IOWA

CS3640

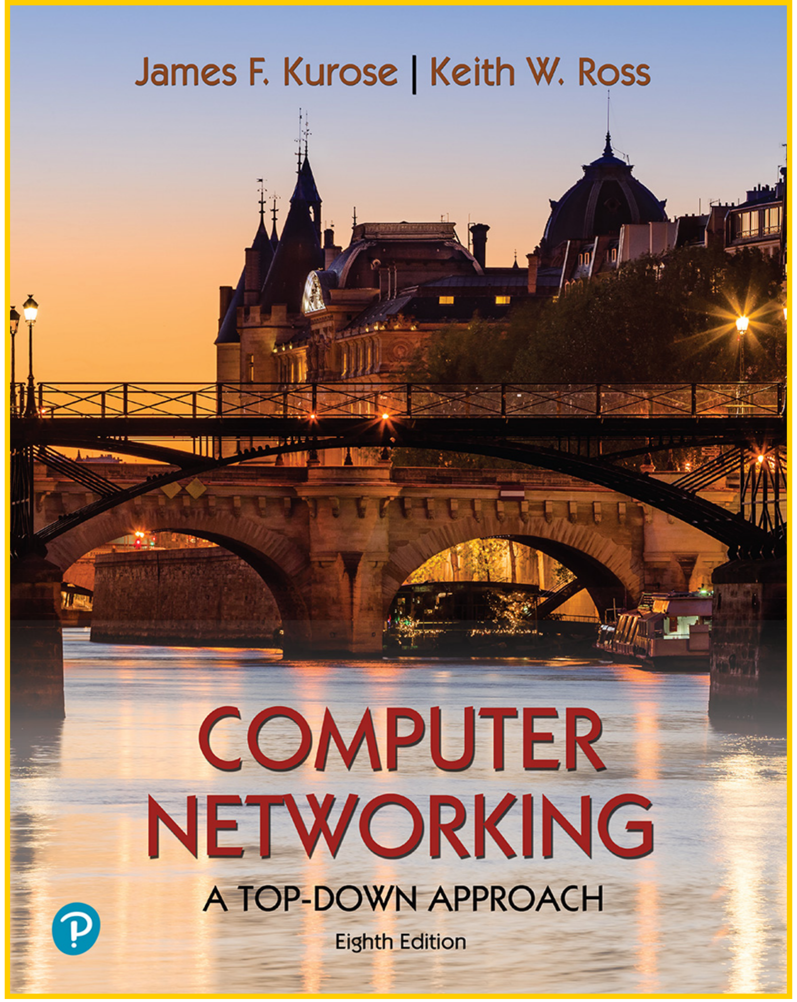
Transport Layer (4): TCP

Prof. Supreeth Shastri
Computer Science
The University of Iowa

Lecture goals

from principles to practice: design and operation of TCP

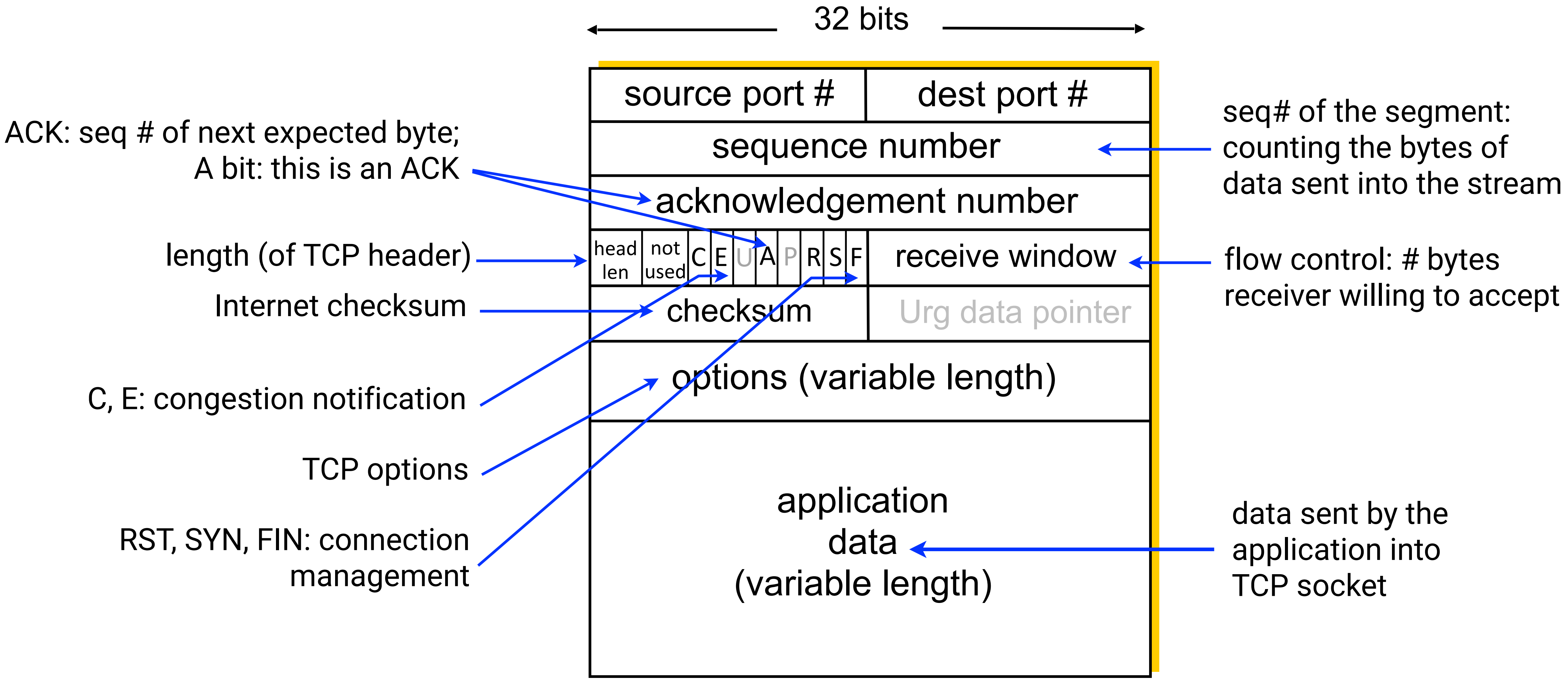
- *Protocol structure*
- *Connection management*
- *Reliable data transfer*
- *Flow and congestion control*



