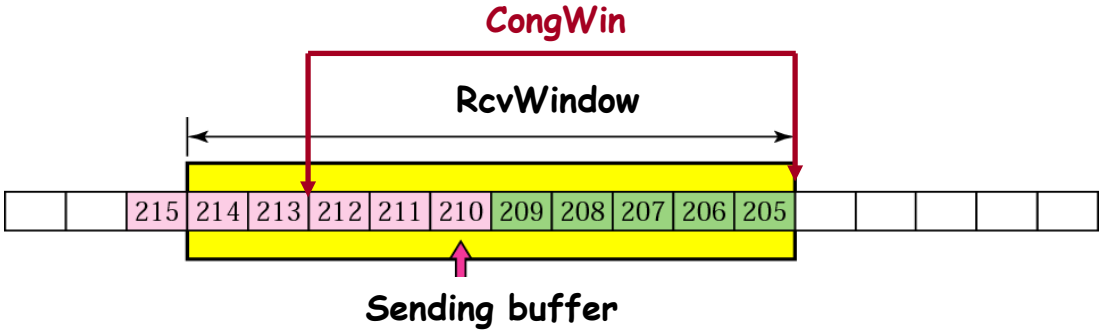
This was ignored in our earlier discussion on TCP Flow Control.

Congestion Window – (CongWin) imposes indirect constraint on rate at which TCP sender can send traffic into network

- amount of unacknowledged data may not exceed minimum of CongWin and RcvWindow !!!

actual window size ≤ min{ RcvWindow, CongWin } →

TCP Congestion Control (cont.)



CongWin is a dynamic parameter which get continually recalculated by the sender – its value changes as a function of perceived network congestion.

TCP Congestion Control (cont.)

How is CongWin size controlled?

1) on every "loss event" ⇒ reduce CongWin
2) on every received ACK ⇒ increase CongWin
   - increase in CongWin is self-clocking – if ACKs arrive at slow rate, CongWin will be increased at slow rate; if ACKs arrive at high rate, CongWin will be increased more quickly

How is congestion "loss event" perceived?

– a TCP sender assumes there is congestion on the path between itself and destination if a TCP segment gets lost/dropped (so called "loss event") indicated through

(a) **timeout** – packet lost, nothing came through after – severe!

(b) **receipt of three duplicate ACKs (fast retransmit) –** packet lost, but some subsequent packets arrived successfully - not as severe!

TCP Congestion Control Algorithm

– detailed algorithm that TCP sender uses to regulate its sending rate – has 3 major components:

1) slow start
2) additive increase
3) multiplicative decrease

congestion avoidance

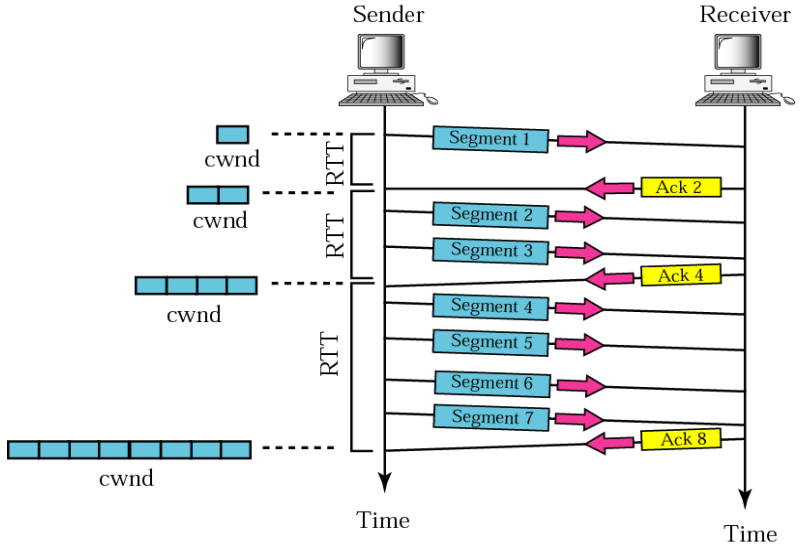
TCP Congestion Control (cont.)

Slow Start

- when TCP connection begins, CongWin is typically initialized to 1 [MSS] ⇒ initial sending rate = MSS/RTT
- for each acknowledged segment, sender increases its CongWin by 1 [MMS] ⇒ CongWin value is doubled every RTT ⇒ CongWin grows exponentially
- "slow start" refers to fact that initial rate is slow but it rumps up exponentially

Slowstart algorithm

```
initialize: Congwin = 1 [MSS]
until (loss event OR CongWin > threshold) {
    for each segment ACKed Congwin++ }
```



Slow-start algorithm works effectively for initializing a connection – it enables the TCP sender to determine quickly a reasonable window size for connection.

