CSCI-1680
Transport Layer Warmup (ish)

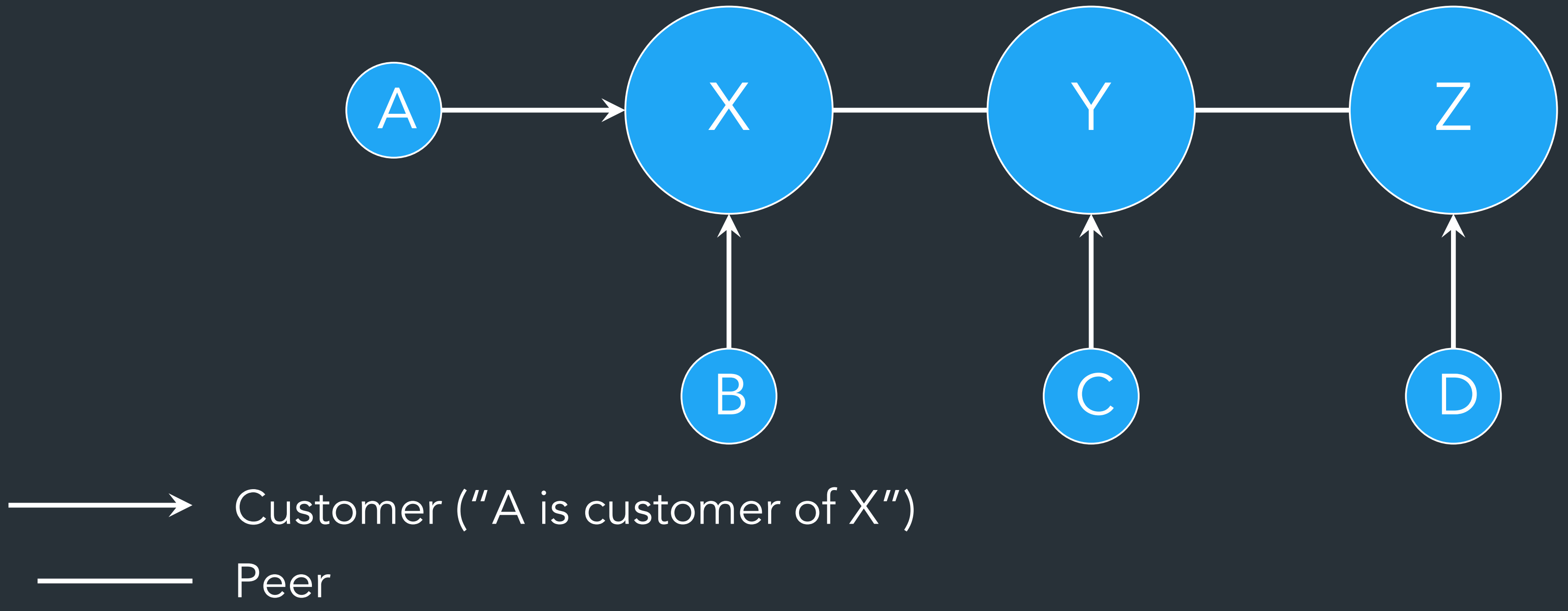
Nick DeMarinis

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?

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



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)

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)