Chapters 3.5, 3.7

IOWA

Structure of the TCP segment

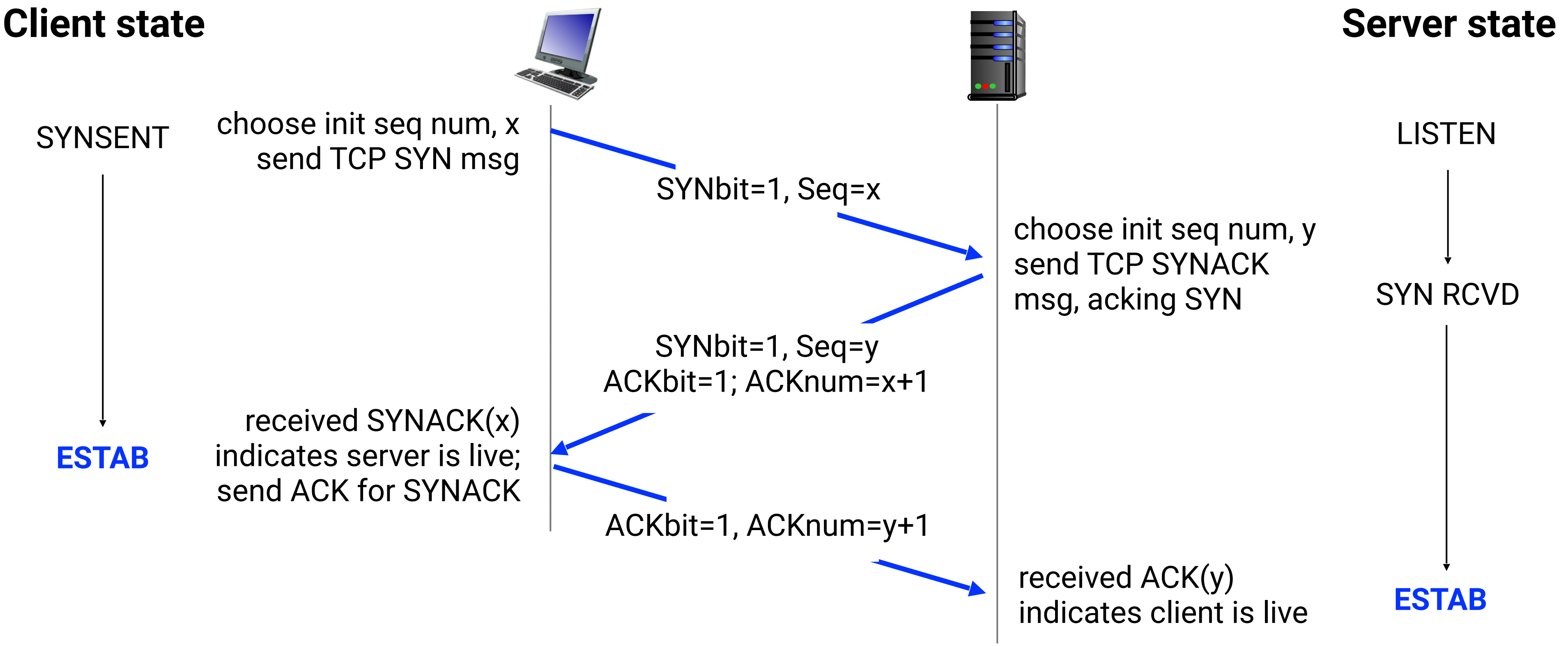


TCP Connection Management

TCP 3-way handshake

TCP is connection-oriented, thus needs a "handshake" before exchanging data

- Goal-1: sender and receiver determine that the other side is willing to establish connection
- Goal-2: sender and receiver agree on connection parameters (e.g., starting sequence #)

