CSCI-1680
Transport Layer Warmup (ish)

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, Jennifer Rexford, Rob Sherwood, David Mazières, Phil Levis, John Jannotti
Warmup

Given the following AS relationships, Which ASes will A know about?

---

<table>
<thead>
<tr>
<th>Advertised by…</th>
<th>Export to…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Everyone</td>
</tr>
<tr>
<td>Peer</td>
<td>Customers only</td>
</tr>
<tr>
<td>Provider</td>
<td>Customers only</td>
</tr>
</tbody>
</table>

---

Customer (“A is customer of X”)

Peer

B  ✓
X  ✓
Y  ✓
C  ✓
Z  NO!
D  NO!
Administrivia: This week

- IP: Due Thursday
  - Signups for grading meetings after that
  - Code cleanup, README, etc after deadline is okay
- HW2: Out today, due in ~2wks
- TCP: Out on Friday
  - Maybe a short intro/gearup on Thursday
This week

• Start of transport layer
• Intro to TCP
One more fun BGP thing...
Anycast

Advertise the same prefix (IP) from multiple places

=> Multiple devices have the same IP!!

• Used to make certain IPs highly available
  – Public DNS: 8.8.8.8 (Google), 1.1.1.1 (Cloudflare)

\[
\text{DNS: google.com} \rightarrow 1.2.3.4
\]

Problems?
Anycast

Advertise the same prefix (IP) from multiple places
 => Multiple devices have the same IP!!
• Used to make certain IPs highly available
  – Public DNS: 8.8.8.8 (Google), 1.1.1.1 (Cloudflare)

=> If you send multiple packets to 8.8.8.8, no guarantee you’re talking to the same server!
=> Protocol must be able to account for this (DNS does, more on this later)
Ports & Sockets
Layers, Services, Protocols

- **Application**
  - Service: user-facing application. Application-defined messages

- **Transport**
  - How to support multiple applications?

- **Network**
  - Moving data between hosts (nodes)

- **Link**
  - Move data across individual links

- **Physical**
  - Service: move bits to other node across link
The story so far

Network layer (L3): move packets between hosts (anywhere on Internet)
How to support multiple applications?

Network layer: moving data between hosts
Transport layer: abstraction for getting data to different applications on a host
How to support multiple applications?

Network layer: moving data between hosts

Transport layer: abstraction for getting data to different applications on a host

- Multiplexing multiple connections at the same IP using port numbers
- Turns series of packets => stream of data/messages

TCP

UDP
How to support multiple applications?

Network layer: moving data between hosts
Transport layer: abstraction for getting data to different applications on a host

- Multiplexing multiple connections at the same IP using port numbers
- Turns series of packets => stream of data/messages

⇒ Provided by OS as sockets
⇒ Use this abstraction to build other application protocols!
The transport layer MAY provide...

- Reliable data delivery *(RETRIES NOTIFICATION ON ERROR)*
- Creating a data stream ➔ makes packets from large data
- Managing throughput/sharing bandwidth ➔ “Congestion control” ➔ “FAIRNESS”
The transport layer MAY provide...

- Reliable data delivery
- Creating a data stream
- Managing throughput/sharing bandwidth
  - "Congestion control"

These are provided by TCP, which is our main focus. However:

⇒ Not required for all transport layer (UDP has none of these)
⇒ Other protocols do this too (eg. QUIC)
Transport Layer

- Transport protocols sit on top of the network layer (IP).
- Can provide:
  - Application-level multiplexing ("ports")
  - Error detection, reliability, etc.
From Lec 2: OSI Model

One or more nodes within the network

End host
Application
Presentation
Session
Transport
Network
Data link
Physical

End host
Application
Presentation
Session
Transport
Network
Data link
Physical

Application Protocol
Transport Protocol
Network Protocol
Link-Layer Protocol

Just care about endpoints

L3
L2
L1
What’s a port number?

- 16-bit unsigned integer, 0-65535
- Ports define a communication endpoint, usually a process/service on the host
What’s a port number?

- 16-bit unsigned integer, 0-65535
- Ports define a communication endpoint, usually a process/service on the host
What’s a port number?

- 16-bit unsigned integer, 0-65535
- Ports define a communication endpoint, usually a process/service on the host
- OS keeps track of which ports map to which applications

Port numbering
- port < 1024: “Well known port numbers” (allocated by IANA officially)
- port > 20000: “ephemeral ports”, for general app use
### Some common ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>20, 21</td>
<td>File Transfer Protocol (FTP)</td>
</tr>
<tr>
<td>22</td>
<td>Secure Shell (SSH)</td>
</tr>
<tr>
<td>23</td>
<td>Telnet (pre-SSH remote login)</td>
</tr>
<tr>
<td>25</td>
<td>SMTP (Email)</td>
</tr>
<tr>
<td>53</td>
<td>Domain Name System (DNS)</td>
</tr>
<tr>
<td>67, 68</td>
<td>DHCP</td>
</tr>
<tr>
<td>80</td>
<td>HTTP (Web traffic)</td>
</tr>
<tr>
<td>443</td>
<td>HTTPS (Secure HTTP over TLS)</td>
</tr>
</tbody>
</table>
How ports work

The kernel maps ports to sockets, which are used in applications like file descriptors to access the network.

