Project 1a due MONDAY
Overview

- Transport layer tasks
  - Process naming/demultiplexing

- User Datagram Protocol (UDP)

- Transport Control Protocol (TCP)
  - State transitions
  - Connection teardown

How was the homework?
A. Trivial
B. Doable with provided materials
C. I had no idea how to answer some parts
D. Homework? There was a homework?
Transport Layer Tasks

- Define and provide specific delivery semantics
  - To which end point?
  - When?
  - How?
  - If?

- Multiplexing

- Reliability
Naming Processes/Services

- Process here is an abstract term for your Web browser (HTTP), Email servers (SMTP), hostname translation (DNS)

- How do we identify for remote communication?
  - Process id or memory address are OS-specific and transient

- So TCP and UDP use **ports**
  - 16-bit integers representing mailboxes that processes “rent”
  - Identify process uniquely as (IP address, protocol, port)
Picking Port Numbers

- We still have the problem of allocating port numbers
  - What port should a Web server use on host X?
  - To what port should you send to contact that Web server?

- Servers typically bind to well-known port numbers
  - e.g., HTTP 80, SMTP 25, DNS 53, … look in /etc/services
  - Ports below 1024 traditionally reserved for well-known services

- Clients use OS-assigned temporary (ephemeral) ports
  - Above 1024, recycled by OS when client finished
User Datagram Protocol (UDP)

- Provides *unreliable message delivery* between processes
  - Source port filled in by OS as message is sent
  - Destination port identifies UDP delivery queue at endpoint
- Connectionless (no state about who talks to whom)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DstPort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CSE 123 – Lecture 6: Transport Layer Protocols
UDP Delivery

Packets arrive

Ports

Message Queues

Application process

Application process

Application process

DeMux

Packets arrive

Kernel boundary

CSE 123 – Lecture 6: Transport Layer Protocols
UDP Checksum

- UDP includes optional protection against errors
  - Checksum intended as an end-to-end check on delivery
  - So it covers data, UDP header, and IP pseudoheader (history)
Applications for UDP

- Streaming media (e.g., live video)
- DNS (Domain Name Service)
- NTP (Network Time Protocol) (synchronizing clocks)
- FPS multi-player video games (e.g., Call of Duty)
- Why might UDP be appropriate for these?
Transmission Control Protocol

- Reliable bi-directional byte stream between processes
  - Uses a sliding window protocol for efficient transfer

- Connection-oriented
  - Conversation between two endpoints with beginning and end

- Flow control (receive window)
  - Prevents sender from over-running receiver buffers
  - (tell sender how much buffer is left at receiver)

- Congestion control (later in term)
  - Prevents sender from over-running network capacity
What corresponds to the receive window size?

A. Segment size
B. Bandwidth*delay product
C. Send buffer space
D. Receive buffer space
TCP Header Format

- Ports plus IP addresses identify a connection (**4-tuple**)

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td>0-2</td>
</tr>
<tr>
<td>DstPort</td>
<td>3-31</td>
</tr>
<tr>
<td>HdrLen</td>
<td>0</td>
</tr>
<tr>
<td>Flags</td>
<td>4-7</td>
</tr>
<tr>
<td>AdvertisedWindow</td>
<td>8-15</td>
</tr>
<tr>
<td>Checksum</td>
<td>16-23</td>
</tr>
<tr>
<td>UrgPtr</td>
<td>24-27</td>
</tr>
<tr>
<td>Options (variable)</td>
<td>28-31</td>
</tr>
<tr>
<td>Data</td>
<td>0-31</td>
</tr>
</tbody>
</table>

CSE 123 – Lecture 6: Transport Layer Protocols
TCP Header Format

- Sequence, Ack numbers used for the sliding window
  - How big a window? Flow control/congestion control determine

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>10</td>
<td>16</td>
<td>31</td>
</tr>
<tr>
<td>SrcPort</td>
<td>DstPort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SequenceNum</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HdrLen</td>
<td>0</td>
<td>Flags</td>
<td>AdvertisedWindow</td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td>UrgPtr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
TCP Header Format

- Flags may be ACK, SYN, FIN, URG, PSH, RST

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Field</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SrcPort</td>
<td>Source Port</td>
</tr>
<tr>
<td>4</td>
<td>DstPort</td>
<td>Destination Port</td>
</tr>
<tr>
<td>10</td>
<td>SequenceNum</td>
<td>Sequence Number</td>
</tr>
<tr>
<td>16</td>
<td>Acknowledgment</td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>31</td>
<td>Flags</td>
<td>ACK, SYN, FIN, URG, PSH, RST</td>
</tr>
<tr>
<td>4</td>
<td>AdvertisedWindow</td>
<td>Advertised Window</td>
</tr>
<tr>
<td>8</td>
<td>UrgPtr</td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>16</td>
<td>Checksum</td>
<td>Checksum</td>
</tr>
<tr>
<td>32</td>
<td>Options (variable)</td>
<td>Options (variable)</td>
</tr>
<tr>
<td>40</td>
<td>Data</td>
<td>Data</td>
</tr>
</tbody>
</table>

CSE 123 – Lecture 6: Transport Layer Protocols
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

+data

What's wrong here?
Two-way handshake?

Active participant (client)

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

Passive participant (server)

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,
Acknowledgment = x + 1

ACK, Acknowledgment = y + 1

+data
For next time…

- Begin to discuss actual *networks*!
- Read Ch 3-3.1 in P&D
- NO CLASS MONDAY (Enjoy MLK Day)
- Project 1b introduced in Discussion TUESDAY