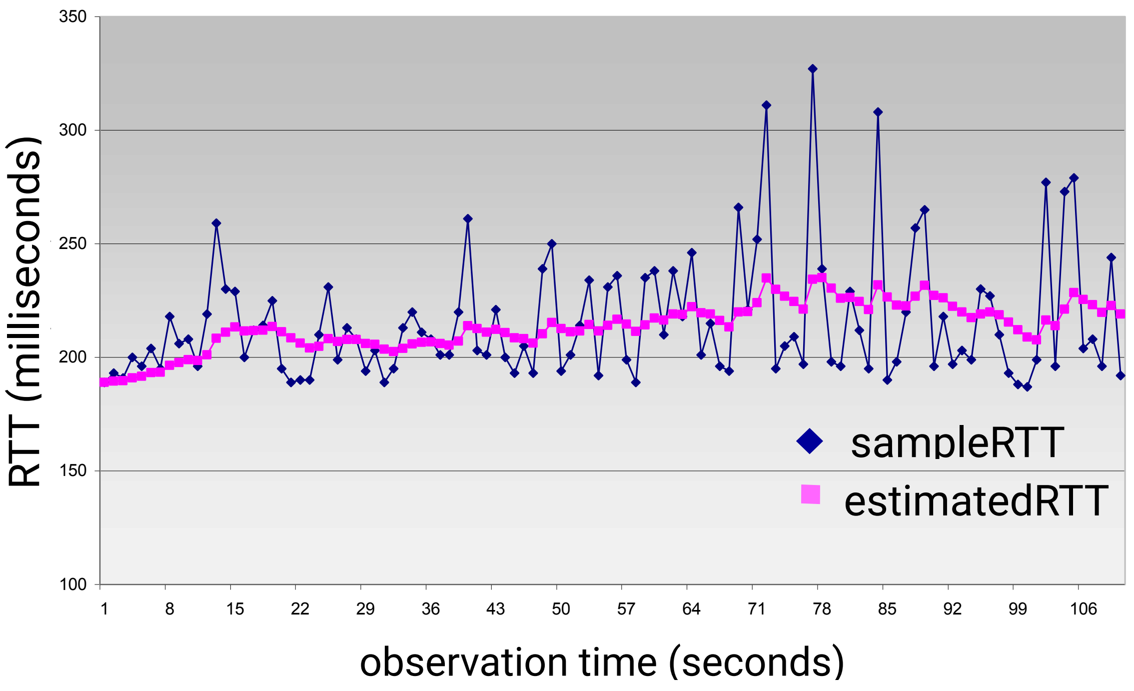


Round Trip Time (RTT) and TCP Timeout

- Key idea: measure time between segment transmission until its ACK receipt
- Such ***SampleRTT*** will vary over time, so we want estimated RTT to be “smoother”

$$\texttt{EstimatedRTT} = (1-\alpha)\texttt{*EstimatedRTT} + \alpha\texttt{*SampleRTT}$$

- exponential weighted moving average (EWMA)
- influence of past sample decreases exponentially fast
- typical value: α = 0.125



Round Trip Time (RTT) and TCP Timeout

- Underestimating timeout value ⇒ unnecessary retransmissions; Overestimating timeout value ⇒ slower loss recovery
- TCP computes timeout interval using **`EstimatedRTT`**
- Larger the variation in **`EstimatedRTT`**, larger the safety margin

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

safety margin

- **`DevRTT`**: EWMA of **`SampleRTT`** deviation from **`EstimatedRTT`**

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

(typically, $\beta = 0.25$)

TCP Reliable Transfer

TCP is a hybrid between GBN and SR protocols

	Go-Back-N	Selective Repeat
ACKs	**Cumulative** i.e., ACK(k) will ACK all packets up to and including #k	**Individual** i.e., ACK(k) just ACKs packet #k
Out of order packets	Receiver discards all out of order packets	Buffers out-of-order packet for later delivery
Buffer size	Sender buffer = N; receiver buffer = 1	Sender buffer = N; Receiver buffer = M
Sender timer	Set for only the oldest unacknowledged packet	Set for every transmitted packet

