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
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
We talked briefly about link-layer reliability:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dropped Packets</td>
<td>Acknowledgments + Timeout</td>
</tr>
<tr>
<td>Duplicate Packets</td>
<td>Sequence Numbers</td>
</tr>
<tr>
<td>Packets out of order</td>
<td>Receiver Window</td>
</tr>
<tr>
<td>Maximizing throughput</td>
<td>Sliding Window (Pipelining)</td>
</tr>
</tbody>
</table>

- Single link: things were easy... 😊
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!
Problem: Reliability + Performance

Other challenges

• *Hosts* have different (and unknown!) resources

• *Network* has unknown resources
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”
- Flow control: If one end stops reading, writes at other eventually stop/fail
- Congestion control: Keeps sender from overloading the network
TCP

• Initially: RFC 793 (1981)
  + 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
- Data Offset
- Reserved
- URG
- ACK
- PSH
- RST
- SYN
- FIN
- Window Size
- Checksum
- Urgent Pointer
- Options
- Data
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
- **Flags…**
Important Header Fields: Flags

- **SYN:**
- **ACK:**
- **FIN:**
- **RST:** reset connection (used for errors)
- **PSH:** push data to the application immediately
- **URG:** whether there is urgent data
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
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

- RFC 793 – TCP v4
- RFC 1122 – includes some error corrections for TCP
- RFC 1323 – TCP Extensions for High Performance [Obsoleted by RFC 7323]
- RFC 1948 – Defending Against Sequence Number Attacks
- RFC 2018 – TCP Selective Acknowledgment Options
- RFC 5681 – TCP Congestion Control
- RFC 6247 – Moving the Undeployed TCP Extensions RFC 1072, 1106, 1110, 1145, 1146, 137
- RFC 6298 – Computing TCP’s Retransmission Timer
- RFC 6824 – TCP Extensions for Multipath Operation with Multiple Addresses
- RFC 7323 – TCP Extensions for High Performance
- RFC 7414 – A Roadmap for TCP Specification Documents
- RFC 9293 – Transmission Control Protocol (TCP)
RFC9293: The One RFC

I hope this will make life easier for the project!

<table>
<thead>
<tr>
<th>RFC</th>
<th>Internet Standard</th>
<th>Request for Comments</th>
<th>Obsoletes:</th>
<th>Updates:</th>
<th>Category:</th>
<th>ISSN:</th>
</tr>
</thead>
<tbody>
<tr>
<td>9293</td>
<td>draft-ietf-tcpm-rfc793bis-28</td>
<td>9293</td>
<td>793, 879, 2873, 6093, 6429, 6528, 6691</td>
<td>1011, 1122, 5961</td>
<td>Standards Track</td>
<td>2070-1721</td>
</tr>
</tbody>
</table>

W. Eddy, Ed.
MTI Systems
August 2022
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
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

deemer@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></td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td><em>.</em></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>
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

- **Passive close**: Can still send!
- **Active close**: Can still receive

Connection Establishment:
- Passive open
- Active open/SYN

TCP States:
- CLOSED
- LISTEN
- SYN_RCVD
- SYN_SENT
- ESTABLISHED
- FIN_WAIT_1
- FIN_WAIT_2
- CLOSING
- LAST_ACK
- TIME_WAIT
- CLOSED

Events:
- SYN/SYN + ACK
- SYN/SYN + ACK
- SYN + ACK
- ACK
- Close/FIN
- FIN/ACK
- FIN/ACK
- Close/FIN
- Ack
- Timeout after two segment lifetimes
Next class

• Sending data over TCP
The IPv4 Header

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>31</th>
<th>bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>IHL</td>
<td>TOS</td>
<td>Total length</td>
<td>Identification</td>
<td>Flags</td>
</tr>
<tr>
<td>TTL</td>
<td>Protocol</td>
<td>Header checksum</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Source address
- Destination address
- Options
- Data

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)