Lecture 10: Transport Layer Protocols
Lecture 10 Overview

- Process naming/demultiplexing

- User Datagram Protocol (UDP)

- Transport Control Protocol (TCP)
  - Three-way handshake
  - Flow control
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)

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

Packets arrive

Ports

Message Queues

Application process

Application process

Application process

DeMux

Kernel boundary

CSE 123 – Lecture 10: Transport Layer
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

![UDP Checksum Diagram]
Applications for UDP

- Streaming media
- DNS (Domain Name Service)
- NTP (Network Time Protocol)
- Why is UDP appropriate for these?
Transmission Control Protocol

- Reliable bi-directional **bytestream** between processes
  - Uses a sliding window protocol for efficient transfer

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

- Flow control
  - Prevents sender from over-running receiver buffers

- Congestion control (next class)
  - Prevents sender from over-running network capacity
TCP Delivery

Application process

TCP
Send buffer

Write bytes

TCP
Receive buffer

Read bytes

Transmit segments
Segment  Segment  ...  Segment
## TCP Header Format

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td>0-3</td>
</tr>
<tr>
<td>DstPort</td>
<td>4-7</td>
</tr>
<tr>
<td>HdrLen</td>
<td>8</td>
</tr>
<tr>
<td>SequenceNum</td>
<td>9-12</td>
</tr>
<tr>
<td>Acknowledgment</td>
<td>13-16</td>
</tr>
<tr>
<td>Flags</td>
<td>17-19</td>
</tr>
<tr>
<td>AdvertisedWindow</td>
<td>20-23</td>
</tr>
<tr>
<td>Checksum</td>
<td>24-27</td>
</tr>
<tr>
<td>UrgPtr</td>
<td>28-31</td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>
**TCP Header Format**

