

Today

- Light intro to TCP
- TCP handshake

Administrivia

- IP grading: look for announcement later today
 - If you fix bugs while working on TCP => push and resubmit
- TCP out today
 - Look for announcement with handout
 - Look for email with a new stencil link (even if keeping same team)
 - Gearup: Thursday 3/12 5pm in CIT 165
 - Intro, how to think about Milestone I (due before spring break)

Lecture 13: TCP Intro



Warmup: You capture the following packets on your router. How many unique connections are present?

Pkt#	Proto	Source		Destination	
		IP	Port	IP	Port
P1	TCP	10.0.0.1	12345	1.2.3.4	80
P2	TCP	10.0.0.2	55444	5.6.7.8	443
P3	TCP	1.2.3.4	80	10.0.0.1	12345
P4	TCP	10.0.0.1	43233	1.2.3.4	80

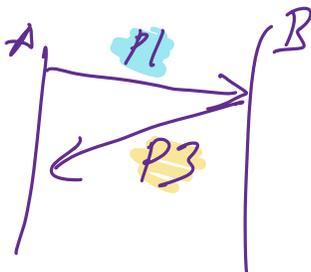
CONN#

1

2 DIFF IP, PORTS

1 RESPONSE TO P1

3



→ SAME HOSTS, BUT DIFFERENT SRC PORT ⇒ DIFFERENT CONN TO SAME DESTINATION (EG. NEW TAB)

5-tuple of (local IP, local port, remote IP, remote port, protocol number) uniquely identifies one connection

⇒ called a "flow"

Transport layer: the story so far

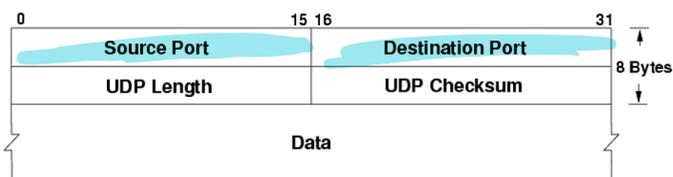
- Provides support for different applications via **ports**
- OS provides interface to applications via **sockets**



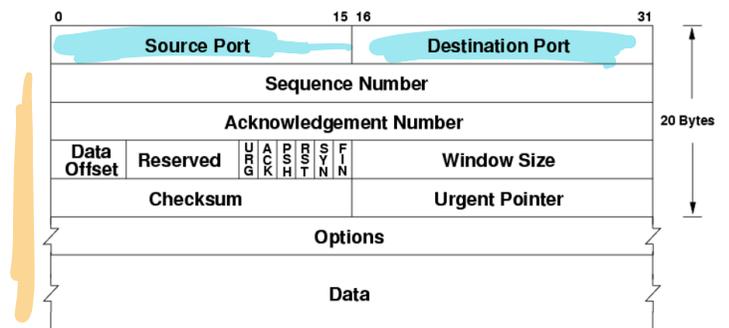
Transport layer is (usually) part of the OS => service provided to apps

The headers

UDP header



TCP header



Motivation: sending a big file

```
cp ~/dir/all-my-files.zip ~/some-other-dir
```

vs.

```
scp ~/dir/all-my-files.zip 1.2.3.4:/some/other/dir
```

Brainstorm: What are some of the challenges involved in implementing the network part?
What features do we need?

Starting point: just our SendIP function:

```
SendIP(destIP, protocol, []bytes)
```

Challenges / problems to solve?

- Need to know where data ends
- Packets may be dropped
- Packets may get corrupted (crumpled up)
- Packets may get reordered