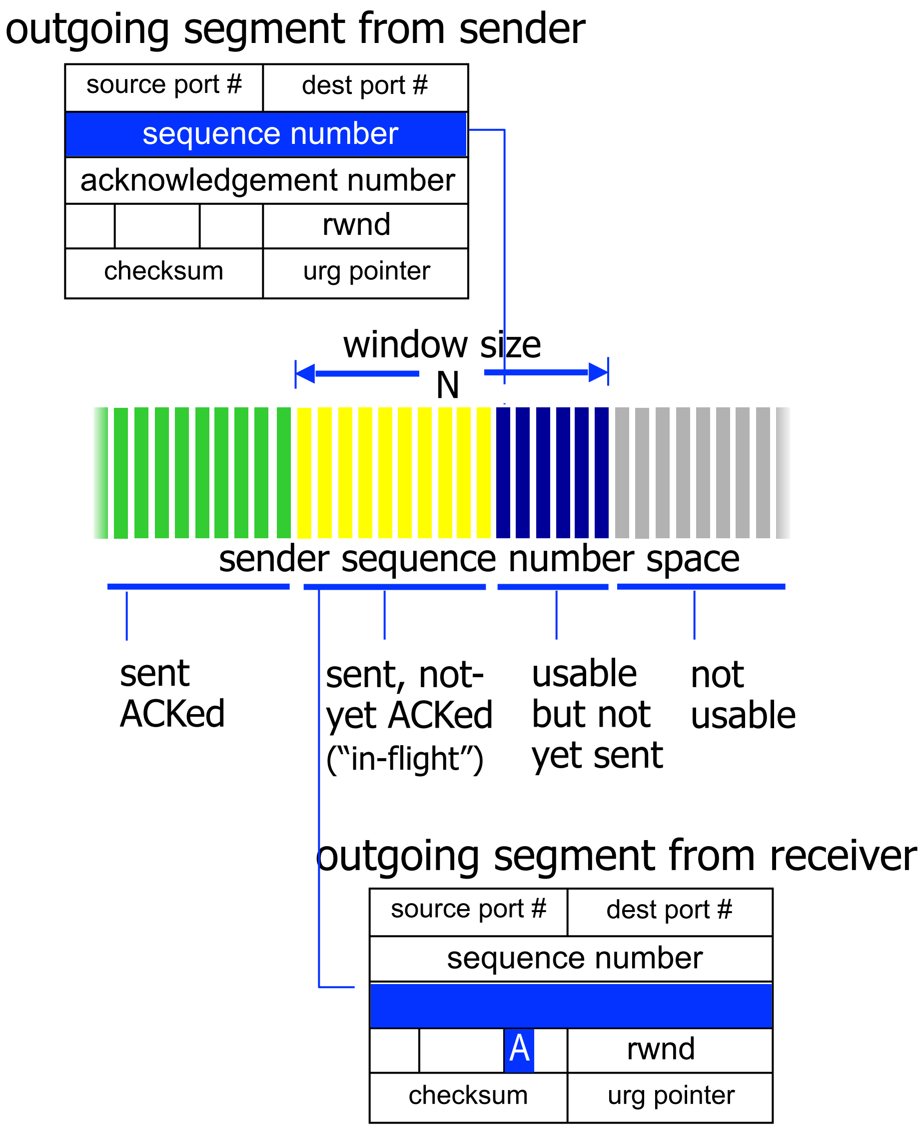
Understanding Sequence and ACK Numbers

Sequence numbers

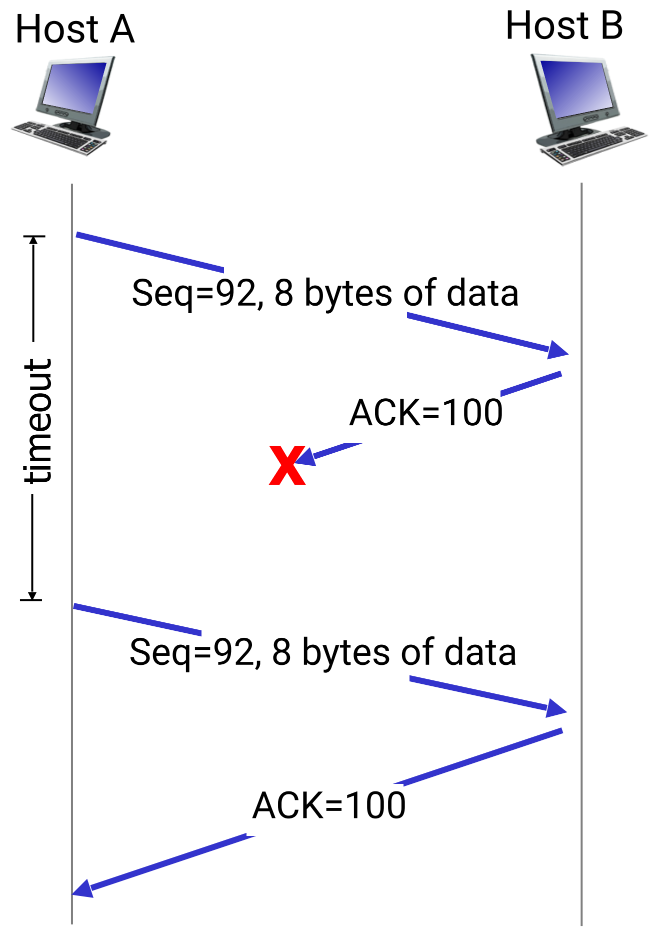
byte stream "number" of first byte in segment's data

