CSCI 466: Networks

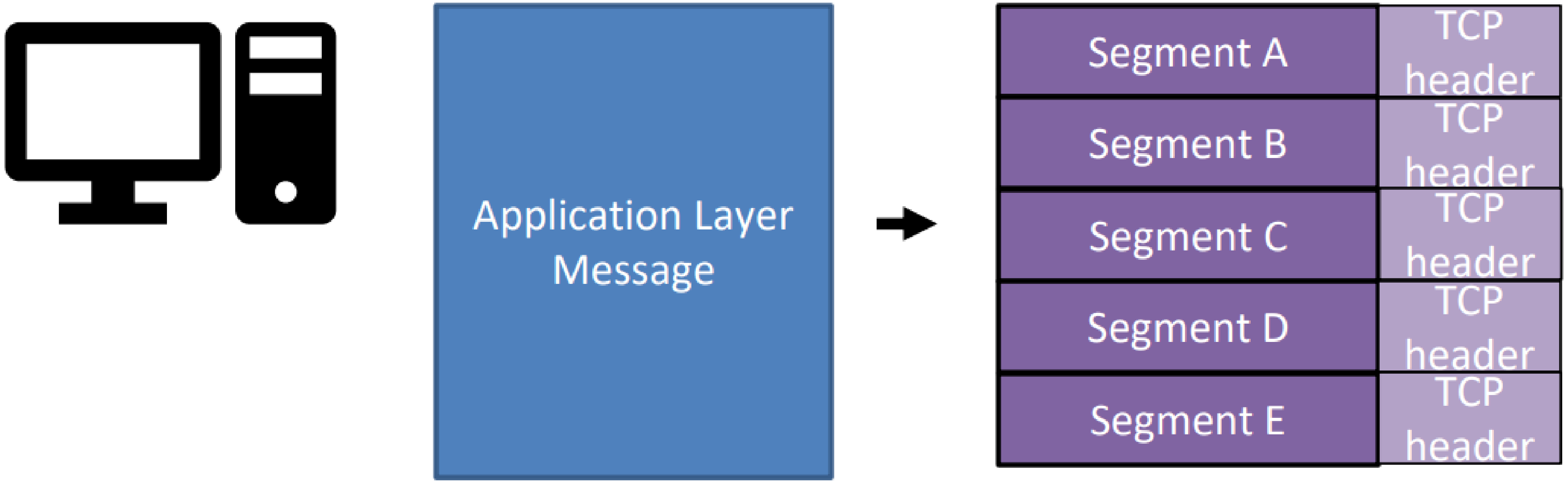
TCP Flow Control, Timeout, Congestion Control

Reese Pearsall

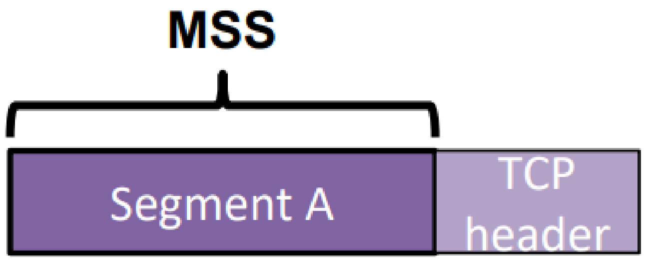
Fall 2023

*All images are stolen from the internet

TCP Flow Control

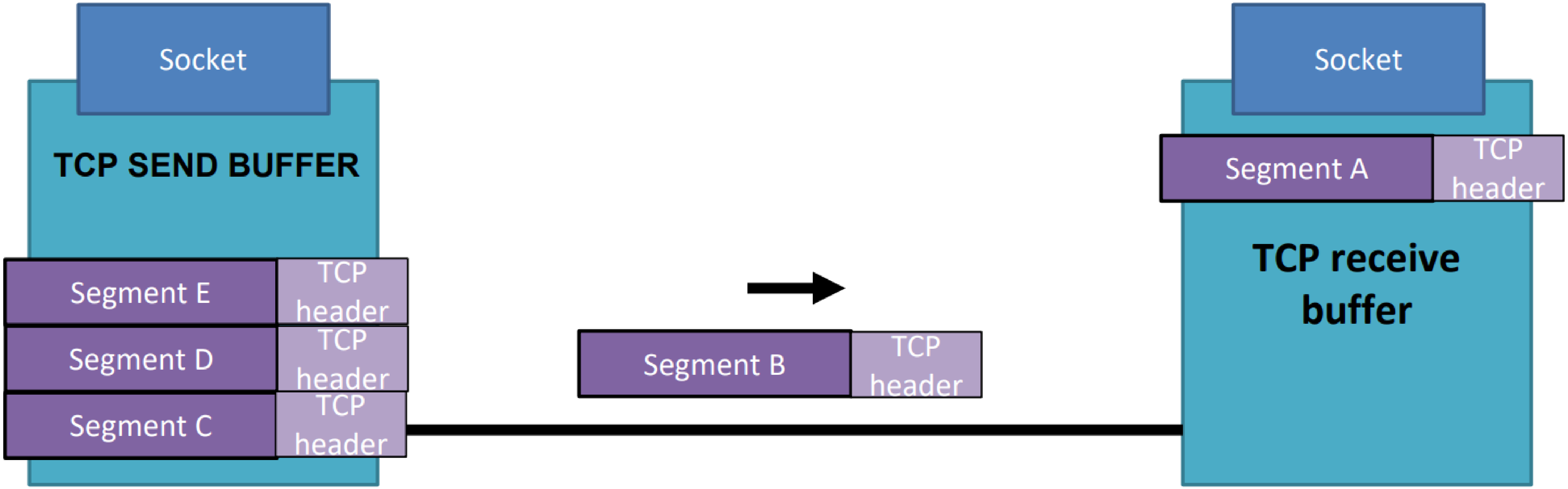


Application layer messages are split into smaller chunks called **segments**



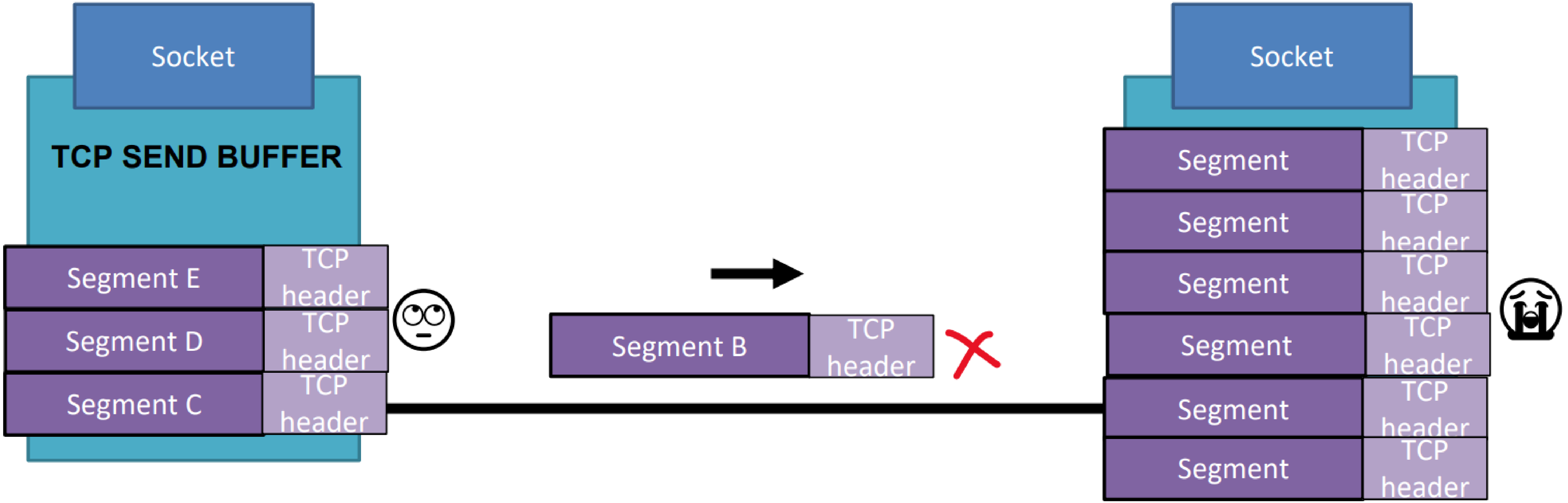
The size of these segments is determined by the **maximum segment size (MSS)**

TCP Flow Control



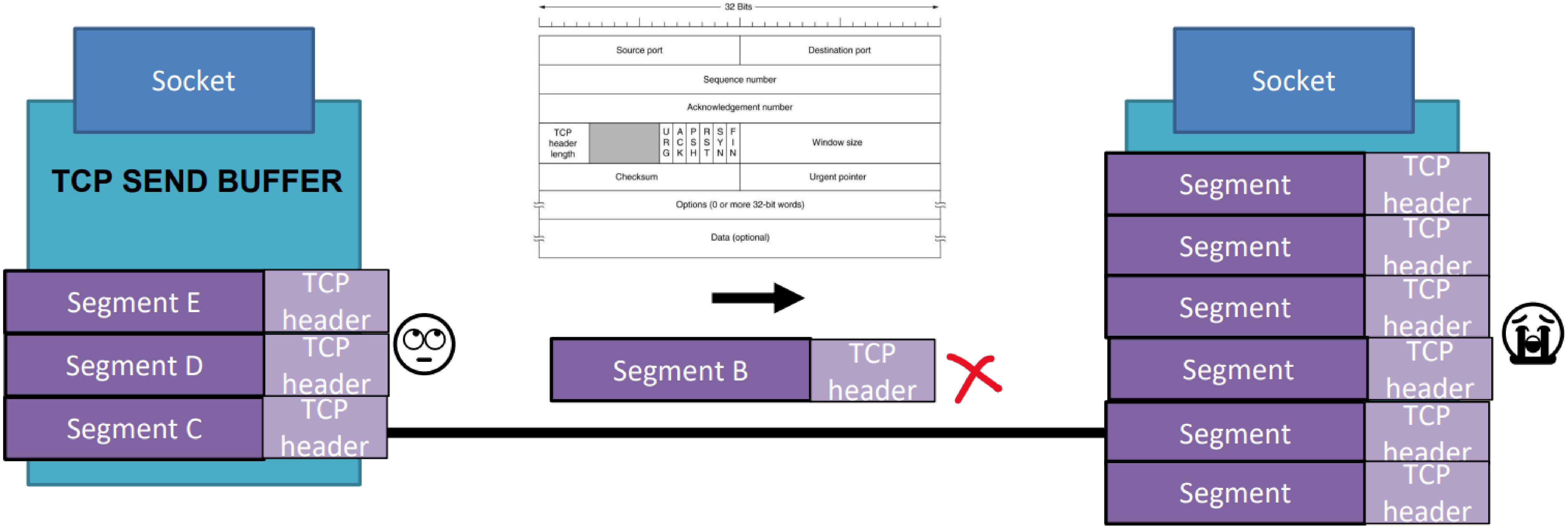
Applications read streams of data from a **TCP buffer**