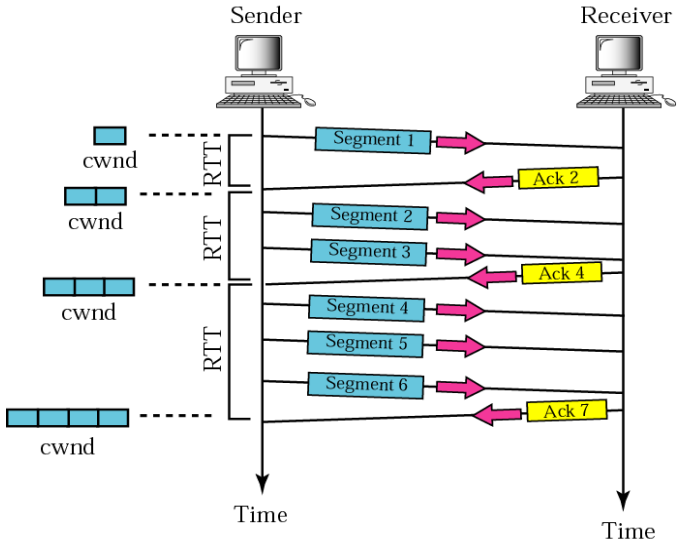
TCP Congestion Control (cont.)

Congestion Avoidance (Additive Increase)

1) set CongWin = 1 [MSS] and perform Slow Start until CongWin = threshold (see step 3))
2) **Additive Increase**: for CongWin > threshold increase CongWin by 1 [MSS] every RTT
3) **Multiplicative Decrease**: on "loss event" set threshold to new value: threshold = CongWin/2, and go to 1) – back to Slow Start

Congestion Avoidance

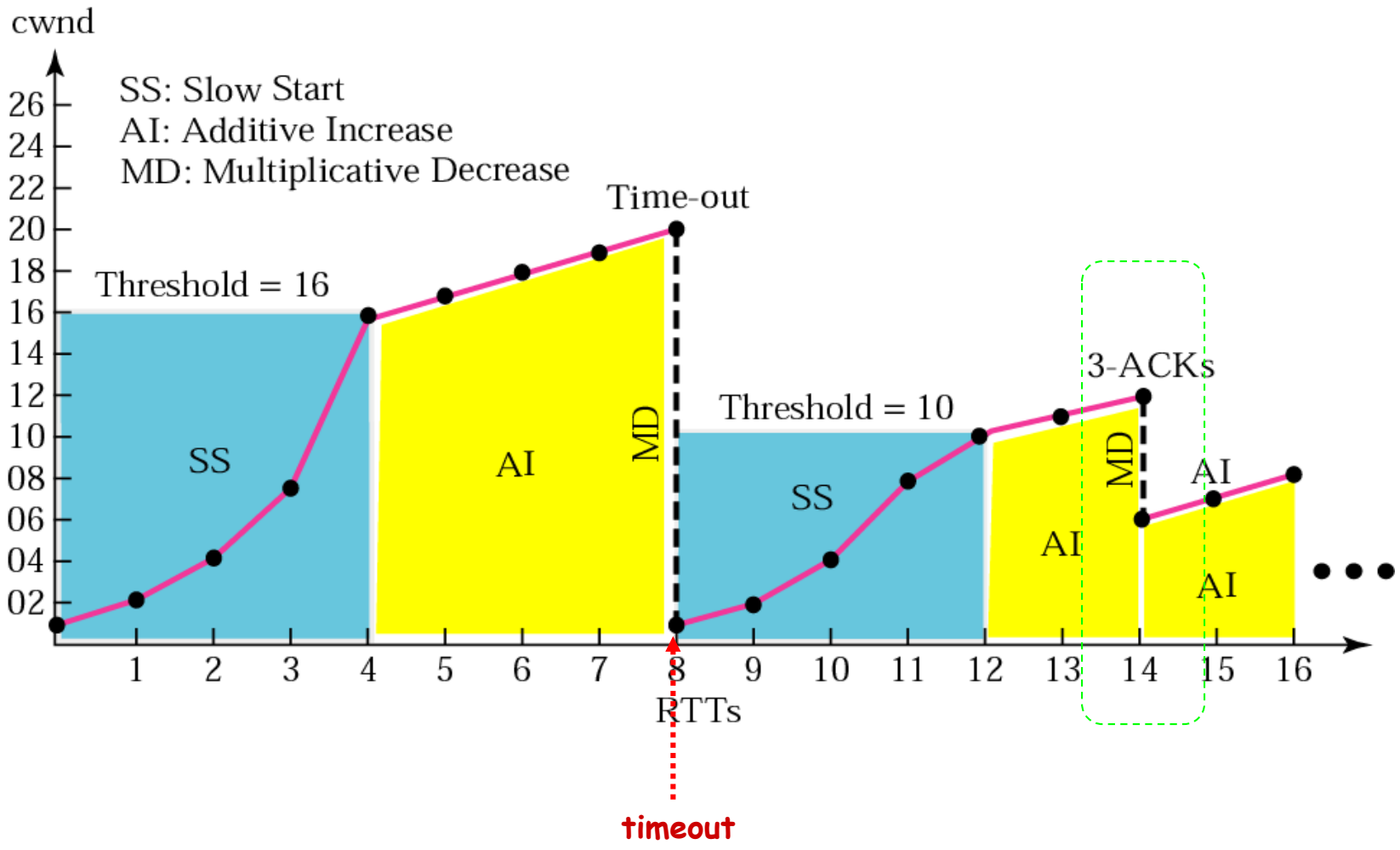
```
/* slowstart is over */
/* CongWin > threshold */
until (loss event) {                1
  for every CongWin segments ACKed CongWin++ }
threshold = Congwin/2
Congwin = 1 [MSS]
perform Slowstart
```