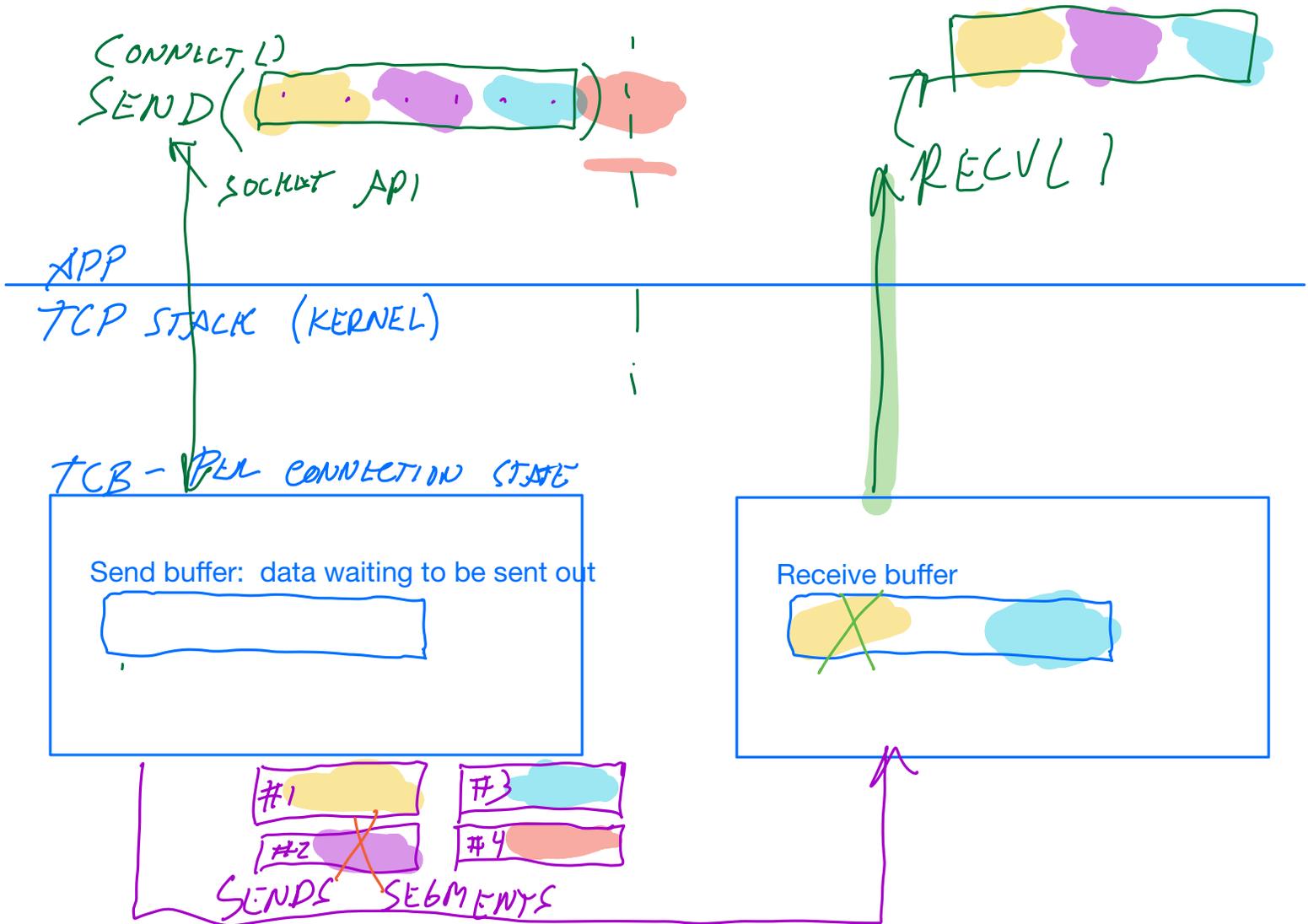
- Sender needs to know the data arrived
- Need to make sure sender doesn't overwhelm receiver
=> "flow control"
- Need to make sure we don't overwhelm networks
=> "congestion control"

Other great challenges from our discussion in class, but not handled by TCP:

- Packets might be intercepted (i.e., eavesdropping)
=> TCP doesn't solve this (instead, we deal with it at other layers--more info later!)

- Packets may take different paths => this part is handled by network layer, but may lead to reordering/dropped packets!

TCP: The Big Picture



Sending side

- Buffers data from app to be sent
- Divides data into segments
- Track which segments have been received, which have been dropped (retransmit on timeout)
- Flow control: if receiver has no more buffer space, stop sending

Receiving side

- Arrange segments in order in recv buffer
- Sends back "acknowledgements" + other info
- App "pulls" data from the receive buffer, frees up more space for data to arrive from network

In practice, both sides of the connection can send and receive (full-duplex)

=> Both sides have send and receive buffers

=> (Can use the same socket to send and receive)

Examples: in wireshark

[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[Timestamps]
[SEQ/ACK analysis]
[Client Contiguous Streams: 1]
[Server Contiguous Streams: 1]
TCP payload (2258 bytes)
TCP segment data (2258 bytes)
[7 Reassembled TCP Segments (18698 bytes): #233(2740), #234(2740), #235(2740), #236(2740), #241(2740), #252(2740), #253(2258)]
Hypertext Transfer Protocol
HTTP/1.1 200 OK\r\n
Response Version: HTTP/1.1