TCP Flow Control



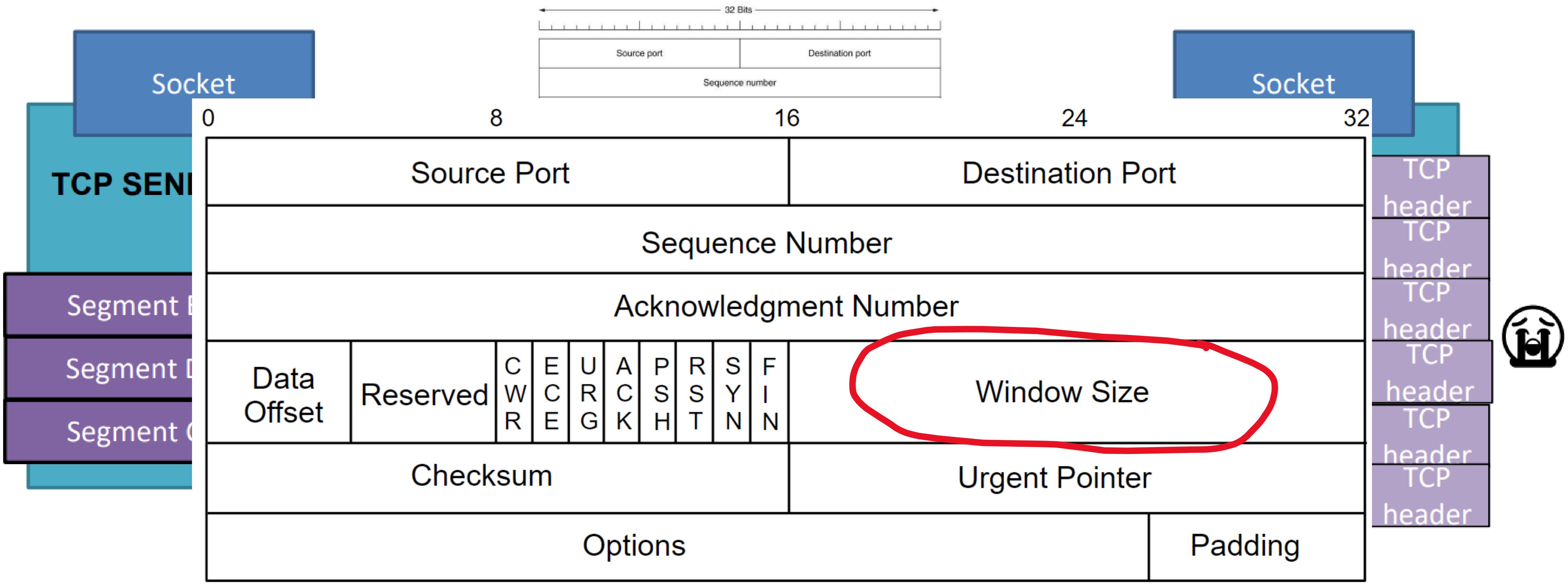
How could we prevent something like this from happening?

TCP Flow Control



We could send back to the sender how much available space we have in our buffer!

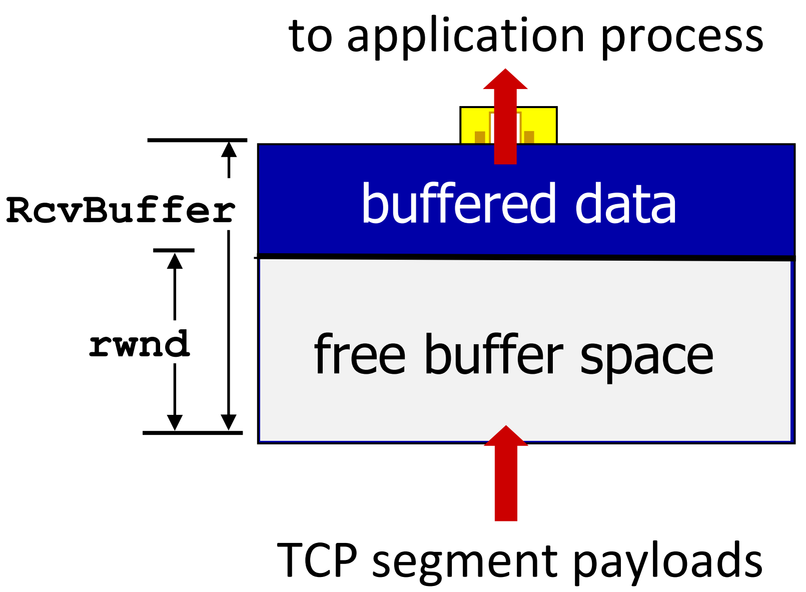
TCP Flow Control



sender how much available space we have in our buffer!

TCP Flow Control

- TCP receiver "advertises" free buffer space in **`rwnd`** field in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems auto-adjust **`RcvBuffer`**
- sender limits amount of unACKed ("in-flight") data to received **`rwnd`**
- guarantees receive buffer will not overflow



TCP receiver-side buffering

TCP Flow Control

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/flow-control/index.html

TCP Timer

What is a good way to determine when to timeout? (aka the length of timer)

1. Too short: premature timeout, unnecessary retransmissions
2. Too long: slow reaction to segment loss

The TCP timeout value should around the same time it take to ….

TCP Timer

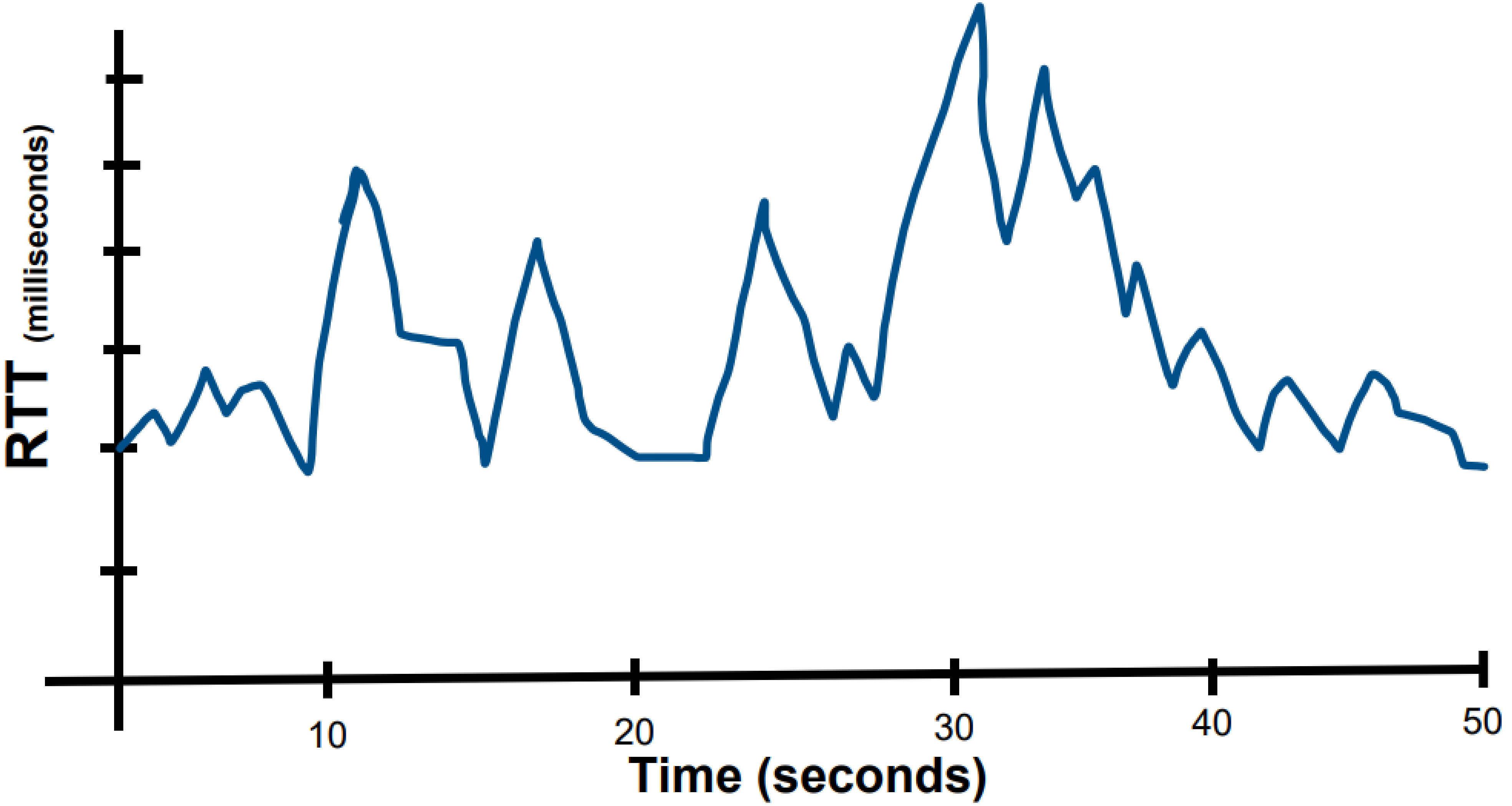
What is a good way to determine when to timeout? (aka the length of timer)

