# TCP Reliability and Congestion Control

GW CSCI 3907/6907 Adv Networking and Distributed Systems

Prof. Timothy Wood

# TCP Properties

**Reliability:** checksums

- Uses a 16 bit hash calculated over header/data as checksum
- Receiver can calculate checksum and verify it matches what is stored in the packet
- Is a checksum perfect?

What to do if checksum doesn't match?

**TCP Segment Header Format**

| Bit # | 0 – 7 | 8 – 15 | 16 – 23 | 24 – 31 |
|---|---|---|---|---|
| 0 | Source Port | | Destination Port | |
| 32 | Sequence Number | | | |
| 64 | Acknowledgment Number | | | |
| 96 | Data Offset | Res | Flags | Window Size |
| 128 | Header and Data Checksum | | | Urgent Pointer |
| 160... | Options | | | |

# TCP Properties

**Reliability:** based on sequence numbers and ACKs

- Client/server start connection with a random sequence number
- On every send, add the total amount of data transmitted
- On receive, reply with ACK specifying last seq number received

What to do…

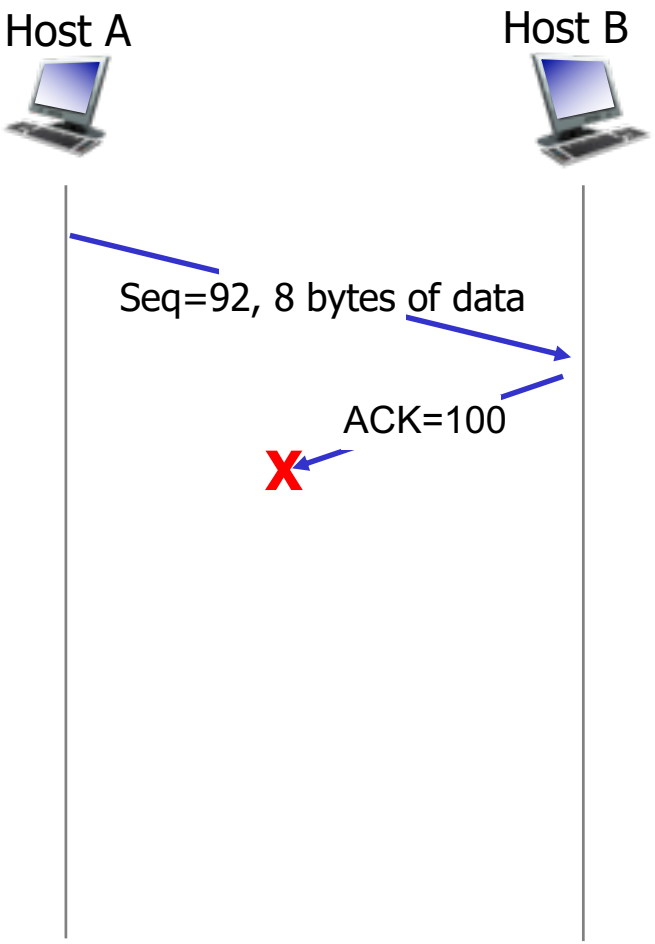
- If no ACK received?
- If receiver

**TCP Segment Header Format**

| Bit # | 0 – 7 | 8 – 15 | 16 – 23 | 24 – 31 |
|---|---|---|---|---|
| 0 | Source Port | | Destination Port | |
| 32 | Sequence Number | | | |
| 64 | Acknowledgment Number | | | |
| 96 | Data Offset | Res | Flags | Window Size |
| 128 | Header and Data Checksum | | Urgent Pointer | |
| 160... | Options | | | |