0330 79 74 65 73 0d 0a 0d 0a 40 69 6d 70 6f 72 74 20 bytes... @import
0340 75 72 6c 28 68 74 74 70 73 3a 2f 2f 63 64 6e 6a url(http s://cdnj
0350 73 2e 63 6c 6f 75 64 66 6c 61 72 65 2e 63 6f 6d s.cloudfl are.com
0360 2f 61 6a 61 78 2f 6c 69 62 73 2f 66 6f 6e 74 2d /ajax/li bs/font-
0370 61 77 65 73 6f 6d 65 2f 34 2e 37 2e 30 2f 63 73 awesome/ 4.7.0/cs
0380 73 2f 66 6f 6e 74 2d 61 77 65 73 6f 6d 65 2e 6d s/font-a wesome.m
0390 69 6e 2e 63 73 73 29 3b 40 69 6d 70 6f 72 74 20 in.css); @import
03a0 75 72 6c 28 68 74 74 70 73 3a 2f 2f 66 6f 6e 74 url(http s://font
03b0 73 2e 67 6f 6f 67 6c 65 61 70 69 73 2e 63 6f 6d s.google apis.com
03c0 2f 63 73 73 3f 66 61 6d 69 6c 79 3d 53 6f 75 72 /css?fam ily=Sour
03d0 63 65 2b 53 61 6e 73 2b 50 72 6f 3a 34 30 30 2c ce+Sans+ Pro:400,
03e0 34 30 30 69 74 61 6c 69 63 2c 36 30 30 2c 36 30 400itali c,600,60
03f0 30 69 74 61 6c 69 63 2c 37 30 30 2c 39 30 30 29 @italic, 700,900)

Packet (2324 bytes) Reassembled TCP (18698 bytes)

From wireshark: this HTTP packet was composed from these TCP segments

(This particular HTTP message contained some Javascript code for a web page. We'll talk more about HTTP later)

Frame 253: Packet, 2324 bytes on wire (18592 bits), 2324 bytes captured (18592 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: Cisco_55:d5:02 (d0:85:43:55:d5:02), Dst: Intel_ac:68:8b (dc:46:28:ac:68:8b)
Internet Protocol Version 4, Src: 146.190.62.39, Dst: 128.148.140.229
Transmission Control Protocol, Src Port: 80, Dst Port: 34642, Seq: 21043, Ack: 994, Len: 2258
Source Port: 80
Destination Port: 34642
[Stream index: 8]
[Stream Packet Number: 25]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 2258]
Sequence Number: 21043 (relative sequence number)
Sequence Number (raw): 1147449151
[Next Sequence Number: 23301 (relative sequence number)]
Acknowledgment Number: 994 (relative ack number)
Acknowledgment number (raw): 2988642979
1000 = Header Length: 32 bytes (8)
Flags: 0x018 (PSH, ACK)
Window: 503
[Calculated window size: 64384]
[Window size scaling factor: 128]
Checksum: 0xe757 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[Timestamps]
[SEQ/ACK analysis]

A TCP header: encapsulated inside IP packet!
(More on what the fields mean on next few pages)

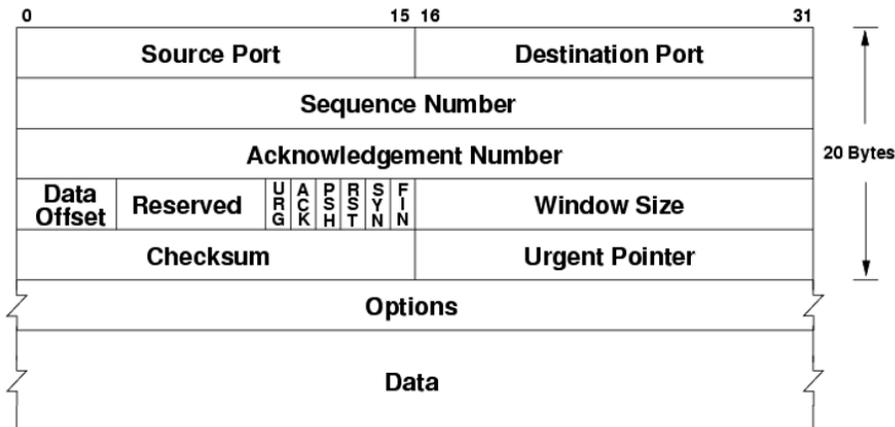
Key features

- Initially defined by: **RFC 793 (1981)** (+ many others now)
- Creates concepts of connections between two endpoints
 - Each connection has its own state
- "Reliable, connection-oriented, byte stream" (we will unpack what this means)

Special note: for project, you should read the (newer) RFC9293 instead. We'll talk about why next class.