1. Too short: premature timeout, unnecessary retransmissions
2. Too long: slow reaction to segment loss

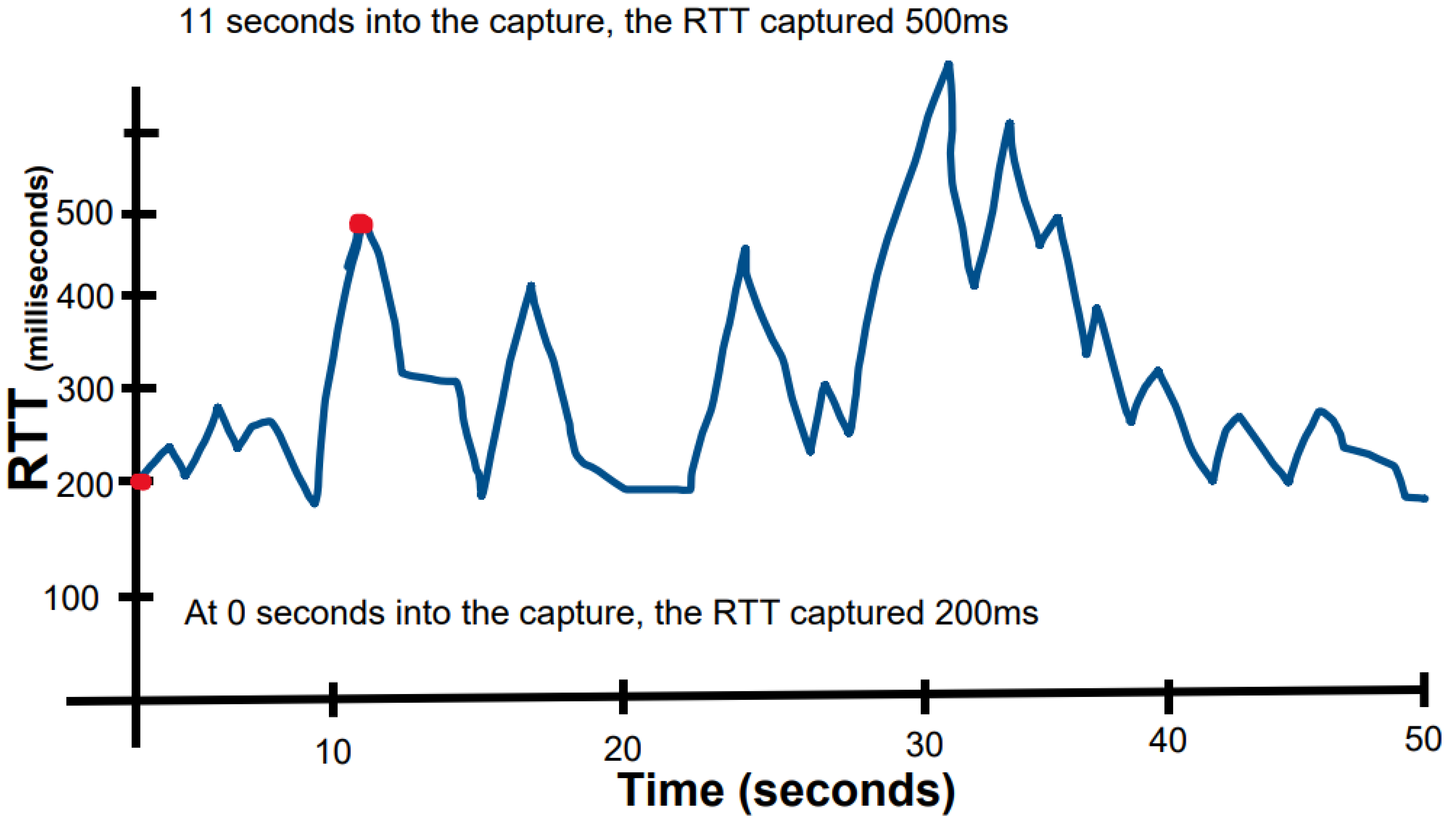
The TCP timeout value should around the same time it take to receive an acknowledgement on a sent packet (on average)

Let’s consider setting it to be a dynamic value!

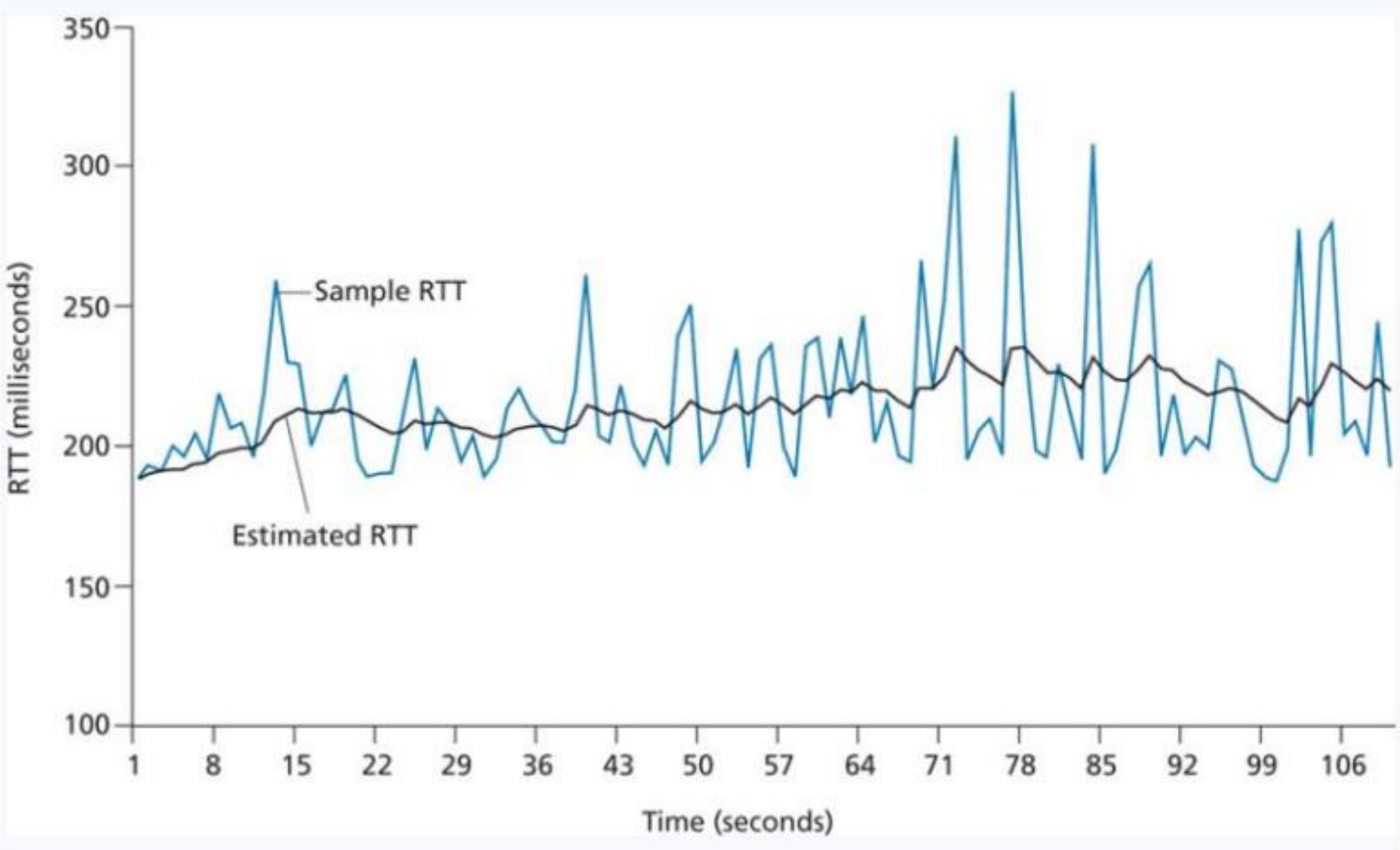
TCP Timeout



TCP Timeout


11 seconds into the capture, the RTT captured 500ms
RTT (milliseconds)
500
400
300
200
100
At 0 seconds into the capture, the RTT captured 200ms
10
20
30
40
50
Time (seconds)