Two modes for using ports/sockets:

- **Listen mode**: apps “bind” to a port to accept new connections.
- **“Outgoing” mode**: make a connection.
- Individual connections use 5-tuple of source-dest port: 
  \[(\text{protocol}, \text{source IP}, \text{source port}, \text{dest IP}, \text{dest port}) \rightarrow \text{connection N}\]

*: Nick made this term up so it has a name.
How ports work

The kernel maps ports to sockets, which are used in applications like file descriptors to access the network.

Two modes for using ports/sockets:

- **Listen mode**: apps “bind” to a port to accept new connections

  => **USE TO RECEIVE PACKETS**

- **“Outgoing” mode**: make a connection to another socket

  => **MAKE OUTGOING CONNECTION**

*: Nick made this term up so it has a name
How ports work

The kernel maps ports to sockets, which are used in applications like file descriptors to access the network.

Two modes for using ports/sockets:

- **Listen mode**: apps “bind” to a port to accept new connections
  => Used to receive/wait for new connections

- **“Normal” mode**: make a connection to another socket
  => Used to make outgoing connections

*: Nick made this term up so it has a name
listen(80)

A
1.2.3.4

B
5.6.7.8

MAKE SOCKET
LISTEN SOCKET
A
1.2.3.4

B
5.6.7.8

\texttt{listen(80)}

\texttt{connect(1.2.3.4, 80)}

\texttt{DIAL}

\texttt{I KNOW AHEAD OF TIME}

\texttt{PICKED BY OS (EPHEMERAL PORTS)}
A must know B is listening on port 80
=> "well known numbers"

When connecting, A's OS picks random source port (eg. 12345),
for its side of connection
A connects to B on port 80 (1.2.3.4 -> 5.6.7.8, Port 12345 -> 80)

B responds to A using this port

A is waiting on this port.
Demo: netcat
How sockets work

Socket: OS abstraction for a network connection (like a file descriptor)

Kernel receives all packets => needs to map each packet to a socket to deliver to app

- **Socket table**: list of all open sockets
- Each socket has some kernel state too (buffers, etc.)

You will build this!!!
How to map packets to sockets?

Kernel table looks something like this:

<table>
<thead>
<tr>
<th>Proto</th>
<th>Local (yours)</th>
<th>Remote (theirs)</th>
<th>Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IP</td>
<td>IP</td>
<td>Socket</td>
</tr>
<tr>
<td>tcp/udp</td>
<td>MY LAPTOP</td>
<td>WEB SERVER</td>
<td>pid</td>
</tr>
<tr>
<td></td>
<td>12345</td>
<td>WEB SERVER</td>
<td>pid...</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>22346</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>WEB SERVER</td>
<td></td>
</tr>
</tbody>
</table>
How to map packets to sockets?

Kernel table looks something like this:

<table>
<thead>
<tr>
<th>Proto</th>
<th>Local (yours)</th>
<th>Remote (theirs)</th>
<th>Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IP</td>
<td>IP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Port</td>
<td>Port</td>
<td></td>
</tr>
<tr>
<td>tcp/udp</td>
<td></td>
<td>(some struct)</td>
<td></td>
</tr>
</tbody>
</table>

Key: 5-tuple of (local IP, local port, remote IP, remote port, protocol)
Value: kernel state for socket (state, buffers, …)
How to map packets to sockets?

*Kernel table looks something like this:*

<table>
<thead>
<tr>
<th>Proto</th>
<th>Local (yours)</th>
<th>Remote (theirs)</th>
<th>Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IP</td>
<td>Port</td>
<td>IP</td>
</tr>
<tr>
<td>tcp</td>
<td>1.2.3.4</td>
<td>12345</td>
<td>5.6.7.8</td>
</tr>
</tbody>
</table>

**Key:** 5-tuple of (local IP, local port, remote IP, remote port, protocol)

**Value:** kernel state for socket (state, buffers, …)
<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>
<tr>
<td>Proto</td>
<td>Local (yours)</td>
<td>Remote (theirs)</td>
<td>Socket</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>----------------</td>
<td>--------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>Port</td>
<td>IP</td>
<td>Port</td>
<td></td>
</tr>
<tr>
<td>tcp</td>
<td>1.2.3.4</td>
<td>12345</td>
<td>5.6.7.8</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>tcp</td>
<td>*</td>
<td>22</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

**Key:** 5-tuple of (local IP, local port, remote IP, remote port, protocol)

Value: kernel state for socket (state, buffers, ...)

What if A does: **listen(22)**
Ports are part of the transport layer

Port numbers are the first two fields of these headers! (Not part of IP!)
An interface to applications

- Ports define an interface to applications
- If you can connect to the port, you can (usually) use it!
Port scanning

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

• Applications have common port numbers
• Network protocols use well-defined patterns

```
deemer@vesta ~/Development % nc <IP addr> 22
SSH-2.0-OpenSSH_9.1
```
Port scanning

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

• Applications have common port numbers
• Network protocols use well-defined patterns

```bash
deemer@vesta ~/Development % nc <IP addr> 22
SSH-2.0-OpenSSH_9.1
```

⇒ Can discover things about the network
⇒ Can learn about open (vulnerable) systems
Port scanning

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

• Applications have common port numbers
• Network protocols use well-defined patterns

⇒ Can discover things about the network
⇒ Can learn about open (vulnerable) systems

Port scanners: try to connect to lots of ports, determine available services, find vulnerable services...
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.
Internet scanning I have done

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