# What happens?

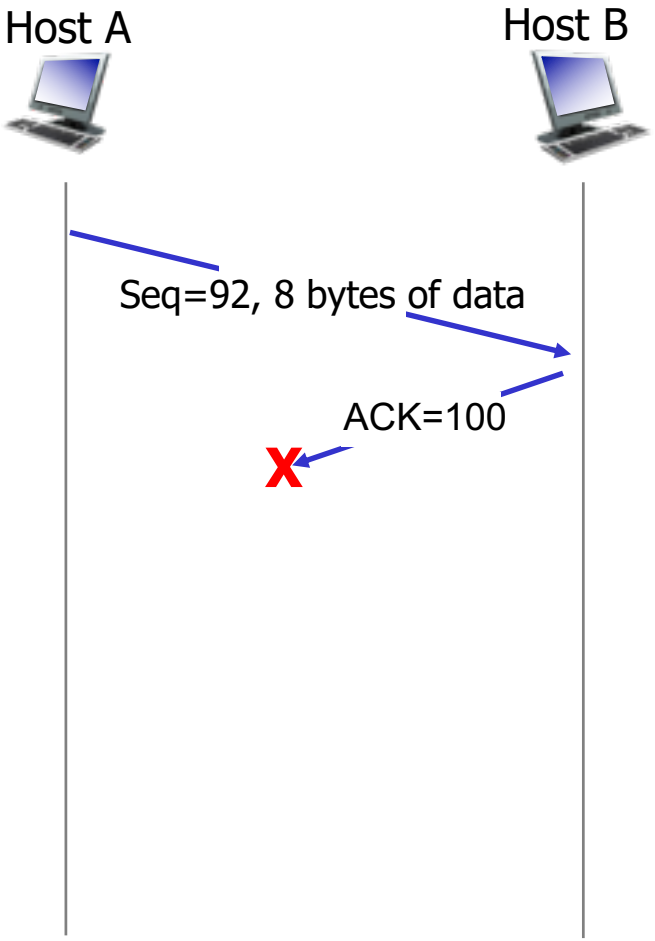




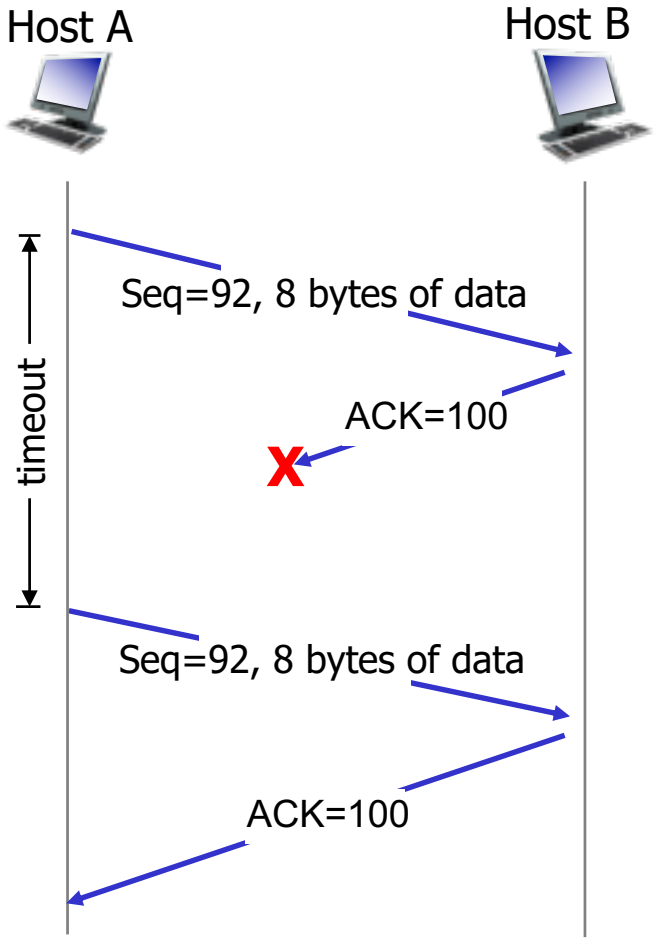

ACK lost

# What happens?






ACK lost




timeout and resend!
(same if original packet lost)

# Wait for ACKs?

Should the **sender** wait for an ACK after each packet before sending another one?

