

CS2200

Systems and Networks

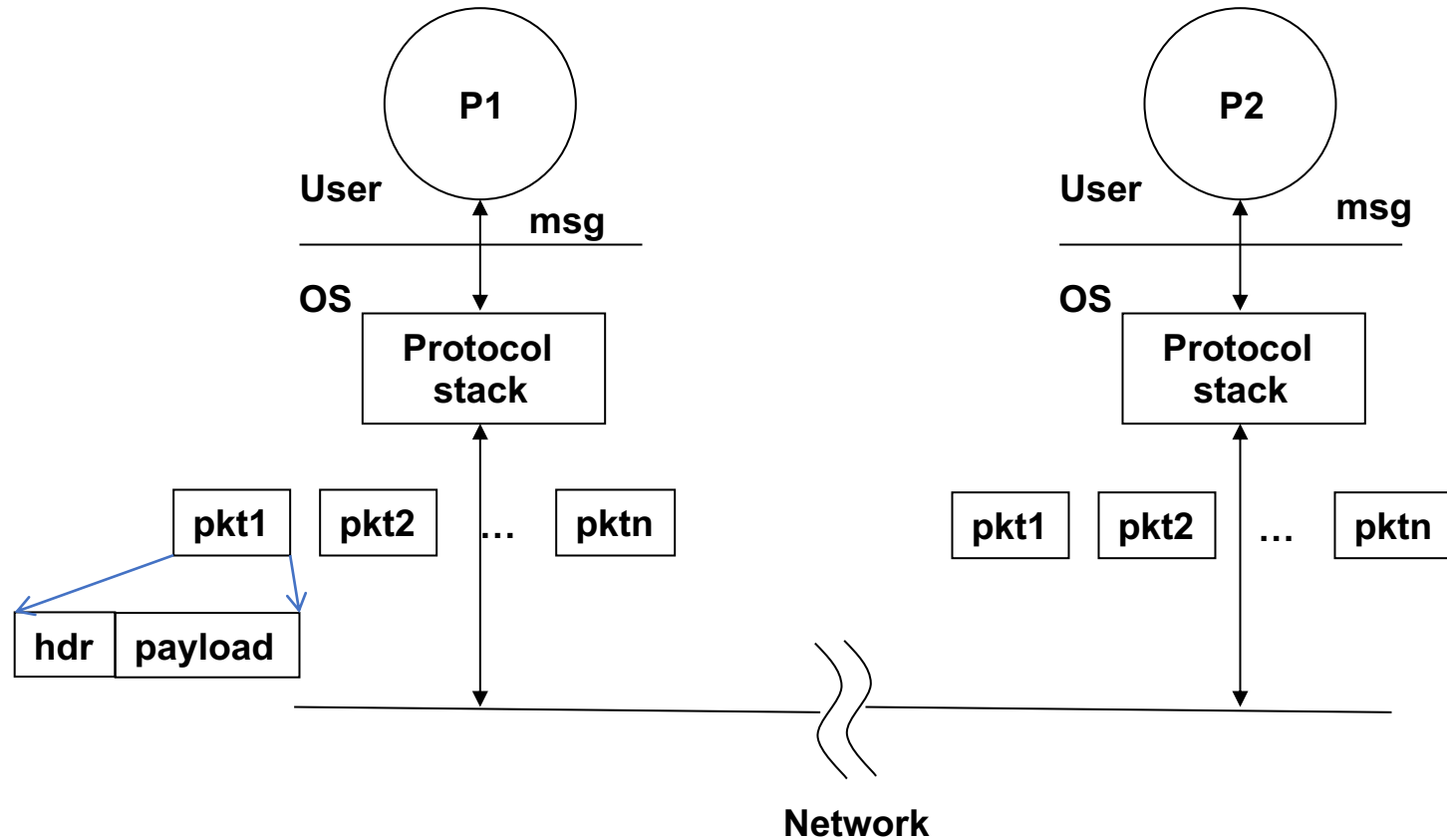
Spring 2022

Lecture 24:

Networking

Alexandros (Alex) Daglis
School of Computer Science
Georgia Institute of Technology
adaglis@gatech.edu

