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

Administrivia: This week

- IP: Due Thursday
  - Signups for grading meetings after that
  - Look for a feedback form

Administrivia: This week

- IP: Due Thursday
  - Signups for grading meetings after that
  - Look for a feedback form
  - Code cleanup, README, etc after deadline is okay

Administrivia: This week

- IP: Due Thursday
  - Signups for grading meetings after that
  - Code cleanup, README, etc after deadline is okay
  - Look for a feedback form

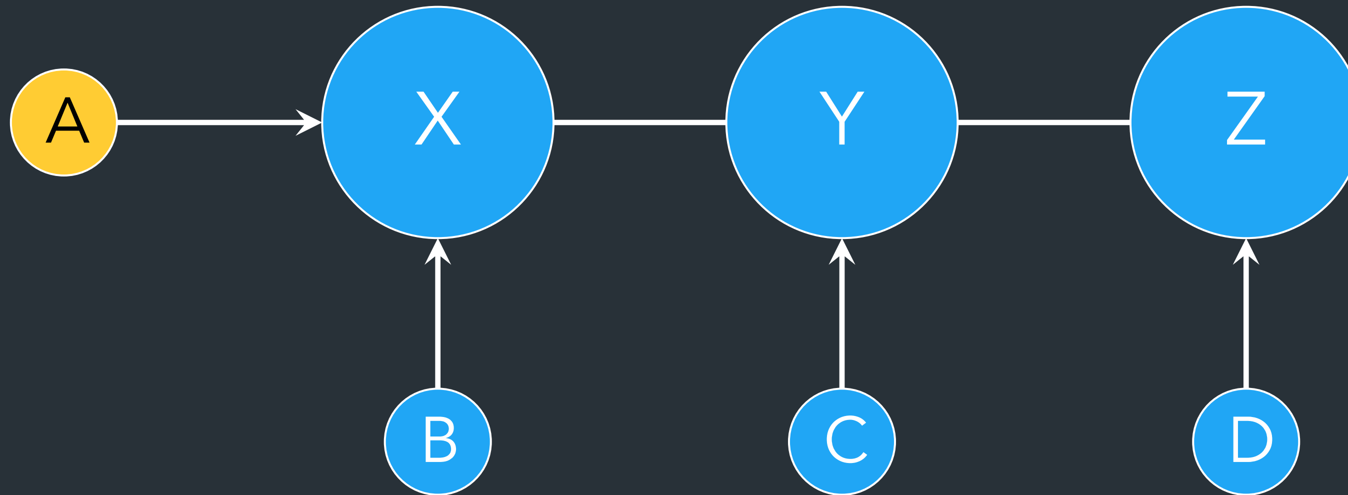
- HW2: Out today, due in >1wk

- TCP: Out on Friday
  - New team form this week (you MAY keep the same team)

Warmup

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

Advertised by…	Export to…
Customer	Everyone
Peer	Customers only
Provider	Customers only

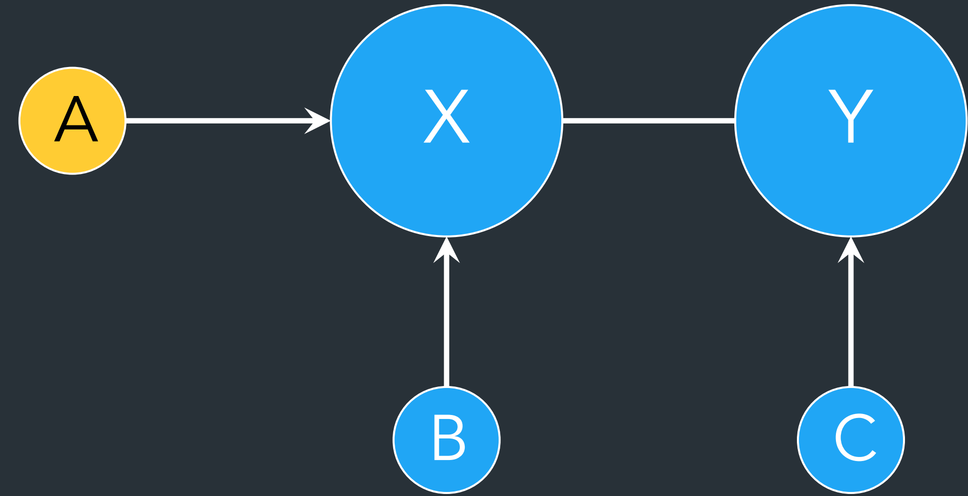


Customer ("A is customer of X")

Peer

Warmup II

What happens if C suddenly starts advertising A's prefix?



Recap: Prefix hijacking

By default, BGP doesn't verify that advertised routes match their owners

Update: *"I can reach prefix 128.148.0.0/16*
hrough ASes 44444 3356 14325 11078"

Recap: Prefix hijacking

By default, BGP doesn't verify that advertised routes match their owners

Update: *"I can reach prefix 128.148.0.0/16*
hrough ASes 44444 3356 14325 11078"

BGP router should ask:
"*Should* AS11078 be originating 138.16.161.0/24?"

Recap: Prefix hijacking

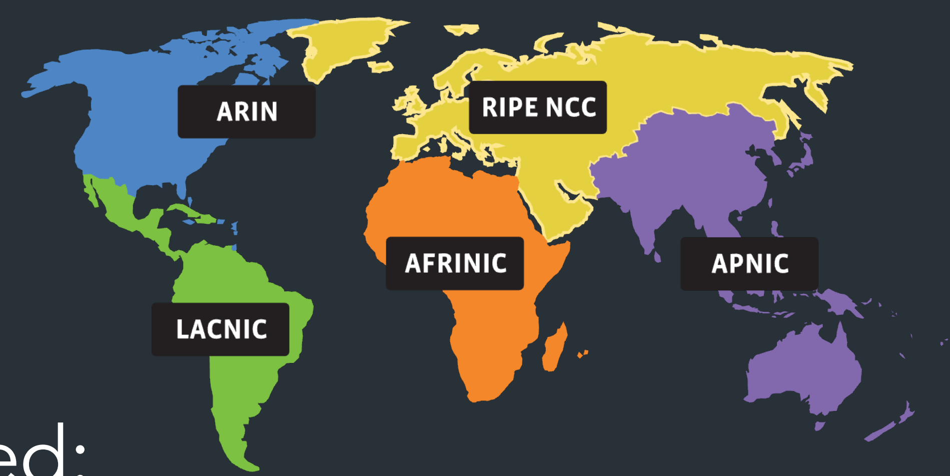
By default, BGP doesn't verify that advertised routes match their owners

Update: *"I can reach prefix 128.148.0.0/16*
hrough ASes 44444 3356 14325 11078"

BGP router should ask:
"*Should* AS11078 be originating 138.16.161.0/24?"