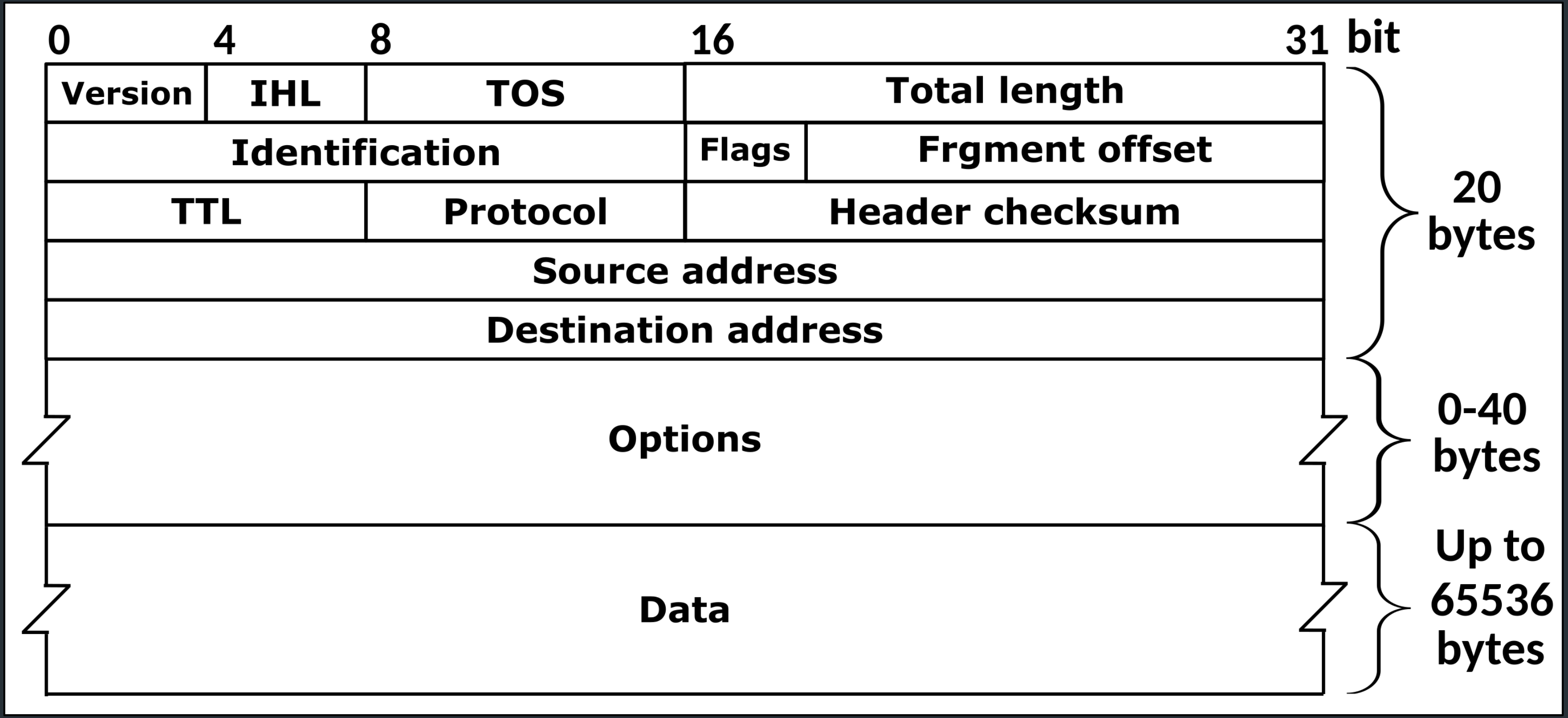
Ports & Sockets

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

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

The transport layer MAY provide…

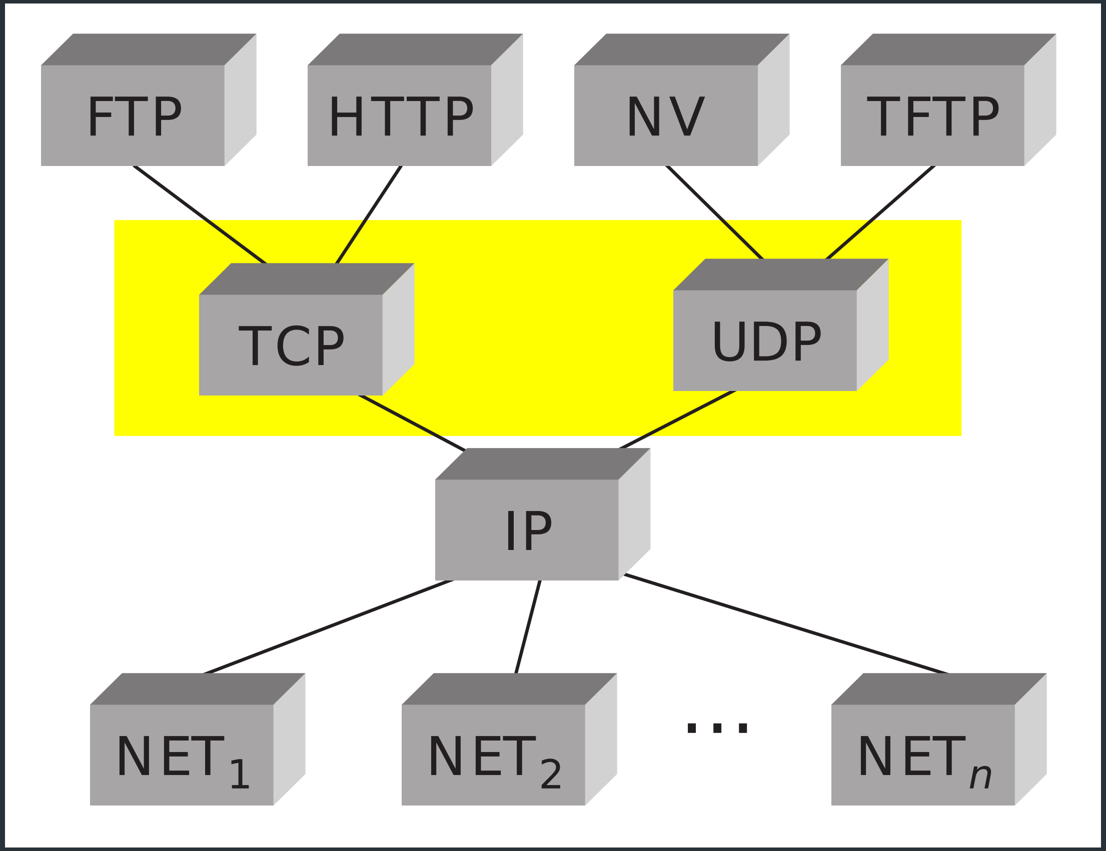
- Reliable data delivery