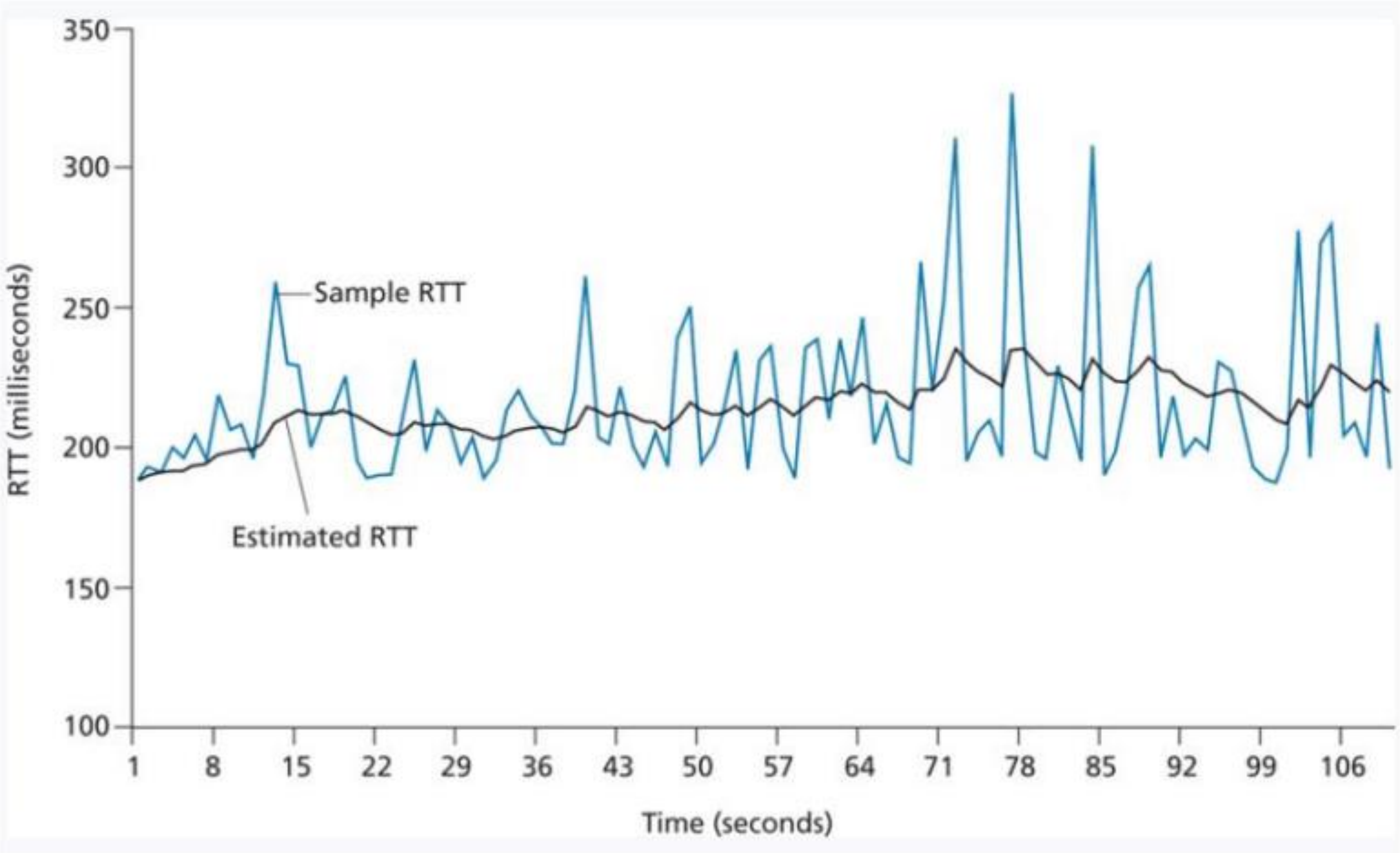
TCP Timeout



$$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}$$

a=0.125

TCP Timeout



In addition, we also want some kind of safety margin

```
DevRTT = (1-β)*DevRTT +
             β*|SampleRTT-EstimatedRTT|
(typically, β = 0.25)
```

TimeoutInterval =

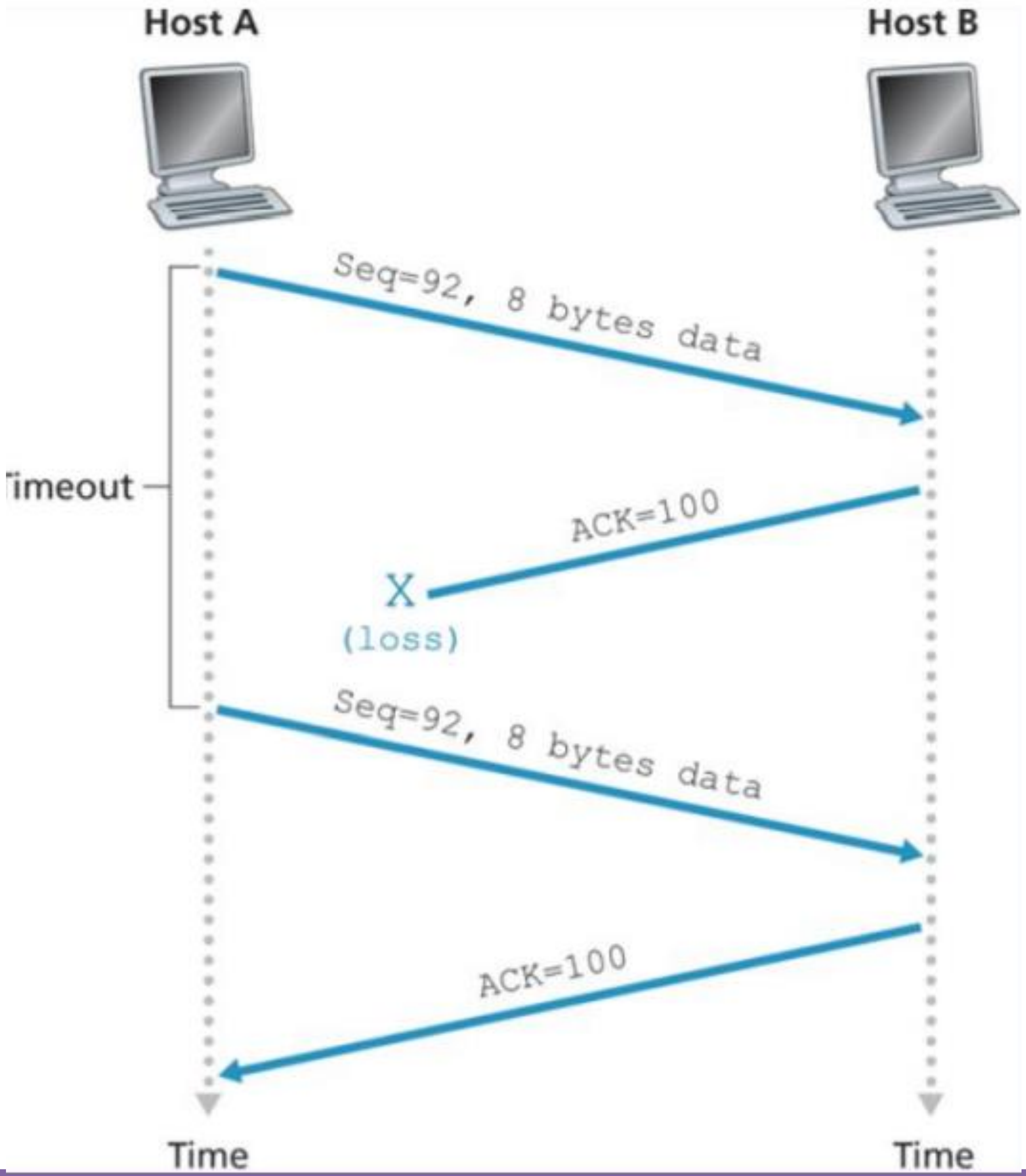
EstimatedRTT + 4*DevRTT

(safety margin)

$$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}$$

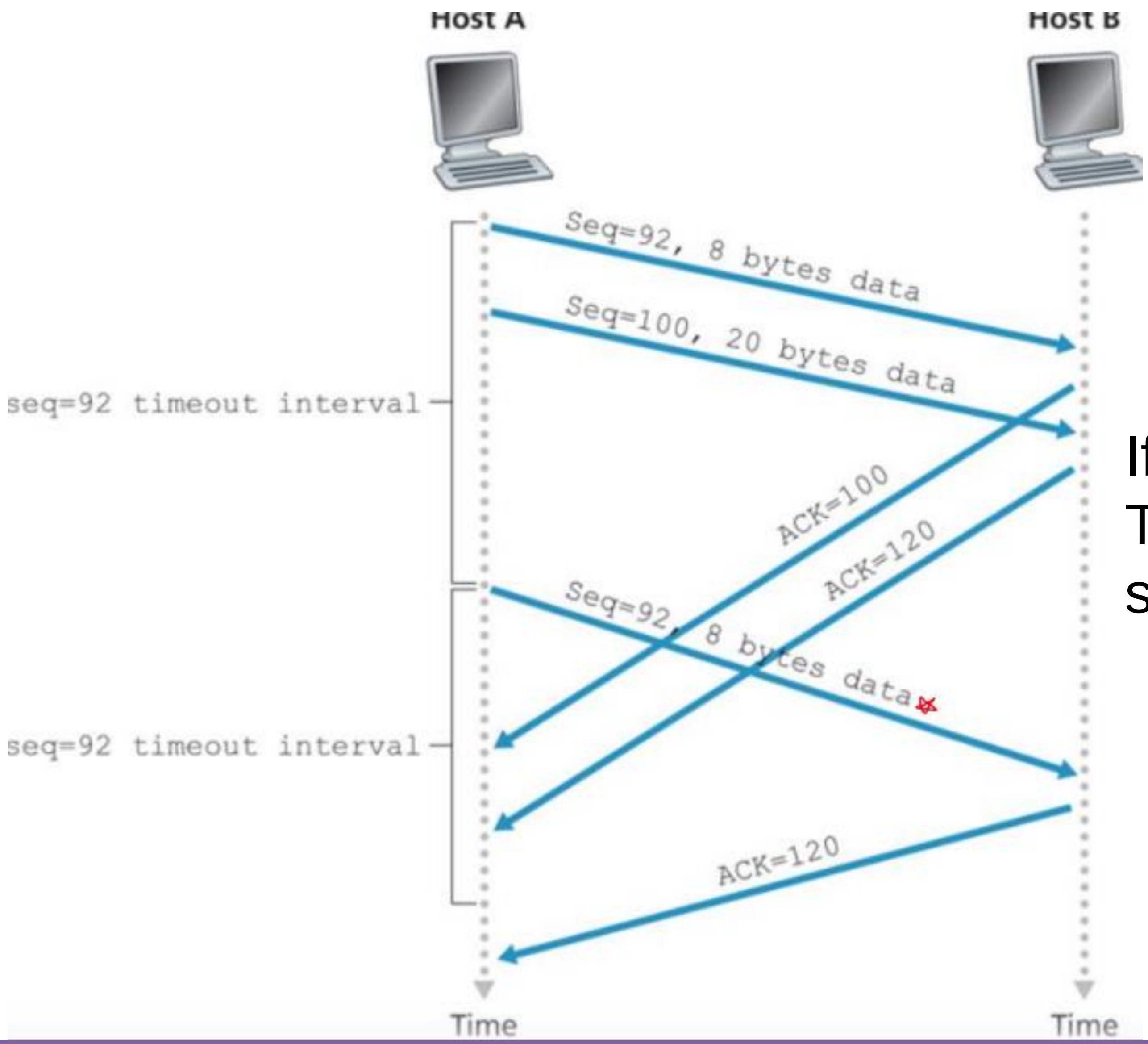
a=0.125

TCP Timeout



TCP retransmits on ACK loss

TCP Timeout



If multiple ACKS are lost/late, TCP only resends the first segment in the sequence