=> Not part of BGP by default. Standards have evolved to help, but adoption is limited.

A modern way: RPKI



Leverages hierarchy of how IPs are allocated:

- Every AS adds a *signature* of its route info in database, signed by authority that allocates addresses
  => ROA (Route Origin Authorization)
- Other ASes can verify ROA signature using cryptography, making it hard to forge

A modern way: RPKI

Leverages hierarchy of how IPs are allocated:

- Every AS adds a *signature* of its route info in database, signed by authority that allocates addresses
  => ROA (Route Origin Authorization)

- Other ASes can verify ROA signature using cryptography, making it hard to forge

- Can avoid
  - Prefix hijacking
  - Addition, removal, or reordering of intermediate ASes

ROAs for OSHEAN (Brown's provider)

Found *4* ROAs and *9* certificates

ROAs

ASN	Prefix	Max Length	IP Family	Trust Anchor	Emitted	Expiration
AS14325	2607:d00::/32	/64	IPv6	ARIN	8/28/2024	in a month
AS14325	131.109.0.0/16	/24	IPv4	ARIN	8/24/2024	in a month

Prefix: 131.109.0.0/16
Max Length: /24
ASN: 14325
Emitted: Sat, 24 Aug 2024 13:00:41 GMT
Validity: Sat, 24 Aug 2024 13:00:41 GMT - Fri, 22 Nov 2024 14:00:41 GMT
Trust Anchor: ARIN
Name: 5f622e78-c575-449a-836d-8c3f5e873fd4
Key: e852ddd445bb44252099dd8b7c39d39388bea544
Parent Key: 6b3fadc67835654f184fbbaa8e7b2408de32dabb
Path: rsync://rpki.arin.net/repository/arin-rpki-ta/5e4a23ea-e80a-403e-b08c-2171da2157d3/a73420cb-b3cc-4b03-bda7-1be204933ae5/f3e09673-5c6e-4340-ad11-4da8dfb8c777/08efbae2-5477-331b-b473-38399917c289.roa

