CSE 222
Graduate Networking

Winter 2001

Lecture 4: Reliable Transmission

Geoffrey M. Voelker

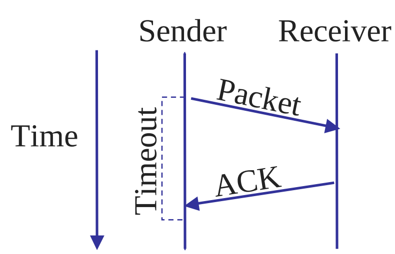
Overview

- Today: Reliable transmission
  - Stop and wait
  - Sliding window
- Next two lectures:
  - Congestion control, congestion avoidance, congestion signals, active queue management

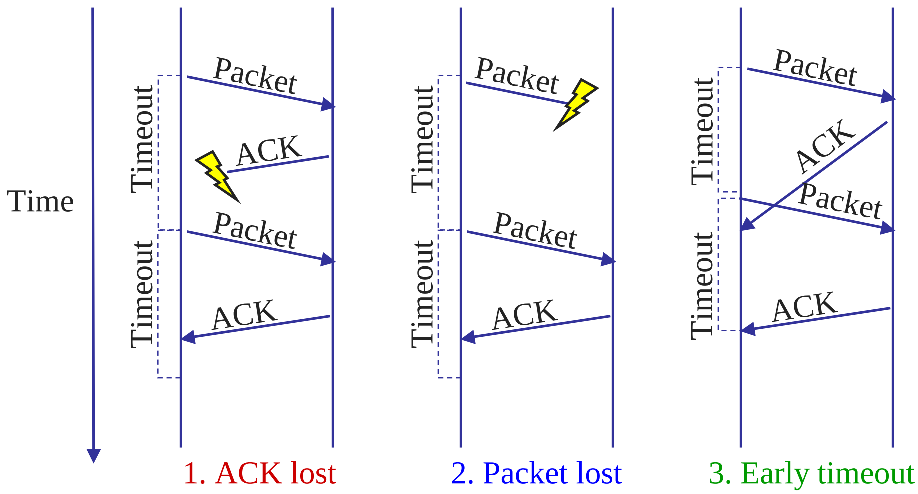
Reliable Transmission

- The Big Q: How do we send a packet reliably when it can be lost somewhere in the network?
- Mechansims
  - Acknowledgements
  - Timeouts
- Simplest reliable protocol: Stop and Wait
  - Send a packet
  - Stop and wait until an acknowledgement arrives from receiver
  - Retransmit if timeout occurs before ACK arrives

Stop and Wait



Recovering From Errors

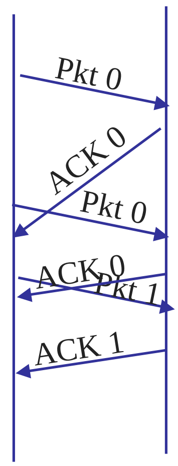


Problems with Stop and Wait

- How does the receiver recognize a duplicate transmission?
  - Solution: Put sequence number in packet
- Performance
  - No pipeline effect
  - For a network with bandwidth BW and delay D, a sender can transmit BW*D bytes before the network is "full"
  - BW*D is known as the bandwidth-delay product, the capacity of a network
  - Solution: Sliding window protocols

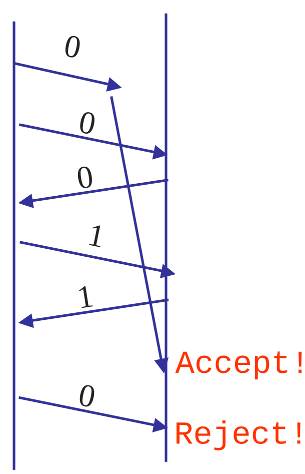
Sequence Numbers

- Detect retransmissions using sequence numbers
  - Both packets and ACKs
- Sequence # in packet is finite, though
- How many bits do we need?
  - One bit for stop and wait
  - Won't send seq #1 until receive ACK for seq #0



What if packets are delayed?

