

Lecture 4:

Transport Layer

CSE 123: Computer Networks

Aaron Schulman



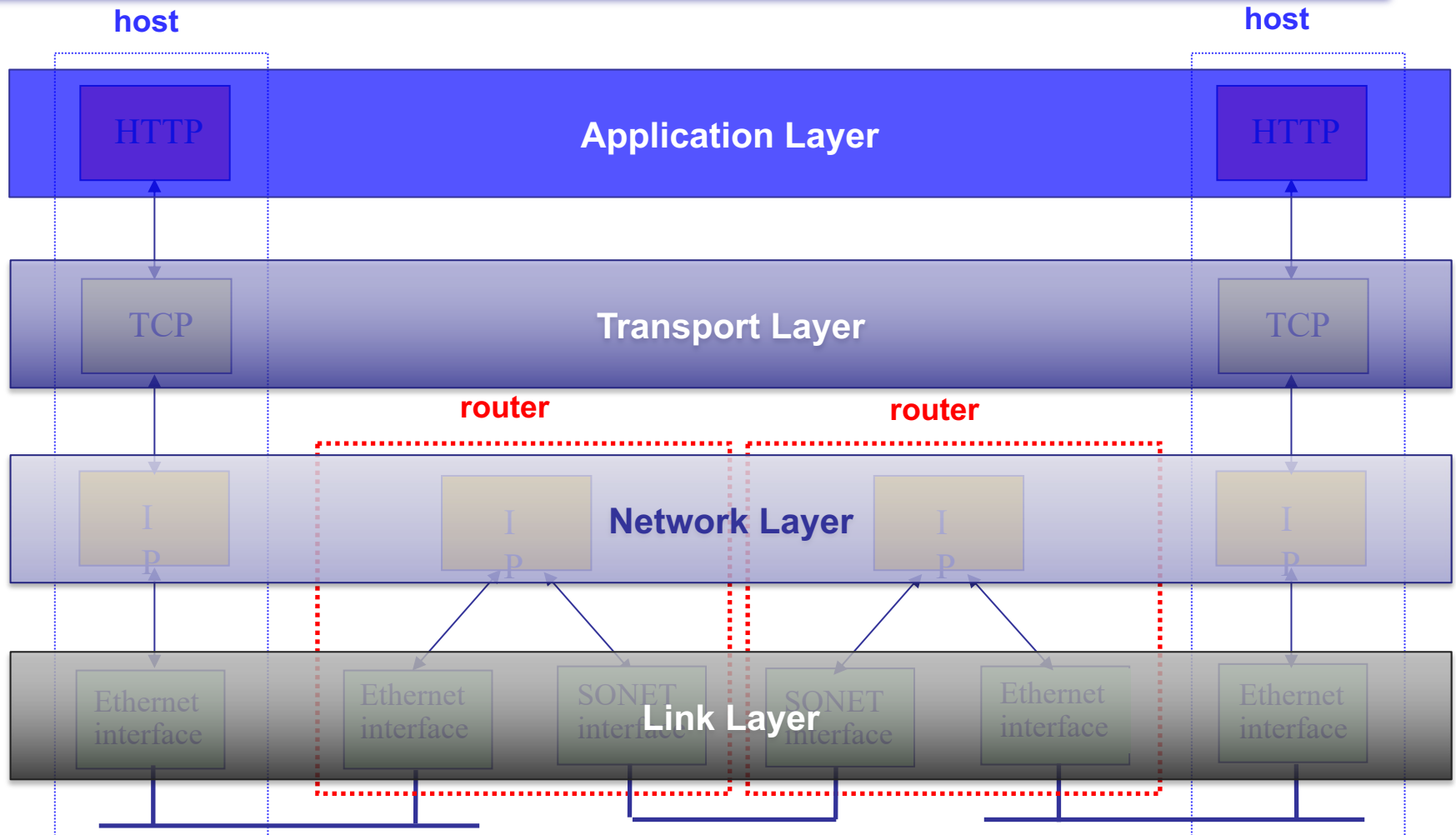


Picking up the Pieces

- Link layer is lossy
 - ◆ We deliberately throw away (drop) corrupt frames!
 - ◆ Packets can be sent faster than link speed (link is a bottleneck)
 - ◆ Infrequent bit errors still lead to occasional frame errors
 - » 10,000+ bits in each frame, if 1 in a billion bits have errors they happen often!
- Things get even harrier if we consider crossing multiple links
 - ◆ In a few lectures, we'll start sending frames on long trips
 - ◆ Each intermediate link might lose, corrupt, *reorder*, etc.
