SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Lec6 – The Transmission Control Protocol

50.012 Networks

Jit Biswas

Cohort 1: TT7&8 (1.409-10)

Cohort 2: TT24&25 (2.503-4)

Introduction

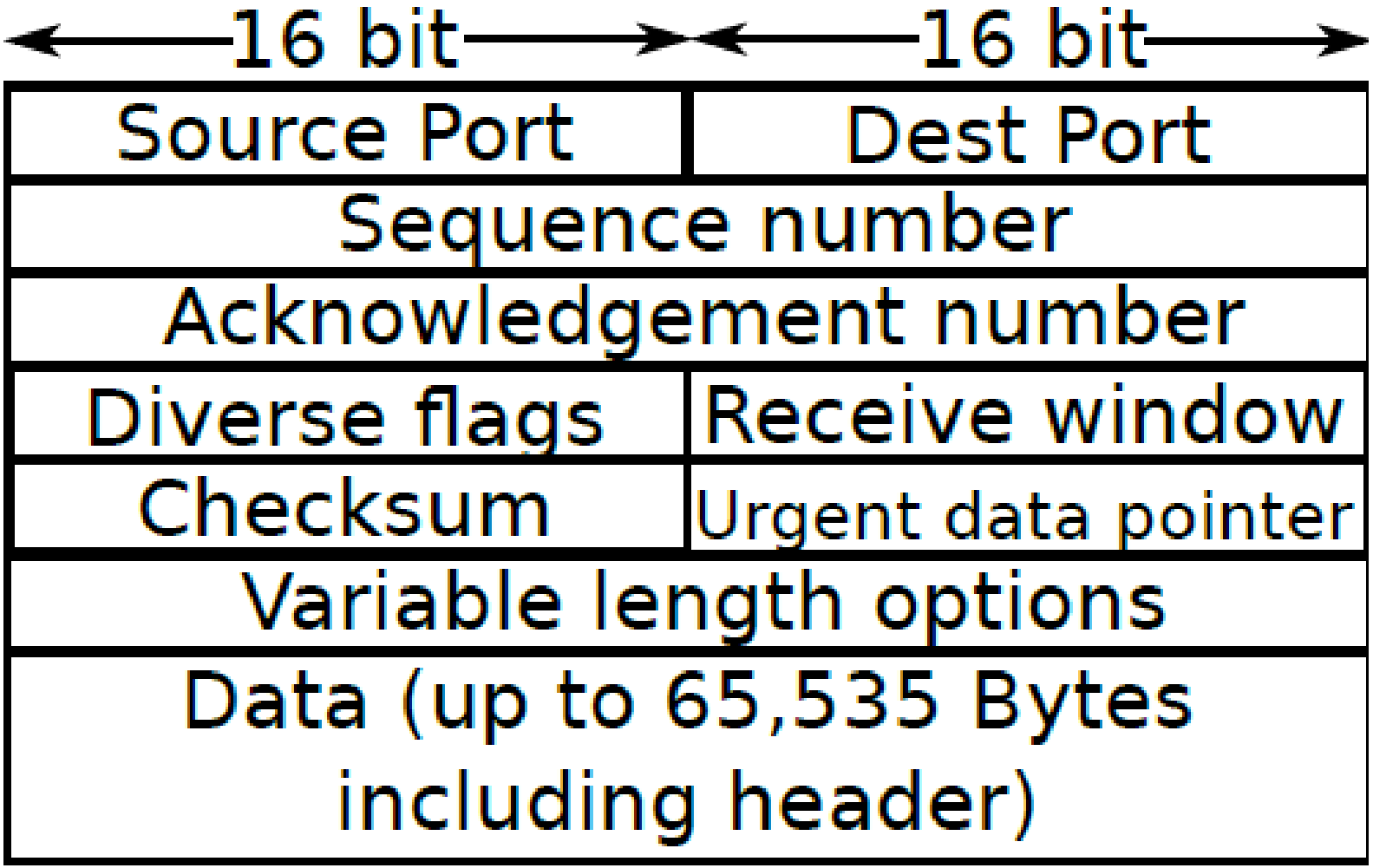
- Today's lecture:
  - TCP in detail
  - Sequence numbers, handshake, flow control
  - Congestions, how TCP handles them
- Note: parts of this slide set are based on Kurose & Ross chapter 3 slide set

The Transmission Control Protocol

TCP Overview

- The Transport Control Protocol (TCP) is the most commonly used transport protocol
- Streaming protocol, as it provides a continuous connection between single sender and receiver
  - Initial handshake is used to set up connection
- Pipelined cumulative ACKs, based on flow and congestion control
  - Sender adapts rate to buffer size of receiver

TCP header structure



Flags in the TCP header

- The 16 *diverse flags* field contains:
  - Header length (due to variable length options)
  - Some unused bits
  - Urgent flag (generally not used)
  - ACK flag (consider ack number acknowledged)
  - Push flag (generally not used)
  - RST, SYN, FIN flags: for connection management, e.g. handshaking

Acknowledgments in TCP

- Segment number is incremented by Byte size of data payload
  - ID numbers can be large - 32 bit ~ 4GByte until overflow
- Acknowledgement number is referring to expected next segment
- ACKs can be integrated in normal segments
- TCP header has explicit acknowledgement number field and ACK flag
- ACK is cumulative - i.e., receiver acknowledges receipt of all previous payload
- ACK transmission can be delayed by Receiver, wait for 500ms if next packet comes

TCP seq. numbers, ACKs

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

sequence numbers:

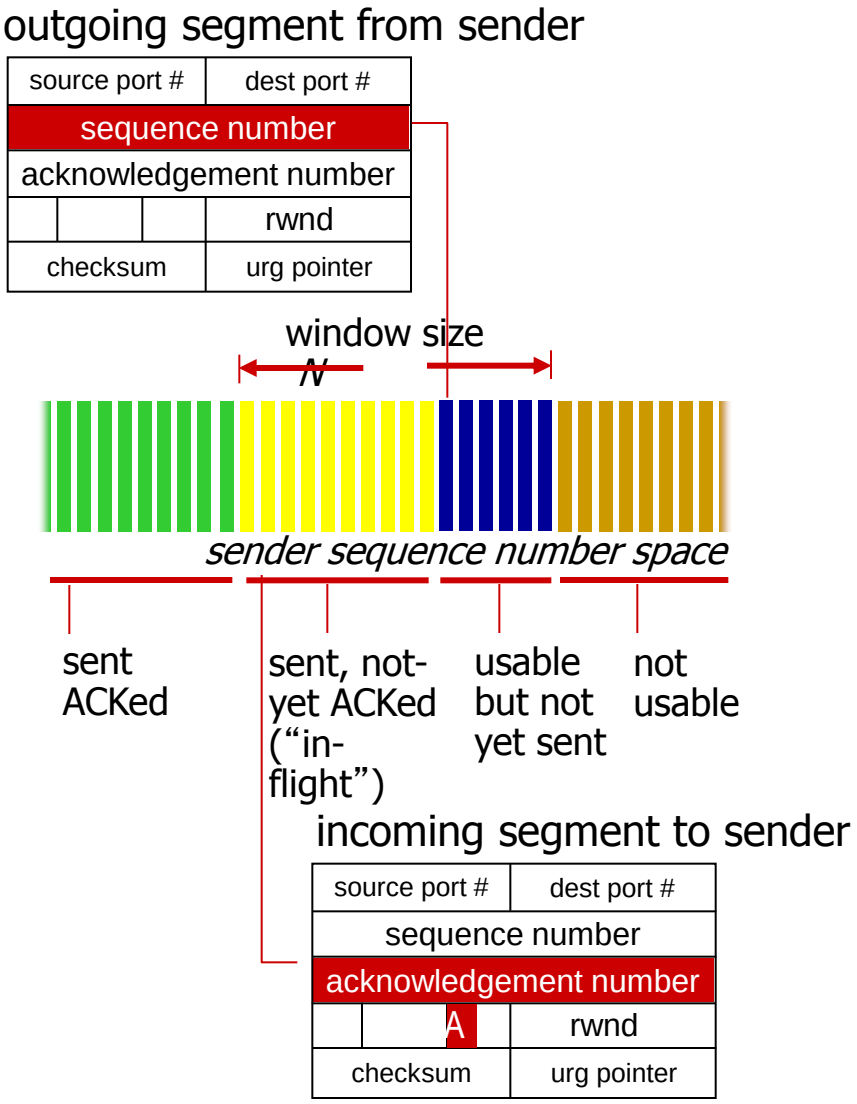
- byte stream "number" of first byte in segment's data