- Never reuse a seq #? Finite…
- Require in order delivery?
- Prevent very late delivery?
  - TTL: Decrement hop count per packet, discard if exceeded
  - Seq #s not reused within delay bound
  - TCP:
    - » Maximum segment lifetime (MSL)
    - » 120 seconds is recommended
    - » Trust implementations to obey

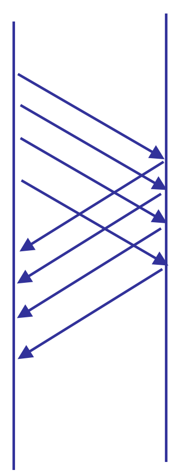


What happens on reboot?

- How do we distinguish packets sent before and after reboot?
  - Can't remember last sequence # used
- Solutions
  - Restart sequence # at 0?
  - Assume boot takes max packet delay?
  - Choose seq # at random and hope?
  - Use stable storage and increment high order bits of seq # on every boot

How do we keep the pipe full?

- Send multiple packets without waiting for the first to be ACKed
- Reliable, unordered delivery:
  - Send new packet after each ACK
  - Sender keeps list of unACK'ed packets and resends after timeout
  - Receiver same as stop & wait
- Prob: What if packet 2 keeps being lost?
  - Need to keep sender from getting ahead of lost packets



Sliding Window

- Receiver has to buffer packet (not pass it up to the application) until all prior packets have arrived
- Sender must prevent buffer overflow at receiver
- Use “sliding window”
  - Circular buffer at sender and receiver
  - # packets in transit <= buffer size
  - Advance window when sender and receiver agree packets at beginning of window have been received

Sender and Receiver State

- Sender
  - Packets send and ACKed (LAR = last ACK received)
  - Packets sent but not ACKed (buffer for retransmission)
  - Packets not yet sent (LFS = last frame sent)
- Receiver
  - Packets received and ACKed (NFE = next frame expected)
  - Packets received out of order (buffer until missing arrive)
  - Packets not yet received (LFA = last frame acceptable)

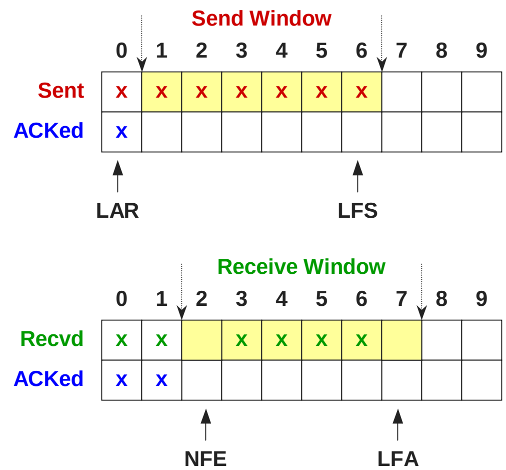
Sliding Window Example

LAR: Last Ack Received

LFS: Last Frame Sent

NFE: Next Frame Expected

LFA: Last Frame Accepted



What if we lose a packet?

- Go back N (TCP)
  - Receiver ACKs "got up through packet k"
    - » If multiple packets received, only one ACK needed
  - OK for receiver to buffer out of order packets
  - On timeout, sender restarts from k+1
- Selective acknowledgement (SACK)
  - Receiver sends ACK for each packet in window
  - On timeout, sender resends only the missing packet
  - Proposed for TCP

Sender Algorithm

- Send full window, set timeout
- On ACK:
  - If it increases LAR (packets sent and ACKed)
    - » Send remaining packets in window
- On timeout:
  - Resend packet LAR + 1 (first packet not yet ACKed)

Receiver Algorithm

- On packet arrival:
  - If packet is the NFE (next frame expected)
    - » Send ACK
    - » Increase NFE
    - » Deliver packets(s) to application (could fill in hole in buffer)
  - Else
    - » Send ACK
    - » Discard if < NFE (duplicate packet)

Can we shortcut the timeout?