Acknowledgement numbers

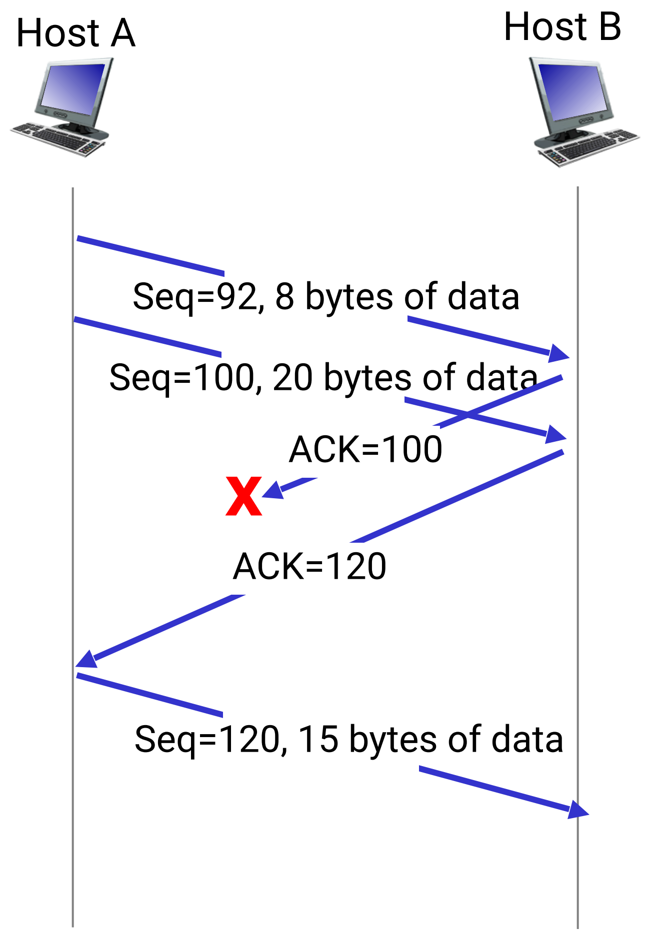
sequence# of next byte expected from the other side



