Announcements

- Mid-semester grades
  - Only included midterm+HW1+HW2 (or 22.5% of class)
  - Project 1 grades available next Tuesday
  - If you got a D+, D, D- or F → must meet with Hui or me
  - 57.5% of class grade remains!

- HW 3 & Project 2
  - Will be posted on Thursday
  - Start early!

Midterm

- Average = 81.5 out of 110, Median = 80.5
- Standard Deviation = 14.7
- Max = 110, Min = 46

Feedback

- Likes:
  - 14: like project
  - 11: like lecture/some lecture
  - 6: like HW
- Dislikes:
  - 13: better project specs
    - Will address in project 2
  - 6: review session
    - Will have a formal review session
    - Please use office hours more
  - 5: more checkpoints/testing scripts
    - Part of the learning experience
  - 4: smaller projects
    - Only gets worse 😞
  - 4: flex days
    - Will address in project 2
Outline

- Transport introduction
- Error recovery
- TCP flow control

Transport Protocols

- Lowest level end-to-end protocol.
  - Header generated by sender is interpreted only by the destination
  - Routers view transport header as part of the payload

Functionality Split

- Network provides best-effort delivery
- End-systems implement many functions
  - Reliability
  - In-order delivery
  - Demultiplexing
  - Message boundaries
  - Connection abstraction
  - Congestion control
  - …

Transport Protocols

- UDP provides just integrity and demux
- TCP adds…
  - Connection-oriented
  - Reliable
  - Ordered
  - Point-to-point
  - Byte-stream
  - Full duplex
  - Flow and congestion controlled
UDP: User Datagram Protocol [RFC 768]

- "No frills," "bare bones" Internet transport protocol
- "Best effort" service, UDP segments may be:
  - Lost
  - Delivered out of order to app
- Connectionless:
  - No handshaking between UDP sender, receiver
  - Each UDP segment handled independently of others

Why is there a UDP?
- No connection establishment (which can add delay)
- Simple: no connection state at sender, receiver
- Small header
- No congestion control: UDP can blast away as fast as desired

UDP, cont.

- Often used for streaming multimedia apps
  - Loss tolerant
  - Rate sensitive
- Other UDP uses (why?):
  - DNS, SNMP
  - Reliable transfer over UDP
  - Must be at application layer
  - Application-specific error recovery

UDP Checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment — optional use!

Sender:
- Treat segment contents as sequence of 16-bit integers
- Checksum: addition (1's complement sum) of segment contents
- Sender puts checksum value into UDP checksum field

Receiver:
- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected
  - But maybe errors nonetheless?

High-Level TCP Characteristics

- Protocol implemented entirely at the ends
- Fate sharing
- Protocol has evolved over time and will continue to do so
  - Nearly impossible to change the header
  - Use options to add information to the header
  - Change processing at endpoints
  - Backward compatibility is what makes it TCP
TCP Header

Flags: SYN FIN RESET PUSH URG ACK

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td>Acknowledgement</td>
</tr>
<tr>
<td>HdrLen</td>
<td>Flags</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent pointer</td>
</tr>
<tr>
<td>Options (variable)</td>
<td>Data</td>
</tr>
</tbody>
</table>

Flags: SYN FIN RESETPUSHURGACK

Evolution of TCP

1975
Three-way handshake
Raymond Tomlinson
in SIGCOMM 75

1974
TCP described by
Vint Cerf and Bob Kahn
in IEEE Trans Comm

1974
TCP described by
Nagle’s algorithm
to reduce overhead
of small packets;
predicts congestion collapse

1982
BSD Unix 4.2
supports TCP/IP

1983
Congestion collapse observed

1984
Nagel’s algorithm
to better estimate
round-trip time

1986
Congestion collapse observed

1987
Karn’s algorithm
to better estimate
round-trip time

1988
4.3BSD Reno
fast retransmit
delayed ACK’s

1989
Van Jacobson’s
algorithms
congestion avoidance and congestion control
(most implemented in 4.2BSD Tahoe)

TCP Through the 1990s

1993
TCP Vegas
(Braden et al)
real congestion avoidance

1994
TCP (Braden)
Transaction

1994
T/TCP
(Shelat)
Transaction

1996
T/TCP
(Floyd et al)
Selective
Acknowledgement

1996
FACK TCP
(Mathis et al)
extension to SACK

Outline

- Transport introduction
- Error recovery & flow control
- TCP flow control
Stop and Wait