ORDERED SEQUENCE OF BYTES

The header



Most important TCP fields:

- Port numbers: handle multiplexing (what connection this is)
- Sequence number (SEQ): where segment is in the data stream (in bytes)
- Acknowledgement number (ACK): the next sequence number expected (also in bytes)
Example: "I have up to byte 1024, give me byte 1025 next"
- Window: how much data you are willing to receive (how much space is in the receive buffer)
- Flags: status bits that indicate where we are in the connection
 - SYN, ACK, FIN, RST,

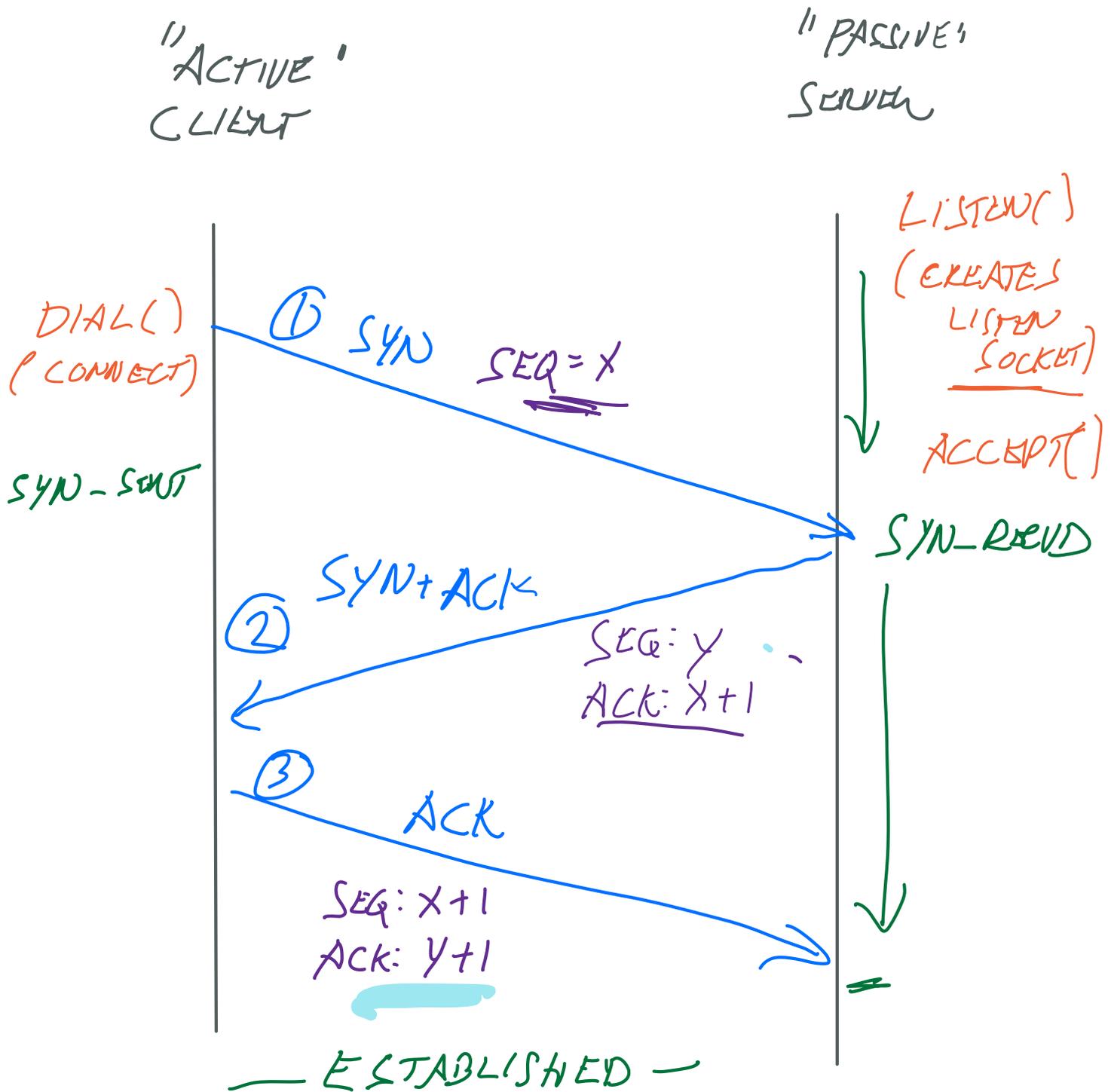
~~Example 1~~

At the socket level: sending a big file

```
func sender() {  
    fd, _ := os.Open("all-my-files.zip")  
  
    conn, _ := net.Dial("1.2.3.4:80")  
    → CONNECT()  
  
    buf := ReadTheWholeFile(fd)  
    conn.Write(buf)  
}
```

```
func receiver() {  
    listenConn, err := net.Listen(":80")  
    clientConn, err := listenConn.Accept()  
  
    buf := make([]byte, . . .)  
    conn.Read(buf)  
  
    fd = os.Open("copy-of-files.zip")  
    fd.Write(buf)  
}
```