TCP Timeout

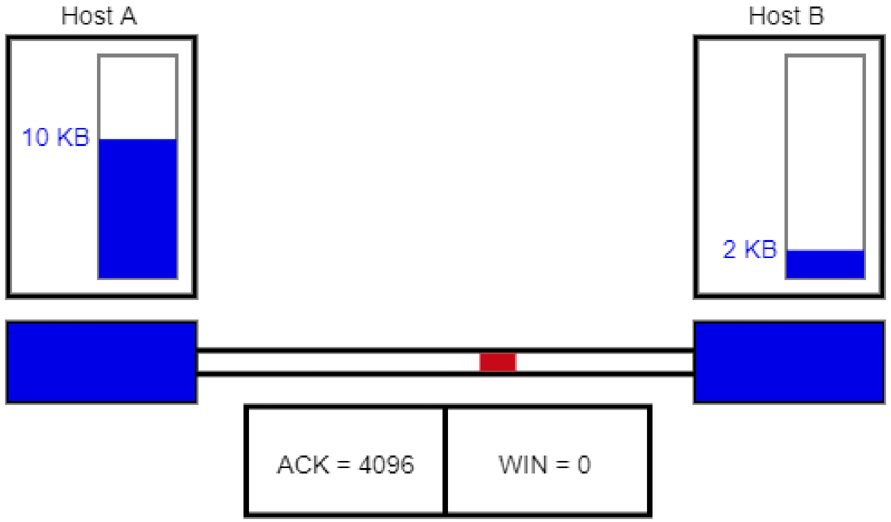
Event	TCP Receiver Action
Arrival of in-order segment with expected sequence number. All data up to expected sequence number already acknowledged.	Delayed ACK. Wait up to 500 msec for arrival of another in-order segment. If next in-order segment does not arrive in this interval, send an ACK.
Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK transmission.	Immediately send single cumulative ACK, ACKing both in-order segments.
Arrival of out-of-order segment with higher-than-expected sequence number. Gap detected.	Immediately send duplicate ACK, indicating sequence number of next expected byte (which is the lower end of the gap).
Arrival of segment that partially or completely fills in gap in received data.	Immediately send ACK, provided that segment starts at the lower end of gap.

Specifics about when/how/why to send ACKs are described in TCP Congestion Control's **RFC (request for comments)**

9293

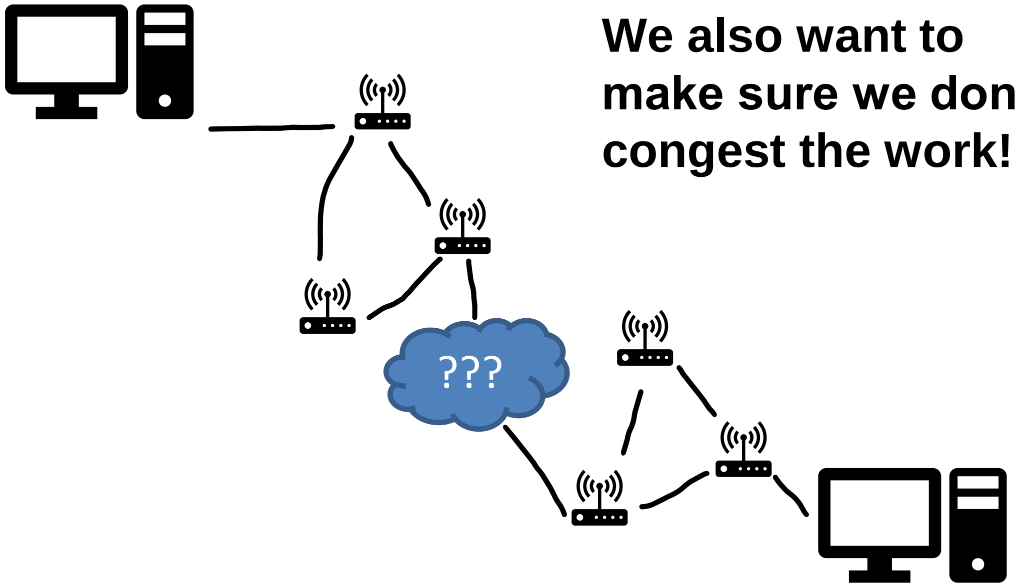
5681

TCP Congestion Control



TCP sends back amount of available buffer space in the receiver

This helps make sure we don't overwhelm the receiver

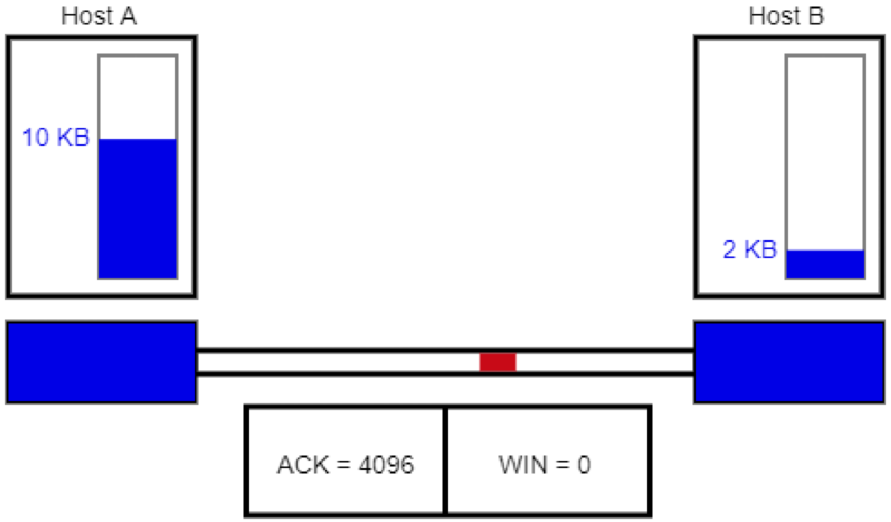


We also want to make sure we don't congest the work!

Issues:

- If the network is congested, we want to slow down our sending rate
- If the network is not congested, we should try to send more stuff

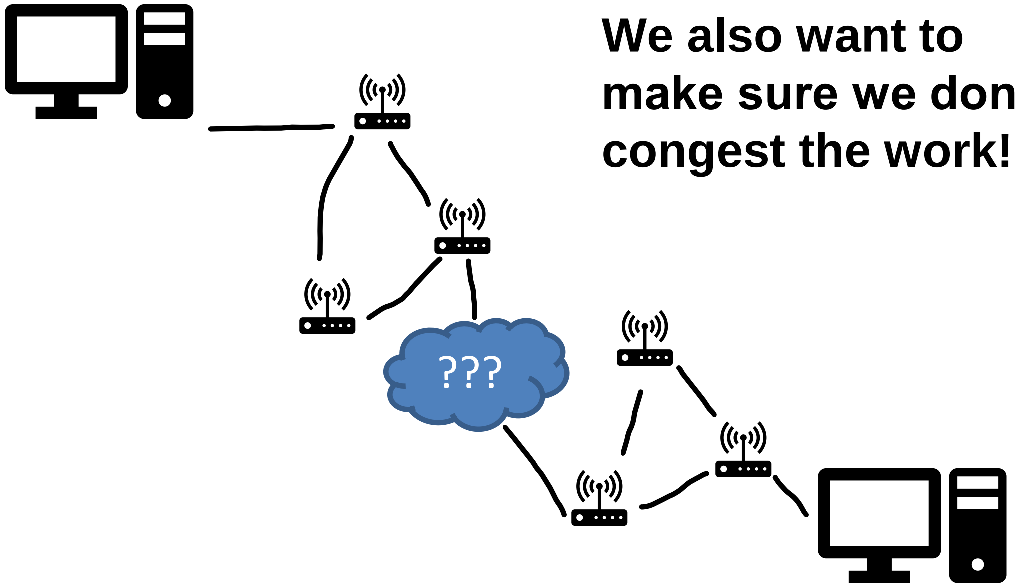
TCP Congestion Control



TCP sends back amount of available buffer space in the receiver

This helps make sure we don't overwhelm the receiver

We also want to make sure we don't congest the work!