- Creating a data stream
- Managing throughput/sharing bandwidth
  - "Congestion control"

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

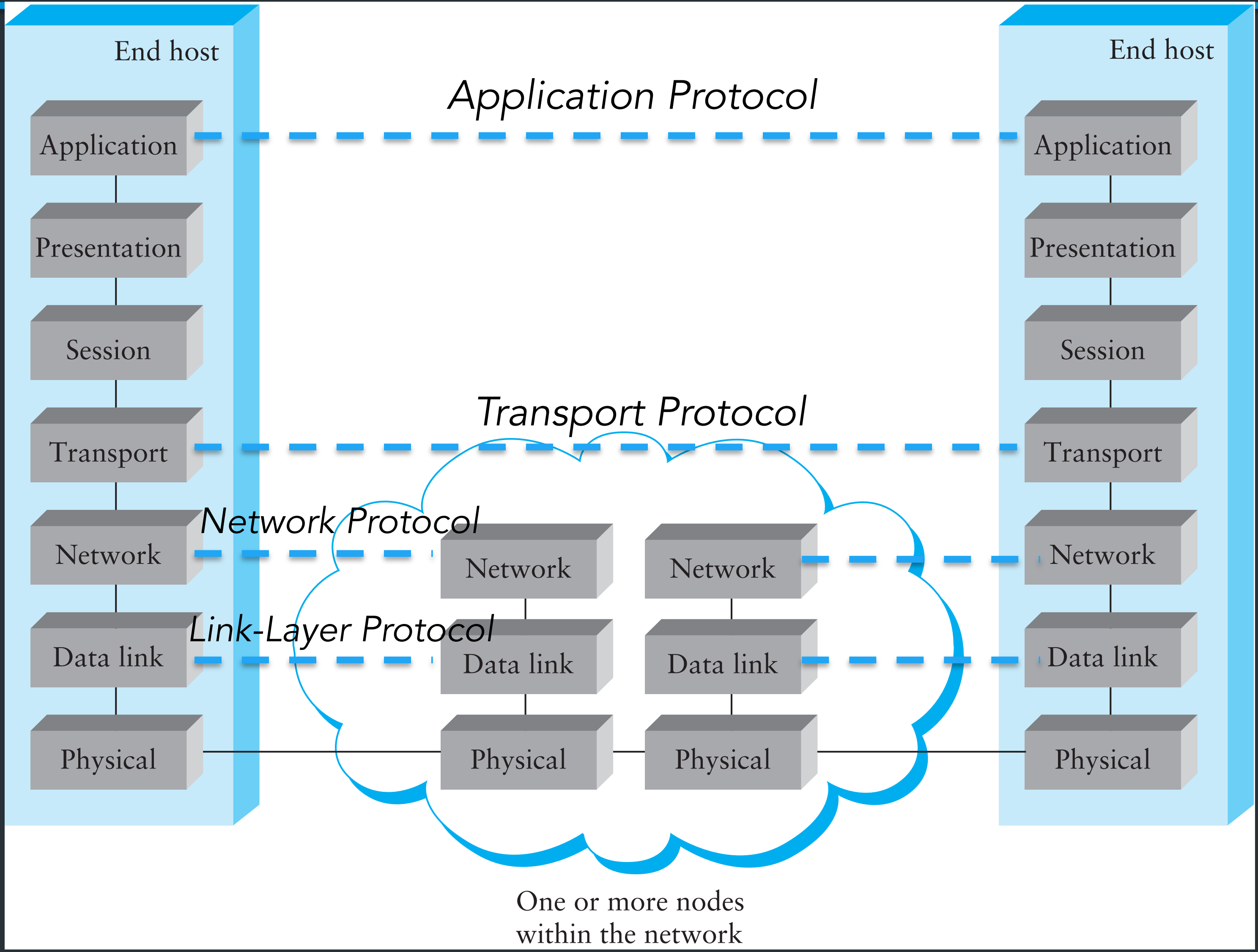
⇒ Not required for all transport layer (UDP has none of these)