IP network connections

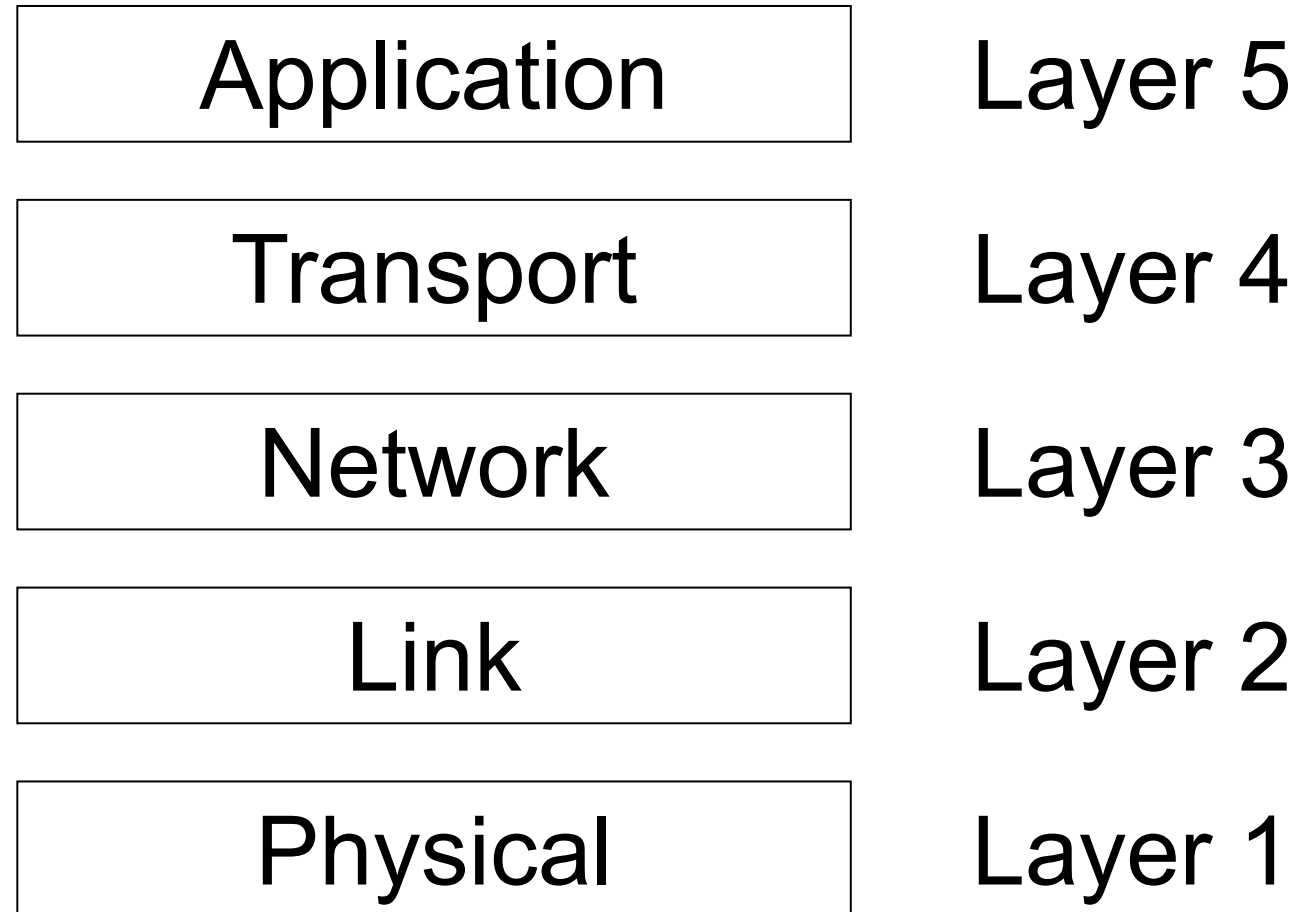


Big deal: Note that the Protocol Stacks are in the hosts, not in the middle of the Network