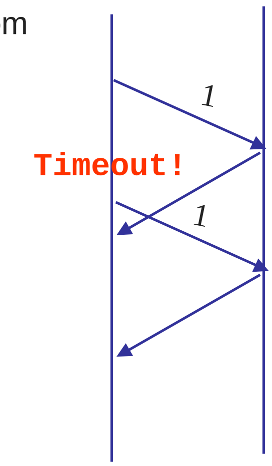
- Problem: If a packet is dropped in the network, the sender has to wait until timeout occurs before reacting
  - Waiting => pipe is not full
- If packets usually arrive in order, an out of order arrival signals a drop
  - Negative ACK (not used by TCP)
    - » Receiver requests missing packet
  - Fast retransmit
    - » Sender detects missing ACK
    - » TCP: When sender gets ACKs that don't advance NFE, resends missing packet

How do we determine timeouts?

- Round trip time (RTT) varies with congestion, route changes, etc.
- If timeout too small, useless retransmits
- If timeout too large, low utilization (pipe not full)
- TCP: Estimate RTT by timing ACKs (JK88)
  - Exponential weighted moving average
  - Account for variability in RTT

Retransmission Ambiguity

- How do we distinguish first ACK from retransmitted ACK?
  - First send to first ACK
    - » What if ACK dropped?
  - Last send to last ACK
    - » What if last ACK dropped?
- Might never be able to correct too short of a timeout!



Retransmission Ambiguity (2)

- TCP: Karn-Partridge
  - Ignore RTT estimates for retransmitted packets
  - Double timeout on every retransmission
- Add sequence #s to retransmissions
  - Retry #1, retry #2…
- TCP proposal: Timestamps
  - Add timestamps into packet header, ACK returns timestamp

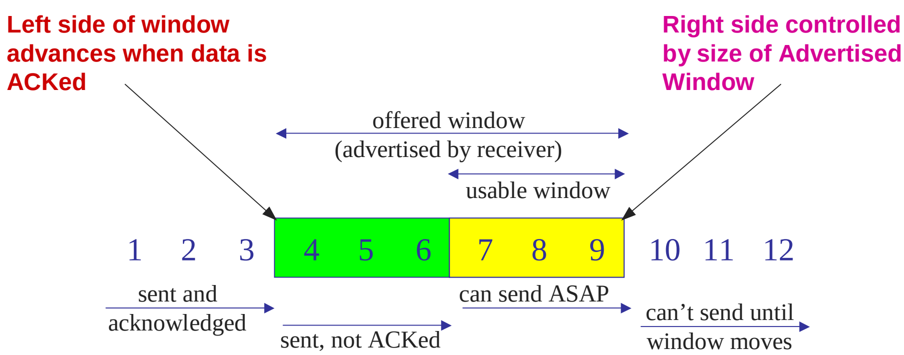
From Sliding Window to TCP

- TCP implements sliding window (reliable, in-order) with after-market customizations
  - Sequence #s:
    - » Count bytes rather than packets
    - » MSL of 120 seconds
  - Packet loss: Go back N
  - Timeouts:
    - » Function of RTT, double on retransmissions (Karn-Partridge)
    - » Fast retransmit when ACK does not advance NFE
- Flow control
- Congestion control (JK88)
  - This is different than flow control, often confused because implementation is intertwined

TCP Flow Control

- Sliding window provides basic flow control
  - Problems:
    - » Receiver sliding window could be smaller than sender's
    - » Sender could produce much faster than receiver consumes
- Only want sender to send as much data as the receiver can buffer
  - Solution: Have receiver tell sender size of free buffer space
  - Advertised Window: Available buffer space at receiver
  - Effective Window: Advertised Window – (LastByteSent – LastByteAcked)
  - Sender only sends up to Effective Window limit

Visualizing the TCP Window



Advertised Window

- If the receiving process does not empty the buffer (e.g., not scheduled), then the sender fills up the receiver's buffer
  - Advertised Window is 0
  - Effective Window goes to 0 when all data is ACKed
- Problem: When can the sender start sending again?
  - No timeouts because all data is ACKed
  - No packets from receiver with a new Advertised Window because receiver isn't running
- Solution: Ping with a segment of 1 byte of data
  - Eventually receiver responds with a new Advert. Window

For Next Time…

- Read 6.1 and 6.3
- Papers will be posted and announced via email…