Trust Anchor | Certificate | ROA file | ROA | Selected

ARIN
Trust Anchor

5e4a23ea-e80a-403e-b08c-2171da2157d3
1 ROAS

a73420cb-b3cc-4b03-bda7-1be204933ae5
1 ROAS

f3e09673-5c6e-4340-ad11-4da8dfb8c777
1 ROAS

5f622e78-c575-449a-836d-8c3f5e873fd4
AS14325

131.109.0.0/16
AS14325

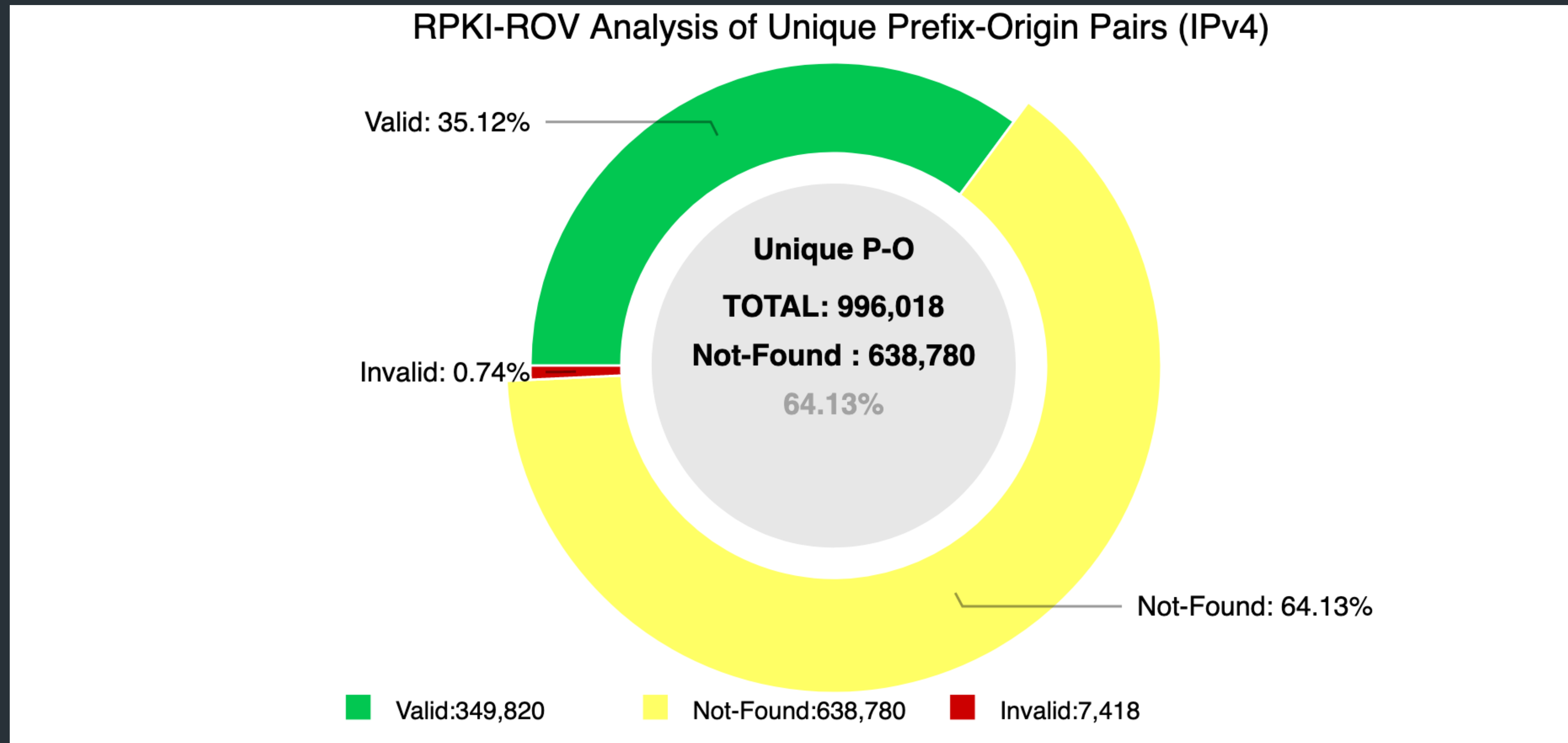
ROAs for Brown

KEY ID: Enter a resource key
TRUST ANCHOR: All
ASN: 11078
PREFIX: Enter an IP prefix
PREFIX MATCH: Less Specific | More Specific

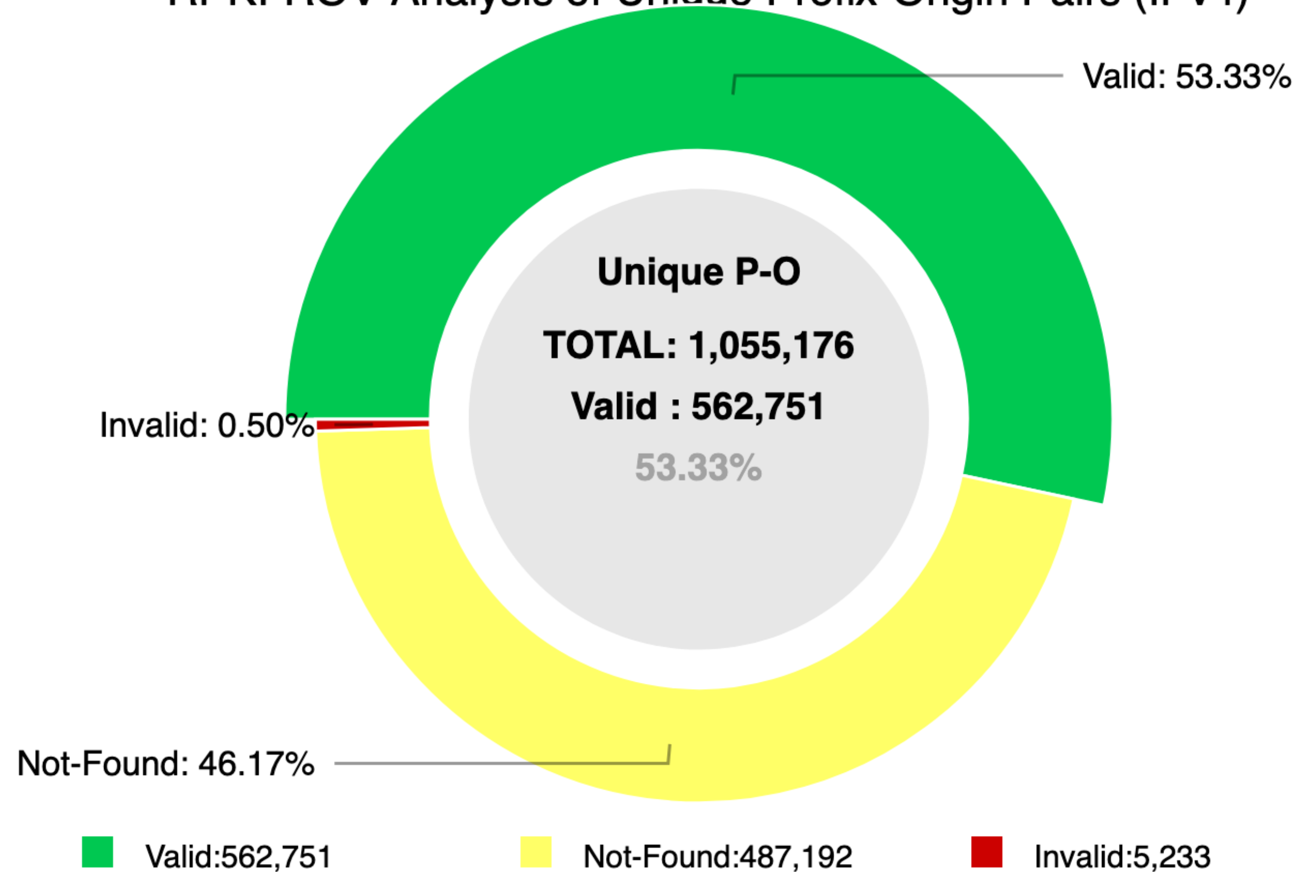
Resource List | Hierarchical View

Found *0* ROAs and *8* certificates

RPKI deployment (2022)



RPKI deployment (2024)



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)

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)

Problems?

Anycast

Advertise the same prefix (IP) from multiple places

=> Multiple devices have the same IP!!