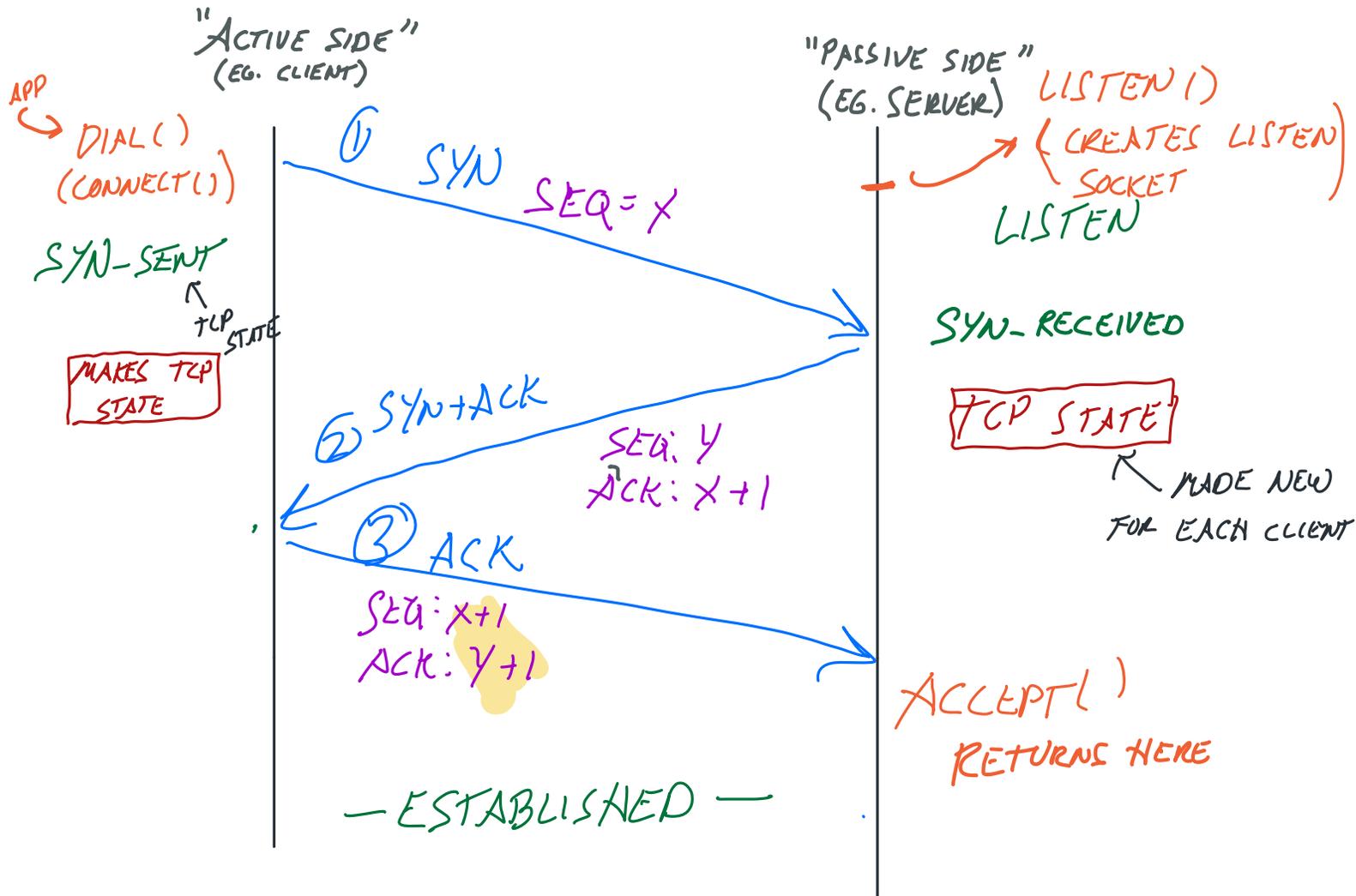
Establishing a connection: The TCP Handshake