 - ◆ Regardless of cause, we'll call loss events **drops**
- We want to provide reliable, in-order delivery
 - ◆ Can—and will—do this at multiple layers



Moving up the Stack



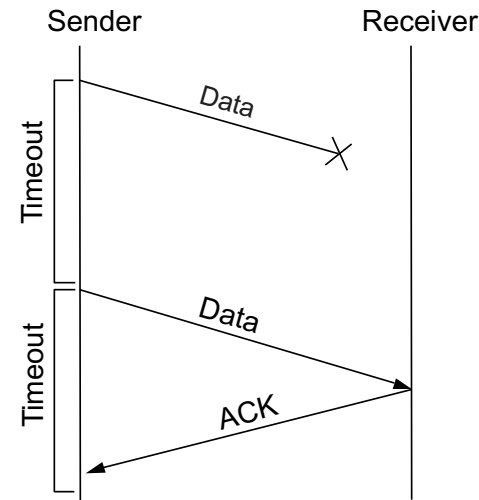
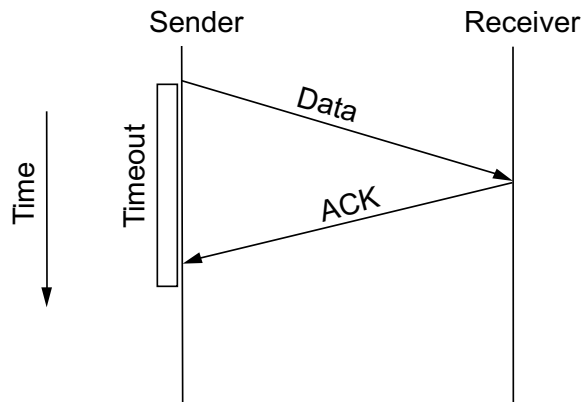


Reliable Transmission

- The **packet**-based version of reliability
 - ◆ How do we reliably send a message when **packets** (not just bits in frames) can be lost/corrupted in the network?
- Two options
 - ◆ **Detect a loss/corruption and retransmit (transport layer)**
 - ◆ Send data redundantly to tolerate loss/corruption



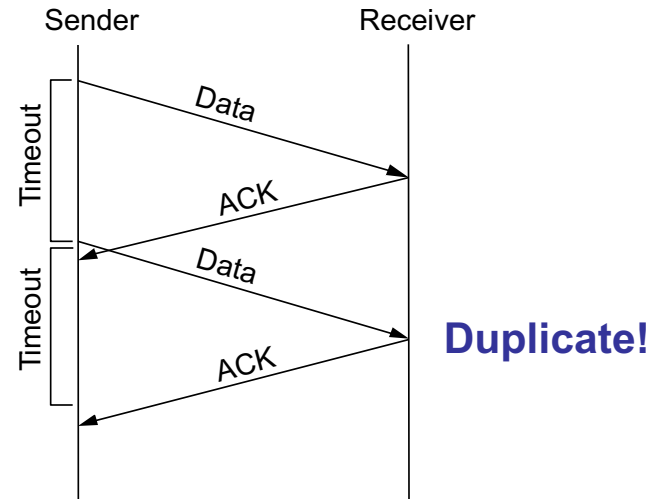
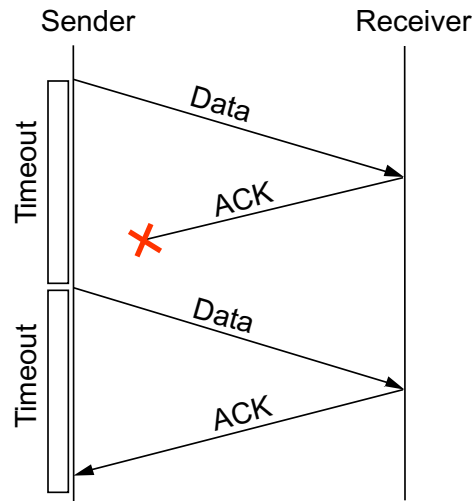
Simple Idea: ARQ