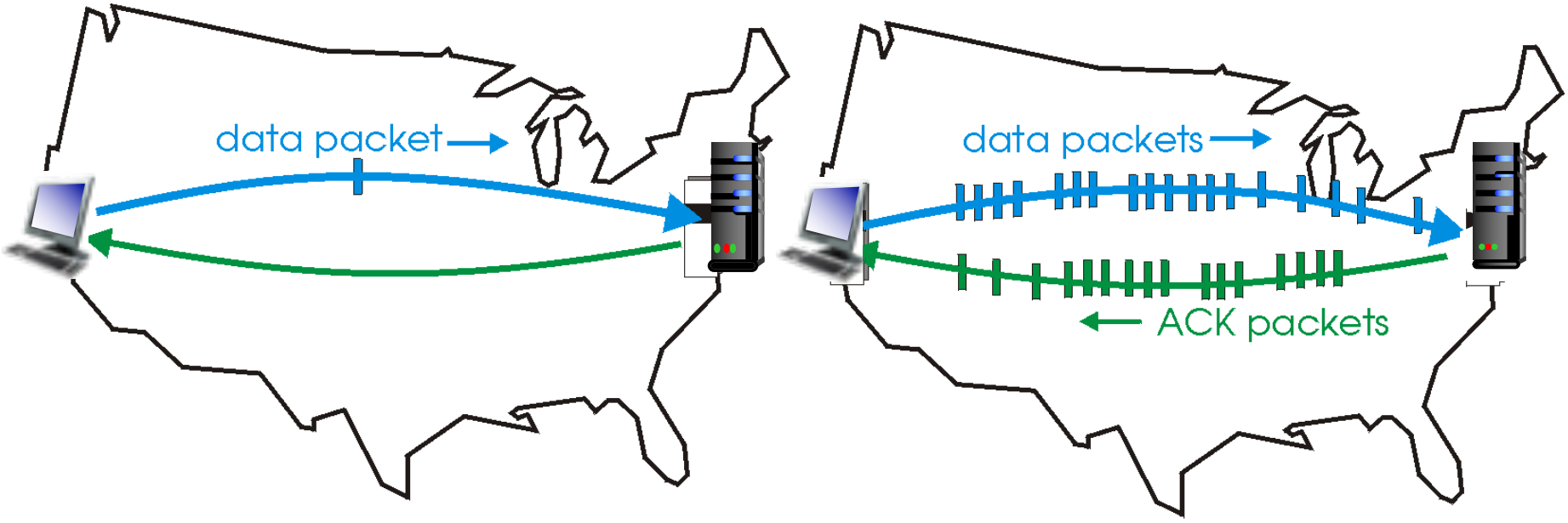
Benefits / Drawbacks?

# Wait for ACKs?



Should the **sender** wait for an ACK after each packet before sending another one?

Benefits / Drawbacks?




(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

# Wait for ACKs?

Should the **sender** wait for an ACK after each packet before sending another one?

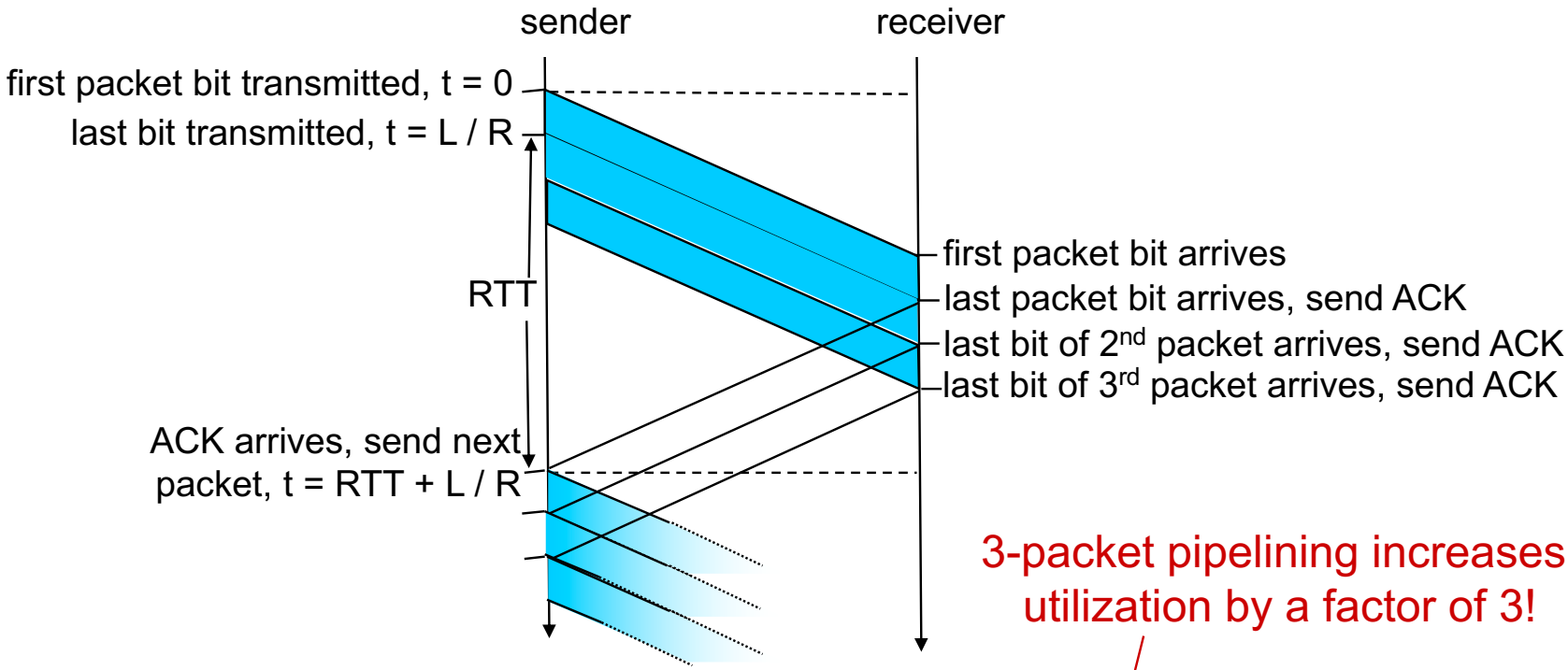
Benefits / Drawbacks?