acknowledgements:

- seq # of next byte expected from other side
- cumulative ACK

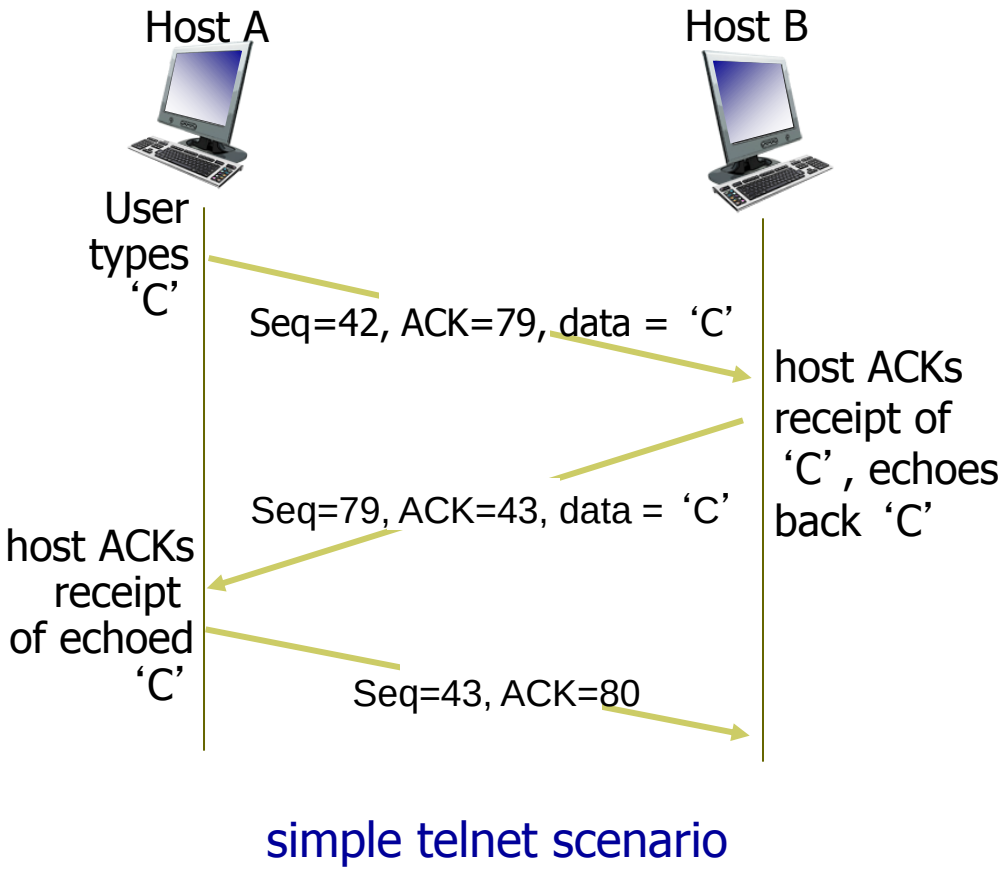
Q: how receiver handles out-of-order segments