- 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)



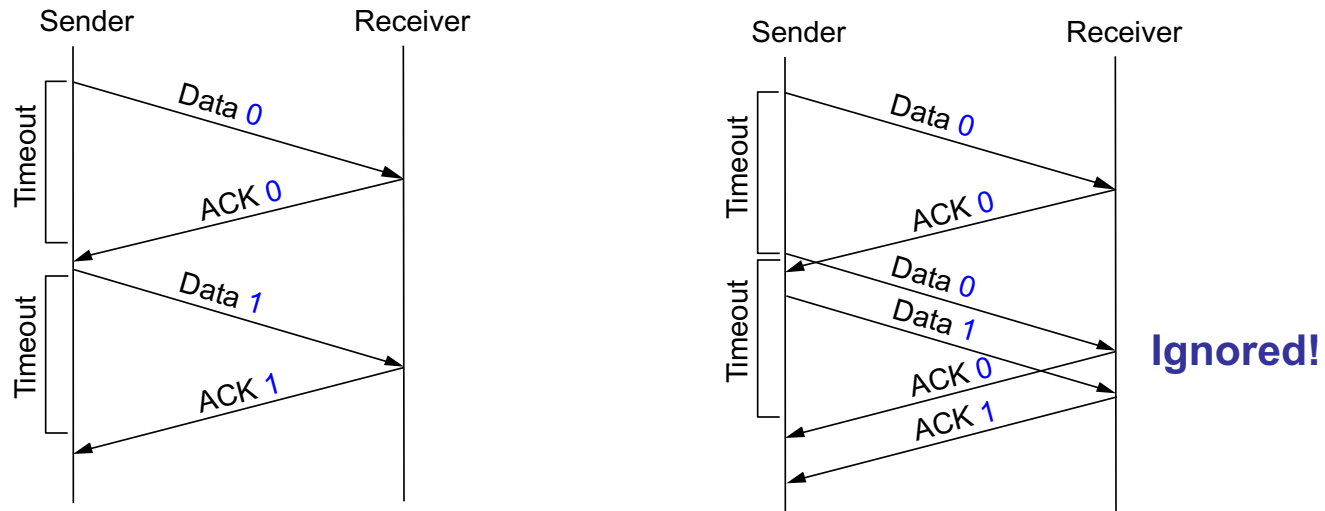
Not So Fast...



- 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!

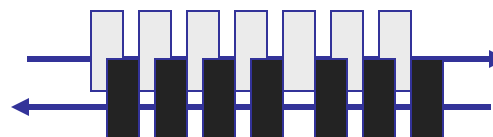
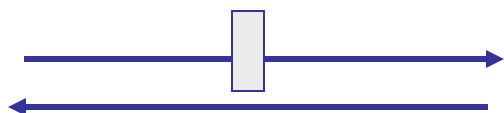