⇒ Other protocols do this too (eg. QUIC)

Transport Layer



From Lec 2: OSI Model



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"
- port > 20000: "ephemeral ports", for general app use

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)

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

  (protocol, source IP, source port, dest IP, dest port) => 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
- "Outgoing" mode*: make a connection to another socket

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

How ports work

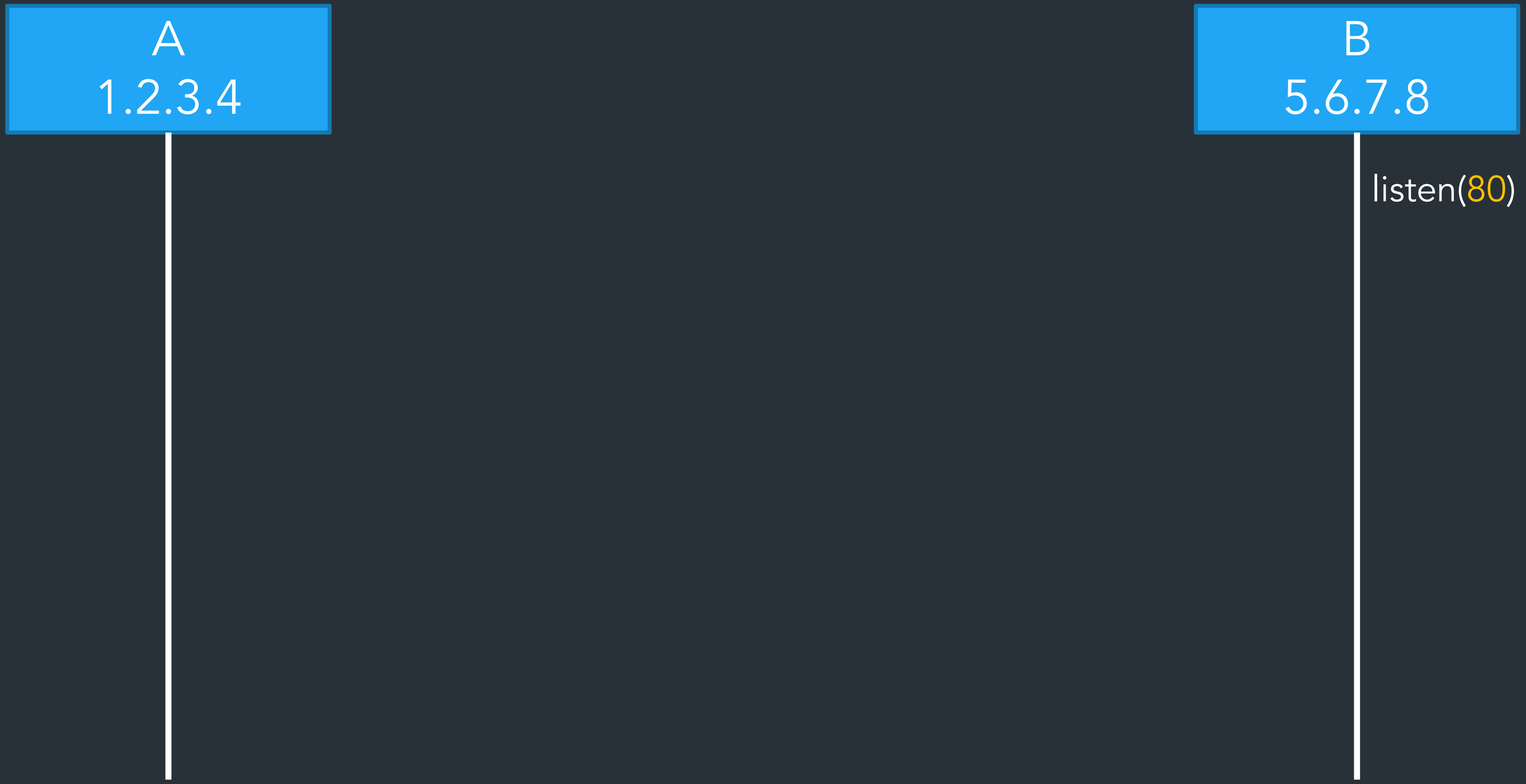
The kernel maps ports to *sockets*, which are used in applications like file descriptors to access the network