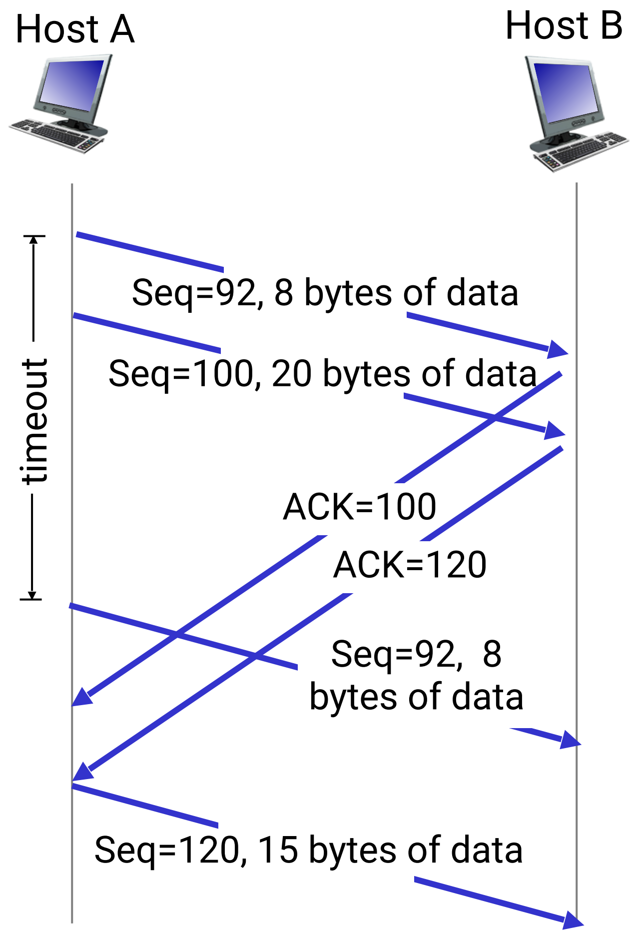
Example Retransmission Scenarios



lost ACK

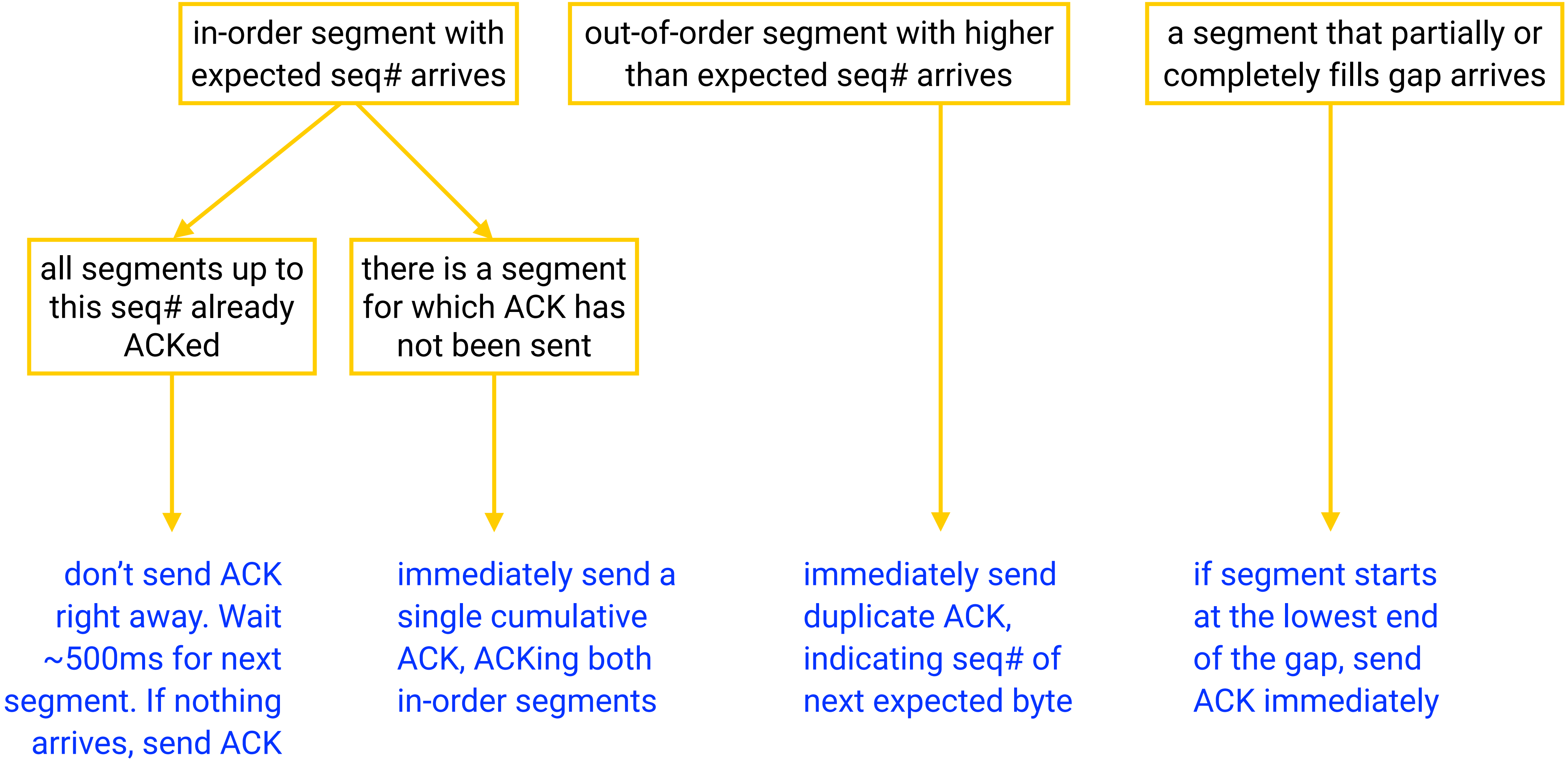


cumulative ACK covers for earlier lost ACK



premature timeout

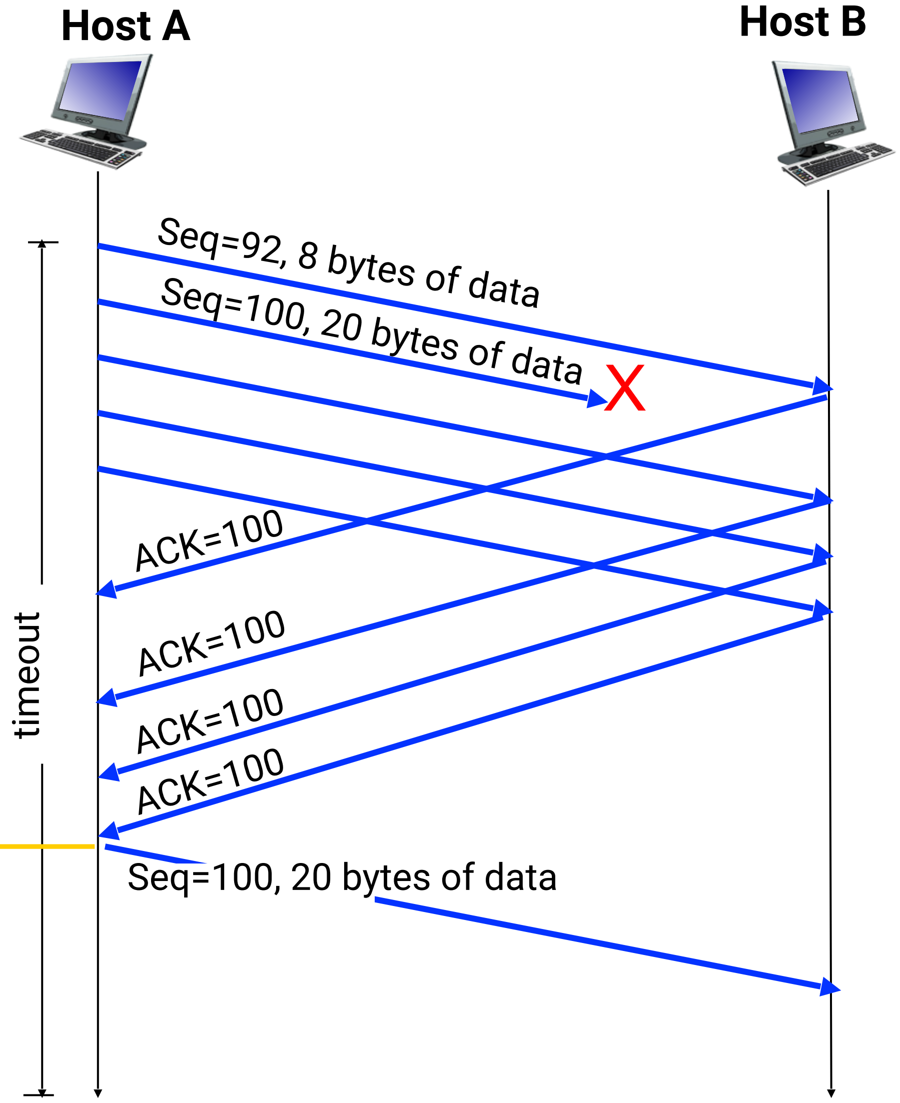
ACK Generation by TCP Receiver (RFC 5681)