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

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

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

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td>DstPort</td>
<td>SequenceNum</td>
<td>Acknowledgment</td>
<td>AdvertisedWindow</td>
</tr>
<tr>
<td>HdrLen</td>
<td>Flags</td>
<td>Checksum</td>
<td>UrgPtr</td>
<td>Options (variable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data</td>
</tr>
</tbody>
</table>
```

CSE 123 – Lecture 10: Transport Layer
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 Maximum Segment Size

- Handshake protocols: setup state between two oblivious endpoints
  - Didn’t need it earlier because link had only two end points
  - 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

Passive participant (server)

Delayed old SYN

Rejected

+data

CSE 123 – Lecture 10: Transport Layer
Three-Way Handshake

- Opens both directions for transfer

Active participant (client)  
SYN, SequenceNum = x
SYN + ACK, SequenceNum = y,
ACK, Acknowledgment = x + 1
+data

Passive participant (server)  
Acknowledgment = y + 1
Some Comments

- We could abbreviate this setup, but it was chosen to be robust, especially against delayed duplicates
  - Three-way handshake from Tomlinson 1975

- Choice of changing initial sequence numbers (ISNs) minimizes the chance of hosts that crash getting confused by a previous incarnation of a connection

- How to choose ISNs?
  - Maximize period between reuse
  - Minimize ability to guess (why?)
TCP State Transitions

CLOSED

LISTEN

SYN_RCVD

SYN_SENT

SYN/SYN + ACK

SYN + ACK/ACK

ESTABLISHED

FIN_WAIT_1

FIN_WAIT_2

CLOSING

TIME_WAIT

CLOSE_WAIT

LAST_ACK

CLOSED

Passive open

Close

Active open/SYN

Send/ SYN

Timeout after two segment lifetimes

CSE 123 – Lecture 10: Transport Layer
Again, with States

Active participant
(client)

Passive participant
(server)

SYN_SENT

SYN, SequenceNum = x

SYN + ACK, SequenceNum = y,
Acknowledgment = x + 1

ACK, Acknowledgment = y + 1

+data

ESTABLISHED

LISTEN

SYN_RCVD

ESTABLISHED

CSE 123 – Lecture 10: Transport Layer
Connection Teardown

- Orderly release by sender and receiver when done
  - Delivers all pending data and “hangs up”

- Cleans up state in sender and receiver

- TCP provides a “symmetric” close
  - Both sides shutdown independently
TCP Connection Teardown

Web server

FIN_WAIT_1
FIN_WAIT_2
TIME_WAIT
CLOSED

Web browser

FIN
ACK
FIN
ACK
CLOSE_WAIT
LAST_ACK
CLOSED

CSE 123 – Lecture 10: Transport Layer
We wait 2*MSL (maximum segment lifetime of 60 seconds) before completing the close
  ◆ Why?

ACK might have been lost and so FIN will be resent
  ◆ Could interfere with a subsequent connection

Real life: Abortive close
  ◆ Don’t wait for 2*MSL, simply send Reset packet (RST)
  ◆ Why?
Flow Control

- Sender must transmit data no faster than it can be consumed by the receiver
  - Receiver might be a slow machine
  - App might consume data slowly

- TCP adjusts the size of the sliding window
  - This is the purpose of the Advertised Window field
TCP Header Format

- Advertised window is used for flow control

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td>0-10</td>
</tr>
<tr>
<td>DstPort</td>
<td>11-20</td>
</tr>
<tr>
<td>SequenceNum</td>
<td>21-30</td>
</tr>
<tr>
<td>Acknowledgment</td>
<td>31-31</td>
</tr>
<tr>
<td>Flags</td>
<td>32</td>
</tr>
<tr>
<td>HdrLen</td>
<td>33</td>
</tr>
<tr>
<td>Checksum</td>
<td>34</td>
</tr>
<tr>
<td>UrgPtr</td>
<td>35</td>
</tr>
<tr>
<td>AdvertisedWindow</td>
<td>36</td>
</tr>
<tr>
<td>Options (variable)</td>
<td>37</td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>
Sender and Receiver Buffering

Sending application

TCP

LastByteWritten

LastByteAcked

LastByteSent

Receiving application

TCP

LastByteRead

NextByteExpected

LastByteRcvd

■ = available buffer

□ = buffer in use

CSE 123 – Lecture 10: Transport Layer
Window-Size Example

Receiver has buffer of size 4 and application doesn’t read

Stall due to flow control here

T=1
SEQ=1
ACK=2; WIN=3

T=2
SEQ=2
ACK=3; WIN=2

T=3
SEQ=3

T=4
SEQ=4

T=5
ACK=4; WIN=1

T=6
ACK=5; WIN=0

CSE 123 – Lecture 10: Transport Layer
Example – Buffer at Sender

<table>
<thead>
<tr>
<th>T</th>
<th>1 2 3 4</th>
<th>5 6 7 8 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>T=1</td>
<td>1 2 3 4</td>
<td>5 6 7 8 9</td>
</tr>
<tr>
<td>T=2</td>
<td>1 2 3 4</td>
<td>5 6 7 8 9</td>
</tr>
<tr>
<td>T=3</td>
<td>1 2 3 4</td>
<td>5 6 7 8 9</td>
</tr>
<tr>
<td>T=4</td>
<td>1 2 3 4</td>
<td>5 6 7 8 9</td>
</tr>
<tr>
<td>T=5</td>
<td>1 2 3 4</td>
<td>5 6 7 8 9</td>
</tr>
<tr>
<td>T=6</td>
<td>1 2 3 4</td>
<td>5 6 7 8 9</td>
</tr>
</tbody>
</table>

- • =acked
- • =sent
- • =advertised
Lots of Icky Details

- Window probes
- Silly Window Syndrome
- Nagle’s algorithm
- PAWS
- Etc…

- Steven’s books “TCP/IP Illustrated (vol 1,2)” is a great source of information on this
TCP applications

- HTTP/WWW
- FTP
- SMTP, POP, IMAP (E-mail)

- Why is TCP well suited to these applications?
Summary

- Transport layer provides demultiplexing
- Different protocols provide various services
  - UDP provides unreliable datagram delivery
  - TCP delivers reliable, in-order bytestreams
- Connection setup/teardown
- Flow control
  - Adjust sliding window to manage receiver buffer
For next time...

- Read Ch 6.3-4 in P&D

- Can still turn in Project 1 for next few days
  - One letter grade penalty per day