CSE 461 – Errors, part 2

David Wetherall
djw@cs.washington.edu
Reliability

• Two strategies to handle errors:
  – Detect and retransmit, or Automatic Repeat reQuest. (ARQ)
  – Error correcting codes, or Forward Error Correction (FEC)

• We’re seen error detecting/correcting codes, now look at retransmissions
Retransmissions, or more formally Automatic Repeat Request (ARQ)

- Sender automatically resends after a timeout until a positive acknowledgment (ACK) is obtained from the receiver
- Receiver automatically acknowledges frames (packets) that are not corrupted or lost in the network
- ARQ is generic name for protocols based on this strategy
Two issues

1. How long to set the timeout?
   - Only easy on a direct link, otherwise timing variability
   - Way too long lowers performance
   - Implies sometimes timeout will be early

2. How to avoid accepting duplicate frames as new
   - Given retransmissions, frame loss, and imprecise timeouts

• Note: we assume no message reordering for links, but allow message reordering for transport protocols
Timeouts

Retransmission timeout depends on round-trip time
- To send frame and receive an acknowledgement
- Need to account for variance on complex paths

**LAN case** – small, regular RTT

**Internet case** – large, varied RTT
Problem cases (due to loss, timeouts)

- In the case of ACK loss (or poor choice of timeout) the receiver can’t distinguish current message from next
The Need for Sequence Numbers

- Frame sequence numbers let receiver tell next frame from duplicate transmission
ACKs need sequence numbers too

• Hm, these things can be tricky!
Stop-and-Wait

- Only one outstanding frame at a time, 0 or 1.
- Retransmissions re-sent with same number.
- Number only needs to distinguish between current and next frame.
  - A single bit will do.
Limitation of Stop-and-Wait

- Lousy performance if wire time $<<$ prop. delay
  - How bad? You do the math
- Want to utilize all available bandwidth
  - Need to keep more data “in flight”
  - How much? Remember the bandwidth-delay product?
- Leads to Sliding Window Protocol
Solution: Allow Multiple Frames in Flight

• This is a form of pipelining
Sliding Window Protocol

- There is some maximum number of un-ACK’ed frames the sender is allowed to have in flight
  - We call this “the window size”
  - Example: window size = 2

Once the window is full, each ACK’ed frame allows the sender to send one more frame.
Sliding Window: Sender

- Assign sequence number to each frame (SeqNum)
- Maintain three state variables:
  - send window size (SWS)
  - last acknowledgment received (LAR)
  - last frame sent (LFS)
- Maintain invariant: LFS - LAR <= SWS

- Advance LAR when ACK arrives
- Buffer up to SWS frames
Sliding Window: Receiver

- Maintain three state variables
  - receive window size (\( RWS \))
  - largest frame acceptable (\( LFA \))
  - last frame received (\( LFR \))
- Maintain invariant: \( LFA - LFR \leq RWS \)

- Frame \( \text{SeqNum} \) arrives:
  - if \( LFR < \text{SeqNum} \leq LFA \) ⇒ accept else discard
  - send ACK to tell sender what has arrived (new or repeat)
- Advance \( LFR \) (and pass to application) as in-order frames arrive
- Need to buffer up to \( RWS \) frames
Acknowledgement options

• Different options are possible:

• Send cumulative ACKs – send ACK for largest frame such that all frames less than this have been received
  – Robust to ACK loss but not packet loss

• Send individual ACKs
  – Robust to packet loss but not ACK loss!

• Can combine:
  – Idea is to tell the sender what frames the receiver already has
  – Usually have cumulative ACK plus hints
Sliding Window Example

Sender

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Receiver

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

09/28/06 CSE461 06au
Sequence Number Space

- \texttt{SeqNum} field is finite; sequence numbers wrap around
- Sequence number space must be larger than the number of outstanding frames
- \texttt{SWS} \leq \texttt{MaxSeqNum}-1 is not sufficient
- \texttt{SWS} < \frac{\texttt{MaxSeqNum}+1}{2} is correct rule
- Intuitively, \texttt{SeqNum} “slides” between two halves of sequence number space
Sliding Window Summary

• It is perhaps the best known algorithm in networking

• First role is to enable reliable delivery of packets
  – Timeouts and acknowledgements
  – This has been our focus

• Second role is to enable in order delivery of packets
  – Receiver doesn’t pass data up to app until it has packets in order

• Third role is to enable flow control
  – We will see this when we get to TCP
  – Prevents fast sender from overflowing slow receiver’s buffer
When to use ARQ or FEC?

- Will depend on the kind of errors and cost of recovery
- Example: Message with 1000 bits, Prob(bit error) 0.001
  - Case 1: random errors
  - Case 2: bursts of 1000 errors

- Q: What to use in Case 1 and 2?
ARQ vs. FEC

- FEC used at low-level to lower residual error rate
- ARQ often used to fix large errors, e.g., packet collision, and with detection to protect against residual errors

- FEC sometimes used at high level too:
  - Real time applications (no time to retransmit!)
  - Nice interaction with broadcast (different receiver errors!)
Example: 802.11

- The standard scheme is:
  - PHY: FEC on data via interleaving and a binary convolutional code or LDPC
    - rates from $\frac{1}{2}$ to $\frac{5}{6}$.
  - PHY header has 16 bit CRC
  - Link: 32 bit CRC on frame and retransmission