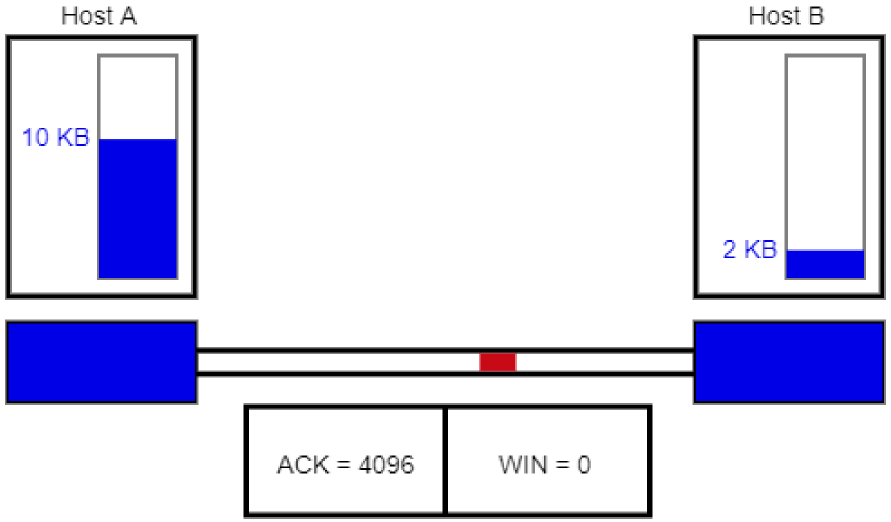


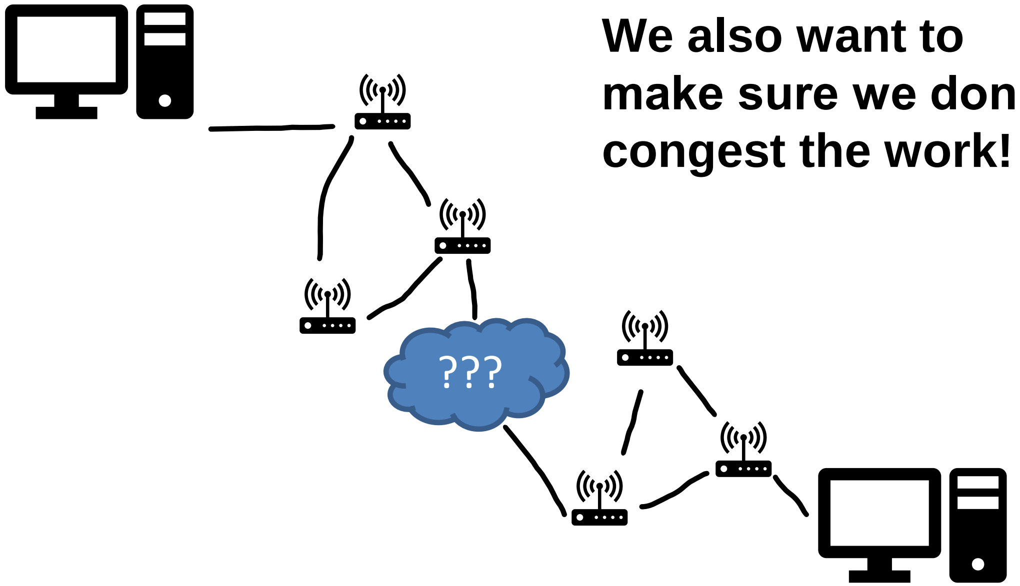
Issues:

- If the network is congested, we want to slow down our sending rate
- If the network is not congested, we should try to send more stuff

From the sender perspective, how could we measure how congested the network is?

TCP Congestion Control





We also want to make sure we don't congest the work!

TCP sends back amount of available buffer space in the receiver

This helps make sure we don't overwhelm the receiver

Issues:

- If the network is congested, we want to slow down our sending rate
- If the network is not congested, we should try to send more stuff

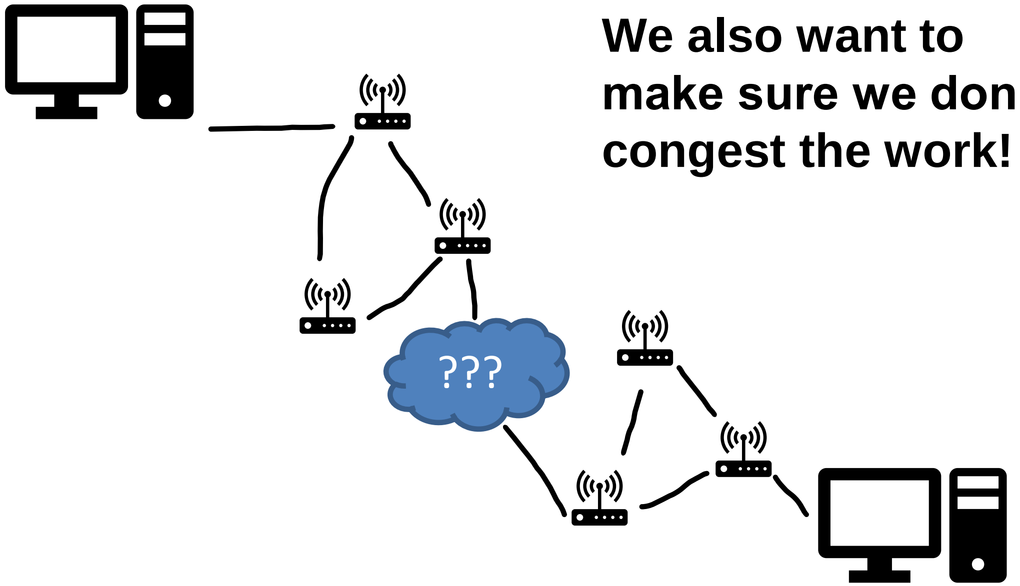
Some ways we could measure how congested the network is

-See how many dropped packets we are getting

-Amount of duplicate ACKs received

-Amount of UnAcked packets

TCP Congestion Control



We also want to make sure we don't congest the work!

TCP sender also has a **congestion window (cwnd)**, which controls the amount of unAck'd that can be sent out

TCP is **self-clocking**

(It uses acknowledgements to trigger, or clock, its increase in congestion window size)

The amount of unacknowledged data at a sender may not exceed the *minimum* of the congestion window and receiving window

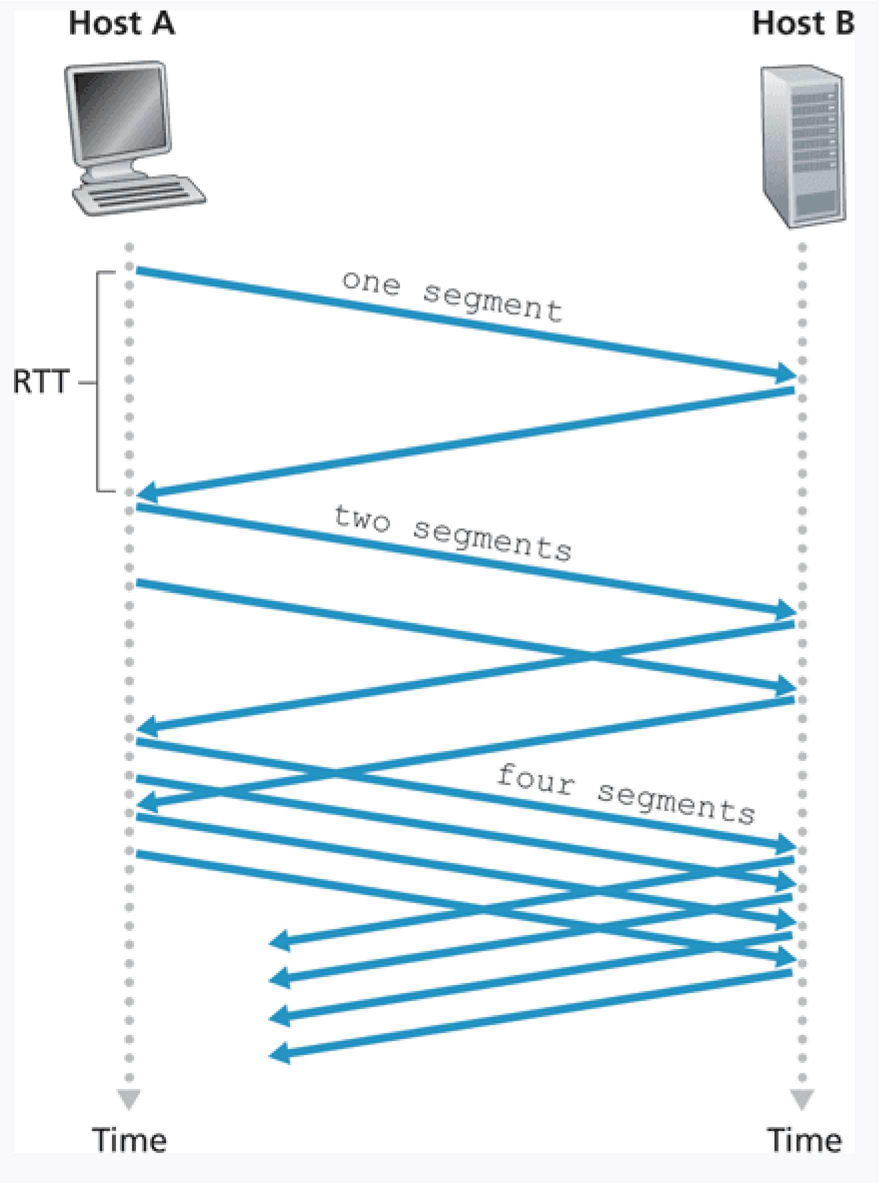
```
LastByteSent – LastByteAcked ≤ min{cwnd, rwnd}
```