- A: TCP spec doesn't say, - up to implementor



Source: K&R Chap 3 foil set

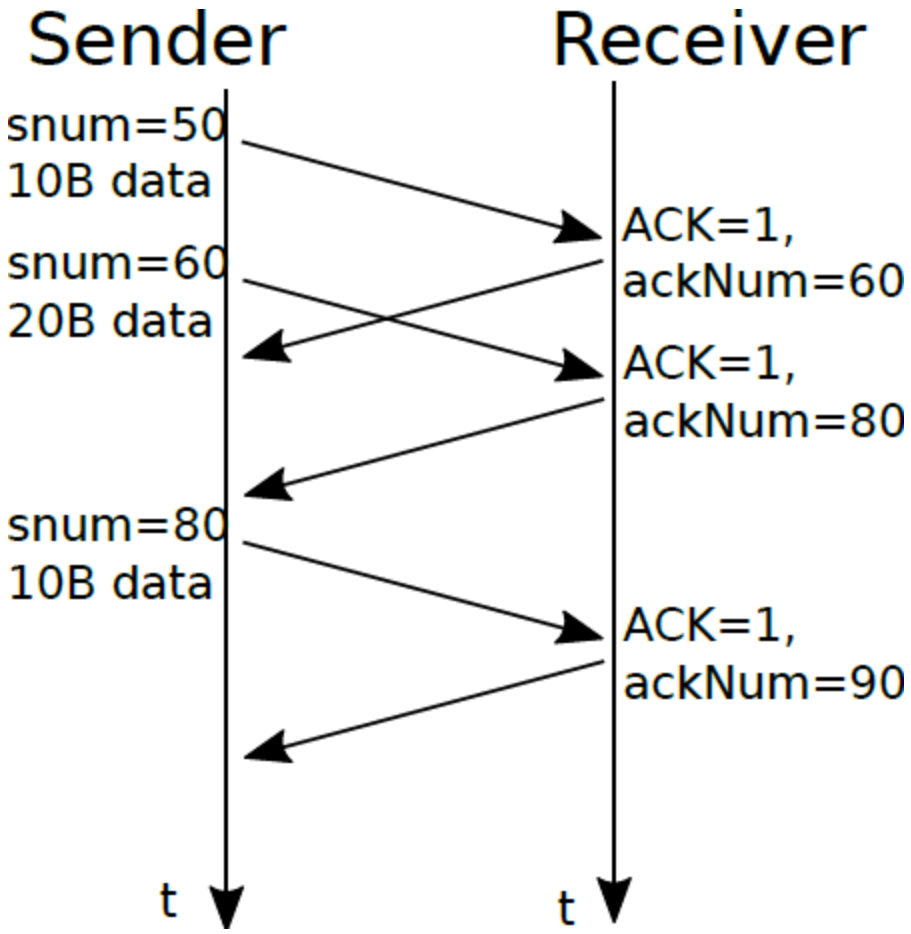
TCP seq. numbers, ACKs



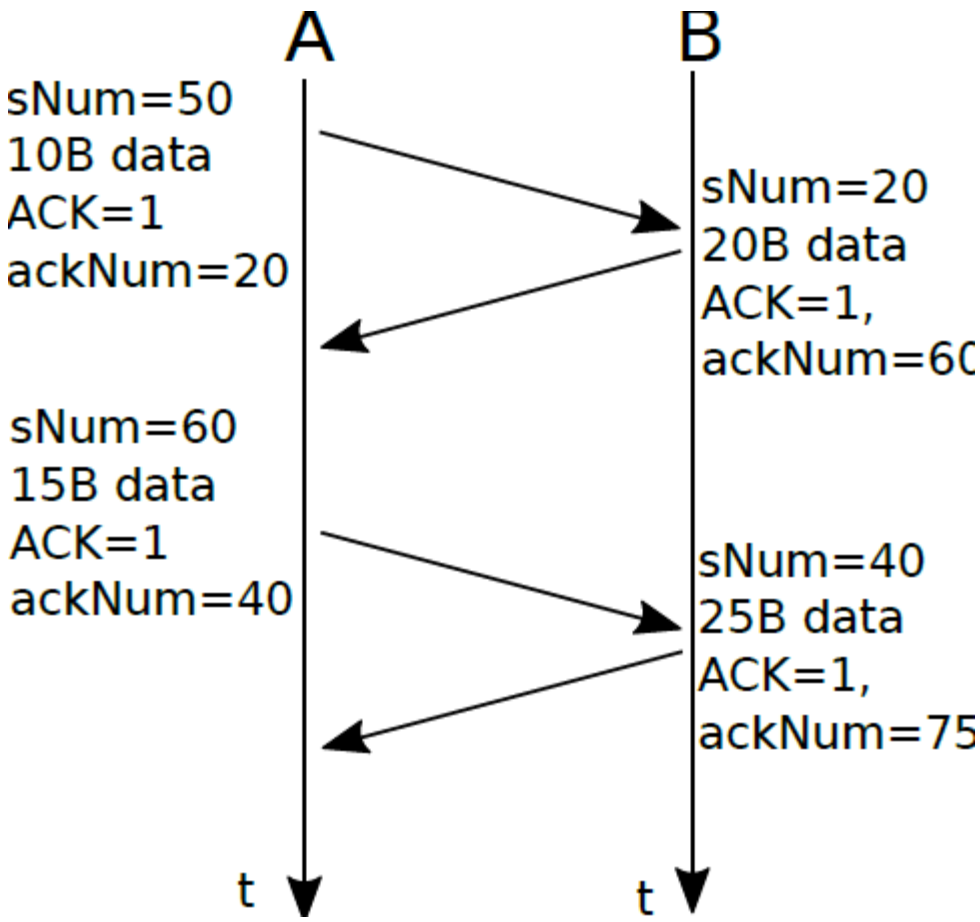
simple telnet scenario

Source: K&R Chap 3 foil set

TCP ACK visualization



TCP Full Duplex ACK



Bi-directional (full duplex) ACK and data exchange

TCP round trip time, timeout

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Q: how to set TCP timeout value?

- longer than RTT
  - but RTT varies
- *too short:* premature timeout, unnecessary retransmissions
- *too long:* slow reaction to segment loss

Q: how to estimate RTT?

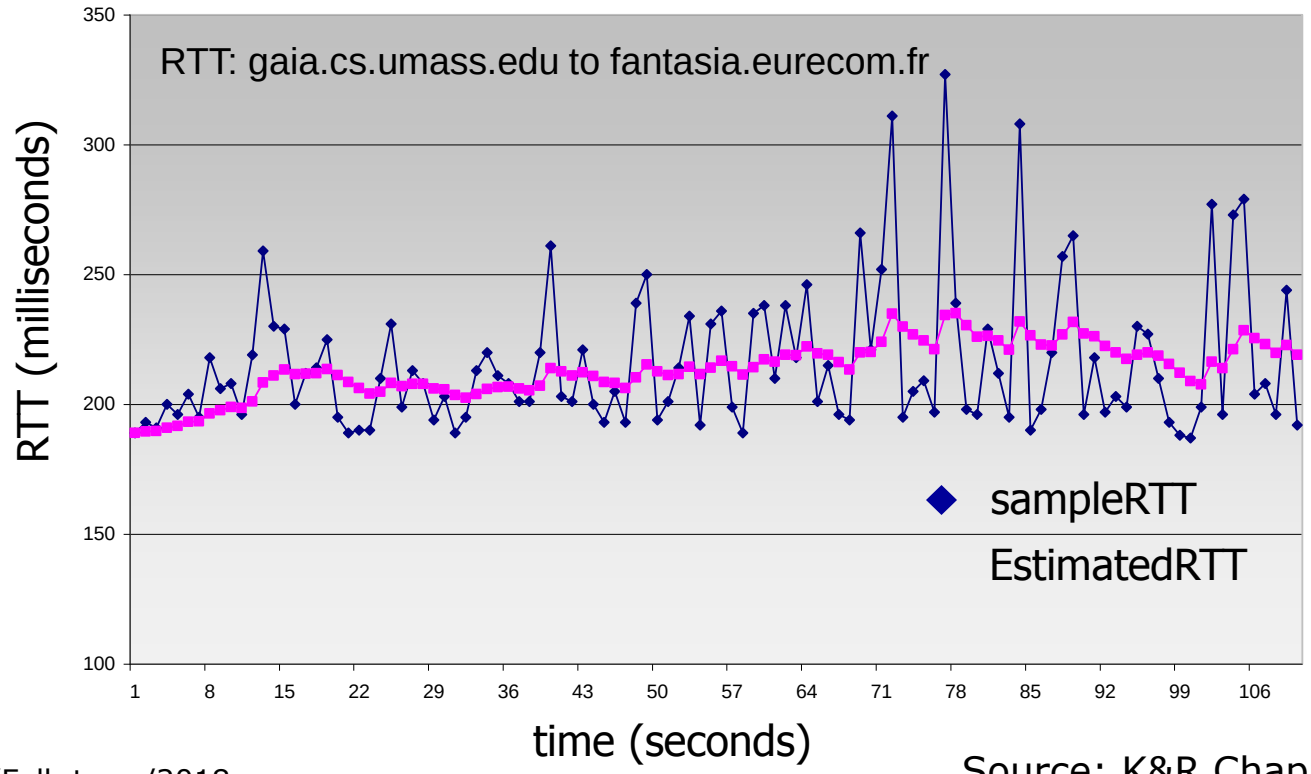
- **SampleRTT**: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **SampleRTT** will vary, want estimated RTT "smoother"
  - average several *recent* measurements, not just current **SampleRTT**