- ARQ
  - Receiver sends acknowledgement (ACK) when it receives packet
  - Sender waits for ACK and timeouts if it does not arrive within some time period
- Simplest ARQ protocol
- Send a packet, stop and wait until ACK arrives

Recovering from Error

- How to recognize a duplicate
- Performance
  - Can only send one packet per round trip

Problems with Stop and Wait

- Use sequence numbers
  - both packets and acks
  - Sequence # in packet is finite
  - How big should it be?
    - For stop and wait?
  - One bit – won’t send seq #1 until received ACK for seq #0

How to Recognize Resends?
How to Keep the Pipe Full?

- Send multiple packets without waiting for first to be acked
  - Number of pkts in flight = window
- Reliable, unordered delivery
  - Several parallel stop & waits
  - Send new packet after each ack
  - Sender keeps list of unack’ed packets; resends after timeout
  - Receiver same as stop & wait
- How large a window is needed?
  - Suppose 10Mbps link, 4ms delay, 500byte pkts
    - 1? 10? 20?
  - Round trip delay * bandwidth = capacity of pipe

Sliding Window

- Reliable, ordered delivery
- Receiver has to hold onto a packet until all prior packets have arrived
  - Why might this be difficult for just parallel stop & wait?
  - Sender must prevent buffer overflow at receiver
- Circular buffer at sender and receiver
  - Packets in transit ≤ buffer size
  - Advance when sender and receiver agree packets at beginning have been received

Sender/Receiver State

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max ACK received Next seqnum</td>
<td>Next expected Max acceptable</td>
</tr>
<tr>
<td>Sender window</td>
<td>Receiver window</td>
</tr>
<tr>
<td>Sent &amp; Acked</td>
<td>Received &amp; Acked</td>
</tr>
<tr>
<td>Sent Not Acked</td>
<td>Acceptable Packet</td>
</tr>
<tr>
<td>OK to Send</td>
<td>Not Usable</td>
</tr>
</tbody>
</table>

Window Sliding – Common Case

- On reception of new ACK (i.e. ACK for something that was not acked earlier)
  - Increase sequence of max ACK received
  - Send next packet
- On reception of new in-order data packet (next expected)
  - Hand packet to application
  - Send cumulative ACK – acknowledges reception of all packets up to sequence number
  - Increase sequence of max acceptable packet
Loss Recovery

- On reception of out-of-order packet
  - Send nothing (wait for source to timeout)
  - Cumulative ACK (helps source identify loss)
- Timeout (Go-Back-N recovery)
  - Set timer upon transmission of packet
  - Retransmit all unacknowledged packets
- Performance during loss recovery
  - No longer have an entire window in transit
  - Can have much more clever loss recovery

Selective Repeat

- Receiver individually acknowledges all correctly received pkts
  - Buffers packets, as needed, for eventual in-order delivery to upper layer
  - Sender only resends packets for which ACK not received
    - Sender timer for each unACKed packet
- Sender window
  - N consecutive seq #'s
  - Again limits seq #s of sent, unACKed packets

Go-Back-N in Action

Selective Repeat: Sender, Receiver Windows
Sequence Numbers

- How large do sequence numbers need to be?
  - Must be able to detect wrap-around
  - Depends on sender/receiver window size
- E.g.
  - Max seq = 7, send win=recv win=7
  - If pkts 0..6 are sent successfully and all acks lost
    - Receiver expects 7,0..5, sender retransmits old 0..6!!!
  - Max sequence must be $\geq$ send window + recv window

Important Lessons

- Transport service
  - UDP $\rightarrow$ mostly just IP service
  - TCP $\rightarrow$ congestion controlled, reliable, byte stream
- Types of ARQ protocols
  - Stop-and-wait $\rightarrow$ slow, simple
  - Go-back-n $\rightarrow$ can keep link utilized (except w/ losses)
  - Selective repeat $\rightarrow$ efficient loss recovery
- Sliding window flow control
  - Addresses buffering issues and keeps link utilized

Next Lecture

- TCP connection setup
- TCP reliability
- Congestion control
- Project 1 & 2