Lecture 5: Flow Control

CSE 123: Computer Networks
Alex C. Snoeren

HW 1 due FRIDAY
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
Stop-and-Wait Performance

- Lousy performance if xmit 1 pkt $<$ propagation 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 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)

CSE 123 – Lecture 5: Flow Control
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”

CSE 123 – Lecture 5: Flow Control
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

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
Sliding Window – Receiver

- 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
Connection Establishment

- Both sender and receiver must be ready before we start to transfer the data
  - Sender and receiver need to agree on a set of parameters
  - Most important: sequence number space in each direction
  - Lots of other parameters: e.g., the window size

- Handshake protocols: setup state between two oblivious endpoints
  - Need to deal with delayed and reordered packets
Two-way handshake?

Active participant (client)

Passive participant (server)

SYN, SequenceNum = x

SYN, SequenceNum = y

What’s wrong here?

+data
Two-way handshake?

Active participant (client)  Passive participant (server)

Old SYN, SequenceNum = x
New SYN, SequenceNum = q
SYN, SequenceNum = y
+data

Delayed old SYN
Rejected
Three-Way Handshake

- Opens both directions for transfer

Active participant (client)

Passive participant (server)

SYN, SequenceNum = \( x \)

SYN + ACK, SequenceNum = \( y \),

ACK, Acknowledgment = \( x + 1 \)

Acknowledgment = \( y + 1 \)

+data
For Next Time

- Read 5-5.1 in P&D
- HW 1 due at the beginning of class Friday
- (Keep) going on the project…