Source: K&R Chap 3 foil set

TCP round trip time, timeout

`EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT`

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$



Source: K&R Chap 3 foil set

TCP timeout: Summary

- The TCP timeout for ACKs is based on continuous measurements
- EstimatedRTT= (1 - α) * EstimatedRTT + α * SampleRTT
- Choice of α determines how quickly RTT window adapts
  - Typical choice: α = 0:125
- Safety margin is then added with second variable
  - DevRTT= (1 - β) * DevRTT + β * | SampleRTT - EstimatedRTT |

TimeoutInterval = EstimatedRTT + 4*DevRTT

estimated RTT

"safety margin"

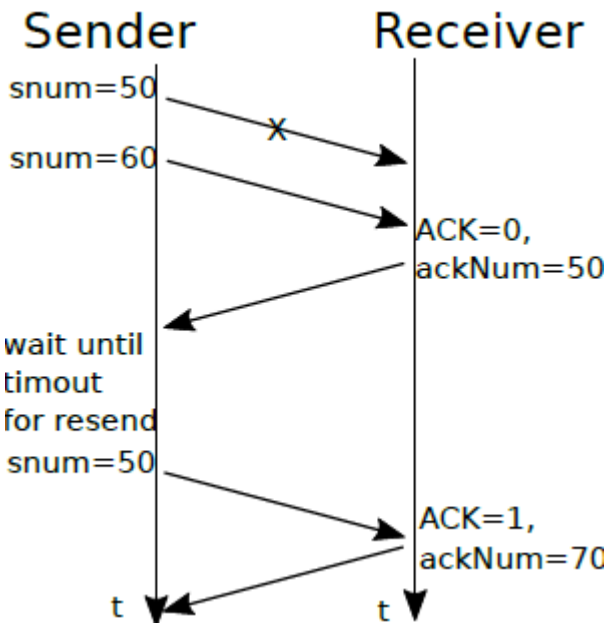
- TCP timeout is typically larger than 500*ms*

TCP timeout handling

- Timer is counting for oldest unACK'ed segment
- If ACK is received: start timer for next segment
- If ACK is not received in time: retransmit old segment, restart timer
  - If receiver has following segments in receive buffer, he will ACK that content as well

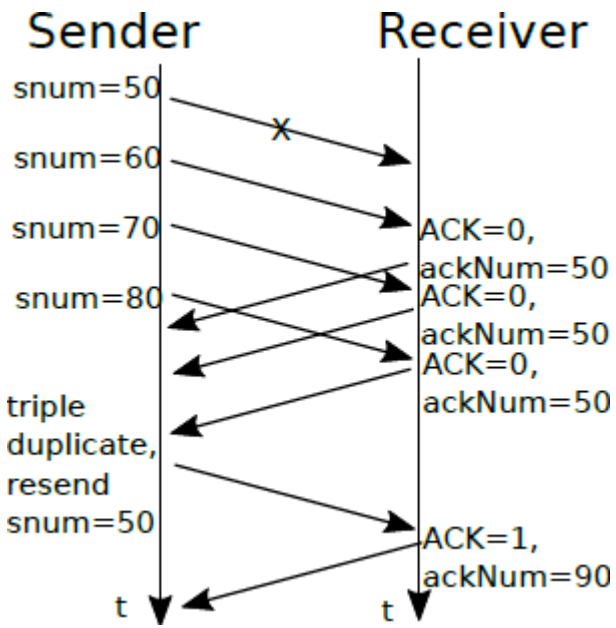
Duplicate ACK

- If receiver receives segment with higher number than expected, she sends duplicate ACK
  - Send a TCP message with the ack number for the segment she is waiting for
  - Without setting the ACK flag
- This tells the sender that the receiver is still waiting for some content



Triple Duplicate ACKs

- If sender receives three of such duplicate ACKs, she will not wait for ACK any longer
  - She will immediately re-send the requested segment
  - Receiver will then send acknowledgement that includes out-of-order segments with higher number



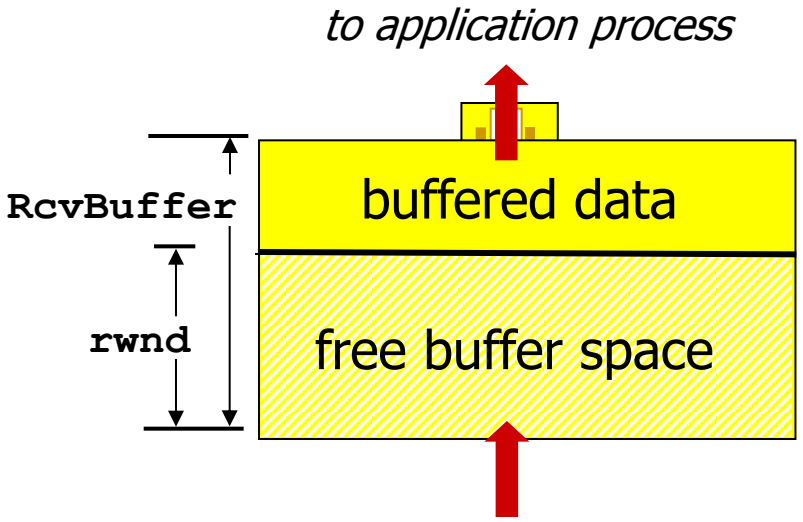
TCP flow control

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

- TCP header contains receive window field
- This specifies the free buffer size on the receiving end
- Sender should make sure that size of un-ACK'ed segments is smaller than this buffer
- This is called flow control
- Example: slow receiver, but fast sender
  - E.g. embedded device downloading data from server
  - Embedded device might be slow in processing buffer content
  - You set the maximal buffer size in your Python code, typically 4096 Bytes

TCP flow control

- receiver "advertises" free buffer space by including **`rwnd`** value in TCP header of receiver-to-sender segments
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unacked ("in-flight") data to receiver's **`rwnd`** value
- guarantees receive buffer will not overflow



receiver-side buffering

Source: K&R Chap 3 foil set

Connection Establishment

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

- TCP uses a 3 message handshake to establish connection
- Client starts with some random sequence number s_c, and sets the SYN bit
- Server chooses its own random sequence number s_s, and sets the SYN and ACK bit
  - Also increments the client's sequence number by one: $a_s = s_c + 1$
- Client responds with ACK of server's sequence number, and increased acknowledge number $a_c = s_s + 1$?