Sequence Numbers



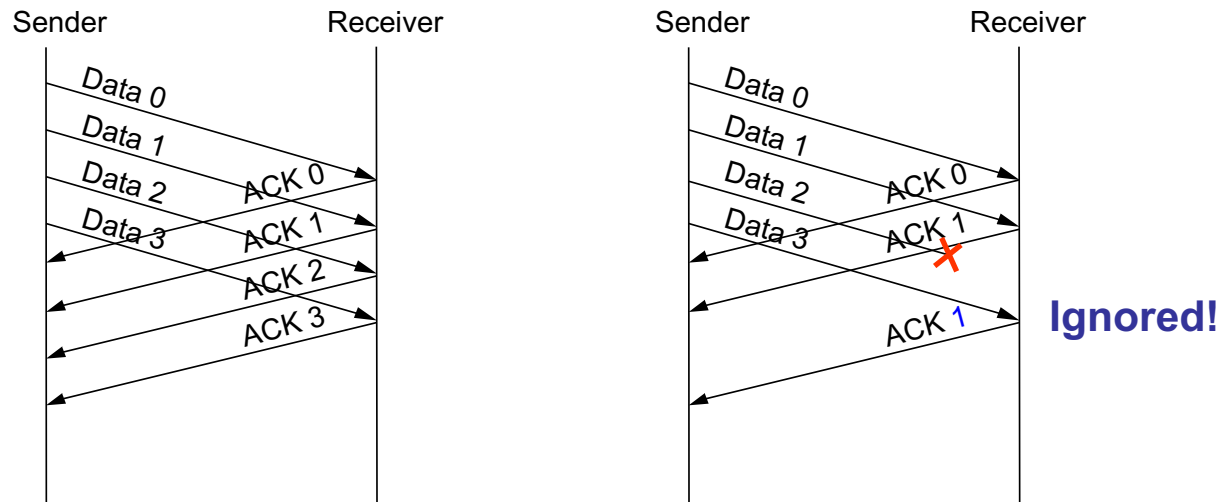
- Sequence numbers solve this problem
 - ◆ Receiver can simply ignore duplicate data
 - ◆ But must still send an ACK! (Why?)
- Simplest ARQ: **Stop-and-wait**
 - ◆ Only allow one outstanding frame at a time

Stop-and-Wait Performance



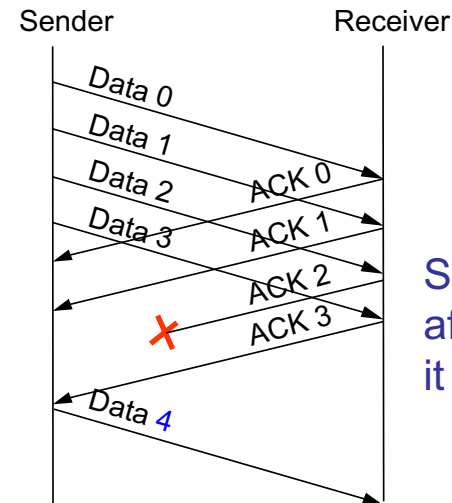
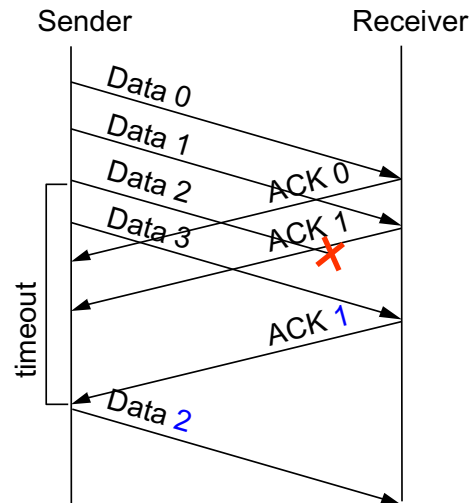
- **Lousy performance** if $\text{time}(\text{xmit 1 pkt}) \ll \text{propagation delay}$
 - ◆ How bad?
- Want to utilize all available bandwidth
 - ◆ Need to keep more data “in flight”
 - ◆ How much? Called the **Bandwidth-Delay Product (BDP)**
- Also limited by accuracy of timeout (how long until gone?)