Two modes for using ports/sockets:

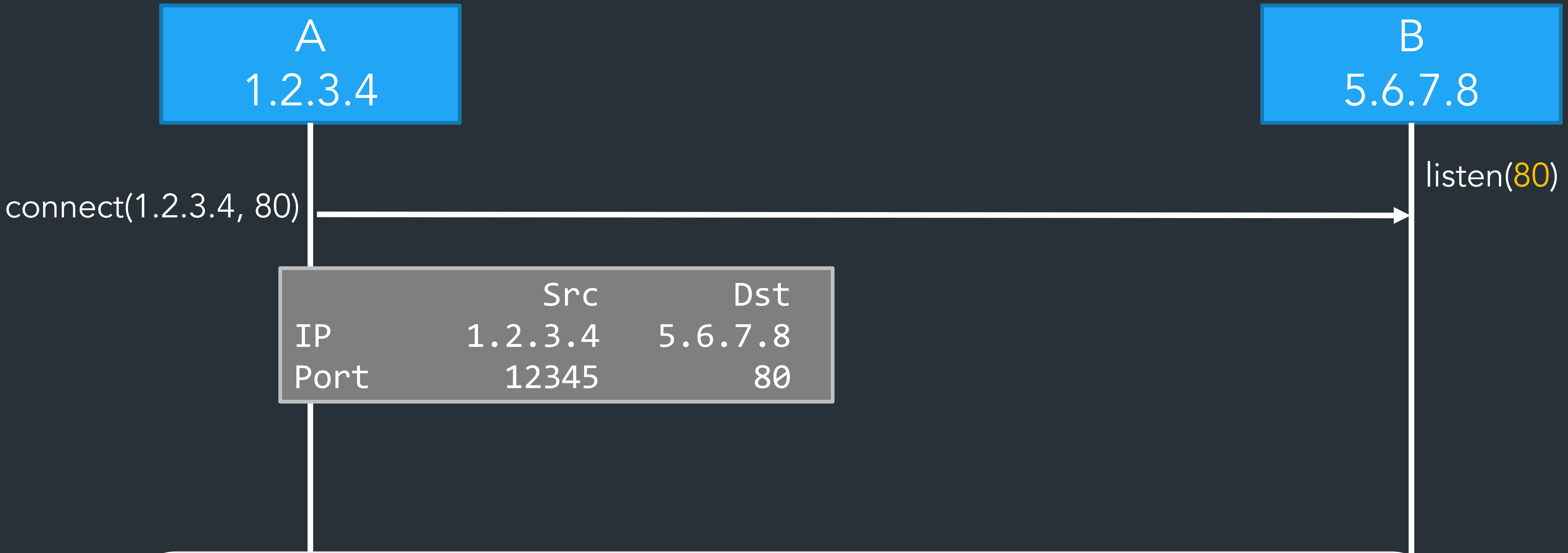
- Listen mode: apps "bind" to a port to accept new connections
  => Used to receive/wait for new connections

