

Lecture 5: Flow Control

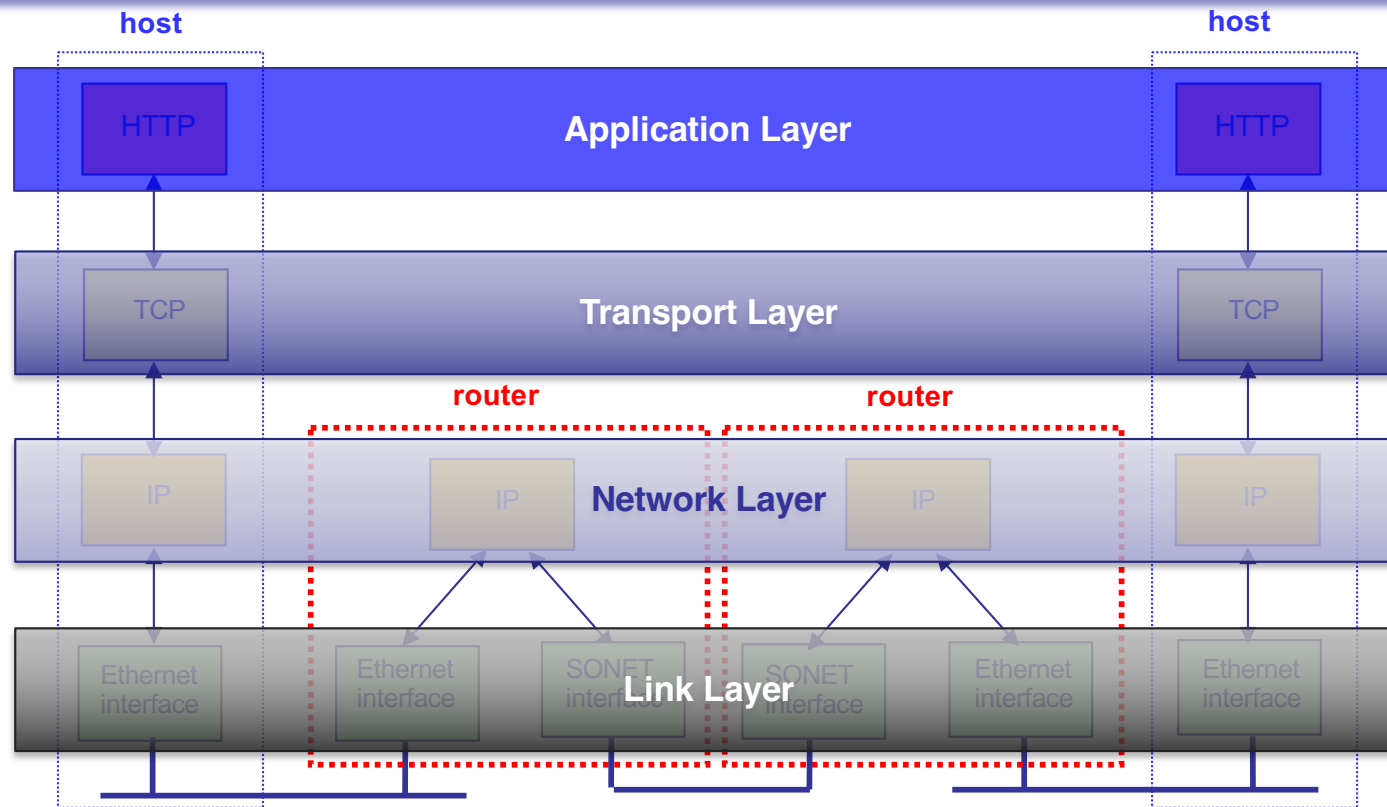
CSE 123: Computer Networks
Alex C. Snoeren