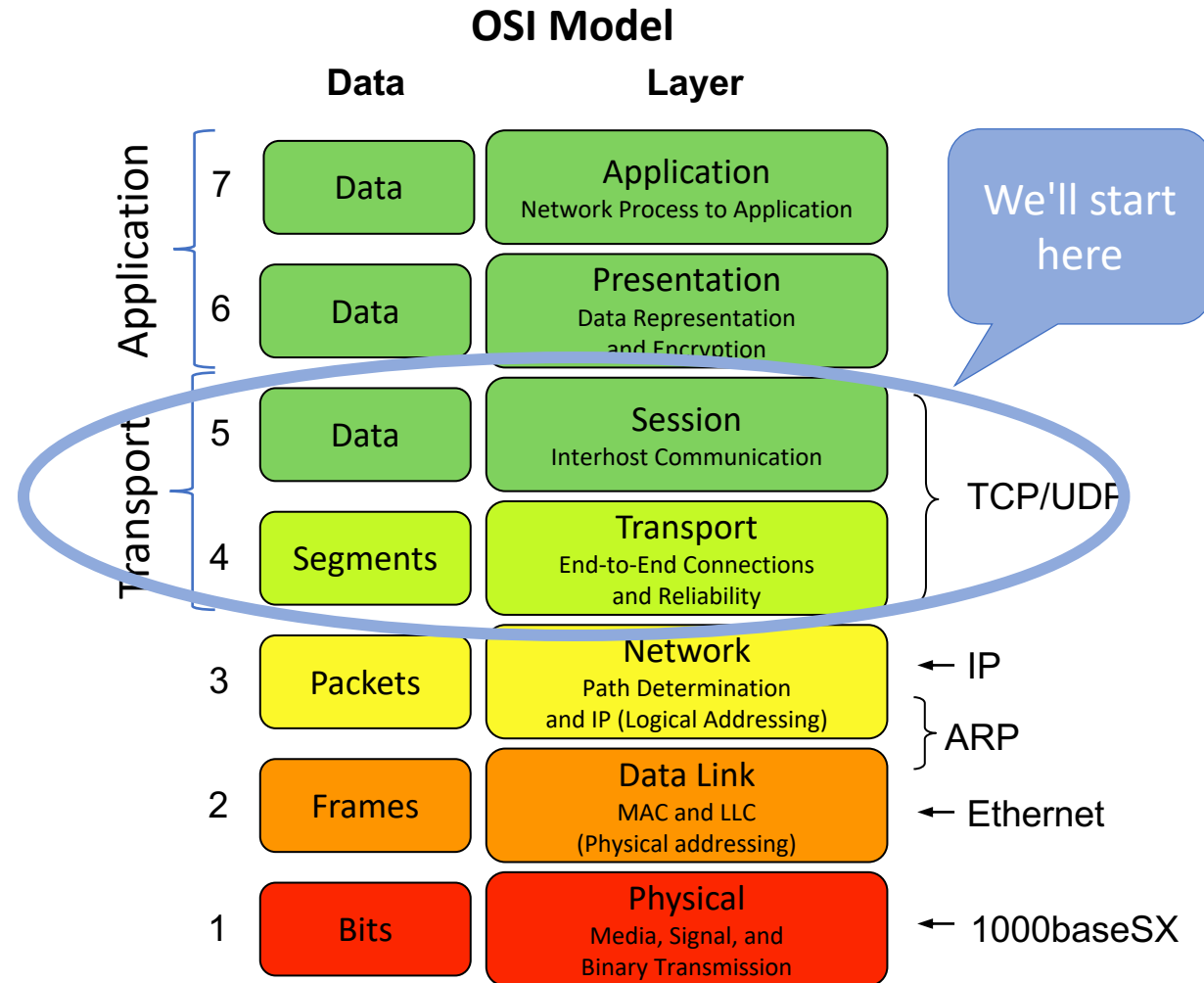
Why does IP need a protocol stack?

- Arbitrary message sizes
- Out of order delivery
- Packet loss
- Bit errors
- Queuing delays