What has happened

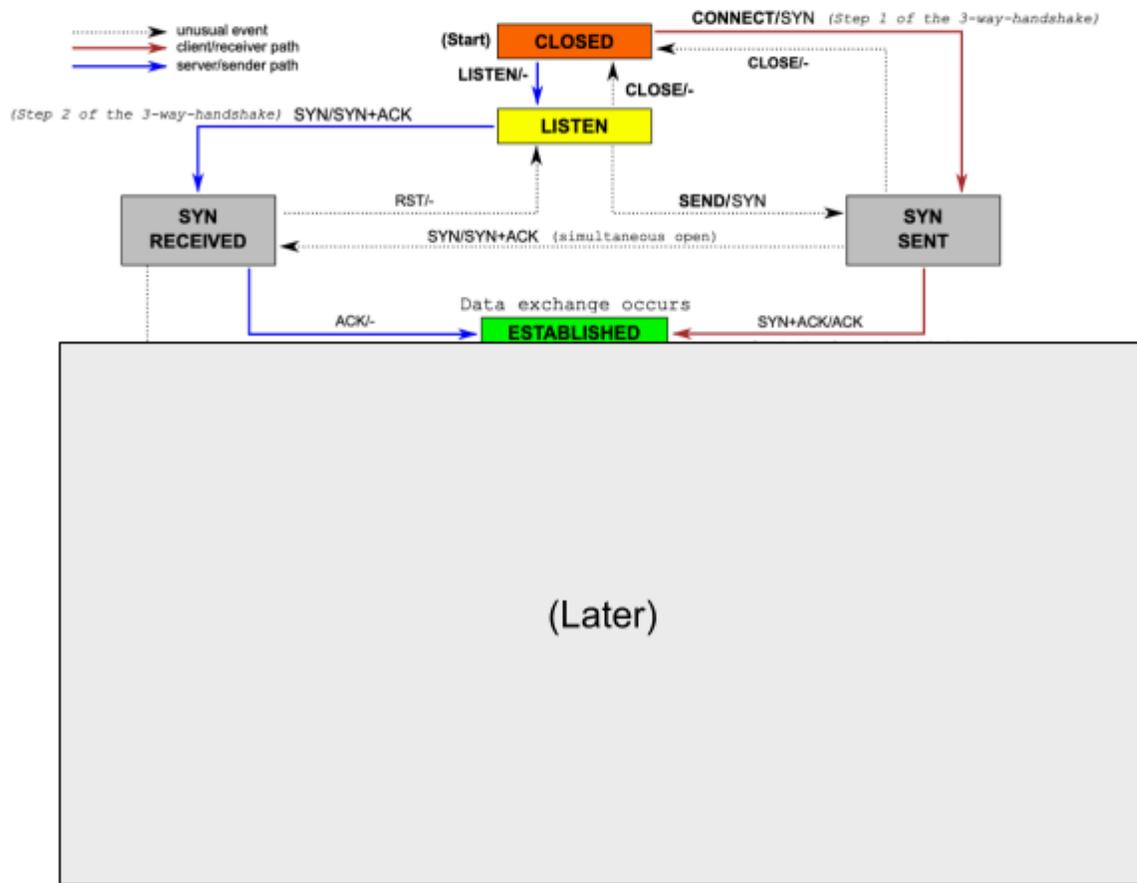
- Sender sends something to receiver
- Both sides agree on a starting sequence number
- Both sides in ESTABLISHED state
- Both sides have a socket representing this connection

Establishing a connection: The TCP Handshake



1. Sender sends SYN with random sequence number X
2. Receiver sends SYN+ACK with its own random sequence number Y, acknowledges sender's sequence number with ACK=X+1
3. Sender acknowledges receiver's sequence num with ACK for Y+1 (packet also has SEQ=X+1, since it comes after packet (1))

⇒ 3-WAY HANDSHAKE



(from [wikipedia](https://en.wikipedia.org/wiki/TCP))

Notation:

- (Event that happens)/(packet sent in response)
- **Bold** indicates Socket API calls (e.g., syscalls)

Sequence numbers: why is the initial sequence number random?

Initial Sequence Number (ISN) => each side picks their own

- Need to make sure sequence numbers aren't reused if host re-uses source port number
- Security reasons: need to make sure adversary can't guess start of connection

