Lecture 5: Reliable Transmission and Flow Control
Picking up the Pieces

- Link layer is lossy
  - We deliberately throw away corrupt frames!
  - Infrequent bit errors still lead to occasional frame errors
    » 10,000+ bits in each frame

- Things get even harder if we consider multiple links
  - In a few lectures, we’ll start sending frames on long trips
  - Each intermediate stop 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

Application Layer

Transport Layer

Network Layer

Link Layer

host

router

host

CSE 123 – Lecture 5: Reliable Transmission
Reliable Transmission

- The packet-based version of the same problem
  - How do we reliably send a message when packets (not just bits) can be lost/corrupted in the network?

- Two options
  - Detect a loss/corruption and retransmit
  - Send data redundantly to tolerate loss/corruption
Receiver sends **acknowledgments** (ACKs)
- Sender assumes transmission didn’t make when doesn’t receive an ACK after some time (timeout)
- 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 also solves this problem: Stop-and-wait
  - Only one outstanding frame at a time
Stop-and-Wait Performance

- Lousy performance if time to xmit 1 pkt << prop. delay
  - How bad?

- Want to utilize all available bandwidth
  - Need to keep more data “in flight”
  - How much? Called the bandwidth-delay product

- Also limited by accuracy of **timeout**: period of time when sender is confident it is lost.
  - More on picking timeouts at the end of the lecture
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 lost packet (or ACK) are ignored (i.e., sent again)
- Simple to implement
  (receiver only has to track one packet at a time)
- Sender controls how much data is “in flight”
Send Window (sender buffer)

- Bound on number of outstanding packets
  - Window “opens” upon receipt of new ACK
  - Window resets entirely upon a timeout
- Limits amount of waste
  - Don’t need to remember all packets ever sent
  - Go-Back-N might still lead to sending lots of duplicates
    » We can do better with selective retransmission: only retransmit missed packets

Go-Back-N Example with window size 3
Sliding Window

- Single mechanism that supports:
  - Multiple outstanding packets
  - 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
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
Receiver:  

- Receiver buffers too:
  - data may arrive out-of-order
  - or 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: \textit{round-trip time} (RTT)
  - Which can vary greatly for reasons well 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 *out-of-order* ACKs from receiver imply that a packet was lost

- Knowing something is lost, we can be proactive!
  - Negative ACK
    » Receiver requests missing packet
  - Fast retransmit
    » When sender receives multiple duplicate acknowledgements resends missing packet

CSE 123 – Lecture 5: Flow Control
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
For Next Time

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