Slow Start vs. Additive Increase – after a certain point it may be unwise to keep increasing the congestion window exponentially, since overshoot could be significant.

TCP Congestion Control (cont.)

Example **[TCP congestion control]**

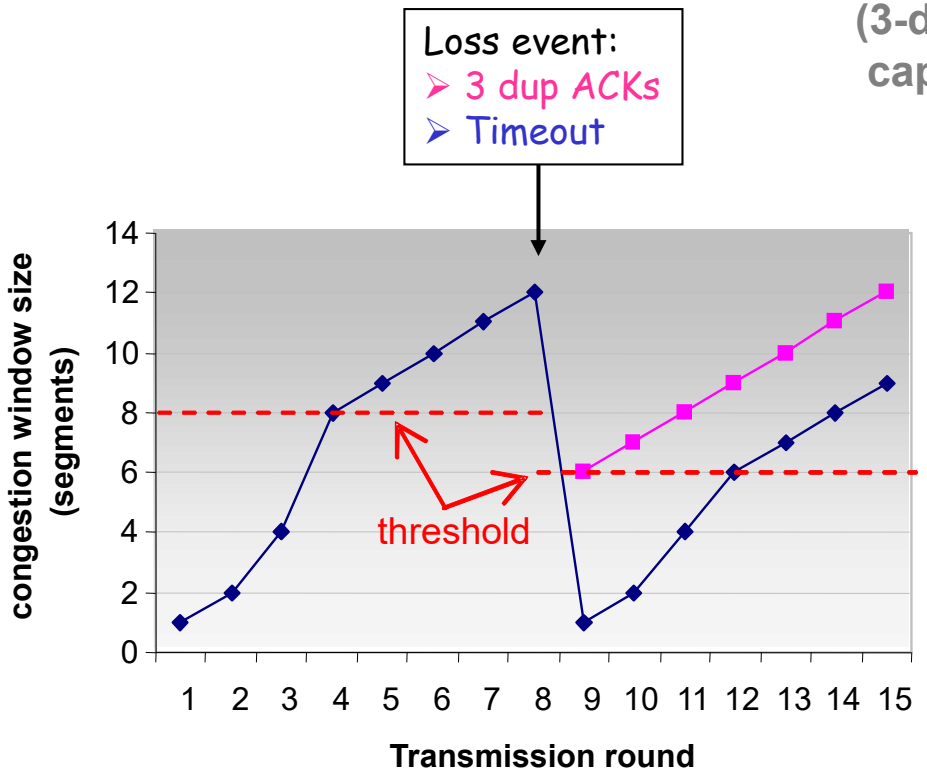


TCP Congestion Control (cont.)

Early-version TCP (Tahoe) vs. New-version TCP (Reno)

- early TCP unconditionally cuts CongWin to 1 MSS, regardless …
- new TCP cancels slow-start phase after a 3-duplicate ACK – instead, does fast retransmit, and fast recovery: set threshold = CongWin/2, CongWin = threshold, and continue with Additive Increase

(3-duplicate ACKs indicate that network is capable of delivering at least some packets)



Reno TCP

```
if (loss event = timeout) {
   CongWin = 1 [MSS]
   perform Slowstart }
if (loss event = 3-duplicate ACK) {
   CongWin = threshold
   perform linear increase }
```

TCP Sender Congestion Control – Overview

Event	State	TCP Sender Action	Commentary
ACK receipt for previously unacked data	Slow Start (SS)	CongWin = CongWin + MSS, if (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
ACK receipt for previously unacked data	Congestion Avoidance (CA)	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
Loss event detected by triple duplicate ACK	SS or CA	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
Timeout	SS or CA	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Enter slow start

TCP Congestion Control (cont.)