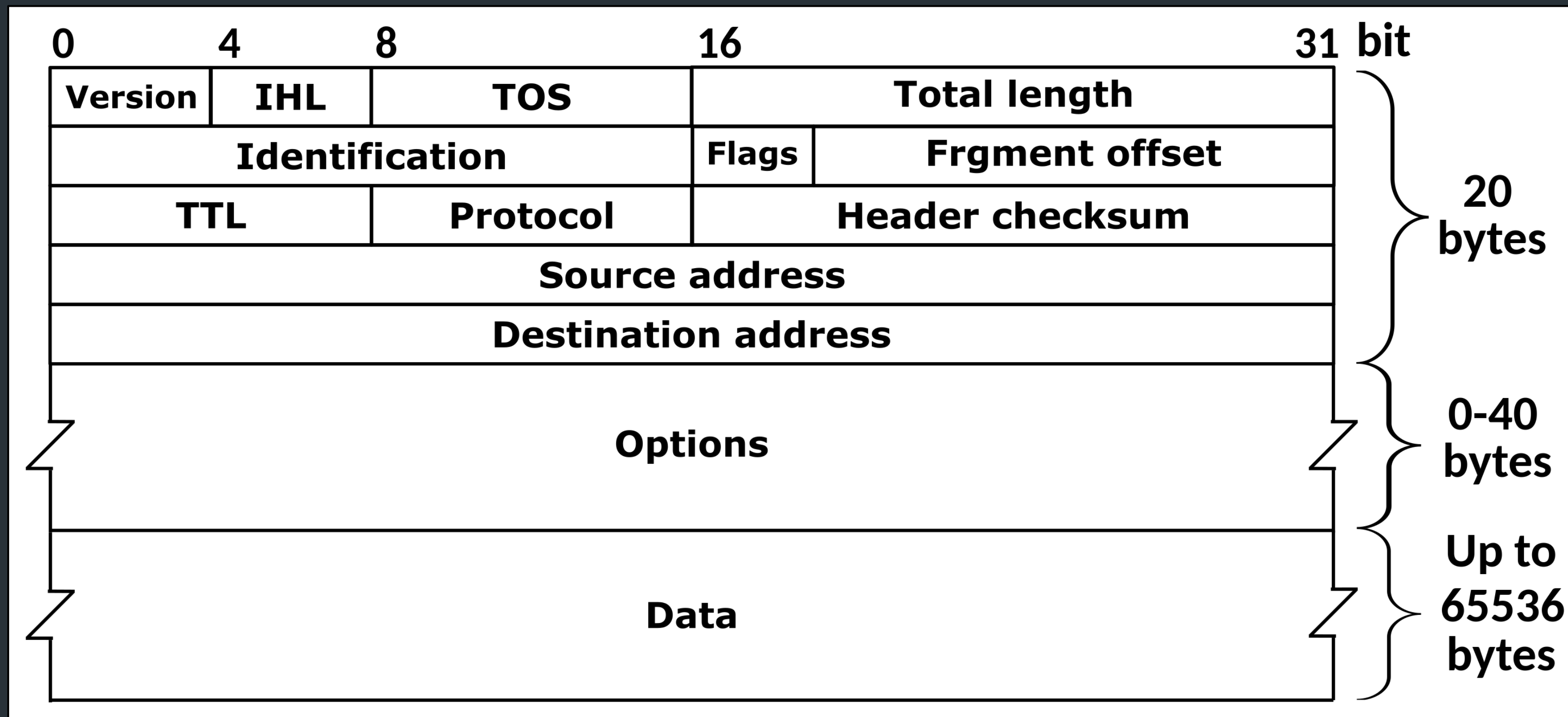
- Used to make certain IPs highly available
  - Public DNS: 8.8.8.8 (Google), 1.1.1.1 (Cloudflare)

=> 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)

Intro to TCP: Ports and Sockets

The story so far

Network layer (L3): move packets between hosts (anywhere on Internet)

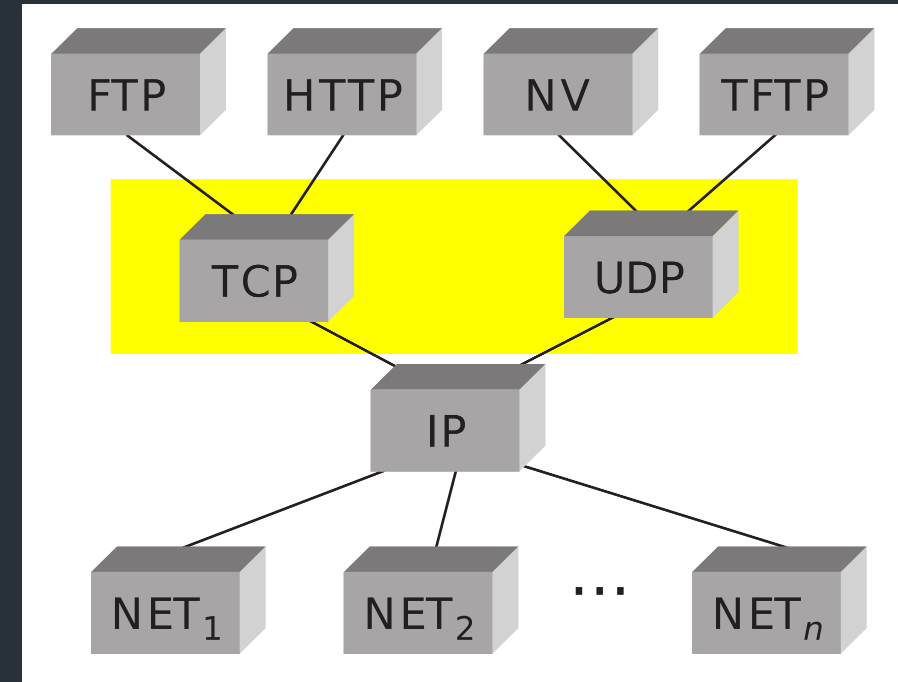


Layers, Services, Protocols

Layer	Description
Application	Service: user-facing application. Application-defined messages
Transport	How to support multiple applications?
Network	Moving data between hosts (nodes)
Link	Move data across <u>individual *links*</u>
Physical	Service: move bits to other node across link