Getting the BDP: 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



Sender can continue after 2's loss because it receives ACK 3

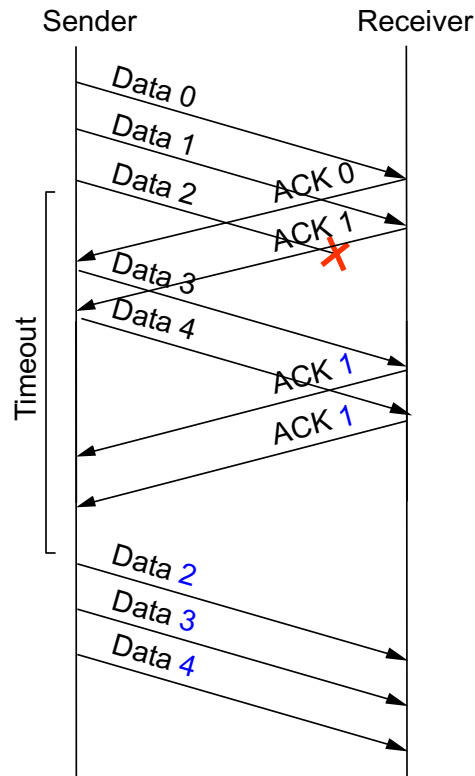
- Retransmit all packets from point of loss
 - ◆ Packets sent after lost packet (or ACK) are ignored (i.e., sent again)
- Simple to implement
(receiver only has to track one packet at a time)
- Go-Back-N might still lead to sending lots of duplicates
 - » We can do better with **selective retransmission**:
only retransmit missed packets



Sliding Window

- Single mechanism that supports:
 - ◆ Multiple outstanding packets (pipelined transmission)
 - ◆ Reliable delivery
 - ◆ In-order delivery
 - ◆ *Flow control*: Don't send more than receiver can handle
- 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

Send Window (sender buffer)

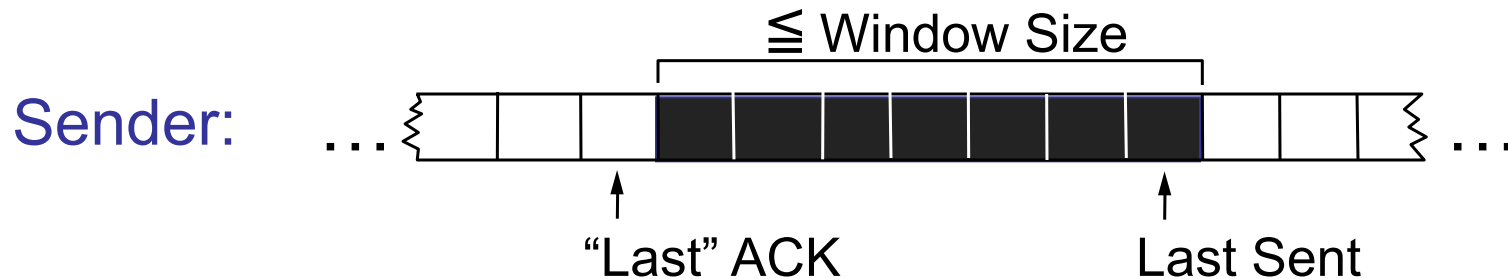


**Go-Back-N Example
with window size 3**

- Bound on number of outstanding packets in memory
 - ◆ Window “moves forward” upon receipt of *new* ACK
 - ◆ Window goes back to beginning upon a timeout
- Limits amount of memory waste
 - ◆ Don't need to remember all packets ever sent