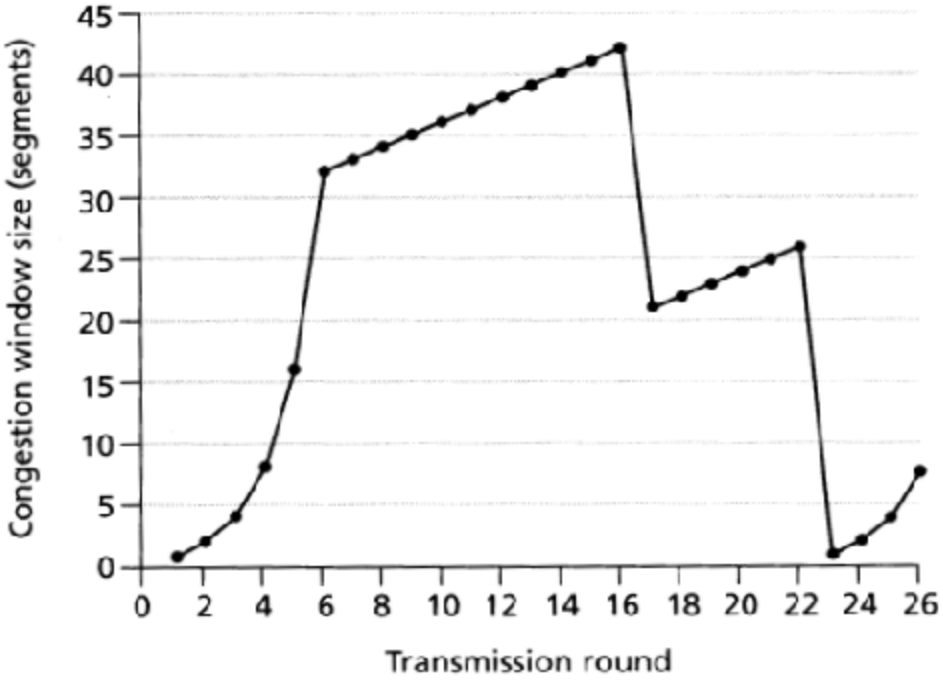
Example **[network / congestion performance measures]**

a) Identify time intervals where TCP is operating in 'slow start'.

[1, 6], [23, 26]

b) Identify time intervals where TCP is operating in 'congestion avoidance'.

[6, 16], [17, 22]



c) After 16th transmission, is lost segment detected by timeout or by duplicate ACK?

duplicate ACK

d) After 22nd transmission, is lost segment detected by timeout or by duplicate ACK?

timeout

e) What is ssthreshold at 1st transmission round?

32

f) What is ssthreshold at 18th transmission round?

21 – ½ of congestion window when last loss was detected

g) What is ssthreshold at 24th transmission round?

13

Characteristic / Description	UDP	TCP
General Description	Simple, high-speed, low-functionality "wrapper" that interfaces applications to the network layer and does little else.	Full-featured protocol that allows applications to send data reliably without worrying about network layer issues.
Protocol Connection Setup	Connectionless; data is sent without setup.	Connection-oriented; connection must be established prior to transmission.
Data Interface To Application	Message-based; data is sent in discrete packages by the application.	Stream-based; data is sent by the application with no particular structure.
Reliability and Acknowledgments	Unreliable, best-effort delivery without acknowledgments.	Reliable delivery of messages; all data is acknowledged.
Retransmissions	Not performed. Application must detect lost data and retransmit if needed.	Delivery of all data is managed, and lost data is retransmitted automatically.
Features Provided to Manage Flow of Data	None	Flow control using sliding windows; window size adjustment heuristics; congestion avoidance algorithms.
Overhead	Very low	Low, but higher than UDP
Transmission Speed	Very high	High, but not as high as UDP
Data Quantity Suitability	Small to moderate amounts of data (up to a few hundred bytes)	Small to very large amounts of data (up to gigabytes)
Types of Applications That Use The Protocol	Applications where data delivery speed matters more than completeness, where small amounts of data are sent; or where multicast/broadcast are used.	Most protocols and applications sending data that must be received reliably, including most file and message transfer protocols.
Well-Known Applications and Protocols	Multimedia applications, DNS, BOOTP, DHCP, TFTP, SNMP, RIP, NFS (early versions)	FTP, Telnet, SMTP, DNS, HTTP, POP, NNTP, IMAP, BGP, IRC, NFS (later versions)

source: http://www.tcpipguide.com/free/t_SummaryComparisonofTCPIPTransportLayerProtocolsUDP.htm