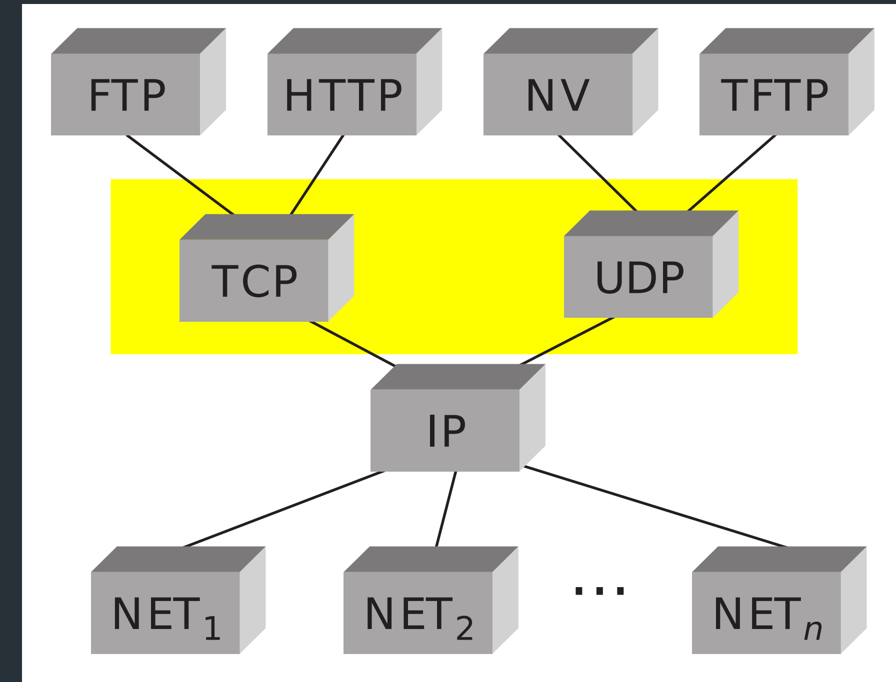
The transport layer: a service provided for applications, usually part of the OS

Examples: TCP, UDP



The transport layer: a service provided for applications, usually part of the OS

Examples: TCP, UDP



Major challenges

- Multiplexing: multiple connections at same IP
- Messaging: packets ?= messages

TCP is one transport-layer protocol
=> Provides a *reliable, connection-oriented, byte stream*

TCP: a reliable, connection-oriented, byte stream

Today's focus: connections

TCP: a reliable, connection-oriented, byte stream

Today's focus: connections

More generally: how does the OS support multiple *applications* using the network?
=> *Not just about TCP!*

How to support multiple applications?

Multiplexing multiple connections at the same IP using port numbers

=> Provided by OS as sockets

=> In general, used by all transport-layer protocols

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?

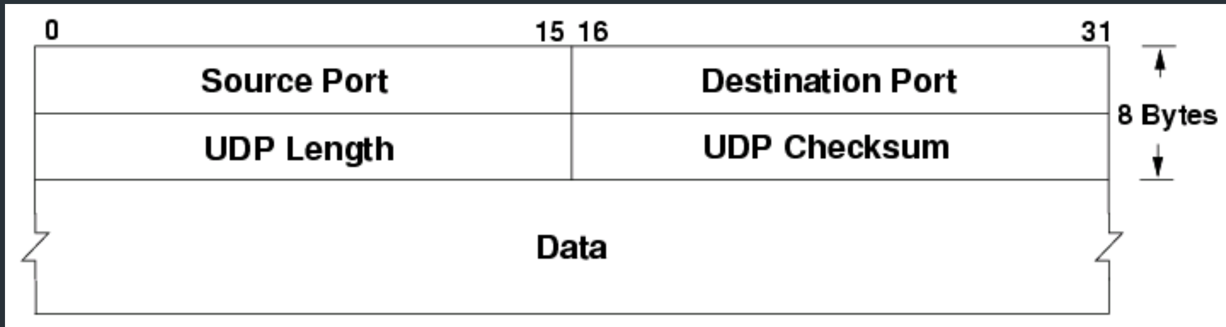
- 16-bit unsigned integer, 0-65535
- Ports define a communication endpoint, usually a process/service on the host
- OS keeps track of which ports map to which applications

Port numbering

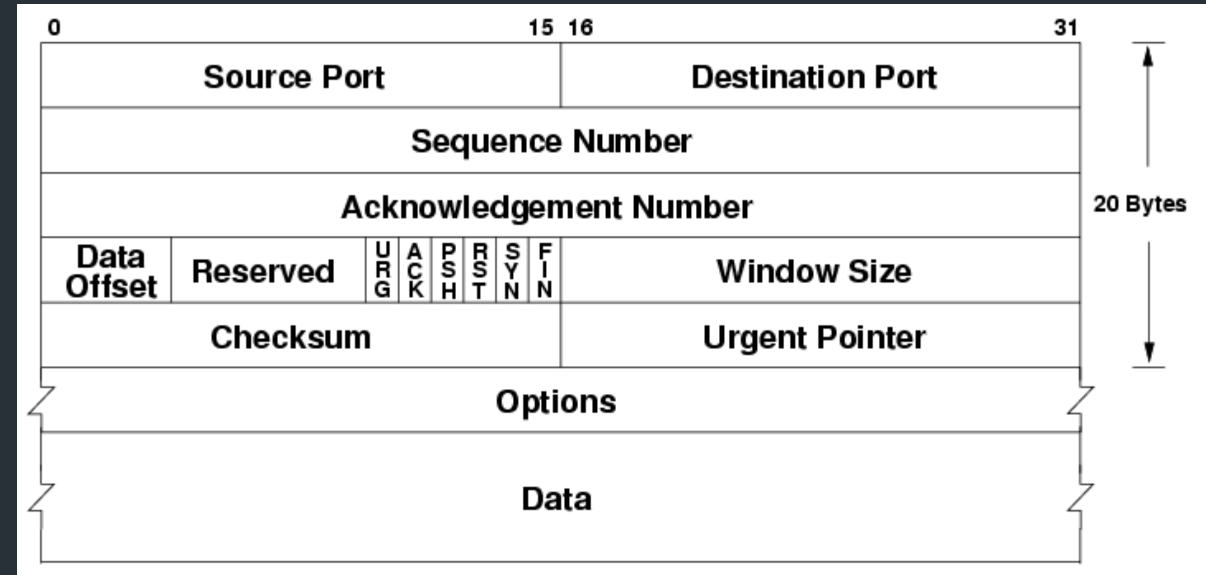
- port < 1024: "Well known port numbers"
- port > 20000: "ephemeral ports", for general app use