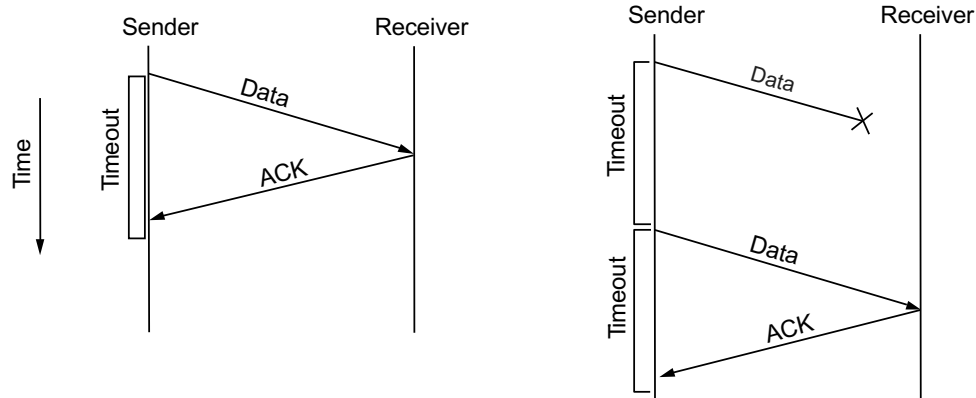
HW 1 due FRIDAY



Moving up the Stack

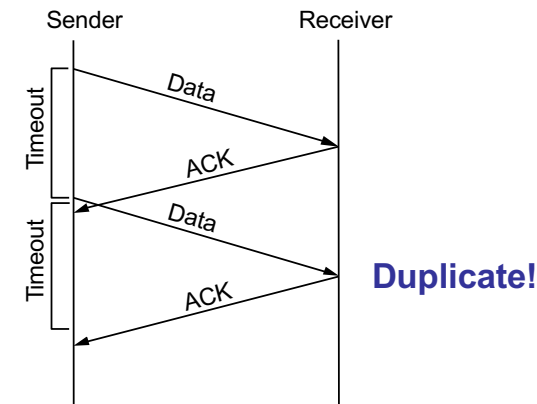
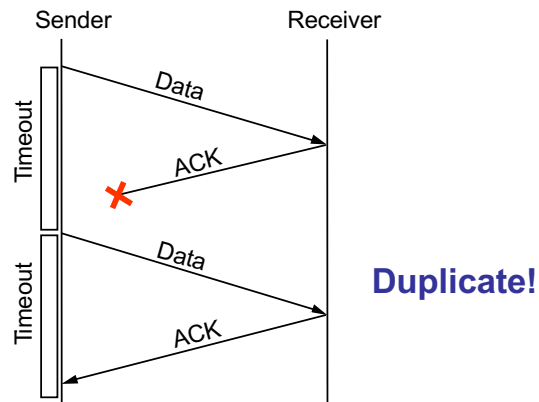


Simple Idea: ARQ



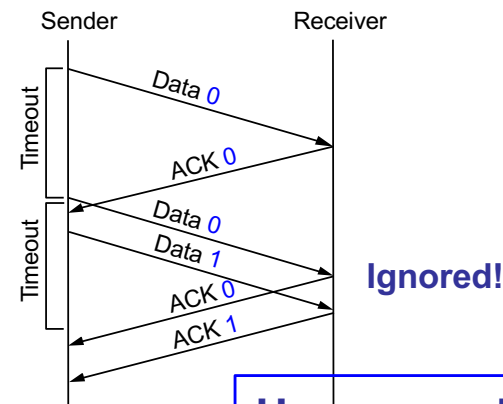
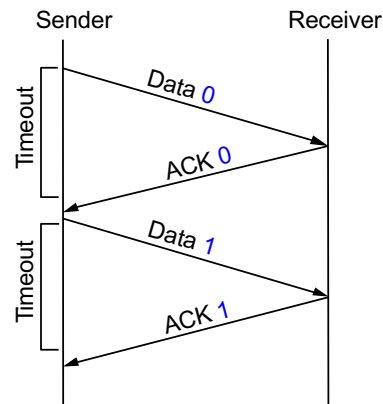
- Receiver sends **acknowledgments** (ACKs)
 - ◆ Sender “times out” and retransmits if it doesn’t receive them
- Basic approach is generically referred to as **Automatic Repeat Request** (ARQ)

Not So Fast...



- Loss can occur on ACK channel as well
 - ♦ Sender cannot distinguish data loss from ACK loss
 - ♦ Sender will retransmit the data frame
- ACK loss—or early timeout—results in duplication
 - ♦ The receiver thinks the retransmission is new data

Sequence Numbers



- Sequence numbers solve this problem
 - ◆ Receiver can simply ignore duplicate data
 - ◆ But must still send an ACK! (Why?)
- Simplest ARQ: **Stop-and-wait**
 - ◆ Only one outstanding frame at a time

How many bits does stop-and-wait need for sequence numbers?

