CSCI-1680
Transport Layer I

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, David Mazières, Phil Levis, John Jannotti
Administrivia

• Later today: Look for message about IP grading
  – Meeting slots start on Friday
• TCP: Draft of assignment out today
• HW3: Due on Monday
Today

Light overview of the transport layer and TCP

– Why we need TCP
– What components are involved
– What you will do in the project
Review P2Q13

- Customer
- Peer

Problem: A doesn’t know about D! (and vice versa)

What topology change would fix?

- Peer X, Z
- Peer A, D
- A becomes customer of Z
- D
- D

(Note: This version corrects an error from the lecture version of the notes - see Edition #232)
Transport Layer

- Transport protocols build on the network layer
- Problem solved: communication among processes
  - Application-level multiplexing provided by ports, OS interface to applications via sockets
  - TCP adds error detection, reliability, etc.
Ports are part of the transport layer

Port numbers are the first two fields of these headers! (Not part of IP!)
User Datagram Protocol

- "Unreliable datagram service"
- Send a message between two ports, and nothing else
- Checksum is pretty useless
Problem: Reliability

What kinds of things can go wrong?
- Dropped packets
- Duplicate packets
- Packets arrive out of order

=> How to handle? Add sequence numbers to each segment, retransmit on timeout
Problem: Reliability

What kinds of things can go wrong?

• Dropped packets
• Duplicate packets
• Packets arrive out of order

=> How to handle? Add sequence numbers to each segment, retransmit on timeout.

Multiple hops and paths => Lots of opportunities for failure!
Other challenges

- Hosts have different (and unknown!) resources
  - Flow control: How much / how fast can we send to receiver?
- Network has unknown resources
  - Varying RTT / latency, link bandwidth
  - Congestion control: Must not overload network (fairness)
Problem: Reliability + Performance

Other challenges

• Hosts have different (and unknown!) resources
  – Flow control: how much can we send to receiver?

• Network has unknown resources
  – Varying RTTs/link bandwidth: need to monitor/adjust send rate
  – Congestion control: must not overload the network

... but, need to maximize throughput => best performance
TCP – Transmission Control Protocol

- Service model: “reliable, connection oriented, full duplex ordered byte stream”
TCP – Transmission Control Protocol

- Service model: “reliable, connection oriented, full duplex ordered byte stream”
- Flow control: If one end stops reading, writes at other eventually stop/fail
TCP – Transmission Control Protocol

- Service model: “reliable, connection oriented, full duplex ordered byte stream”
- Flow control: If one end stops reading, writes at other eventually stop/fail
- Congestion control: Keeps sender from overloading the network
TCP

  + Many more RFCs now!
- Was born coupled with IP, later factored out
- End-to-end protocol
  - Minimal assumptions on the network
  - All mechanisms run on the end points

=> But wait, what if you had link-layer reliability instead?
Why not provide X on the network layer?

X = Reliability, security, message ordering…

• **Might be too costly**
  – Extra features aren’t free: don’t burden protocols that don’t need them!
  – Cost could be extra latency, extra computation, …

• **Features might be conflicting**
  – Timeliness vs. in-order delivery

• **Might not be enough**
  – Example: reliability
End-to-end argument

• Functions placed at lower levels of a system may be redundant or of little value
  – They may need to be performed at a higher layer anyway
• But they may be justified for performance reasons
  – Or just because they provide most of what is needed
  – Example: retransmissions

=> Takeaway: Important to weigh the costs and benefits at each layer!
TCP Header

- Source Port
- Destination Port
- Sequence Number
- Acknowledgement Number
- Window Size
- Checksum
- Urgent Pointer
- Options
- Data

Flags

How much space is left in the buffer you have received?