Ports are part of the transport layer

UDP



TCP



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

Some common ports

Port	Service
20, 21	File Transfer Protocol (FTP)
22	Secure Shell (SSH)
23	Telnet (pre-SSH remote login)
25	SMTP (Email)
53	Domain Name System (DNS)
67, 68	DHCP
80	HTTP (Web traffic)
443	HTTPS (Secure HTTP over TLS)

Ports and connections in TCP

Think back to Snowcast:

```
func main() {
    listenConn, err := net.Listen("tcp", "127.0.0.1:5000")

    for {
            clientConn, err := listenConn.Accept()

        go handleClient(clientConn)
    }
}
```

Ports and connections in TCP

To implement TCP, we need:

- Wait for new connections

- Individual connections between server and client
  => Separate data streams for multiple clients at a time!

How ports/sockets work

Two modes for using ports/sockets

- Listen (or "passive") mode

- Normal (or "active") mode

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

How ports/sockets work

Two modes for using ports/sockets

- Listen (or "passive") mode: apps "bind" to a port to accept new connections

- Normal (or "active") mode: a specific connection to another socket (probably on a different system)

*: 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

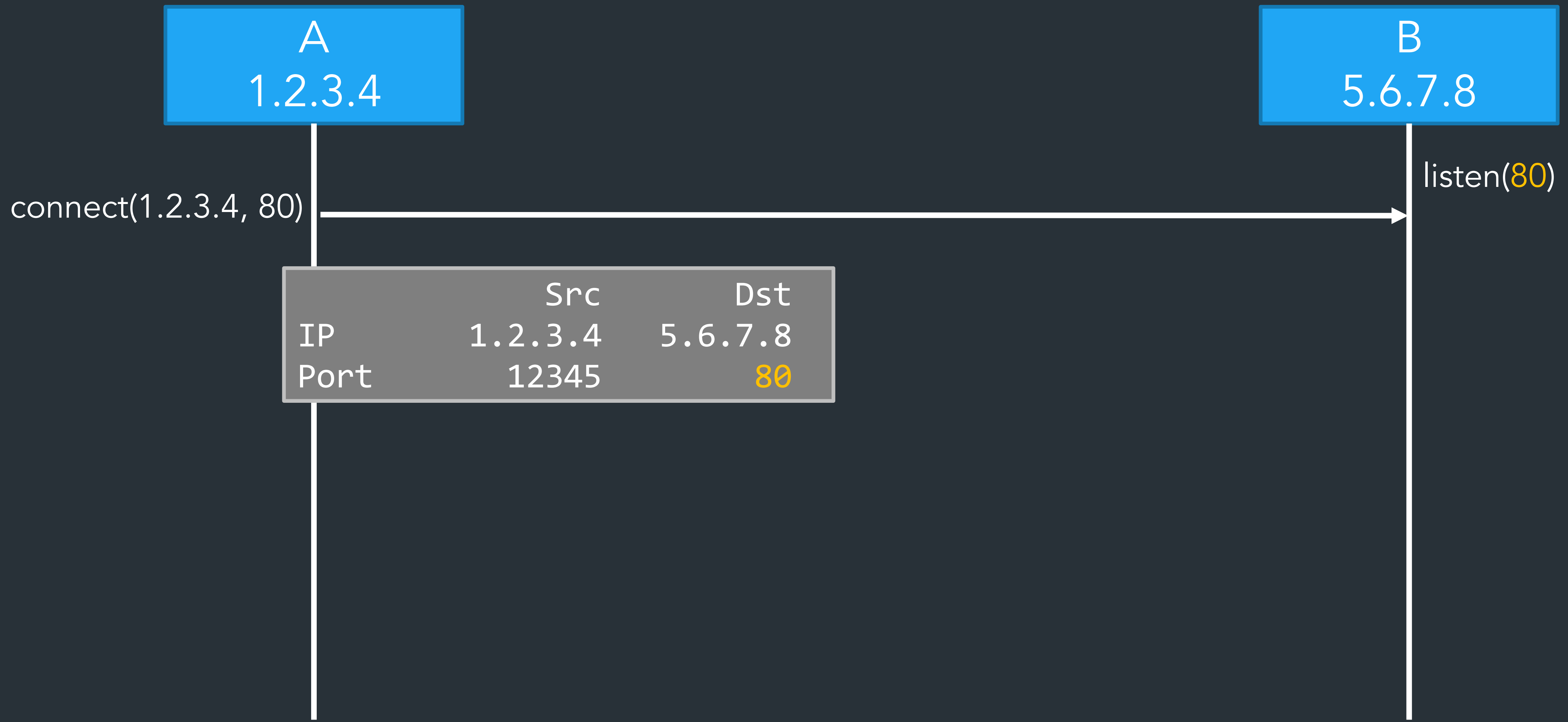
- "Normal" mode*: make a connection to another socket