Connection Termination

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

- 2 Messages required for connection termination
- Either side sends segment with FIN set to 1
- Other side replies with ACK (and possibly FIN) bit set
- Until both sides sent FIN, connection is still alive
  - This enables the receiver of FIN to finish his transmission of data

TCP Congestion Control

Question

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

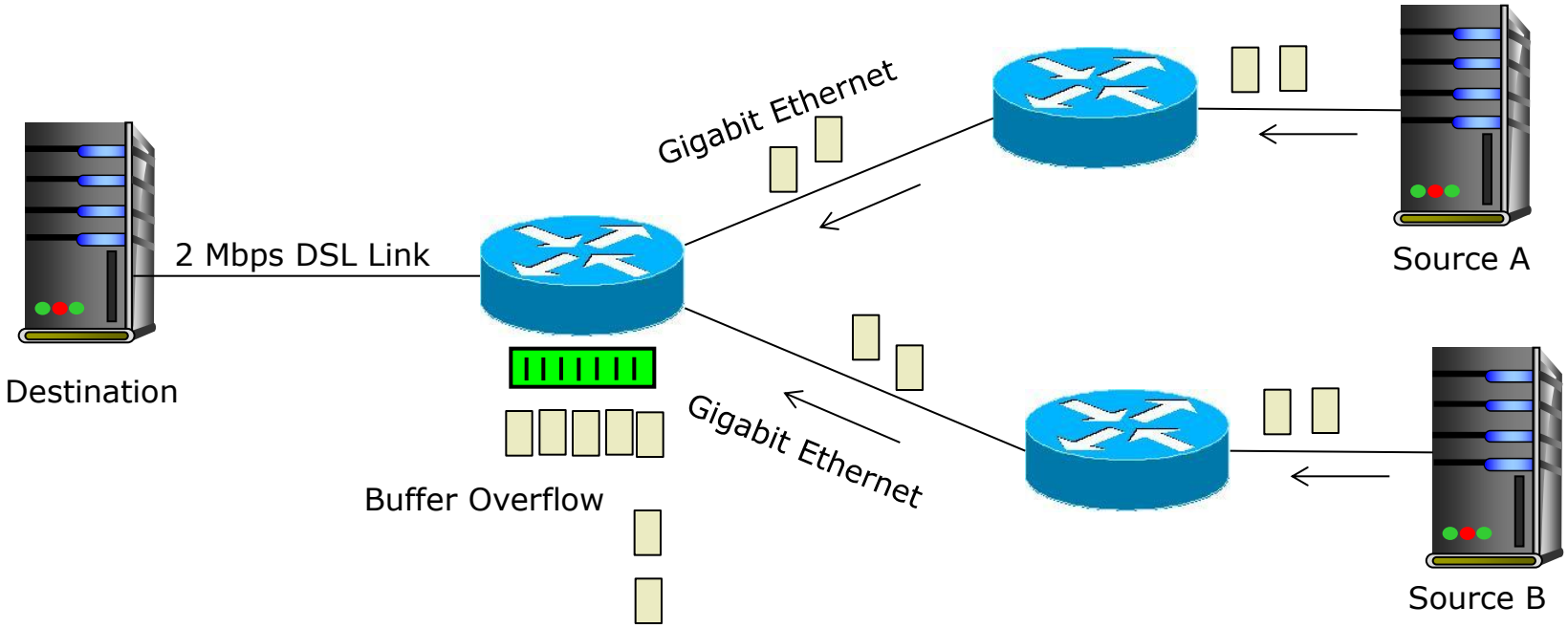
What is the root cause of congestion in the network?

Question

What is the root cause of congestion in the network?

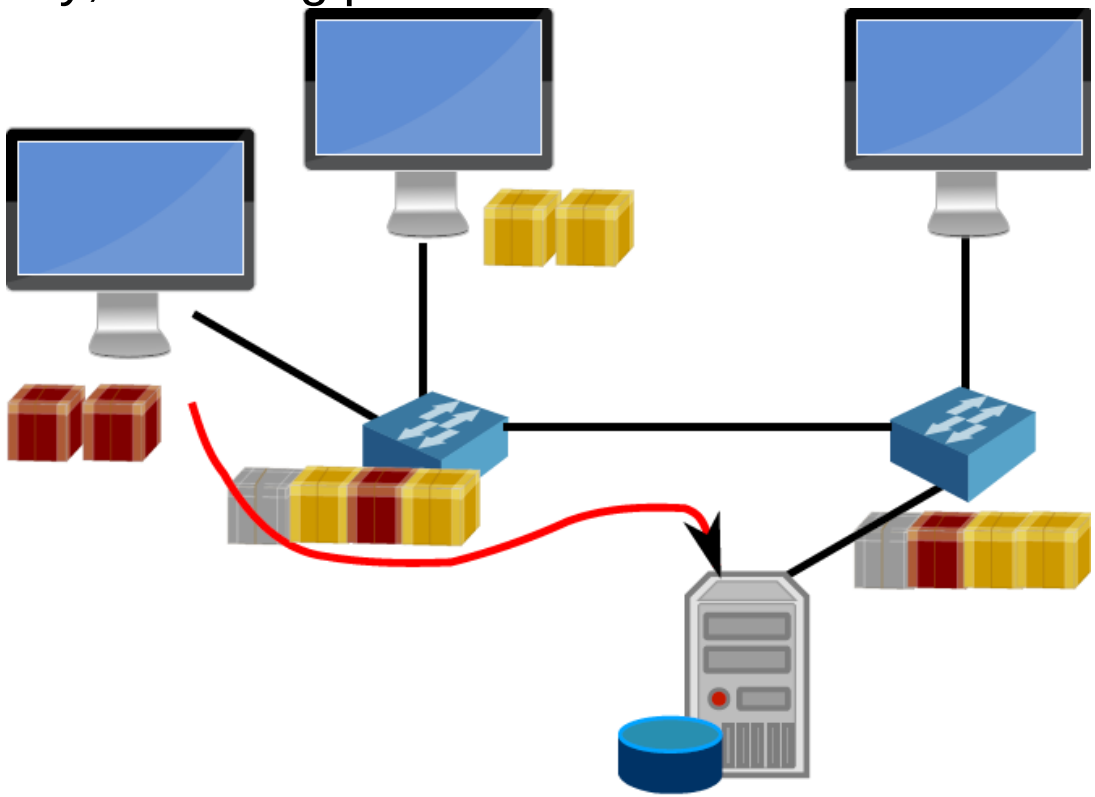
Too many packets injected at routers leading to:

- Network congestion
- Network collapse



Congestion

- Congestion is a network layer problem:
  - Network link cannot transmit packets fast enough
  - Forwarding routers will start to fill up transmission buffers
- Buffers at routers are finite
  - So eventually, incoming packets at router cannot be stored and are discarded



Effects of Congestion

- Retransmissions by sender
  - Additional effort for sender and links towards the congested router
  - If retransmitted too early, same segment can end up in buffer twice
- Congestions can also hurt the "back channel" of ACKs
- So a one-way congestion can hurt both directions
- Example: ADSL connection with low upload bandwidth
  - Uploading with maximal rate will hinder simultaneous downloads