Do the math!

# Pipelining Sends

Waiting for each ACK makes very poor use of our available bandwidth!

- Better to send a "window" of packets as a pipeline

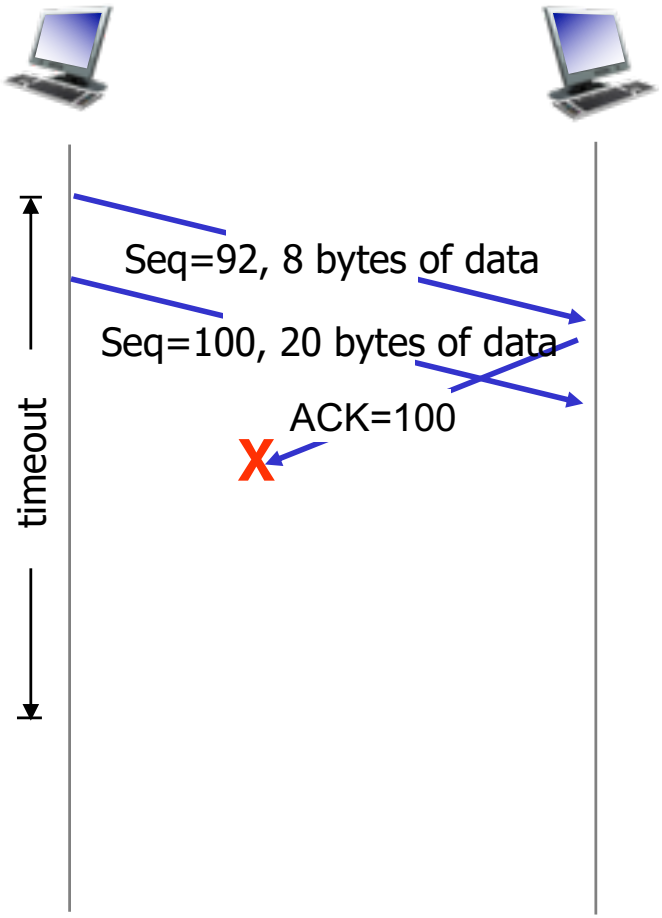



3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3L/R}{RTT + L/R} = \frac{.0024}{30.008} = 0.00081$$

