Announcements

- Mid-semester grades
  - Based on (ckpt 1 & ckpt2) + midterm + HW1 + HW2
  - NOTE: GRADES DO NOT REFLECT LATE PENALTIES!
  - 25.4% of class
  - If you got a D+, D, or F → must meet with Roger or me

- Worry but don’t panic
  - 74.6% of class grade remains!
  - Grading is likely a bit harsher than end-of-semester

- Project 1 grades – coming soon – lots of handin issues

Midterm

- Avg: 62
- Med: 67
- STD: 13.5
- Max: 79
- Min: 28

What was hard...

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect</td>
<td>15</td>
<td>24</td>
<td>9</td>
<td>14</td>
<td>17</td>
</tr>
<tr>
<td>Average</td>
<td>12.8</td>
<td>18.4</td>
<td>6.3</td>
<td>11</td>
<td>12.5</td>
</tr>
</tbody>
</table>

Crypto  Bridging & Routing
Feedback (positive)

**Likes**
- 12 project
- 8 recitations
- 7 HW
- 6 TA
- 2 IRC
- Slides
- Jokes!

**More**
- 2 lab/tool assignments
- 2 examples

Feedback (negative)

**Project/HW**
- 10 release all at start/ clarity/changes
- 2 Due date conflicts
- Smaller HWs
- HW clarity
- Project open-ended
- Recitations run over time

**Lectures**
- 4 Integration with project/HW
- 3 not enough detail/too much material
- Need less repetition
- First HW date vs. lectures

**Other**
- 2 Study guide for exam/practice
- 2 3-free points
- more office hours
- Use bboard more

Outline

- Transport introduction
- Error recovery & 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 segment format

<table>
<thead>
<tr>
<th>Source port #</th>
<th>Dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Checksum</td>
</tr>
<tr>
<td>Length, in bytes of UDP segment, including header</td>
<td></td>
</tr>
<tr>
<td>Application data (message)</td>
<td></td>
</tr>
<tr>
<td>UDP segment format</td>
<td></td>
</tr>
</tbody>
</table>
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 (and parts of IP header)
- 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
    - These do change sometimes
    - Change processing at endpoints
  - Backward compatibility is what makes it TCP

TCP Header

<table>
<thead>
<tr>
<th>Flags:</th>
<th>SYN</th>
<th>FIN</th>
<th>RESET</th>
<th>PUSH</th>
<th>URG</th>
<th>ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source port</td>
<td>Destination port</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence number</td>
<td>Acknowledgment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HdrLen</td>
<td>Flags</td>
<td>Advertised window</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent pointer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (variable)</td>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Evolution of TCP

- 1975: Three-way handshake by Raymond Tomlinson in SIGCOMM ’75
- 1974: TCP described by Vint Cerf and Bob Kahn
- 1983: BSD Unix 4.2 supports TCP/IP
- 1982: TCP & IP in 4.2BSD, later in 4.3BSD
- 1986: Karn’s algorithm observed
- 1988: Nagle’s algorithm to reduce overhead of small packets; predicts congestion collapse
- 1987: Congestion collapse observed
- 1990: Van Jacobson’s algorithms congestion avoidance and congestion control (most implemented in 4.3BSD Tahoe)
TCP Through the 1990s

1993
TCP Vegas (Brakmo et al)
- delay-based congestion avoidance

1994
T/TCP (Braden)
- Transaction TCP

1994
ECN (Floyd)
- Explicit Congestion Notification

1994
Mee
- New Reno startup and loss recovery

1996
SACK TCP (Floyd et al)
- Selective Acknowledgment

1996
FACK TCP (Mathis et al)
- Extension to SACK

Outline

- Transport introduction
- Error recovery & 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

- ACK lost
- Packet lost
- Early timeout
- DUPLICATE PACKETS!!!
Problems with Stop and Wait

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

How to Recognize Resends?

- 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 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, 500 byte 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
**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

**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
Go-Back-N in Action

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

Selective Repeat: Sender, Receiver Windows

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

• Congestion control
• TCP Reliability