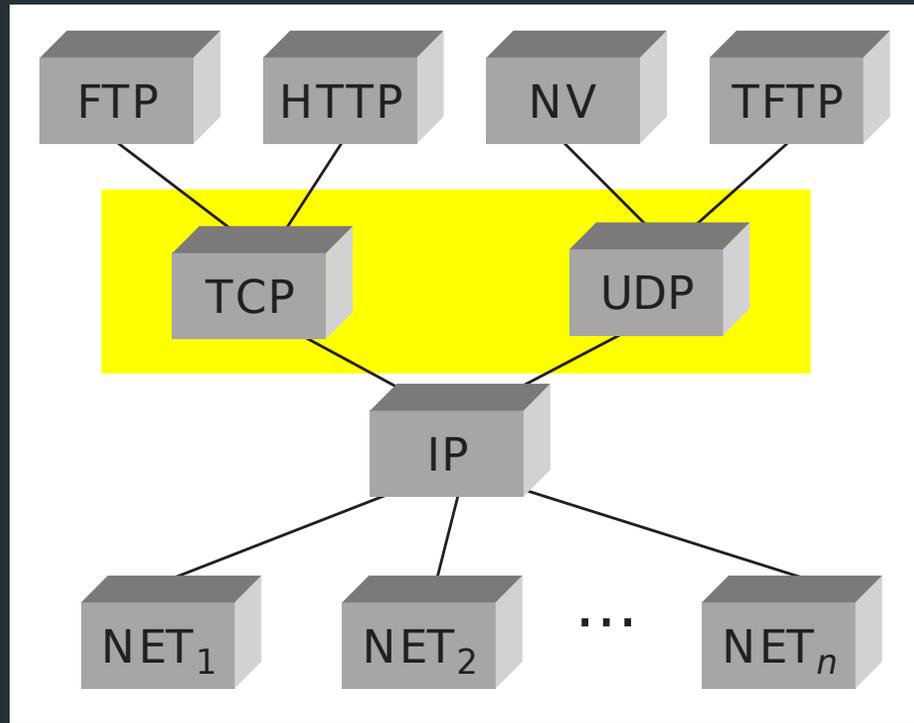
Official way: RFC9293 Sec 3.4.1 describes a procedure based on a timer and a cryptographic hash

=> for our project, just pick a random integer :)

Relative sequence numbering in Wireshark:

> Ethernet II, Src: Apple_cd:6a:23 (c8:89:f3:cd:6a:23), Dst: IntelCor_63:c4:45 (0	0000	00 1b 21 63 c4 45 c8 89 f3 cd 6a 23 08 00 45 00
> Internet Protocol Version 4, Src: 172.17.48.156, Dst: 172.17.48.22	0010	00 40 00 00 40 00 40 06 81 e3 ac 11 30 9c ac 11
✓ Transmission Control Protocol, Src Port: 49719, Dst Port: 22, Seq: 0, Len: 0	0020	30 16 c2 37 00 16 77 42 38 e5 00 00 00 00 b0 02
Source Port: 49719	0030	ff ff b7 2a 00 00 02 04 05 b4 01 03 03 06 01 01
Destination Port: 22	0040	08 0a 0d c7 46 c0 00 00 00 00 04 02 00 00
[Stream index: 8]		
[Conversation completeness: Complete, WITH_DATA (31)]		
[TCP Segment Len: 0]		
Sequence Number: 0 (relative sequence number)		
Sequence Number (raw): 2000828645		
[Next Sequence Number: 1 (relative sequence number)]		
Acknowledgment Number: 0		
Acknowledgment number (raw): 0		
1011 = Header Length: 44 bytes (11)		
> Flags: 0x002 (SYN)		

Next few pages contain some older notes -- these cover the same content, but have some more pictures. Read on if you're interested / want more info!