Where this segment is in the data stream

LAST_SEQUENCE # YOU HAVE RECEIVED
Important Header Fields

- **Ports**: multiplexing
- **Sequence number**
  - Where segment is in the stream (in bytes)
- **Acknowledgment Number**
  - Next expected sequence number
- **Window**
  - How much data you’re willing to receive

\[
\text{if window} = 0, \text{ sender must stop.}
\]
Important Header Fields: Flags

- SYN: "Synchronize" - start a connection
- ACK: ACK field says what byte to expect next (most packets are all ACKs)
- FIN: used to close connection
- RST: reset connection (used for errors)
- PSH: push data to the application immediately
- URG: whether there is urgent data

"I am done sending"
Important Header Fields: Flags

- **SYN**: establishes connection ("synchronize")
- **ACK**: this segment ACKs some data (all packets except first)
- **FIN**: close connection (gracefully)
- **RST**: reset connection (used for errors)
- **PSH**: push data to the application immediately
- **URG**: whether there is urgent data
Less important header fields

- **Checksum**: Very weak, like IP
  - Has weird semantics ("pseudo header"), more on this later…

- **Data Offset**: used to indicate TCP options (mostly unused)
- **Urgent Pointer**: No one uses this over the internet.
TCP Standards: The Many RFCs

- RFC793 (Original)
- RFC1122 (Some corrections)
- RFC5681 (Congestion control)
- RFC7414 (Roadmap to TCP RFCs)
- Various Errata...
# TCP Standards: The Many RFCs

<table>
<thead>
<tr>
<th>RFC documents</th>
<th>[ edit ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC 793 – TCP v4</td>
<td></td>
</tr>
<tr>
<td>RFC 1122 – includes some error corrections for TCP</td>
<td></td>
</tr>
<tr>
<td>RFC 1323 – TCP Extensions for High Performance [Obsoleted by RFC 7323]</td>
<td></td>
</tr>
<tr>
<td>RFC 1948 – Defending Against Sequence Number Attacks</td>
<td></td>
</tr>
<tr>
<td>RFC 2018 – TCP Selective Acknowledgment Options</td>
<td></td>
</tr>
<tr>
<td>RFC 5681 – TCP Congestion Control</td>
<td></td>
</tr>
<tr>
<td>RFC 6247 – Moving the Undeployed TCP Extensions RFC 1072, 1106, 1110, 1145, 1146, 1378</td>
<td></td>
</tr>
<tr>
<td>RFC 6298 – Computing TCP’s Retransmission Timer</td>
<td></td>
</tr>
<tr>
<td>RFC 6824 – TCP Extensions for Multipath Operation with Multiple Addresses</td>
<td></td>
</tr>
<tr>
<td>RFC 7323 – TCP Extensions for High Performance</td>
<td></td>
</tr>
<tr>
<td>RFC 7414 – A Roadmap for TCP Specification Documents</td>
<td></td>
</tr>
<tr>
<td>RFC 9293 – Transmission Control Protocol (TCP)</td>
<td></td>
</tr>
</tbody>
</table>
RFC9293: The One RFC!!!
I hope this will make life easier for the project!
ESTABLISHING A TCP CONNECTION

LISTEN

SERVER (LISTEN)

LISTENING ON A PORT

SYN RECEIVED

3-WAY HANDSHAKE

SET UP COMM.

AGREE ON STARTING SEQ NUMBERS.

CLIENT

P(ACC) (CONNECT)

SYN + ACK
SEQ = y
ACK = x + 1

ACK SEQ = x + 1
ACK = y + 1

ESTABLISHED

ACCEPT() RETURNS NEW
Establishing a Connection

- Three-way handshake
  - Two sides agree on respective initial sequence nums
- If no one is listening on port: OS may send RST
- If server is overloaded: ignore SYN
- If no SYN-ACK: retry, timeout
Establishing a Connection

• Three-way handshake
  – Two sides agree on respective initial sequence nums
• If no one is listening on port: OS may send RST
• If server is overloaded: ignore SYN
• If no SYN-ACK: retry, timeout
Establishing a Connection

- Three-way handshake
  - Two sides agree on respective initial sequence nums
- If no one is listening on port: OS may send RST
- If server is overloaded: ignore SYN
- If no SYN-ACK: retry, timeout
Establishing a Connection

- Three-way handshake
  - Two sides agree on respective initial sequence numbers
- If no one is listening on port: OS may send RST
- If server is overloaded: ignore SYN
- If no SYN-ACK: retry, timeout
How do we tell two connections apart?
How do we tell two connections apart?