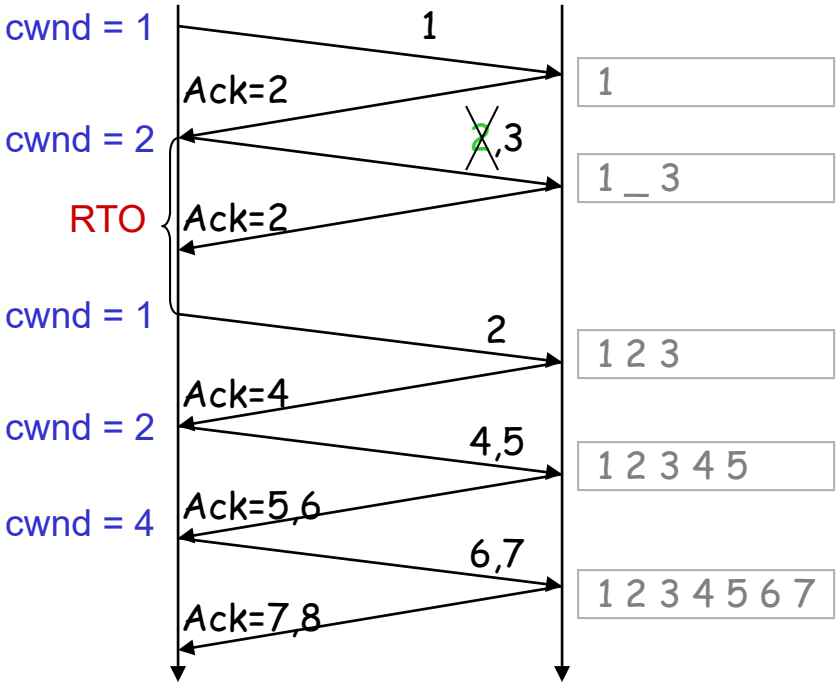
Example [TCP congestion control - Final Exam, Fall 2004]

Consider a TCP Reno flow that sends 7 packets. Assume the TCP source experiences exactly one packet loss before finishing the transmission. Let:

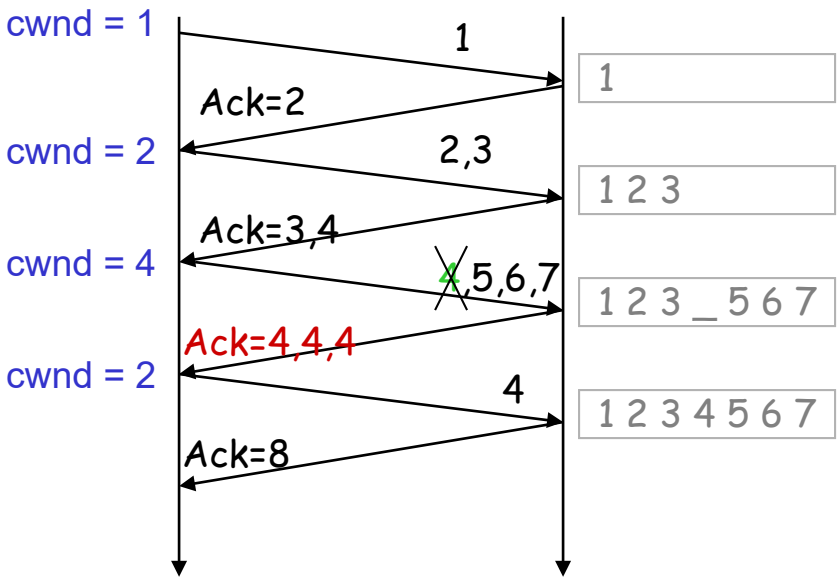
- **Threshold = 10;**
- **d denote the one-way propagation delay between the source and the receiver;**
- **RTO denote the TCP retransmission timeout;**
- **T denote the total time it takes the source to transmit all packets.**

Determine which packet has to be dropped to result in minimum and which one in maximum T. Compute T in both cases as a function of d and RTO.

Note: Assume the TCP connection has been fully established prior to sending the 7 packets in question. Also, the TCP implements both fast recovery and fast retransmission.



T = RTO + 4*(2d) = RTO + 8d



T = 4*(2d) = 8d