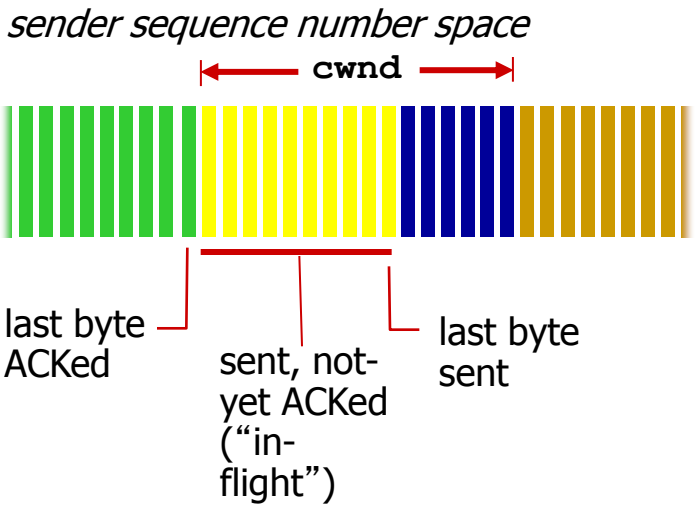
Handling Congestion

- Two approaches: end-to-end or network-assisted
- End-to-end:
  - Both parties recognize segment loss, start to reduce rate
  - This is done in TCP
- Network-assisted:
  - Congested router notifies the sender or recipient
  - TCP Explicit Congestion Notification (ECN) is an extension for TCP that allows this
  - Router sets a flag in IP header if link is congested

TCP Congestion control

- Approach: maximize transmission rate in fair way
  - Flow control ensures that received can handle data
  - Congestion control ensures that intermediate links are not overloaded
- Two phases
  - *Slow start* mode at beginning of TCP connection
  - *Congestion avoidance* mode after first dropped ACK
- In both phases the *congestion window* (cwnd) is adapted

TCP Congestion Control: details



- sender limits transmission:

$$\texttt{LastByteSent-LastByteAcked} \leq \texttt{cwnd}$$

- **cwnd** is dynamic, function of perceived network congestion

TCP sending rate:

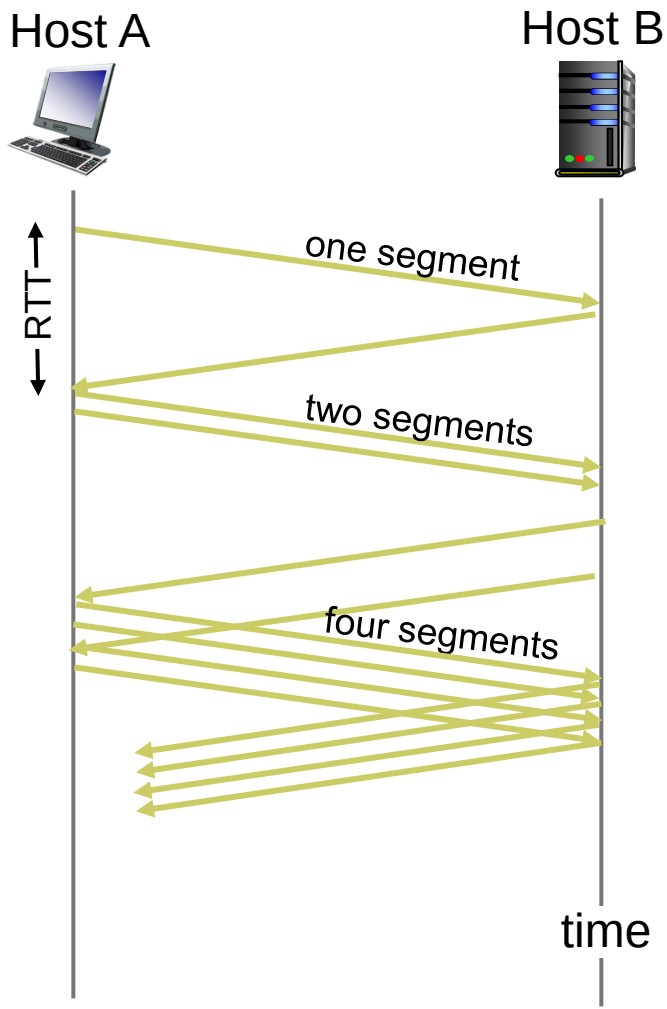
- *roughly:* send cwnd bytes, wait RTT for ACKS, then send more bytes

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

Source: K&R Chap 3 foil set

TCP Slow Start

- when connection begins, increase rate exponentially until first loss event:
  - initially **`cwnd`** = 1 MSS
  - double **`cwnd`** every RTT
  - done by incrementing **`cwnd`** for every ACK received
- *summary:* initial rate is slow but ramps up exponentially fast



Source: K&R Chap 3 foil set

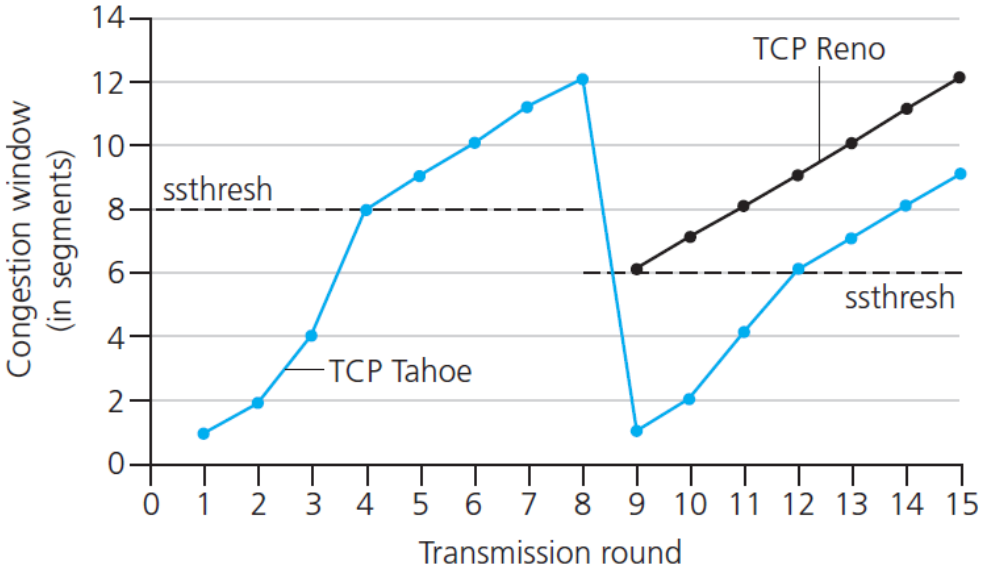
TCP: detecting, reacting to loss

- loss indicated by timeout:
  - **cwnd** set to 1 MSS;
  - window then grows exponentially (as in slow start) to threshold, then grows linearly
- loss indicated by 3 duplicate ACKs: TCP RENO
  - dup ACKs indicate network capable of delivering some segments
  - **cwnd** is cut in half window then grows linearly
- TCP Tahoe always sets **cwnd** to 1 (timeout or 3 duplicate acks)

TCP: switching from slow start to CA

Q: when should the exponential increase switch to linear?

A: when **cwnd** gets to 1/2 of its value before timeout.



