Lecture 14: Transport Layer Protocols
Announcements

- Homework #2 and Midterms to be returned at end of class
  - Midterm mean 51, median 52

- Project #2
  - Up now, due Dec 4th at noon
  - SRMP: Build a sliding window protocol with flow control
  - We provide framework for you to plug into
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)

```
<table>
<thead>
<tr>
<th></th>
<th>SrcPort</th>
<th>DstPort</th>
<th>Checksum</th>
<th>Length</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>6</td>
<td>14</td>
<td>31</td>
<td></td>
</tr>
</tbody>
</table>
```
UDP Delivery

Packets arrive

Ports

Message Queues

DeMux

Application process

Application process

Application process

Kernel boundary
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

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

```
01 6 3 1
```
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

Write bytes

TCP
Send buffer

Transmit segments
Segment Segment ... Segment

TCP
Receive buffer

Application process

Read bytes

...
TCP Header Format

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

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>10</th>
<th>16</th>
<th>31</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>0</td>
<td>Flags</td>
<td>Checksum</td>
<td>UrgPtr</td>
</tr>
</tbody>
</table>

Options (variable)

Data
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位</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>SrcPort</td>
</tr>
<tr>
<td>4-7</td>
<td>DstPort</td>
</tr>
<tr>
<td>8-15</td>
<td>SequenceNum</td>
</tr>
<tr>
<td>16-23</td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>24</td>
<td>HdrLen</td>
</tr>
<tr>
<td>25-27</td>
<td>Flags</td>
</tr>
<tr>
<td>28-31</td>
<td>AdvertisedWindow</td>
</tr>
<tr>
<td></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

<table>
<thead>
<tr>
<th>0 4 10 16 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
</tr>
<tr>
<td>SequenceNum</td>
</tr>
<tr>
<td>Acknowledgment</td>
</tr>
<tr>
<td>HdrLen 0 Flags AdvertisedWindow</td>
</tr>
<tr>
<td>Checksum</td>
</tr>
<tr>
<td>Options (variable)</td>
</tr>
<tr>
<td>Data</td>
</tr>
</tbody>
</table>
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 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

Passive participant (server)

Delayed old SYN

Rejected

+data
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 ➔ +data
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
Again, with States

Active participant (client)

SYN_SENT

SYN, SequenceNum = x

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

ACK, Acknowledgment = y + 1

ESTABLISHED

Passive participant (server)

LISTEN

SYN_RCVD

ESTABLISHED

+data

CSE 123 – Lecture 9: 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

CLOSE_WAIT

LAST_ACK

ACK

CLOSED

FIN

ACK

CLOSED
The TIME_WAIT State

- 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
Sender and Receiver Buffering

- **Sending application**
  - LastByteWritten
  - LastByteAcked
  - LastByteSent

- **Receiving application**
  - LastByteRead
  - NextByteExpected
  - LastByteRcvd

- Black = available buffer
- Light blue = buffer in use
Window-Size Example

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

Stall due to flow control here
Example – Buffer at Sender

- **T=1**: 1 2 3 4 5 6 7 8 9
- **T=2**: 1 2 3 4 5 6 7 8 9
- **T=3**: 1 2 3 4 5 6 7 8 9
- **T=4**: 1 2 3 4 5 6 7 8 9
- **T=5**: 1 2 3 4 5 6 7 8 9
- **T=6**: 1 2 3 4 5 6 7 8 9

- **T** = Times
- **=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
- Enjoy Thanksgiving