TCP fast retransmit

if sender receives 3 ACKs for same data ("triple duplicate ACKs"), it is likely that unacknowledged segment is lost, so don't wait for timeout, instead resend that segment now

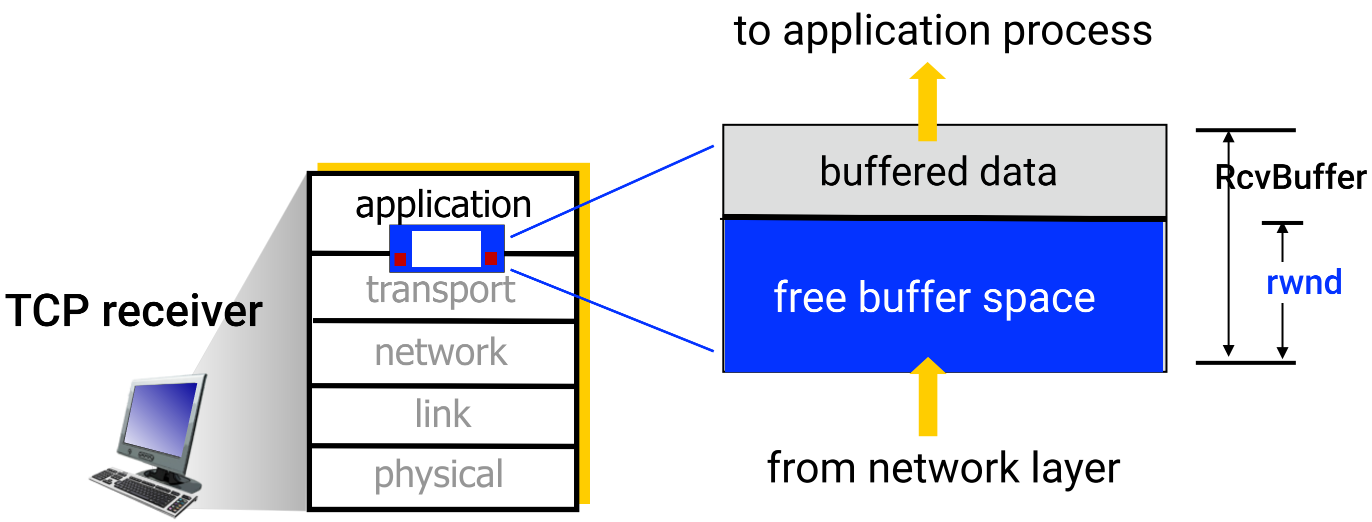
Receipt of triple duplicate ACKs indicates 3 segments received after a missing segment, so lost segment is likely. Retransmit!



TCP Flow Control

Key idea

let the receiver control the sender, so sender won't overflow receiver's buffer by transmitting too much, too fast



- TCP receiver advertises its free buffer space in **rwnd** field in TCP header
- rwnd is typically set to 4kB, while its full range is 0 to 64kB (16-bit field)
- managed internally by the TCP/IP stack, and could be modified via socket options()
- sender limits amount of unacknowledged, in-flight data to receiver's **rwnd**, thereby guaranteeing that receive won't experience buffer overflow

TCP Congestion Control

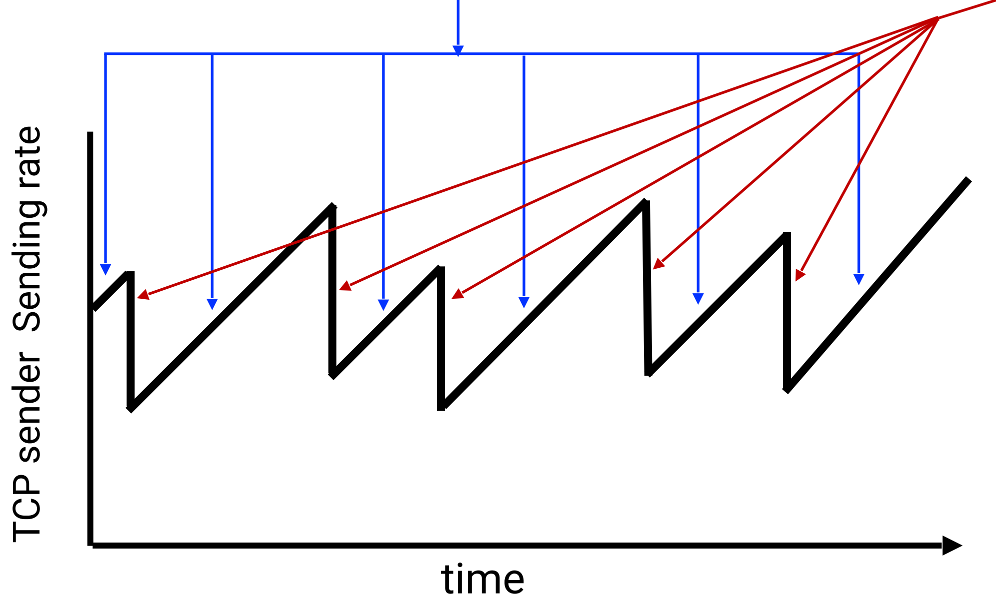
Key idea: *senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event*