TCP Congestion Control Algorithm

TCP Algorithm to prevent network congestion

- **Slow Start**
- Congestion Avoidance
- Fast recovery

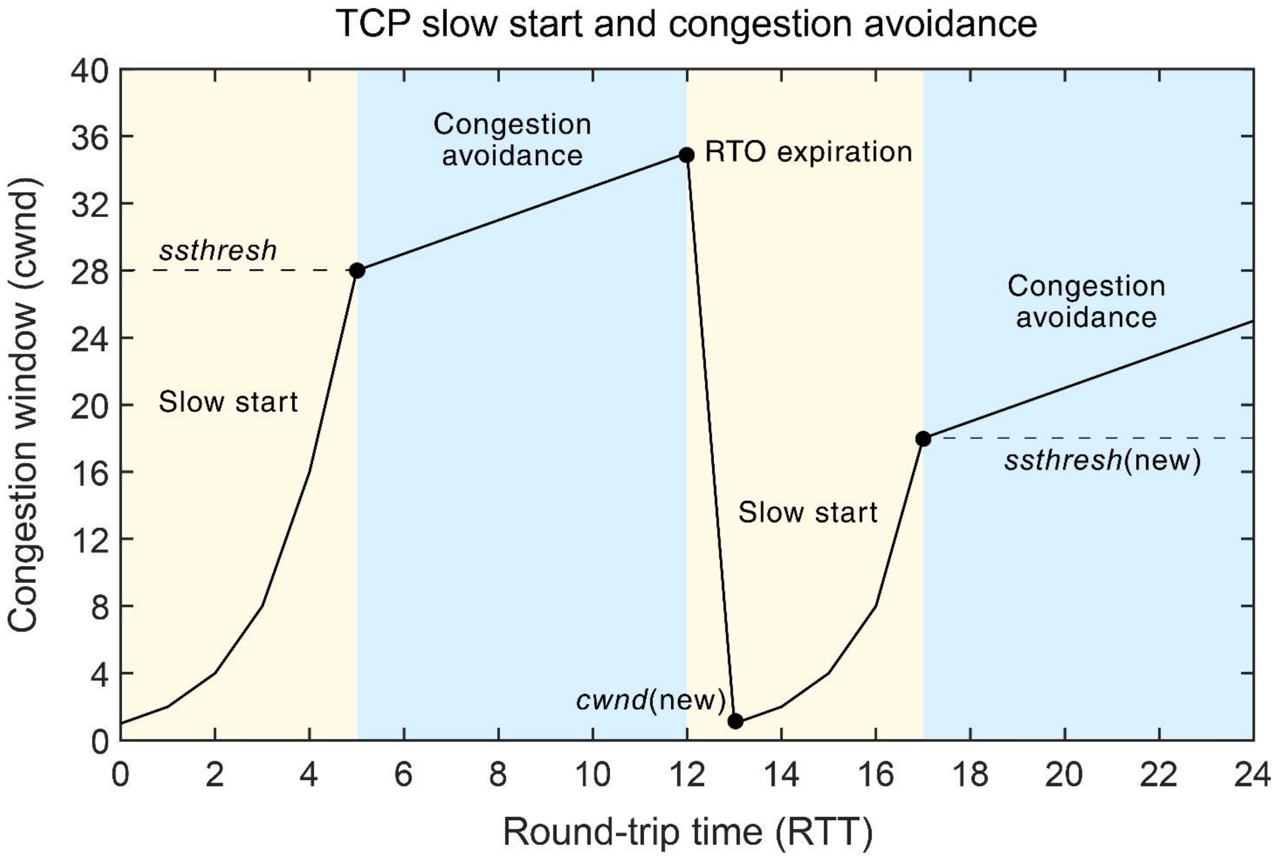
Start sending slow, but exponentially grows up to a *threshold*



TCP Congestion Control Algorithm

TCP Algorithm to prevent network congestion

- Slow Start
- **Congestion Avoidance**
- Fast recovery



Linearly increase congestion window for each ACK received

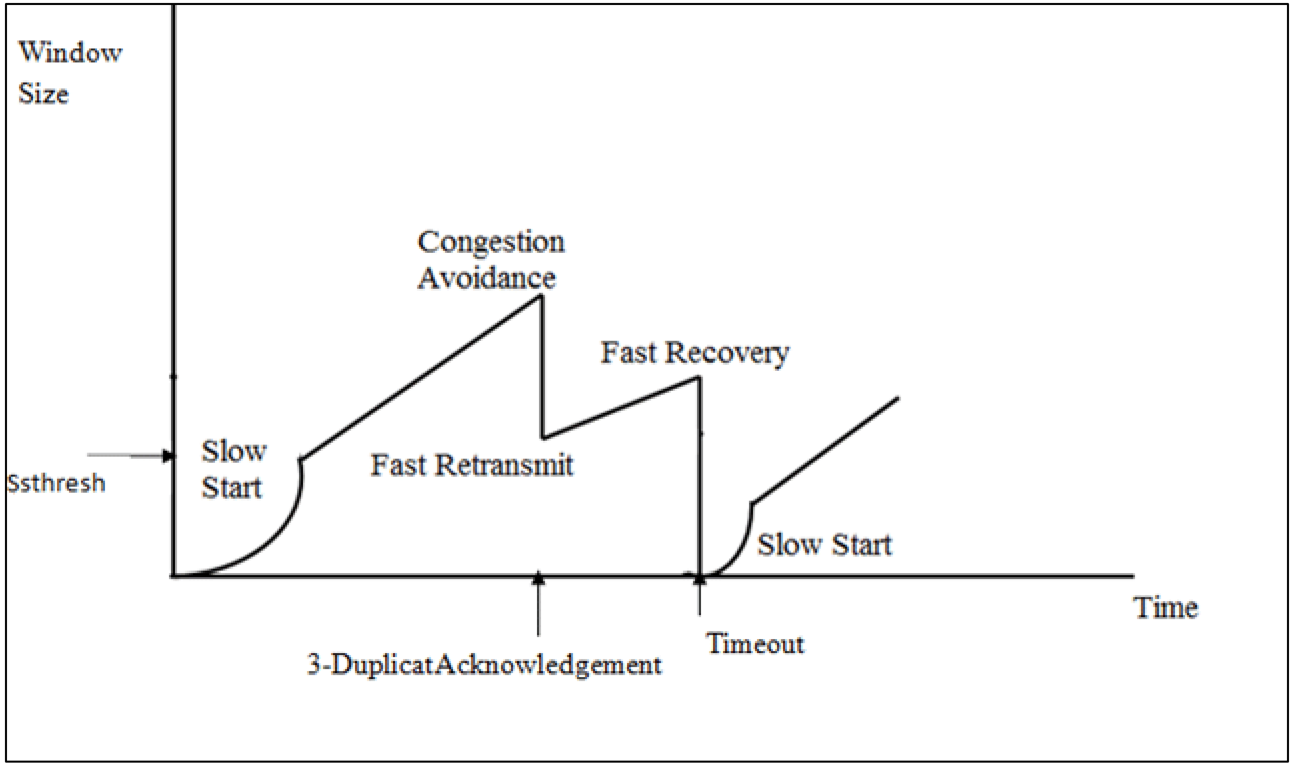
When a loss event occurs, significantly decrease congestion window and slow down transmission rate, and enter **fast recovery**

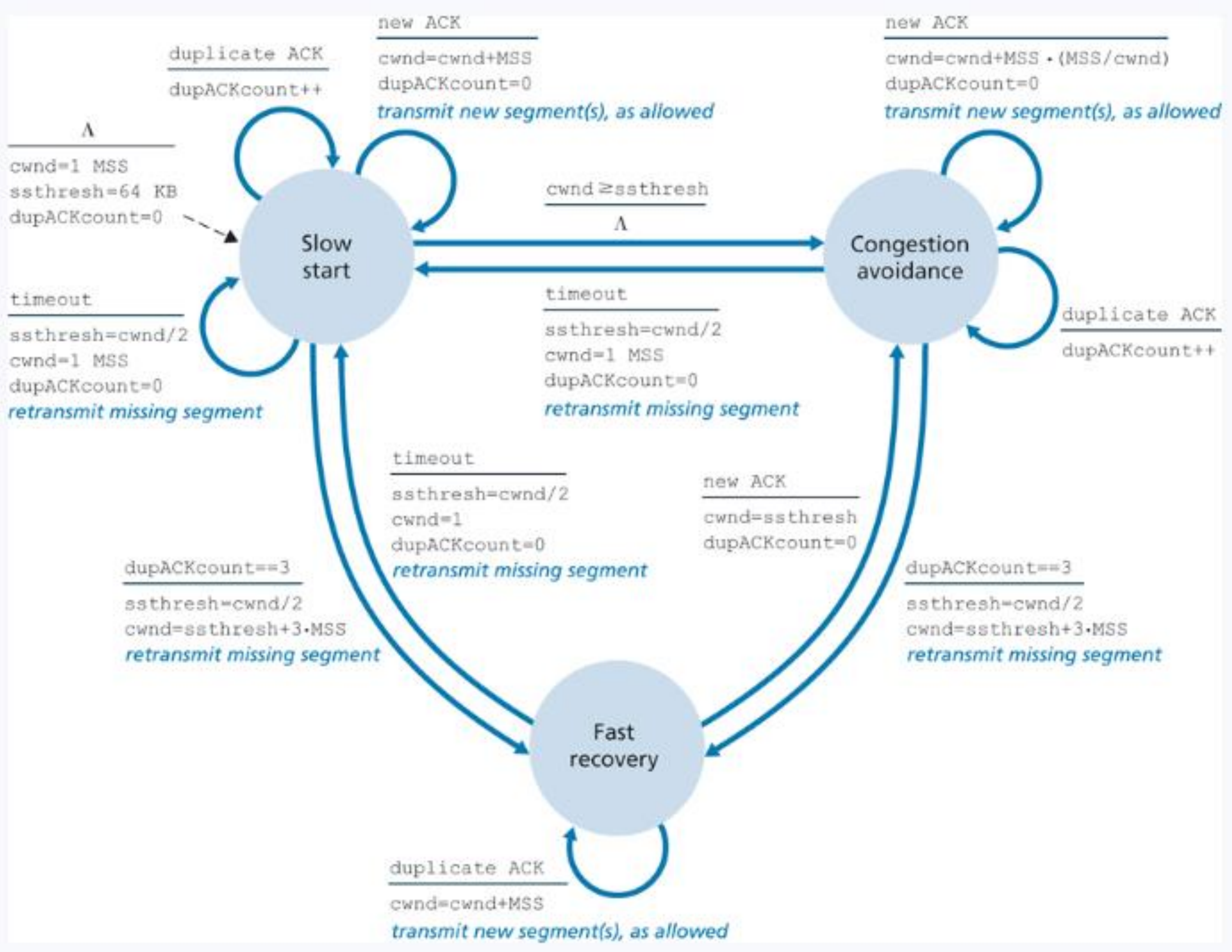
TCP Congestion Control Algorithm

TCP Algorithm to prevent network congestion

- Slow Start
- Congestion Avoidance
- **Fast recovery**

  Upon knowledge of packet loss, throttle the TCP connection and start off slow again