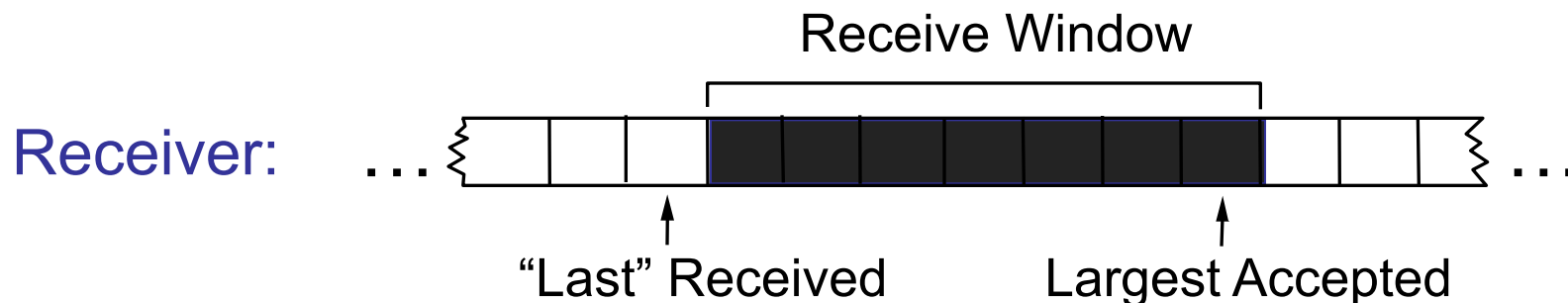


Sliding Window – Sender



- 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 Repeat: timer per packet, resend as needed

Sliding Window – Receiver



- Receiver buffers packets that arrive...
 - ◆ ...*out-of-order*, they are stored to provide in-order
 - ◆ ...*faster* than can be consumed by higher layers
 - » **Flow control**: tell sender how much receive window is left
 - Sender will only send enough packets to fill the receive window

Deciding When to Retransmit



- How does sender know when a packet has been lost?
 - ◆ Ultimately sender uses **timeouts** to decide when to retransmit
- But how long should the timeout be?
 - ◆ Too long: inefficient (large delays, poor use of bandwidth)
 - ◆ Too short: may retransmit unnecessarily (causing extra traffic)
- Right timer is based on link latency: **round-trip time** (RTT)
 - ◆ Which can vary greatly for reasons well see later
 - ◆ Latest transport protocols (e.g., Google QUIC) have sophisticated schemes for computing timeout for retrainsmissions.

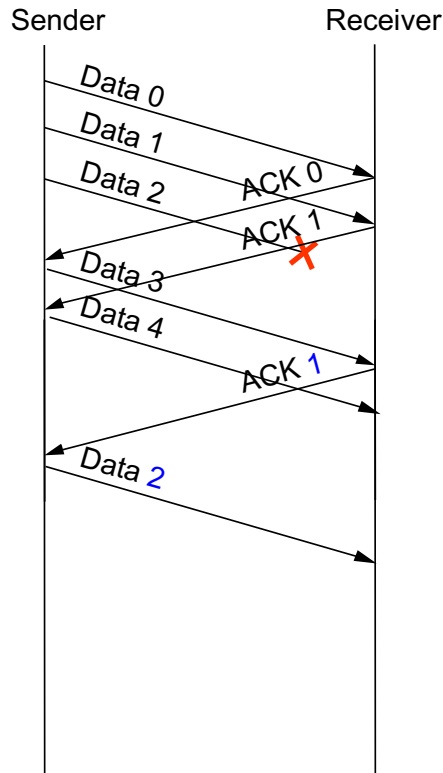
Can we shortcut the timeout?



- Timeout is long in practice
 - ◆ Lots of variation in RTT so timeout must be conservative
- If packets are usually *in order* then *out-of-order* packets or ACKs imply that a packet was lost
- Knowing something is lost, we can be proactive!
 - ◆ Negative ACK (receiver side informs of missing packet)
 - » Receiver requests missing packet
 - ◆ Fast retransmit (sender side)
 - » When sender receives multiple duplicate acknowledgements resends missing packet



Fast retransmit



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

Overview of all Transport Layer Tasks



- Define and provide specific delivery semantics
 - ◆ To which end point (which application/process)?
 - ◆ When?
 - ◆ How?
 - ◆ If?
- Multiplexing different processes between the same hosts
- ... and Reliability (if the application needs it)



For Next Time

- Read 5-5.1 in P&D
- Homework out Today and due in 1 week
- (Keep) going on the project...