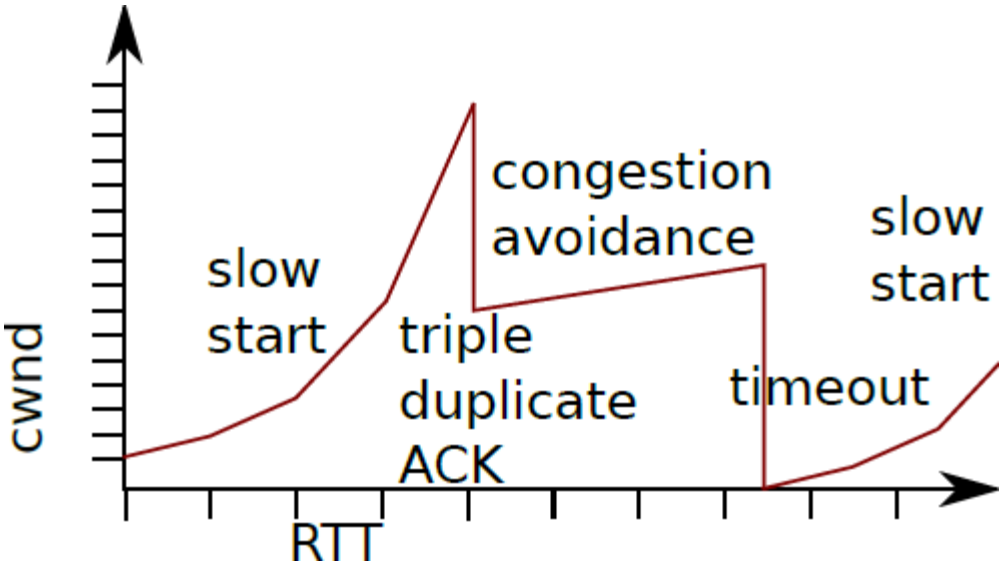
Implementation:

- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event

Source: K&R Chap 3 foil set

TCP Slow start and congestion avoidance

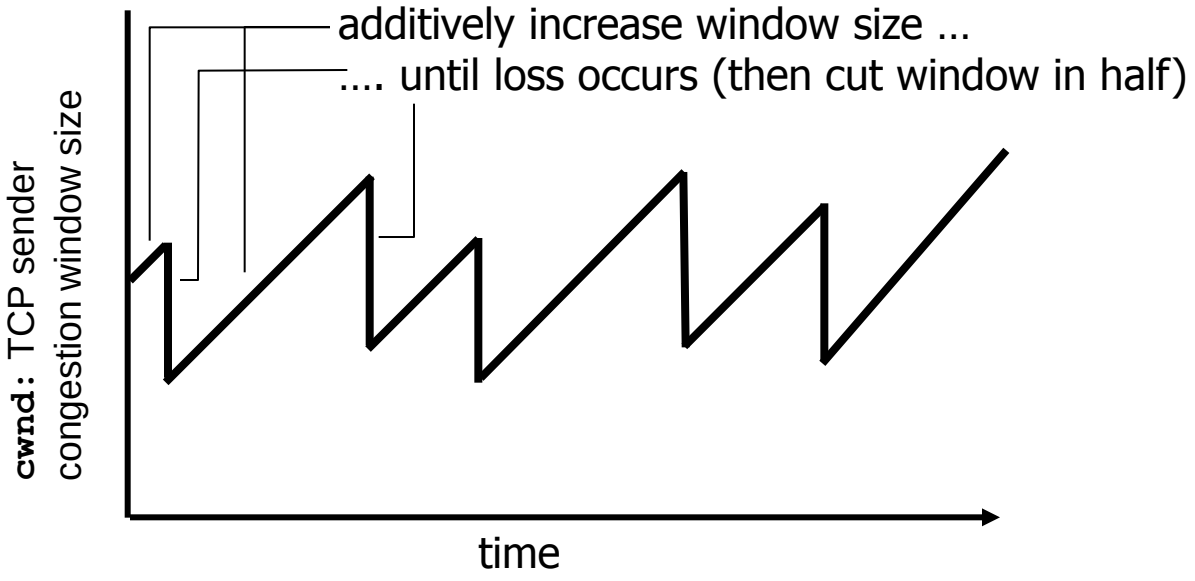
- At connection start: *slow* start phase, cwnd=1
- cwnd is then *increased* by one for each ACK until loss occurs
- When loss is detected
  - By triple-duplicate-ACK: cwnd is cut by half and *congestion avoidance mode*
  - By timeout: start in slow start phase from 1 again



TCP congestion control:
Additive Increase Multiplicative Decrease (AIMD)

- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **cwnd** by 1 MSS every RTT until loss detected
  - *multiplicative decrease*: cut **cwnd** in half after loss

AIMD saw tooth behavior: probing for bandwidth



Source: K&R Chap 3 foil set

Activity 6: TCP Congestion Window

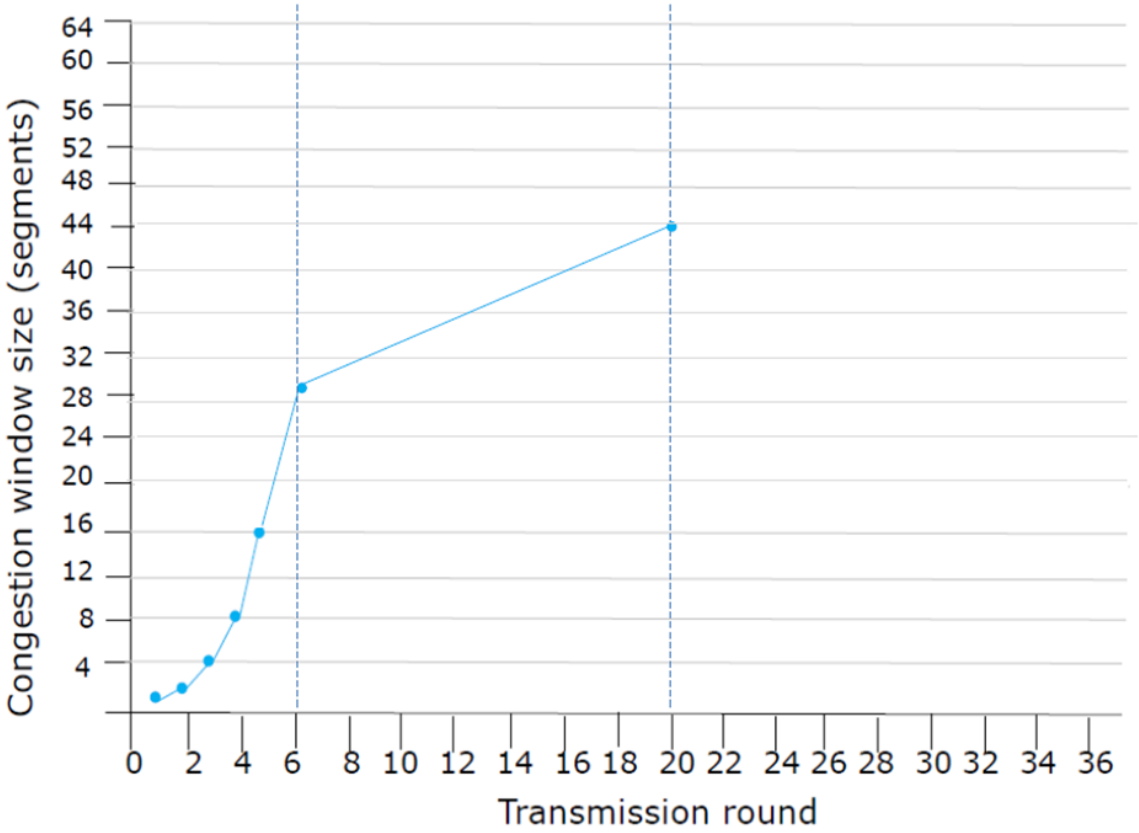
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Consider the congestion window plot shown in the figure below. Assuming TCP Reno is the protocol, answer the following questions with a brief explanation where needed:

a. Identify the TCP slow start period

b. Identify the TCP congestion avoidance period

c. After the 20th transmission round, segment loss is detected by triple duplicate ack. Plot the graph from transmission rounds 21 to 25. Describe what you have plotted in text form, and submit your answer on eDimension. You will finish the rest of the problem as your homework.