- **Port numbers**
  - 5-tuple (proto., source IP, source port, dest IP, dest port) => 1 Connection
  - Kernel maintains *socket table*: maps (5-tuple) => Socket
Connections + Port Numbers

*Client picks random, "ephemeral," source port - would be different for each connection, even from same host.*

**Tuple of** \((\text{source IP}, \text{source port}), (\text{dest IP}, \text{dest port}), \text{protocol})**

*Don't need for project (all are TCP) => one connection*

**The kernel has a socket table that maps** s-tuple \(\Rightarrow\) connection state.
How do we tell two connections apart?

• Port numbers
  – 5-tuple (proto., source IP, source port, dest IP, dest port) => 1 Connection
  – Kernel maintains socket table: maps (5-tuple) => Socket

• Sequence numbers don’t start at 0!
  – Start from “random” Initial Sequence Number (ISN)
  – If a 5-tuple is reused => new ISN, so sequence numbers likely out of range from past connection
Netstat

```
deeemer@vesta ~/Development % netstat -an
Active Internet connections (including servers)

<table>
<thead>
<tr>
<th>Proto</th>
<th>Recv-Q</th>
<th>Send-Q</th>
<th>Local Address</th>
<th>Foreign Address</th>
<th>(state)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>10.3.146.161.51094</td>
<td>104.16.248.249.443</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>10.3.146.161.51076</td>
<td>172.66.43.67.443</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp6</td>
<td>0</td>
<td>0</td>
<td>2620:6e:6000:900.51074</td>
<td>2606:4700:3108:443</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>10.3.146.161.51065</td>
<td>35.82.230.35.443</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>10.3.146.161.51055</td>
<td>162.159.136.234.443</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>10.3.146.161.51038</td>
<td>17.57.147.5.5223</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp6</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.51036</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.51036</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>127.0.0.1.14500</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
</tbody>
</table>
```
Below is extra content we will cover later.

(Feel free to read ahead, though!)
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
Summary of TCP States

Connection Establishment

- Passive open
- Send/SYN
- SYN/ SYN + ACK
- SYN + ACK/ACK
- FIN/ACK
- FIN/ACK
- FIN + ACK/ACK
- ACK
- ACK + FIN/ACK
- Timeout after two segment lifetimes
- Active open/SYN
- Close
- Close
- Close/FIN
- CLOSE_WAIT
- LAST_ACK
- CLOSED
Summary of TCP States
Summary of TCP States

Active close: Can still receive
Summary of TCP States

**Passive close:** Can still send!

**Active close:** Can still receive
Next class

• Sending data over TCP
The IPv4 Header

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>IHL</td>
<td>4</td>
</tr>
<tr>
<td>TOS</td>
<td>8</td>
</tr>
<tr>
<td>Total length</td>
<td>16</td>
</tr>
<tr>
<td>Identification</td>
<td>20</td>
</tr>
<tr>
<td>Flags</td>
<td>24</td>
</tr>
<tr>
<td>Fragment offset</td>
<td>28</td>
</tr>
<tr>
<td>TTL</td>
<td>29</td>
</tr>
<tr>
<td>Protocol</td>
<td>30</td>
</tr>
<tr>
<td>Header checksum</td>
<td>31</td>
</tr>
<tr>
<td>Source address</td>
<td>32</td>
</tr>
<tr>
<td>Destination address</td>
<td>40</td>
</tr>
<tr>
<td>Options</td>
<td>56</td>
</tr>
<tr>
<td>Data</td>
<td>65536</td>
</tr>
</tbody>
</table>

20 bytes
0-40 bytes
Up to 65536 bytes
The IPv4 Header

Defined by RFC 791
RFC (Request for Comment): defines network standard
If we have time: Port scanning
Port scanning

What can we learn if we just start connecting to well-known ports?

• Can discover things about the network
• Can learn about vulnerabilities
Large-scale port scanning

• Can reveal lots of open/insecure systems!
• Examples:
  – shodan.io
  – VNC roulette
  – Open webcam viewers..
  – …
Disclaimer

• Network scanning is easy to detect

• Unless you are the owner of the network, it’s seen as malicious activity

• If you scan the whole Internet, the whole Internet will get mad at you (unless done very politely)

• Do NOT try this on the Brown network. I warned you.
Scanning I have done

- Scanned IPv4 space for ROS (Robot Operating System)
- Found ~200 “things” using ROS (some robots, some other stuff)