Lecture 6: Transport Layer Protocols

CSE 123: Computer Networks
Alex C. Snoeren

HW 1 due NOW
Overview

- Process naming/demultiplexing
- User Datagram Protocol (UDP)
- Transport Control Protocol (TCP)
  - State transitions
  - Connection teardown
Today: Transport Layer

Application Layer

Transport Layer

Network Layer

Link Layer

host

router

host

HTTP

TCP

IP

Ethernet interface

SONET interface

Ethernet interface

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

- Ports
- Message Queues
- Application process
- Kernel boundary
- DeMux
- Packets arrive
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 **bytestream** between processes
  - Uses a sliding window protocol for efficient transfer

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

- Flow control (last lecture)
  - 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
TCP Delivery

Application process

TCP

Send buffer

Write
bytes

Transmit segments

Segment

Segment

Segment

TCP

Receive buffer

Application process

Read
bytes

CSE 123 – Lecture 6: Transport Protocols
TCP Header Format

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

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

Data
TCP Header Format

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

```
0  4  10  16  31
SrcPort | DstPort
        |        
SequenceNum
        |        
Acknowledgment
HdrLen | 0 | Flags | AdvertisedWindow
        |   |       |
Checksum | UrgPtr
        |     |
Options (variable)
Data
```
TCP Header Format

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

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SrcPort</td>
</tr>
<tr>
<td>1-2</td>
<td>DstPort</td>
</tr>
<tr>
<td>3-4</td>
<td>SequenceNum</td>
</tr>
<tr>
<td>5-6</td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>7-31</td>
<td>Options (variable)</td>
</tr>
<tr>
<td>32</td>
<td>Checksum</td>
</tr>
<tr>
<td>33-34</td>
<td>UrgPtr</td>
</tr>
<tr>
<td>35-31</td>
<td>AdvertisedWindow</td>
</tr>
<tr>
<td>32</td>
<td>Data</td>
</tr>
</tbody>
</table>
Three-Way Handshake

- Opens both directions for transfer

Active participant (client)

SYN, SequenceNum = x

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

ACK, Acknowledgment = y + 1

Passive participant (server)

+data

CSE 123 – Lecture 6: Transport Protocols
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
- ESTABLISHED
- FIN_WAIT_1
- FIN_WAIT_2
- CLOSING
- TIME_WAIT
- CLOSE_WAIT
- LAST_ACK
- CLOSED

TCP State Transitions:

- Passive open
- Active open/SYN
- Close
- Send/ SYN
- SYN/SYN + ACK
- SYN + ACK/ACK
- FIN/ACK
- FIN_WAIT_1
- FIN_WAIT_2
- CLOSING
- TIME_WAIT
- LAST_ACK
- CLOSED

- Timeout after two segment lifetimes

CSE 123 – Lecture 6: Transport Protocols
Again, with States

Active participant (client)

SYN_SENT

SYN, SequenceNum = x

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

ACK, Acknowledgment = y + 1

ESTABLISHED

+data

Passive participant (server)

LISTEN

SYN_RCVD

ESTABLISHED
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
The TIME_WAIT State

- We wait $2*\text{MSL}$ (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 $2*\text{MSL}$, simply send Reset packet (RST)
  - Why?
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
For next time…

- Beginning to discuss networks
- Read Ch 3-3.1 in P&D