- A. 1**
- B. 2**
- C. Depends on timeout**
- D. I don't know**

Stop-and-Wait Performance



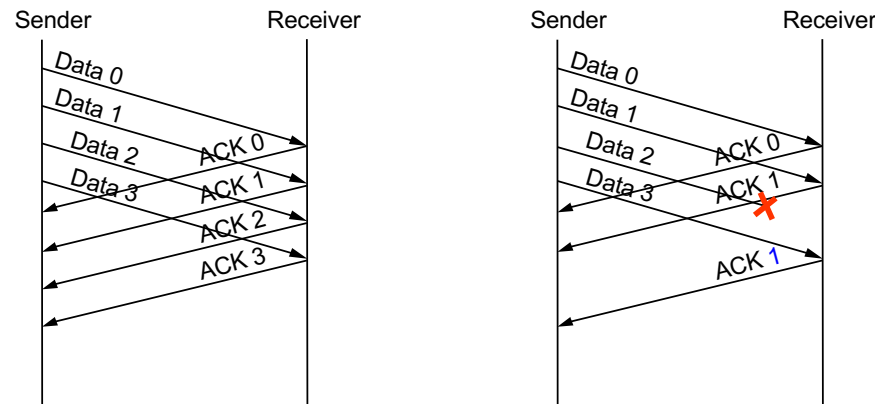
- Lousy performance if time to transmit 1 packet (**serialization delay**) $\ll$ time to get to the receiver (**propagation delay**)
 - ◆ How bad? Depends on **round trip time (RTT)**
- Want to utilize all available bandwidth
 - ◆ Need to keep more data “in flight”
 - ◆ How much? Called the **bandwidth-delay product**
- Also limited by quality of timeout (how long?)

How many bits do we want to keep in flight?

- A. Bandwidth * one-way delay
- B. Bandwidth * round-trip delay
- C. Depends on timeout
- D. I don't know

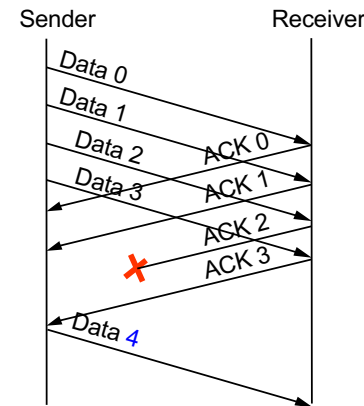
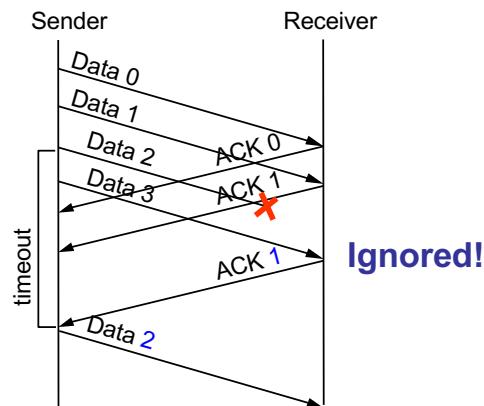


Pipelined Transmission



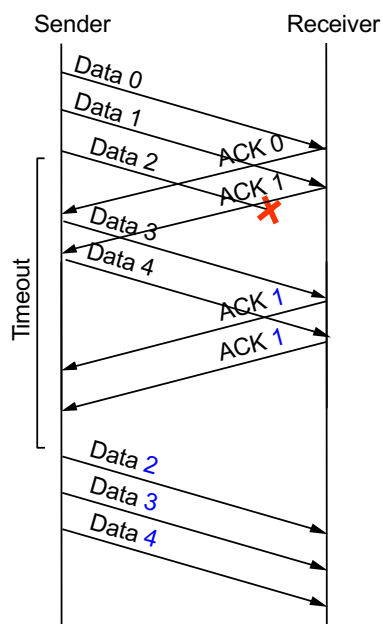
- Keep multiple packets “in flight”
 - ♦ Allows sender to make efficient use of the link
 - ♦ Sequence numbers ensure receiver can distinguish frames
- Sender buffers outstanding un-acked packets
 - ♦ Receiver ACKs the highest *consecutive* frame received
 - » ACKs are **cumulative** (covers current frame and all previous)

Go-Back-N



- Retransmit all packets from point of loss
 - ◆ Packets sent after loss event are ignored (i.e., sent again)
- Simple to implement (receiver doesn't need to buffer)
- Sender controls how much data is “in flight”

Send Window



Go-Back-N Example
with window size 3

- Bound on number of outstanding packets
 - ◆ Window “opens” upon receipt of *new* ACK
 - ◆ Window resets entirely upon a timeout
- Limits amount of waste
 - ◆ Still lots of duplicates in Go-Back-N
 - ◆ We can do better with **selective retransmission**

