Lecture 7:
Sliding Windows

CSE 123: Computer Networks
Stefan Savage

UCSD CSE
From last class: 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
  - Called *alternating bit protocol* in book (1 bit sequence number)
Stop-and-Wait Performance

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

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

- Also limited by quality of timeout (how long?)
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)
  - What to do after a loss?
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 very simple)
- Sender controls how much data is “in flight”
How many packets in flight?

Send Window

- 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
  - We can do better with selective retransmission

Go-Back-N Example with window size 3
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 Repeat: timer per packet, resend as needed
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 ACK choices:
- Cumulative, Selective (exempt missing frames), Negative
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 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 imply that a packet was lost
  - Negative ACK
    » Receiver requests missing packet
  - Fast retransmit
    » When sender receives multiple duplicate acknowledgements resends missing packet
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
  - Today’s TCP identifies a loss if there are three duplicate ACKs in a row
Is ARQ the Only Way?

- No. We could use redundancy
  - Send additional data to compensate for lost packets

- Why not use retransmission?
  - Broadcast media with lots of receivers
    - If each one ACK/NAK then hard to scale
      - Lots of messages
      - Lots of state
    - Heterogeneous receivers
      - Some receivers can handle 500kbps others 100Mbps
      - E.g., variable quality wireless reception
  - Highly lossy or very long delay channels (e.g., satellite)
Forward Error Correction

- Use **erasure codes** to redundantly encode $k$ data frames into $m > k$ encoded frames
  - E.g., Reed Solomon Codes, Tornado codes, Raptor codes, etc

- Multicast/broadcast encoded frames speculatively

- A receiver can reconstruct message from *any* $k$ frames in the set of $m$ encoded frames
Summary

- The primary way we achieve reliability over an unreliable channel is **automatic repeat request**
  - Explicitly notify sender of received packets by sending an acknowledgement (ACK)
  - Resend packets that are not ACKed within some timeframe

- We can make this efficient by having many unACKed packets simultaneously in flight

- Sliding Window is the standard technique to manage this process and keep the sender and receiver in sync
  - And limit the amount of data needing to be buffered

- Can use pattern of ACKs to infer losses by making assumptions about packet reordering
For Next Time

- Keep reading 2.6 in P&D
- Keep going on the project…