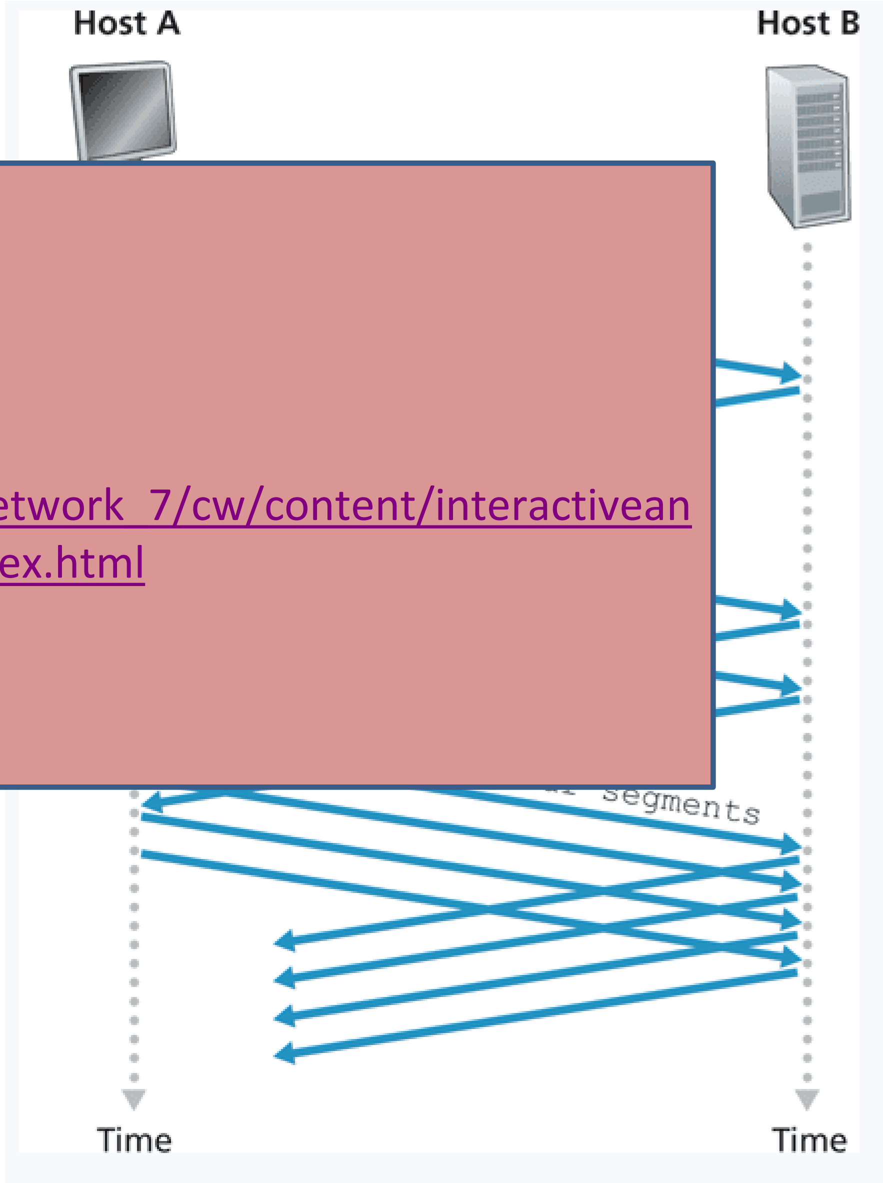

Slow start
Congestion avoidance
Fast recovery
Λ
cwnd=1 MSS
ssthresh=64 KB
dupACKcount=0
duplicate ACK
dupACKcount++
new ACK
cwnd=cwnd+MSS
dupACKcount=0
transmit new segment(s), as allowed
new ACK
cwnd=cwnd+MSS • (MSS/cwnd)
dupACKcount=0
transmit new segment(s), as allowed
cwnd ≥ ssthresh
Λ
timeout
ssthresh=cwnd/2
cwnd=1 MSS
dupACKcount=0
retransmit missing segment
timeout
ssthresh=cwnd/2
cwnd=1 MSS
dupACKcount=0
retransmit missing segment
duplicate ACK
dupACKcount++
timeout
ssthresh=cwnd/2
cwnd=1
dupACKcount=0
retransmit missing segment
new ACK
cwnd=ssthresh
dupACKcount=0
dupACKcount==3
ssthresh=cwnd/2
cwnd=ssthresh+3•MSS
retransmit missing segment
dupACKcount==3
ssthresh=cwnd/2
cwnd=ssthresh+3•MSS
retransmit missing segment
duplicate ACK
cwnd=cwnd+MSS
transmit new segment(s), as allowed

TCP Congestion Control Algorithm

TCP Alg

- Slow
- Con
- **Fast**

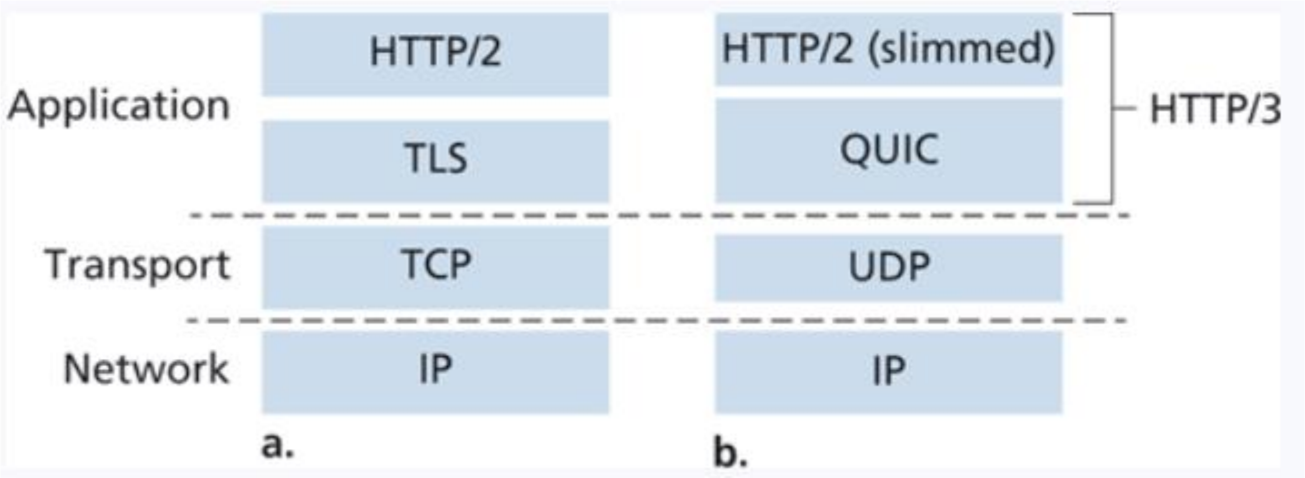
Upon knowledge of packet loss, throttle the TCP connection and start off slow again



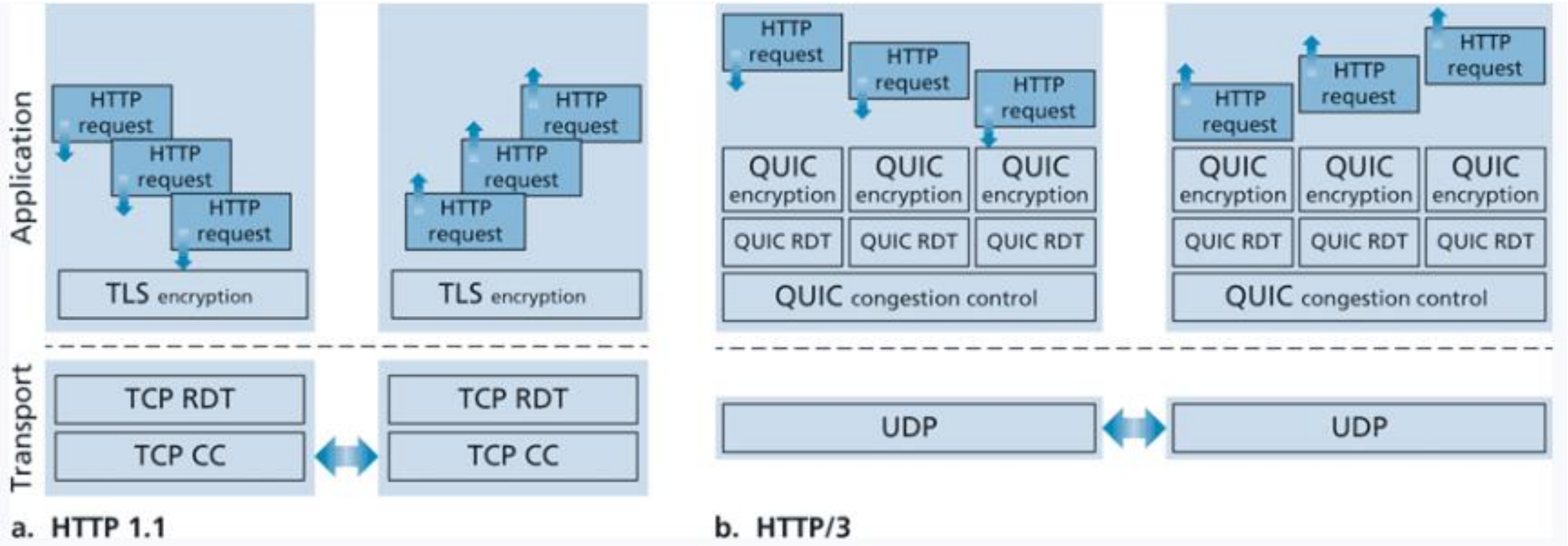
Animation time!

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/tcp-congestion/index.html

Current transport layer implementation



Transport layer protocols and congestion control is still a heavily researched area!



FIN

UDP
second packet sent
but the first one is missing
UDP
fifth packet sent
and where are the others?
UDP
last packet sent