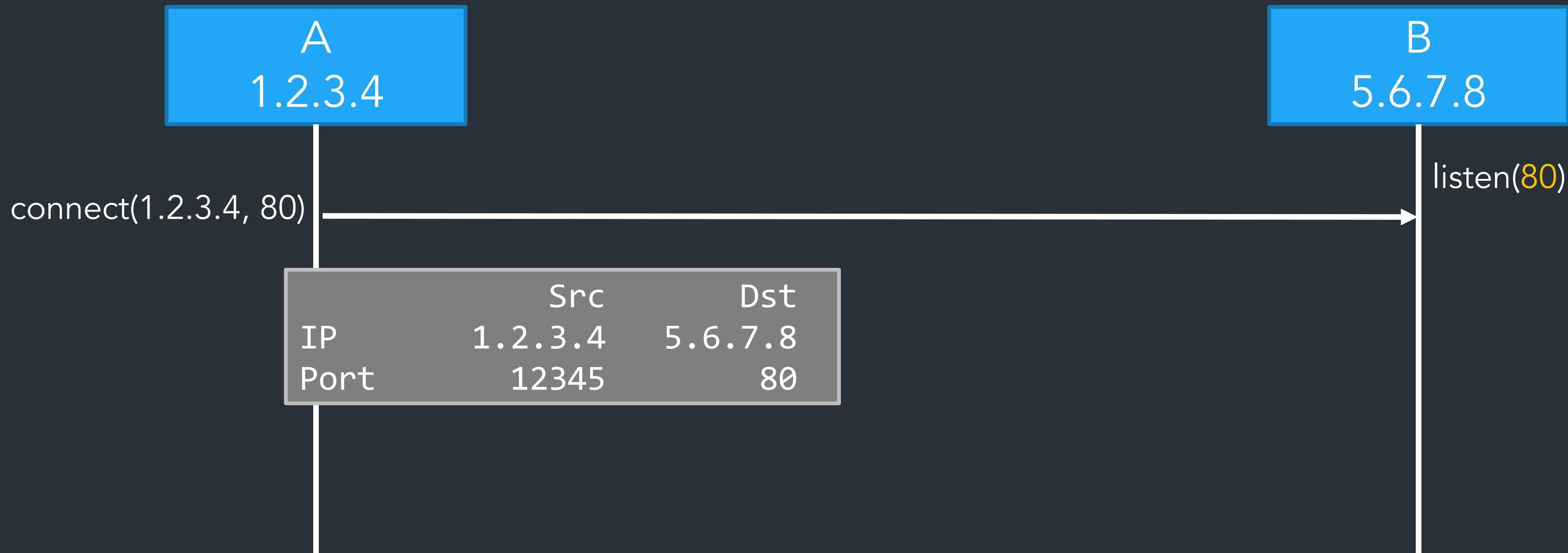
  => Used to make outgoing connections

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

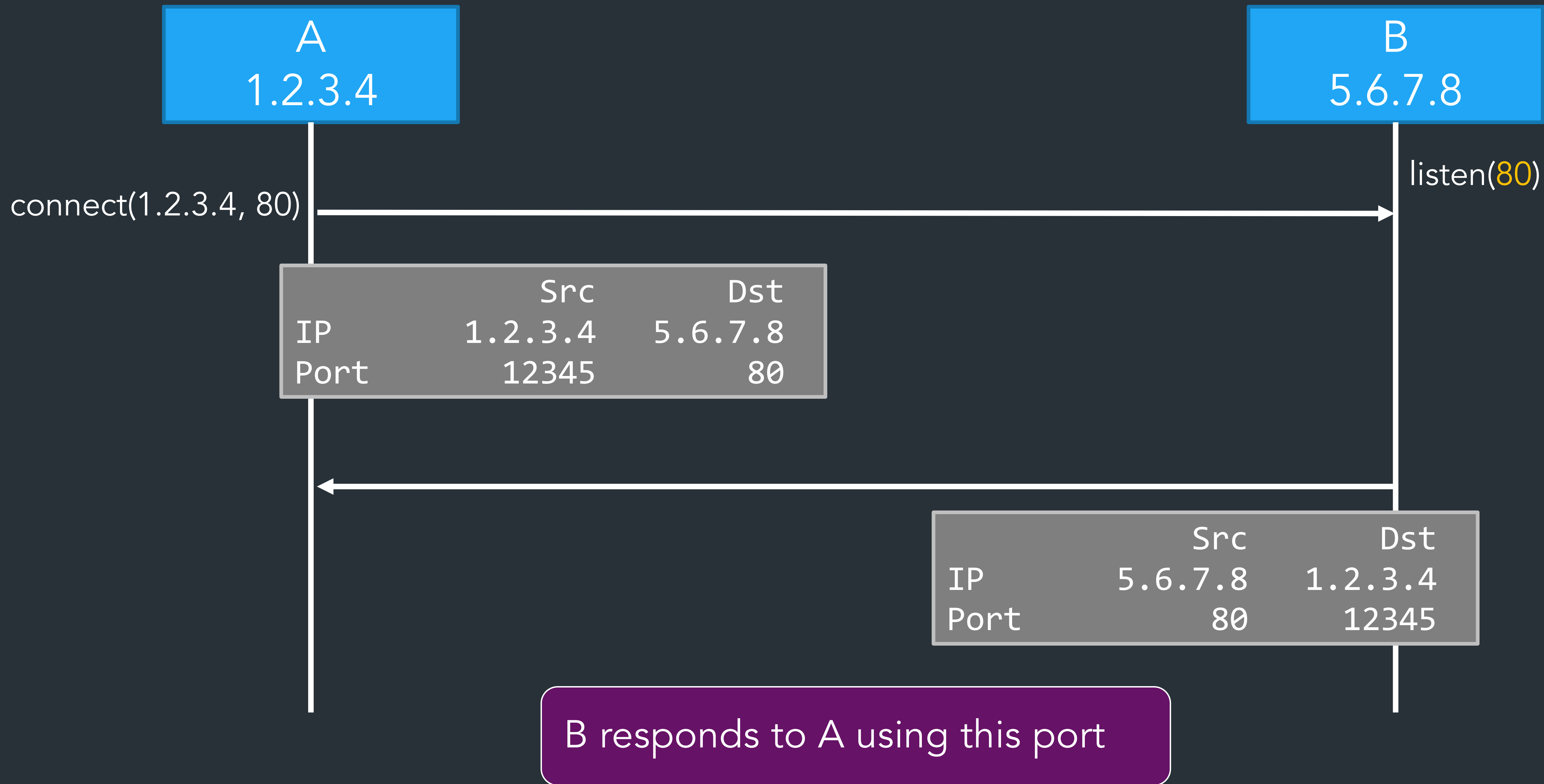
A
1.2.3.4

B
5.6.7.8


A
1.2.3.4
B
5.6.7.8
listen(80)
connect(1.2.3.4, 80)
Src
Dst
IP
1.2.3.4
5.6.7.8
Port
12345
80



- 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
1.2.3.4
B
5.6.7.8
listen(80)
connect(1.2.3.4, 80)
Src
Dst
IP
1.2.3.4
5.6.7.8
Port
12345
80
Src
Dst
IP
5.6.7.8
1.2.3.4
Port
80
12345
B responds to A using this port

Demo: netstat

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

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:

Proto	Local (yours)		Remote (theirs)		Socket
	IP	Port	IP	Port	
					(some struct)
					…

How to map packets to sockets?