Additive Increase

increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease

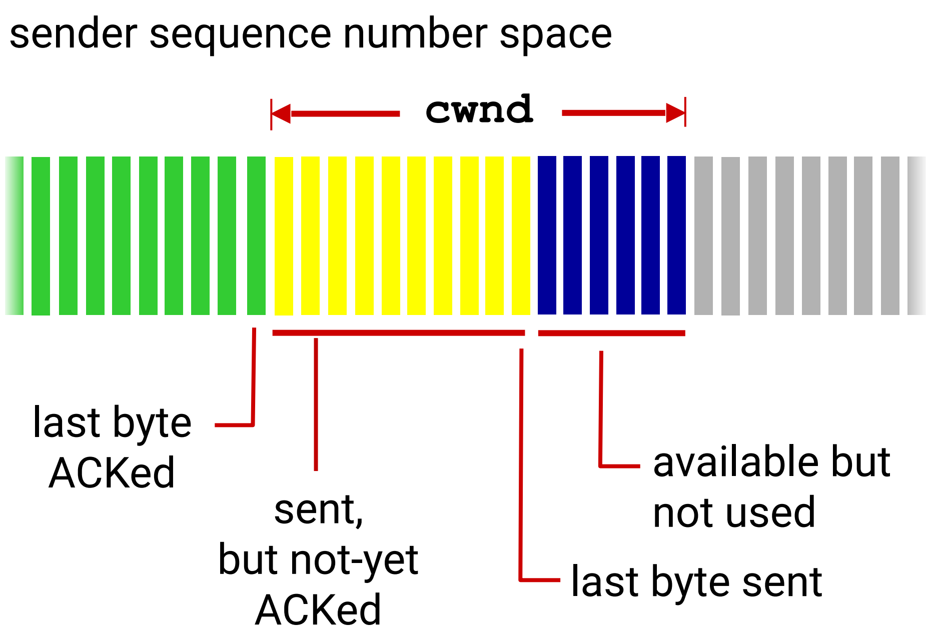
cut sending rate in half at each loss event (e.g., triple dup ACKs)



AIMD

- sawtooth behavior: probing for bandwidth
- a distributed, asynchronous algorithm
- shown to optimize network-wide flow rates

Classical TCP Implementation



TCP sender limits transmission:

`LastByteSent - LastByteAcked ≤ cwnd`

`cwnd` is dynamically adjusted in response to observed network congestion events

send cwnd bytes, wait RTT for ACKS, then send more bytes

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

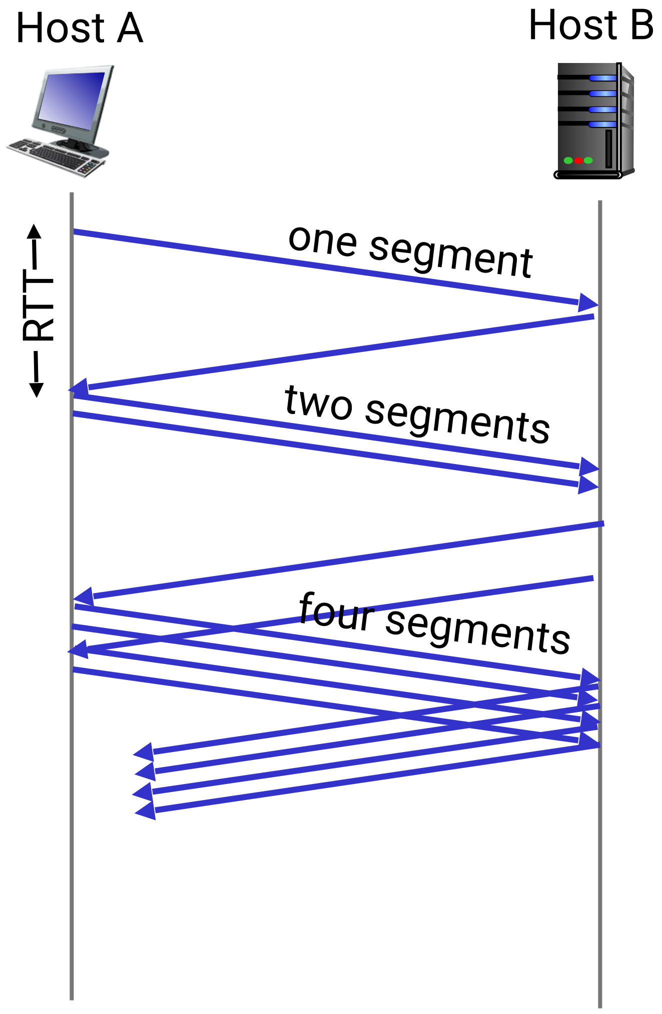
Two Phases: Slow Start and Congestion Avoidance

Slow Start: when a connection begins, increase sending rate exponentially until the first loss event

- start with `cwnd` = 1 MSS
- double `cwnd` every RTT i.e., increment `cwnd` for every ACK received

Congestion Avoidance: switch from exponential increase to linear increase when the connection hits first timeout

- set `ss-threshold` = `cwnd/2`
- switch to additive increase anytime `cwnd` reaches this level in the future



Spot Quiz (ICON)