Where in the world is Stefan Savage?
Moving

Starting Thu Oct 13, class will move to Center Hall 109
Lecture 5: Reliable Transmission

CSE 123: Computer Networks
Stefan Savage
Last Time...

- Framing: How to translate a bitstream into separate "units of transfer"
  
  \[ \ldots 10101011100100111000010110 \]

- Sentinel based framing, length-based framing, clock based framing

- Error detection – CRC, parity, Internet checksum
Today: reliable data delivery

Neither snow, nor rain, nor heat, nor gloom of night stays these couriers from the swift completion of their appointed rounds.
Picking up the Pieces

- Link layer is lossy
  - We deliberately threw away corrupt frames last lecture
    » Infrequent bit errors still lead to occasional frame errors
      ■ 10,000+ bits in each frame
  - We also need to drop frames when they arrive too fast
    (no buffering available when the frame arrives)

- Things get even harder if we consider multiple links
  - Soon we’ll start sending frames across multiple “hops”
  - Each intermediate stop might lose, corrupt, reorder, etc.
  - Regardless of cause, we’ll call loss events drops

- Goal: we want to provide reliable, in-order delivery
  - But how? The channel loses packets…
Stepping back: A thought experiment

- You want to send a long letter to your friend
  - The only medium available to either of you is postcards
  - Postcards get lost in the mail, delayed, damaged, reordered
- How do you ensure that your friend receives the letter?
Reliable Transmission

- The data networking version of the same problem
  - How do we reliably send a message when packets can be lost/corrupted in the network?

- Two options
  - Detect a loss/corruption and retransmit
  - Send data redundantly to tolerate loss/corruption
Rest of lecture

- Automatic Repeat Request (ARQ)
  - Acknowledgements (ACKs) and timeouts
  - Stop-and-Wait protocol

- Implementing ARQ efficiently
  - Sliding Windows
  - Selective retransmission vs Go-Back-N
  - Detecting losses

- Forward Error Correction
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 to ACKs 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
  - Called alternating bit protocol in book (1 bit sequence number)
What if packets are delayed?

- One bit not enough… what to do?

- Never reuse a seq #?
  - Seq #s could be really big

- Require in-order delivery?
  - Hard to guarantee in some networks

- Prevent very late delivery?
  - Limit lifetime of each packet (drop pkt if not delivered in n seconds)
  - Seq #s not reused within delay bound
  - Approximate with big seq #s

Accept!  
Reject!
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 to wait?)
Pipelined Transmission

- Keep multiple packets “in flight”
  - Allows sender to make efficient use of the link
  - Large seq numbers ensure receiver can distinguish frames
- Sender buffers outstanding un-acked packets
  - Receiver ACK 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”
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 (resending pkt 3 and 4)
  - 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 only to in-order data
- What to do on a timeout?
  - Go-Back-N: send all unacknowledged data on timeout
  - Selective Repeat: timer per packet, resend each as needed
Sliding Window – Receiver

- Receiver buffers too:
  - data may arrive out-of-order; placed in proper order
  - or faster than can be consumed—flow control (another class)

- 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 when we study congestion control
Can we shortcut the timeout?

- Timeout can be long in practice

- If packets are *usually* in order then out-of-order data implies to the receiver that a packet was lost
  - Negative ACK
    - Receiver requests missing packet
  - Fast retransmit
    - Receiver always ACKs last in-order packet
    - 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
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 (e.g., update to TV guide)
    - If each one ACK/NAK then hard to scale
      - Lots of messages
      - Lots of state
    - Heterogeneous receivers
      - Some receivers 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
A “Digital Fountain” file broadcast protocol
For Next Time

- Media Access
- Read 2.6-2.7 in P&D