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

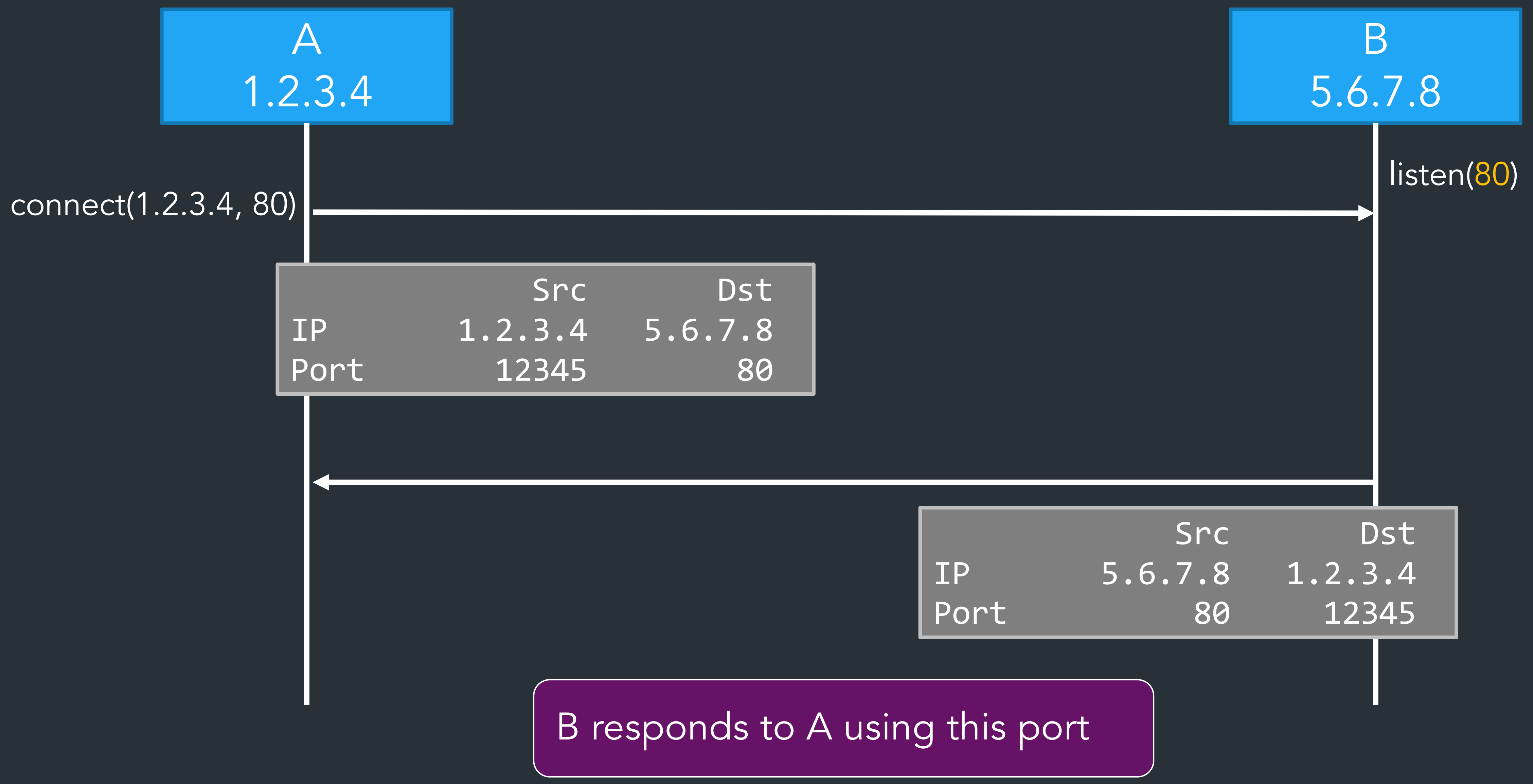
*: Nick made this term up so it has a name


A
1.2.3.4
B
5.6.7.8
listen(80)


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: 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:

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

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:

Proto	Local (yours)		Remote (theirs)		Socket
	IP	Port	IP	Port	
tcp	1.2.3.4	12345	5.6.7.8	80	(some struct)
					…
…	…	…	…	…	…

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

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
```

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)
…	…	…	…	…	…

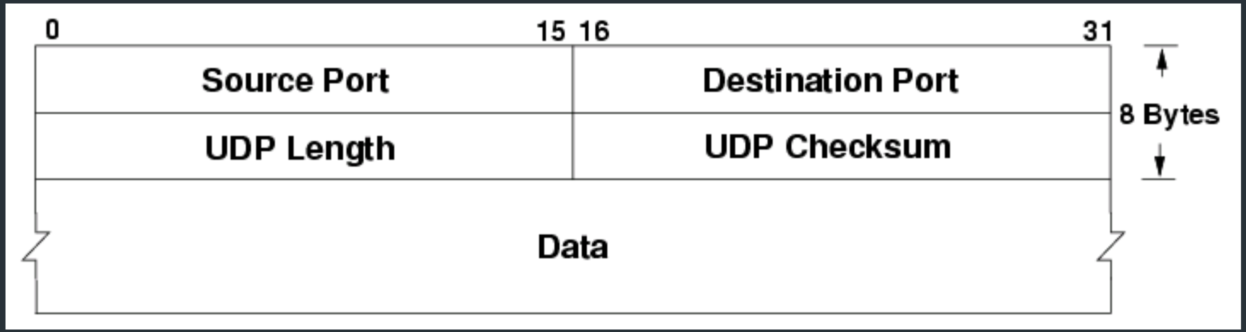
Key: 5-tuple of (local IP, local port, remote IP, remote port, protocol)

=> For listen sockets, some fields may be blank

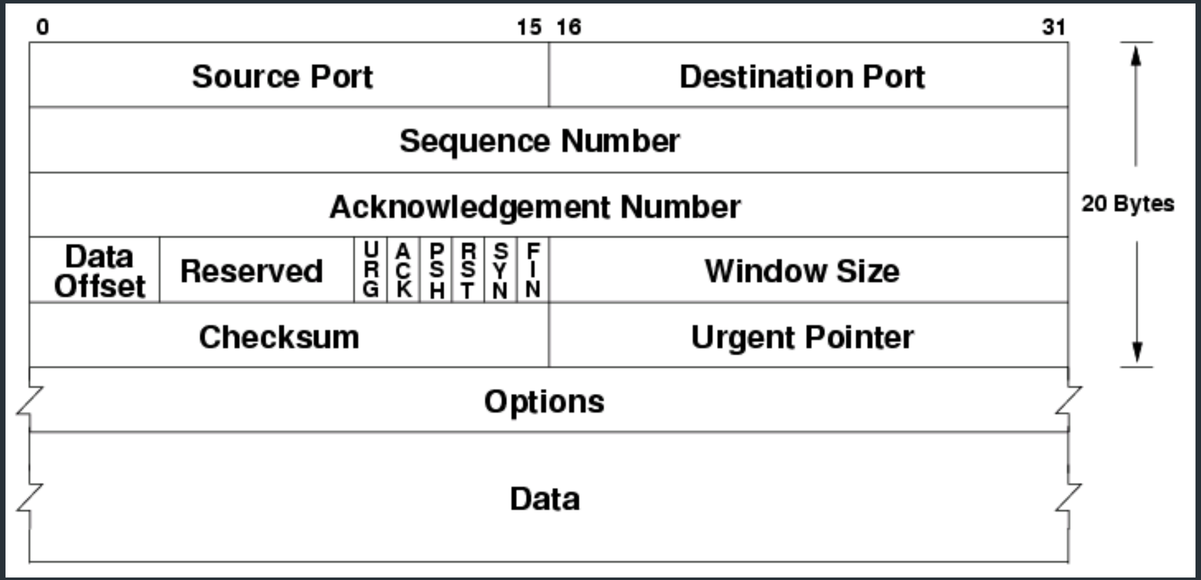
Value: kernel state for socket (state, buffers, …)

Ports are part of the transport layer

UDP



TCP



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!

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
```

$\Rightarrow$ Can discover things about the network
$\Rightarrow$ 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)