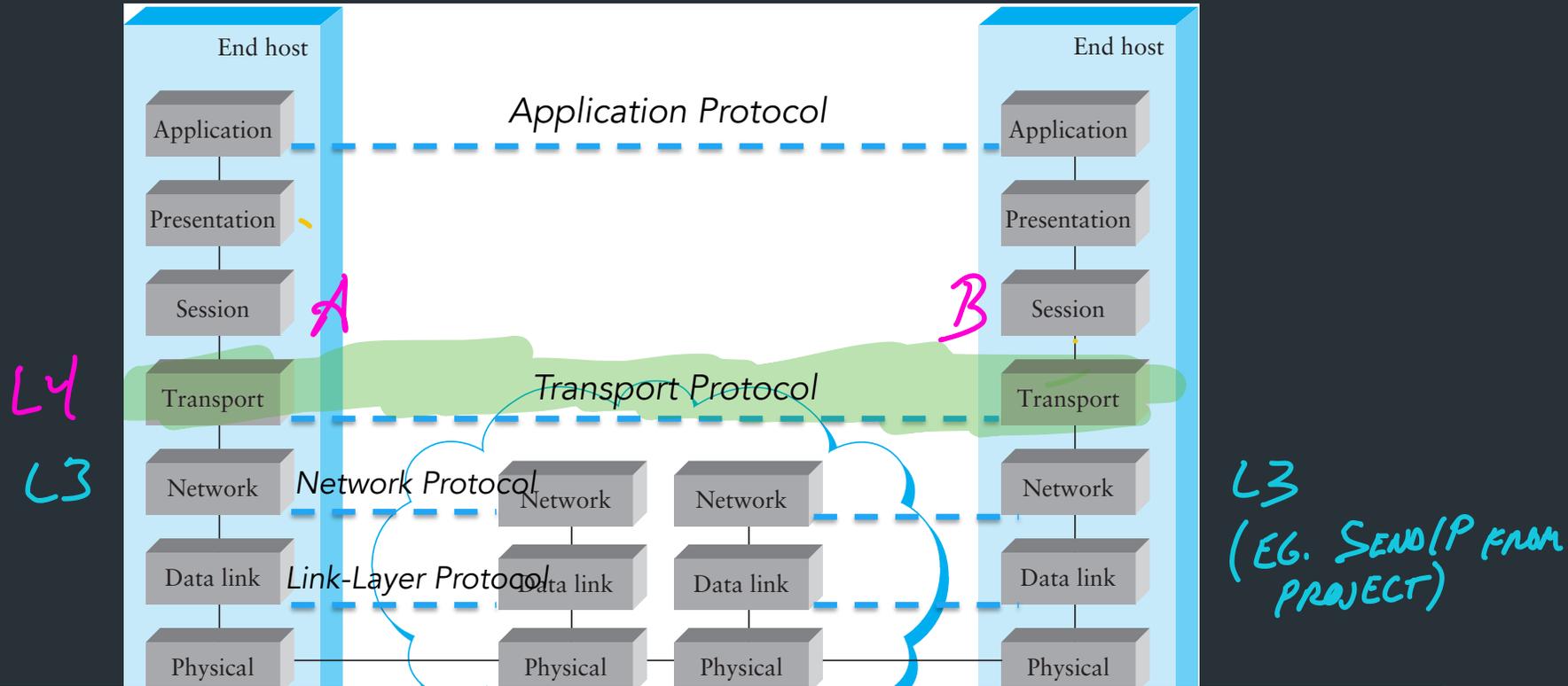
Transport layer: the story so far

- Provides support for different applications via ports
- OS provides interface to applications via sockets

IDEA OF
CONNECTIONS
FOR APPS
✓

⇒ For now: transport layer is part of OS, service provided to apps

From Lec 2: OSI Model

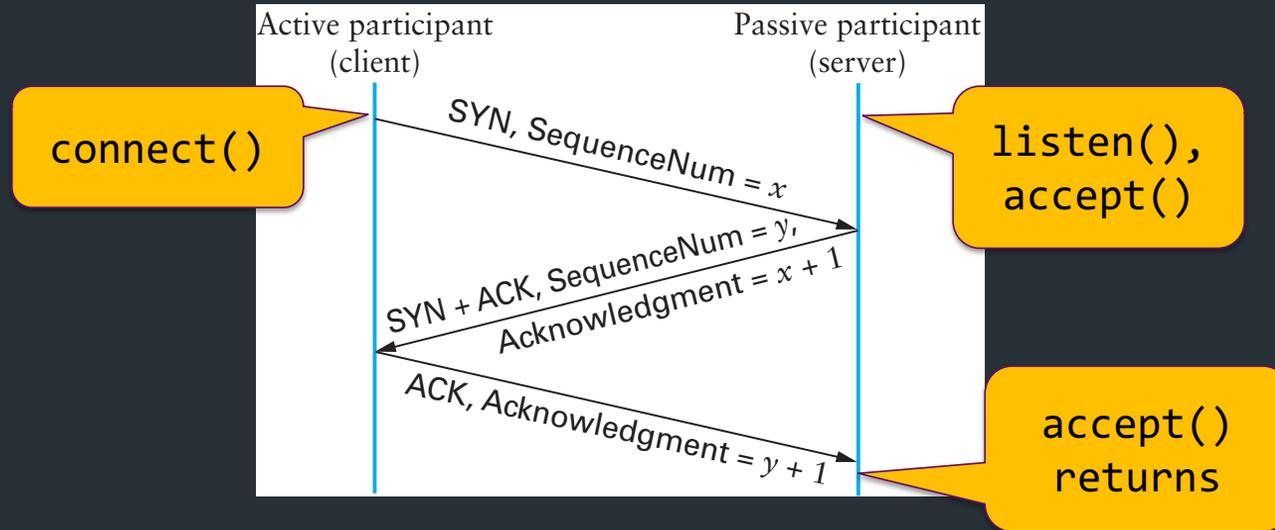


New abstraction: builds on network layer

=> Just care about endpoint, not individual routers on path

=> All mechanisms run on endpoints (not routers) => Why?

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
- If a 5-tuple is reused => new ISN, so sequence numbers likely out of range from past connection