Homework 3: TCP Congestion Window cont'd

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Continued from Activity 5:

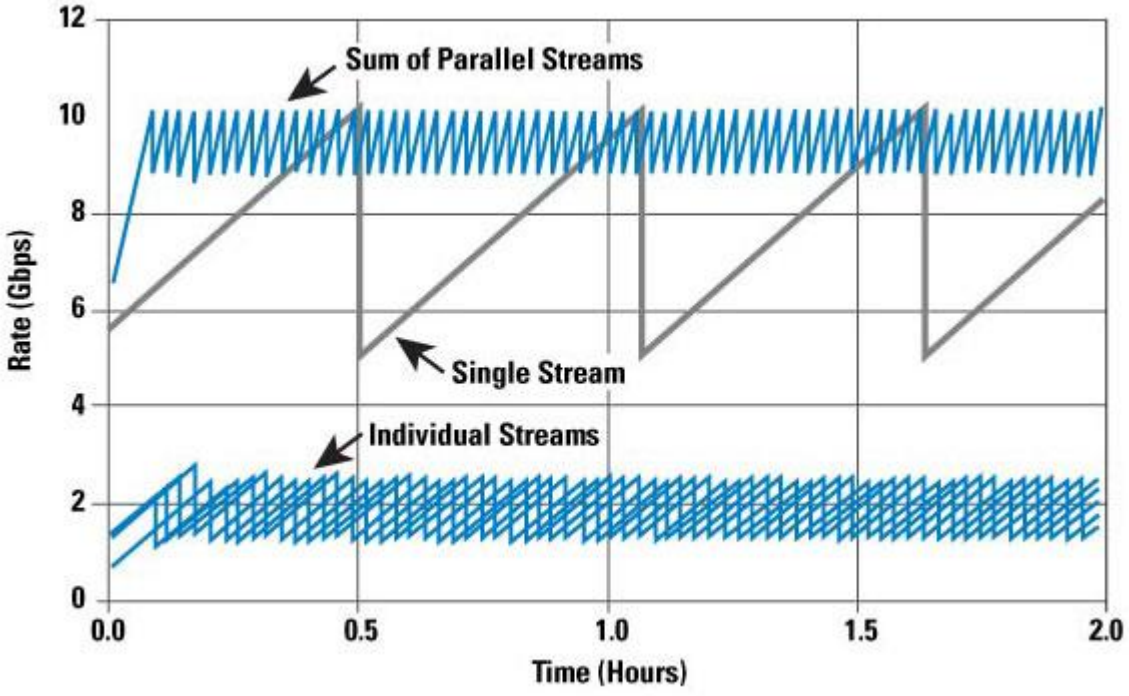
After the 25th transmission round, segment loss is detected by timeout.

- Plot the graph from transmission rounds 26 to 30.
- What is the initial value of ssthresh at the first transmission round?
- What is the value of ssthresh at the 23rd transmission round?
- What is the value of ssthresh at the 29th transmission round?
- During what transmission round is the 75th segment sent?
- Assuming packet loss is detected after the 30th round, by receipt of a triple duplicate ack, what will be the values of the congestion window size and of ssthresh?
- Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 20th round. What are the ssthresh and congestion window size at the 23rd round?

Please submit your plot and answers on eDimension. An enlarged copy of the starter graph from Activity 6 is available on eDimension.

TCP Congestion avoidance mode

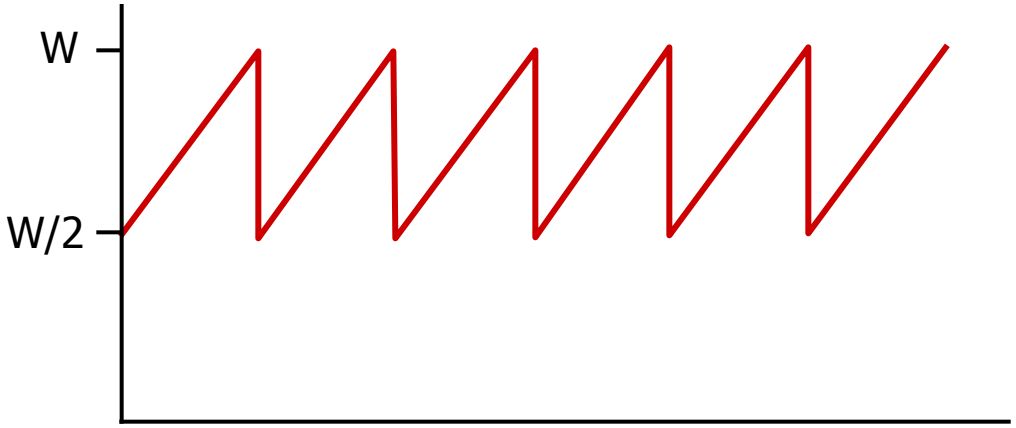
- Increases rate additively (+1 per RTT time) until loss occurs
- again
- This will result in a saw-tooth pattern for the rate



TCP throughput

- avg. TCP thruput as function of window size, RTT?
  - ignore slow start, assume always data to send
- W: window size (measured in bytes) where loss occurs
  - avg. window size (# in-flight bytes) is ¾ W
  - avg. thruput is 3/4W per RTT

$$\text{avg TCP thruput} = \frac{3}{4}\frac{W}{RTT} \text{ bytes/sec}$$



TCP and Fairness

- With multiple parallel TCP streams, every stream gets roughly the same share of available bandwidth
- Every stream is increasing transmission window with roughly same speed (+1/RTT)
- Every stream backs off by cutting transmission window in half
- Hosts can get a somewhat unfair advantage by creating *n* parallel TCP connections
  - They will get ~ *n* times the bandwidth they would get otherwise

Example: parallel TCP connections by your browser to a website

ECN: Network assisted Congestion Control

- ECN Bits set at IP Datagram Header at congested router
- ECN Echo bit set in rcvr to sndr TCP ACK segment
- Sender reduces cwnd

Recent Variants of TCP/UDP

- DDCP – Datagram Congestion Control Protocol
- QUIC – Quick UDP Internet Connections
- DCTCP – Data Center TCP
- TFRC – TCP Friendly Rate Control
- … more to come

Conclusion

- We continued our design challenge
  - We discussed error correction vs. ACKs
  - Pipelined ACKs
  - Timeouts and window size
- We introduced TCP
  - Header fields
  - ACKs and mechanisms
  - Flow control
  - Congestion and rate adaption