# What happens?

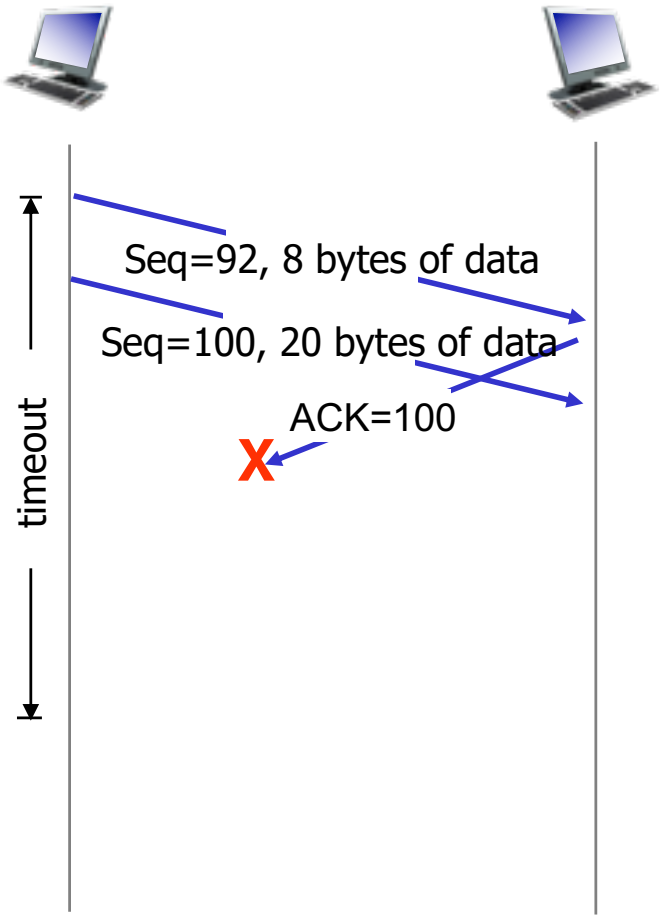




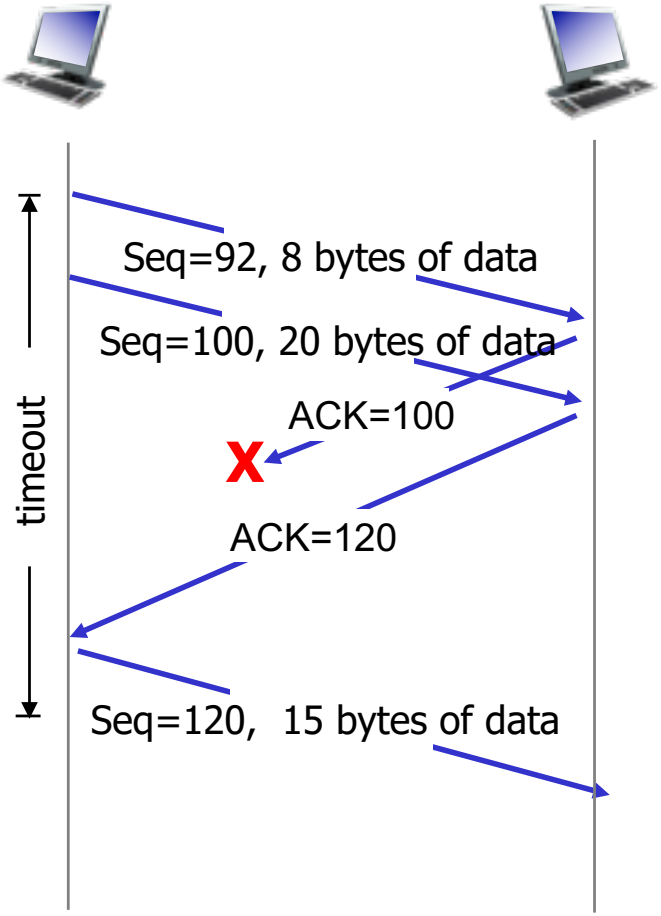

1st ACK lost

# What happens?






1st ACK lost

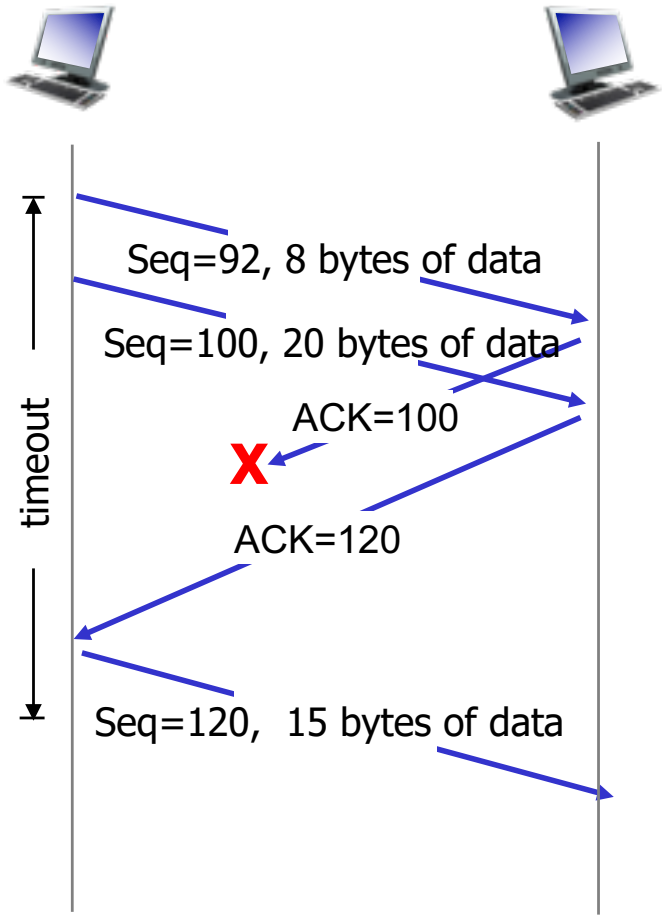



2nd ACK is cumulative!

# Cumulative ACKs

ACK 120 means ALL bytes up to that point are received

Why use cumulative instead of individual ACKs?




2nd ACK is cumulative!

# Wait for ACKs?

Should the **receiver** immediately send an ACK?

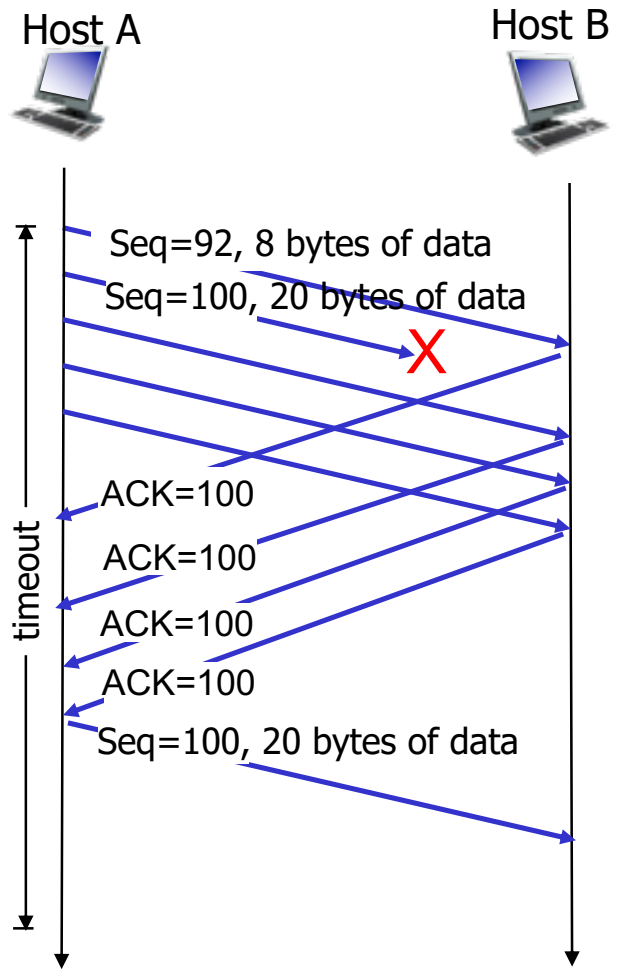
Benefits / Drawbacks?

# TCP Reliability



| *event at receiver* | *TCP receiver action* |
|---|---|
| arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed | delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK |
| arrival of in-order segment with expected seq #. One other segment has ACK pending | immediately send single cumulative ACK, ACKing both in-order segments |
| arrival of out-of-order segment higher-than-expect seq. # . Gap detected | immediately send *duplicate ACK*, indicating seq. # of next expected byte |
| arrival of segment that partially or completely fills gap | immediate send ACK, provided that segment starts at lower end of gap |

# TCP Fast Retransmit




fast retransmit after sender receipt of triple duplicate ACK

# How many packets to send?

Using a larger **window** leads to better link utilization

- *So why not just use a window of 1,000,000?*

Benefit of large window?

