Announcements

• HW3 due today
• Start working on HW4
• HW5 posted
• In-class student presentations
• No TA office hours this week
  – Makeup hours next week
Today’s Topics

• Transport Protocols
  – UDP
  – TCP
  – EWMA
Transport Layer

- Transport protocols sit on top of the network layer (IP)
- Can provide:
  - Application-level multiplexing ("ports")
  - Error detection, reliability, etc.
Error Detection

• Idea: add redundant information to catch errors in packet

• Three examples:
  – Parity
  – Internet Checksum
  – CRC
Reliable Delivery

• Error detection can discard bad packets
• Problem: if bad packets are lost, how can we ensure reliable delivery?
  – Exactly-once semantics = at least once + at most once
At Least Once Semantics

• How can the sender know packet arrived at least once?
  – Acknowledgments + Timeout

• Stop and Wait Protocol
  – S: Send packet, wait
  – R: Receive packet, send ACK
  – S: Receive ACK, send next packet
  – S: No ACK, timeout and retransmit
Stop and Wait Problems

• Duplicate data
• Duplicate acks
• Can’t fill pipe
• Difficult to set the timeout value
Sliding Window Protocol

- Still have the problem of keeping pipe full
  - Generalize approach with > 1-bit counter
  - Allow multiple outstanding (unACKed) frames
  - Upper bound on unACKed frames, called *window*
UDP – User Datagram Protocol

• Unreliable, unordered datagram service
• Adds multiplexing, checksum
• End points identified by *ports*
  – Scope is an IP address (interface)
• Checksum aids in error detection

UDP Checksum

• Uses the same algorithm as the IP checksum
  – Set Checksum field to 0
  – Sum all 16-bit words, adding any carry bits to the LSB
  – Flip bits to get checksum (except 0xffff->0xffff)
  – To check: sum whole packet, including sum, should get 0xffff

• How many errors?
  – Catches any 1-bit error
  – Not all 2-bit errors

• Optional in IPv4: not checked if value is 0
Pseudo Header

0 7 8 15 16 23 24 31
+------------------------+--------+
| source address         |        |
+------------------------+--------+
| destination address    |        |
+------------------------+--------+
| zero  | protocol| UDP length |
+------------------------+--------+

• UDP Checksum is computer over pseudo-header prepended to the UDP header
  – For IPv4: IP Source, IP Dest, Protocol (=17), plus UDP length

• Benefits? Problem?
  – Is UDP a layer on top of IP?

http://www.postel.org/pipermail/end2end-interest/2005-February/004616.html
Next Problem: Reliability

<table>
<thead>
<tr>
<th>Problem</th>
<th>Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dropped Packets</td>
<td>Acknowledgments + Timeout</td>
</tr>
<tr>
<td>Duplicate Packets</td>
<td>Sequence Numbers</td>
</tr>
<tr>
<td>Packets out of order</td>
<td>Receiver Window</td>
</tr>
<tr>
<td>Keeping the pipe full</td>
<td>Sliding Window (Pipelining)</td>
</tr>
</tbody>
</table>
Transport Layer Reliability

• Extra difficulties
  – Multiple hosts
  – Multiple hops
  – Multiple potential paths

• Need for connection establishment, tear down
  – Analogy: dialing a number versus a direct line

• Varying RTTs
  – Both across connections and during a connection
  – Why do they vary? What do they influence?
Extra Difficulties (cont.)

• Out of order packets
  – Not only because of drops/retransmissions
  – Can get very old packets (up to 120s), must not get confused

• Unknown resources at other end
  – Must be able to discover receiver buffer: flow control

• Unknown resources in the network
  – Should not overload the network
  – But should use as much as safely possible
  – Congestion Control (next class)
TCP – Transmission Control Protocol

- **Service model:** “reliable, connection oriented, full duplex byte stream”
  - Endpoints: <IP Address, Port>
- **Flow control**
  - If one end stops reading, writes at other eventually stop/fail
- **Congestion control**
  - Keeps sender from overloading the network (next lecture)
TCP

- Specification
  - RFC 793 (1981), RFC 1222 (1989, some corrections), RFC 5681 (2009, congestion control), ...
- Was born coupled with IP, later factored out
  - We talked about this, don’t always need everything!
- End-to-end protocol
  - Minimal assumptions on the network
  - All mechanisms run on the end points
- Alternative idea:
  - Provide reliability, flow control, etc, link-by-link
  - Does it work?
### TCP Header

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- **Source Port** | **Destination Port**
- **Sequence Number**
- **Acknowledgment Number**
- **Data** | **Reserved**
- **Offset** | **Reserved**
- **Checksum**
- **Options**
- **Padding**
Header Fields

• Ports: multiplexing
• Sequence number
  – Correspond to \textit{bytes}, not packets!
• Acknowledgment Number
  – Next expected sequence number
• Window: willing to receive
  – Lets receiver limit SWS (even to 0) for flow control
• Data Offset: \# of 4 byte header + option bytes
• Flags, Checksum, Urgent Pointer
Header Flags

- **URG:** whether there is urgent data
- **ACK:** ack no. valid (all but first segment)
- **PSH:** push data to the application immediately
- **RST:** reset connection
- **SYN:** synchronize, establishes connection
- **FIN:** close connection
Establishing a Connection

- **Three-way handshake**
  - Two sides agree on respective initial sequence nums
- If no one is listening on port: server sends RST
- If server is overloaded: ignore SYN
- If no SYN-ACK: retry, timeout
Connection Termination

• FIN bit says no more data to send
  – Caused by close or shutdown
  – Both sides must send FIN to close a connection

• Typical close
Summary of TCP States

- **Unsynchronized**
- **Synchronized**

**Connection Establishment**
- **Active open/SYN**
- **Passive open**
- **Close**

**Passive close:**
- Can still send!

**Active close:**
- Can still receive

**Timeout after two segment lifetimes**
TIME_WAIT

• Why do you have to wait for 2MSL in TIME_WAIT?
  – What if last ack is severely delayed, AND
  – Same port pair is immediately reused for a new connection?
• Solution: active closer goes into TIME_WAIT
  – Waits for 2MSL (Maximum Segment Lifetime)
• Can be problematic for active servers
  – OS has too many sockets in TIME_WAIT, can accept less connections
    • Hack: send RST and delete socket, SO_LINGER = 0
  – OS won’t let you re-start server because port in use
    • SO_REUSEADDR lets you rebind
Reliable Delivery

• TCP retransmits if data corrupted or dropped
  – Also retransmit if ACK lost

• When should TCP retransmit?

• Challenges in estimating RTT
  – Dynamic
  – No additional traffic
Smoothing RTT

• RTT measurement can have large variation
• Need to smooth the samples
  – One RTT measurement = one sample
• Some ways to smooth the sample
  – Average of the whole sequence
  – Windowed Mean
• Problems?
EWMA

• EWMA: Exponentially Weighted Moving Average
• Give greater weight to recent samples
  – Why?

http://en.wikipedia.org/wiki/Moving_average
EWMA

• Estimate RTT
• $RTT(t) = \alpha \times RTT(t-1) + (1-\alpha) \times \text{newEst}$

$\alpha = 0.8$

<table>
<thead>
<tr>
<th>Time</th>
<th>RTT</th>
<th>newEst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>10</td>
</tr>
</tbody>
</table>