Kernel table looks something like this:

Proto	Local (yours)		Remote (theirs)		Socket
	IP	Port	IP	Port	
tcp/udp	10.0.0.1	12345	1.2.3.4	80	(some struct)
	10.0.0.1	55444	5.6.7.8	443	(some struct)
…	…	…	…	…	…

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)

Proto	Local (yours)		Remote (theirs)		Socket
	IP	Port	IP	Port	
tcp	1.2.3.4	12345	5.6.7.8	80	(normal struct)
tcp	*	22	*	*	(listen struct)
...	...	...	...	...	...

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

=> For listen sockets, some fields may be blank

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

Netstat

```
deemer@vesta ~/Development % netstat -an
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  10.3.146.161.51094     104.16.248.249.443     ESTABLISHED
tcp4       0      0  10.3.146.161.51076     172.66.43.67.443       ESTABLISHED
tcp6       0      0  2620:6e:6000:900.51074 2606:4700:3108::.443   ESTABLISHED
tcp4       0      0  10.3.146.161.51065     35.82.230.35.443       ESTABLISHED
tcp4       0      0  10.3.146.161.51055     162.159.136.234.443    ESTABLISHED
tcp4       0      0  10.3.146.161.51038     17.57.147.5.5223       ESTABLISHED
tcp6       0      0  *.51036                *.*                    LISTEN
tcp4       0      0  *.51036                *.*                    LISTEN
tcp4       0      0  127.0.0.1.14500        *.*                    LISTEN
```

An interface to applications

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

Problems?

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

```
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

```
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 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…
  - …

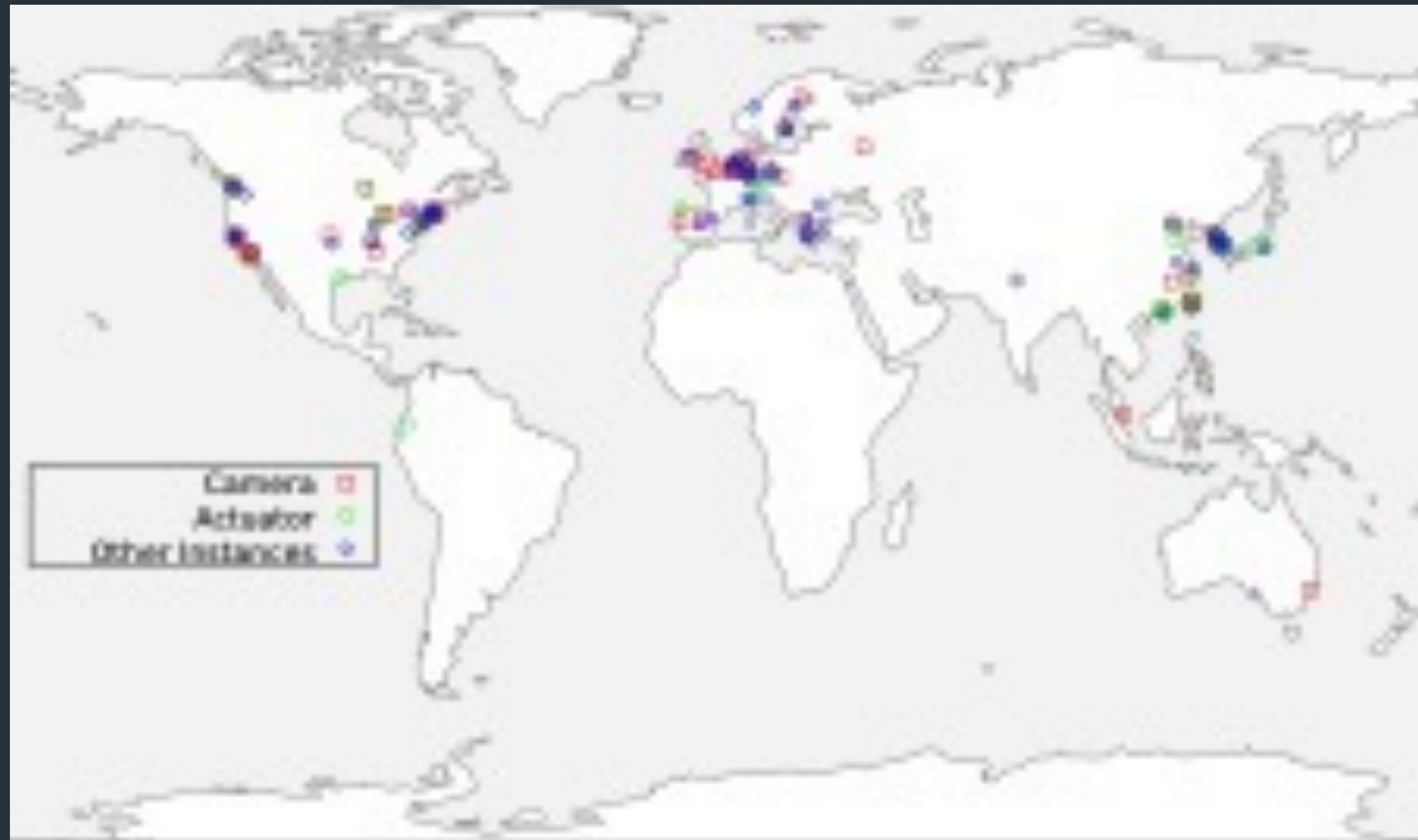
Disclaimer

- 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)



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)

From Lec 2: OSI Model