Announcements

- Mid-semester grades
  - Based on project1 + midterm + HW1 + HW2
  - 42.5% of class
  - If you got a D+, D, D- or F → must meet with Dave or me
  - 57.5% of class grade remains!

Feedback (positive)

- Likes:
  - 12: lectures (or some aspect of lectures)
  - 8: project
  - 4: recitations
  - 3: HW
  - 2: cookies

- More:
  - 3: more practical apps/tools (IRC, IPTV, BitTorrent, P2P, ethereal)
  - 3: more examples/more animations/more details
  - 3: more overview/summaries
  - 2: more project advice

Feedback (negative)

- Project/HW
  - 5: project writeup
  - 5: need more/more complex checkpoints/need at beginning
  - 4: HW tedious/poorly written
  - 2: project too hard
  - HW not covered in lecture
  - HW makeup grade

- Lectures
  - 3: define terms/acronym/memorization hell
  - 3: textbook bad/relationship to lectures unclear
  - lecture relationship to the book
  - Srini’s lectures are slow-paced

- Interaction
  - email vs. bboard
  - more direct answers on bboard
  - want review session for exam
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 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

<table>
<thead>
<tr>
<th>Flags</th>
<th>Source port</th>
<th>Destination port</th>
<th>Sequence number</th>
<th>Acknowledgement</th>
<th>Advertised window</th>
<th>HdrLen</th>
<th>Flags</th>
<th>Options (variable)</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>URG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Flags: SYN, FIN, RESET, PUSH, URG, ACK

TCP Through the 1990s

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

1994
- ECN (Floyd)
- Explicit Congestion Notification

1996
- New Reno startup and loss recovery

1998
- FACK TCP (Mathis et al.)
- extension to SACK

Outline

- Transport introduction
- Error recovery & flow control

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
- Nagle’s algorithm to reduce overhead of small packets, predict congestion collapse

1982
- 4.2BSD Unix supports TCP/IP

1983
- Congestion collapse observed

1984
- Nagel’s algorithm to reduce overhead of small packets, predict congestion collapse

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

1985
- Karn’s algorithm to better estimate round-trip time

1986
- Congestion collapse observed

1986
- Van Jacobson’s algorithms congestion avoidance and congestion control (most implemented in 4.3BSD Tahoe)

1987
- 4.3BSD Reno fast retransmit delayed ACK

1990
- 4.3BSD Reno fast retransmit delayed ACK

TCP Through the 1990s

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

1994
- T/TCP (Braden)
- Transaction TCP

1994
- SACK TCP (Floyd et al.)
- Selective Acknowledgement

1995
- ECN (Floyd)
- Explicit Congestion Notification

1996
- New Reno startup and loss recovery

1998
- FACK TCP (Mathis et al.)
- extension to SACK
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

- 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

- Recovering from Error
  - Packet lost
  - Early timeout
  - Duplicate packets!!!
How to Keep the Pipe Full?

- Send multiple packets without waiting for first to be acknowledged
  - Number of packets in flight = window
- Reliable, unordered delivery
  - Several parallel stop & waits
  - Send new packet after each acknowledgment
  - Sender keeps list of unacknowledged packets; resends after timeout
  - Receiver same as stop & wait
- How large a window is needed?
  - Suppose 10Mbps link, 4ms delay, 500-byte packets
    - Round trip delay * bandwidth = capacity of pipe
    - 10 Mbps * 4 ms = 40 packets

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</td>
<td>Next expected</td>
</tr>
<tr>
<td>Next seqnum</td>
<td>Max acceptable</td>
</tr>
</tbody>
</table>

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 packets 0..6 are sent successfully and all acknowledgments are lost
    - Receiver expects 7,0..5, sender retransmits 0..6!!!
- Max sequence must be ≥ 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 → mostly just IP service
  - TCP → congestion controlled, reliable, byte stream

- Types of ARQ protocols
  - Stop-and-wait → slow, simple
  - Go-back-n → can keep link utilized (except w/ losses)
  - Selective repeat → efficient loss recovery

- Sliding window flow control
  - Addresses buffering issues and keeps link utilized

Next Lecture

- Congestion control
- TCP Reliability

EXTRA SLIDES

The rest of the slides are FYI
Ponder This…

• A bus station is where a bus stops.
• A train station is where a train stops.
• A work station is where…

• Maybe that explains why it was so hard getting project 1 done 😊 …. ouch