Drawback of large window?

# How many packets to send?

Using a larger **window** leads to better link utilization

- *So why not just use a window of 1,000,000?*

Benefit of large window?

- Can send more data before waiting for ACK

Drawback of large window?

- With cumulative ACK, it can be hard for sender to know exactly what it needs to resend
- Sending at too high a rate may cause higher packet loss!

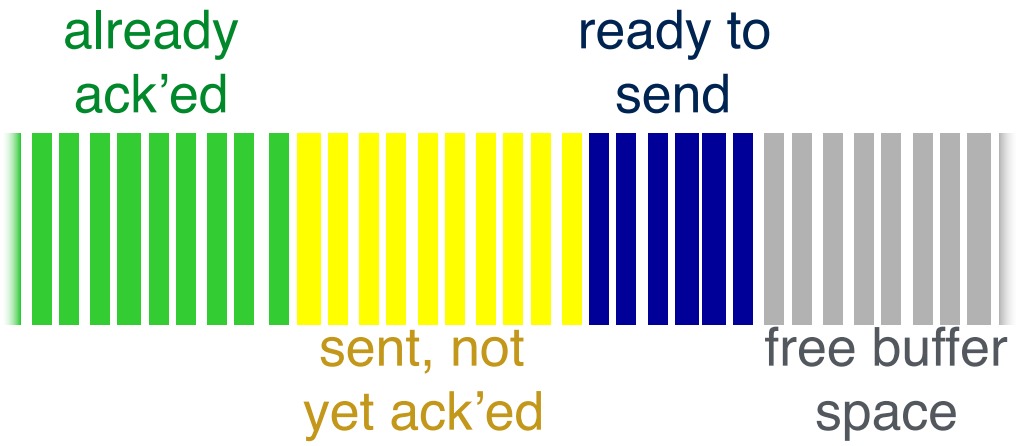
This leads to why TCP does Congestion Control!

- We'll just cover the basics…

# Windows

Window size controls # of packets in flight

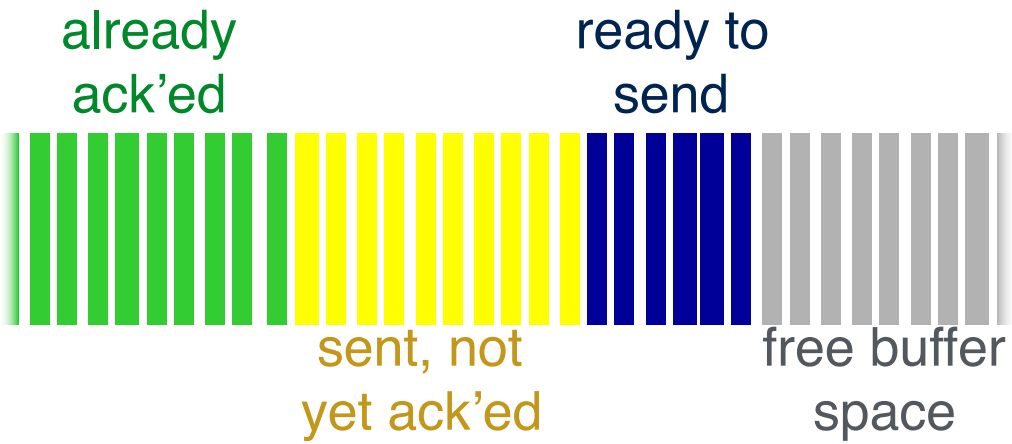
**Sender's view of Sequence Numbers**



# Windows

Window size controls # of packets in flight

**Sender's view of Sequence Numbers**




**Receiver's view of Sequence Numbers**


Remember that TCP is bidirectional, so this all happens twice!

# Congestion Control Basics

How should we adjust window size?

- Let's assume client is sending a large file to server

# Congestion Control Basics

How should we adjust window size?

- Let's assume client is sending a large file to server

Startup:

- use a small window since you don't know anything about receiver

No drops for a while:

```
window_size++;     // Send faster!
```

Drop detected:

```
window_size = window_size/2;  // whoa! slow down!
```

Additive Increase, Multiplicative Decrease (AIMD)