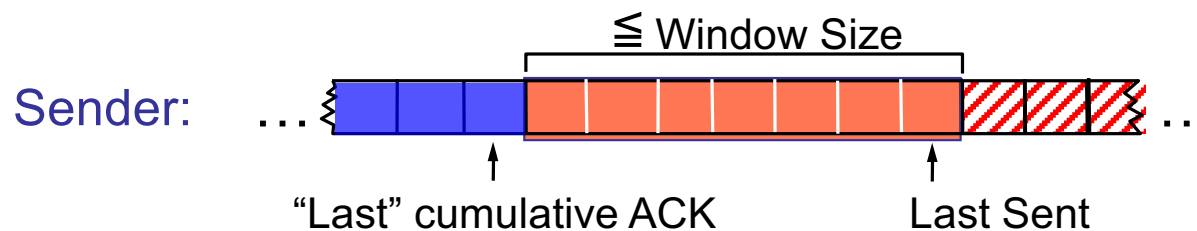
Sliding Window



- Single mechanism that supports:
 - ◆ Multiple outstanding packets
 - ◆ Reliable delivery
 - ◆ In-order delivery
 - ◆ Flow control
- Sender and receiver each maintain “window” abstractions to track outstanding packets
 - ◆ At the core of all modern ARQ protocols
- Go-Back-N is a special case
 - ◆ Receive window size of one

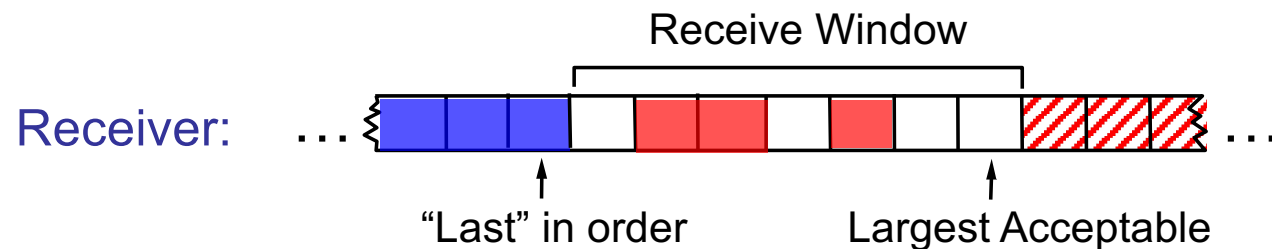


Sliding Window – Sender



- Window bounds outstanding unACKed data
 - ♦ Implies need for buffering at sender
- “Last” ACK applies to **in-order** data
- What to do on a timeout?
 - ♦ Go-Back-N: resend all unacknowledged data on timeout
 - ♦ Selective Retransmit: timer per packet, resend as needed

Sliding Window – Receiver



- Receiver buffers too:
 - ♦ data may arrive out-of-order
 - ♦ or faster than can be consumed
 - » **Flow control:** tell sender how much buffer left at receiver
- Receiver sends an ACK for every segment:
 - ♦ **Cumulative**, Selective (exempt missing frames), or Negative (only missing frames)



Deciding When to Retransmit

- How do you know when a packet has been lost?
 - ♦ Ultimately sender uses timers to decide when to retransmit
- But how long should the timer be?
 - ♦ Too long: inefficient (large delays, poor use of bandwidth)
 - ♦ Too short: may retransmit unnecessarily (causing extra traffic)
- Right timer is based on the **round-trip time (RTT)**
 - ♦ Which can vary greatly for reasons we'll see later



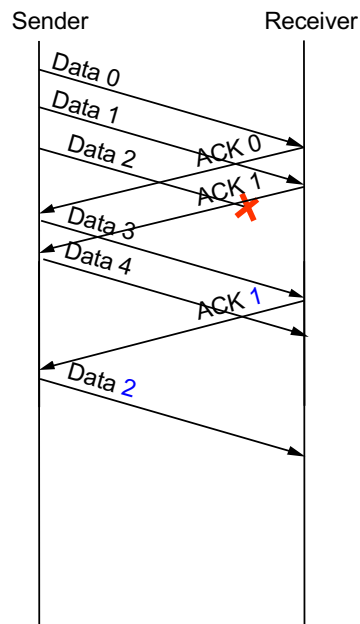
Can we shortcut the timeout?

- Timeout is long in practice
 - ♦ Lots of variation in RTT and timeout must be conservative
- If packets are usually *in order* then duplicate cumulative ACKs imply that a packet was lost
 - ♦ Fast retransmit
 - » When sender receives multiple duplicate acknowledgements resends missing packet

**Duplicate ACKs for packet N
indicate which packet was
lost?**

- A. Packet N-1**
- B. Packet N**
- C. Packet N+1**
- D. Packet N+2**
- E. You can't tell**

Fast retransmit



- Don't bother waiting
 - ◆ Receipt of duplicate acknowledgement (**dupACK**) indicates loss
 - ◆ Retransmit immediately
- Used in TCP
 - ◆ Need to be careful if frames can be reordered
 - ◆ TCP Reno identifies a loss if there are **three** duplicate ACKs in a row

For Next Time



- Read 5-5.2 in P&D
- HW 1 due at the beginning of class Friday
- (Keep) going on the project...