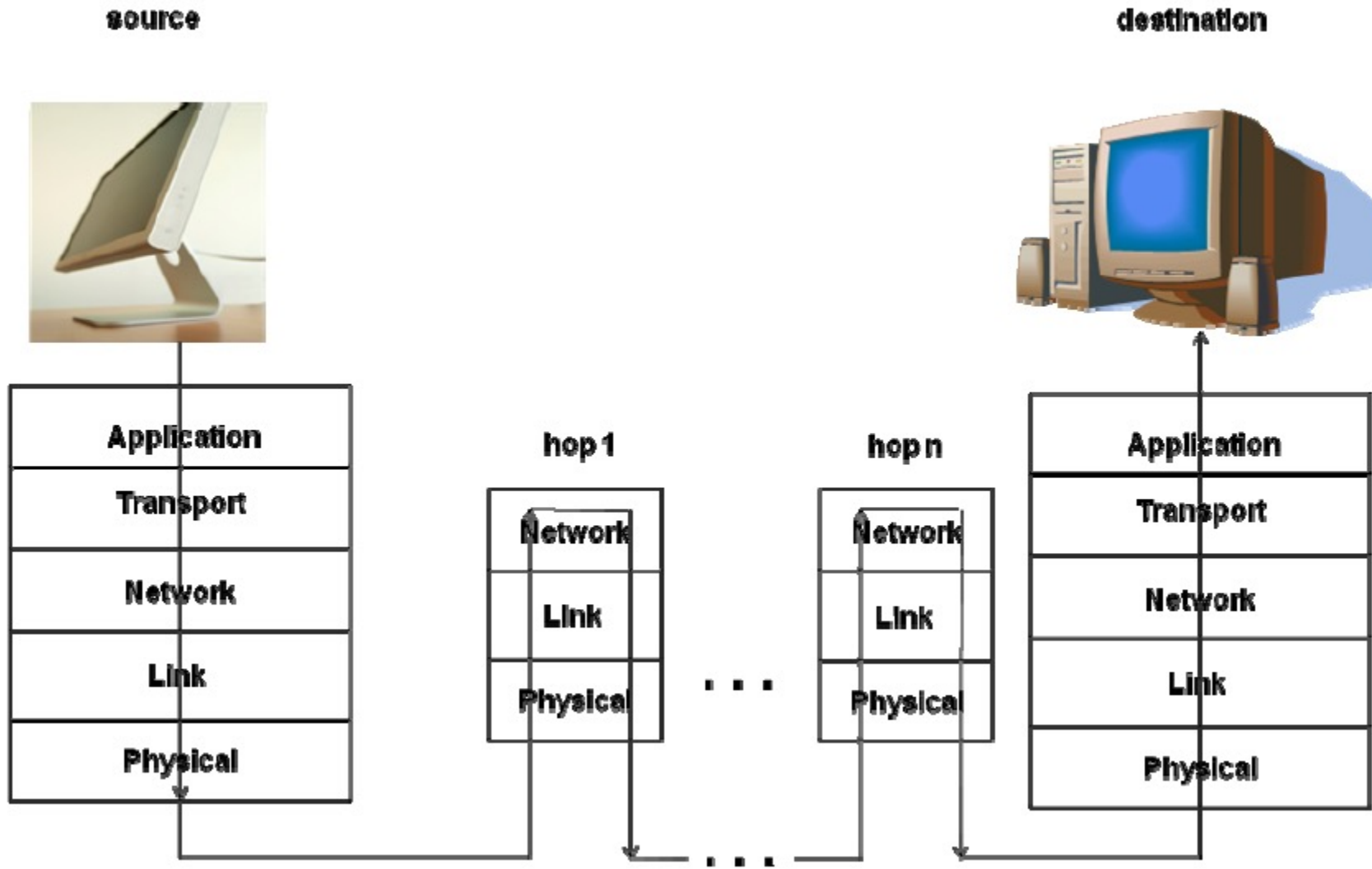
Internet Protocol Stack



The OSI Model



Getting from here to there...

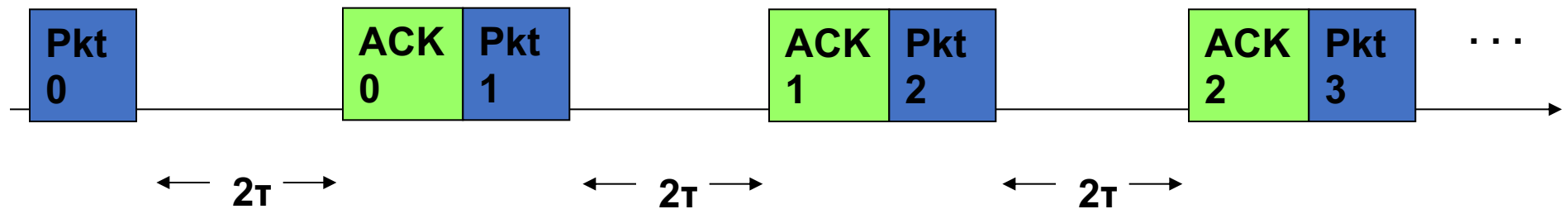


Transport (session) protocol

We're going to talk about these protocol types

- Stop and wait
- Pipelined transmission
- Reliable pipelined transmission

Stop and wait (for ack)



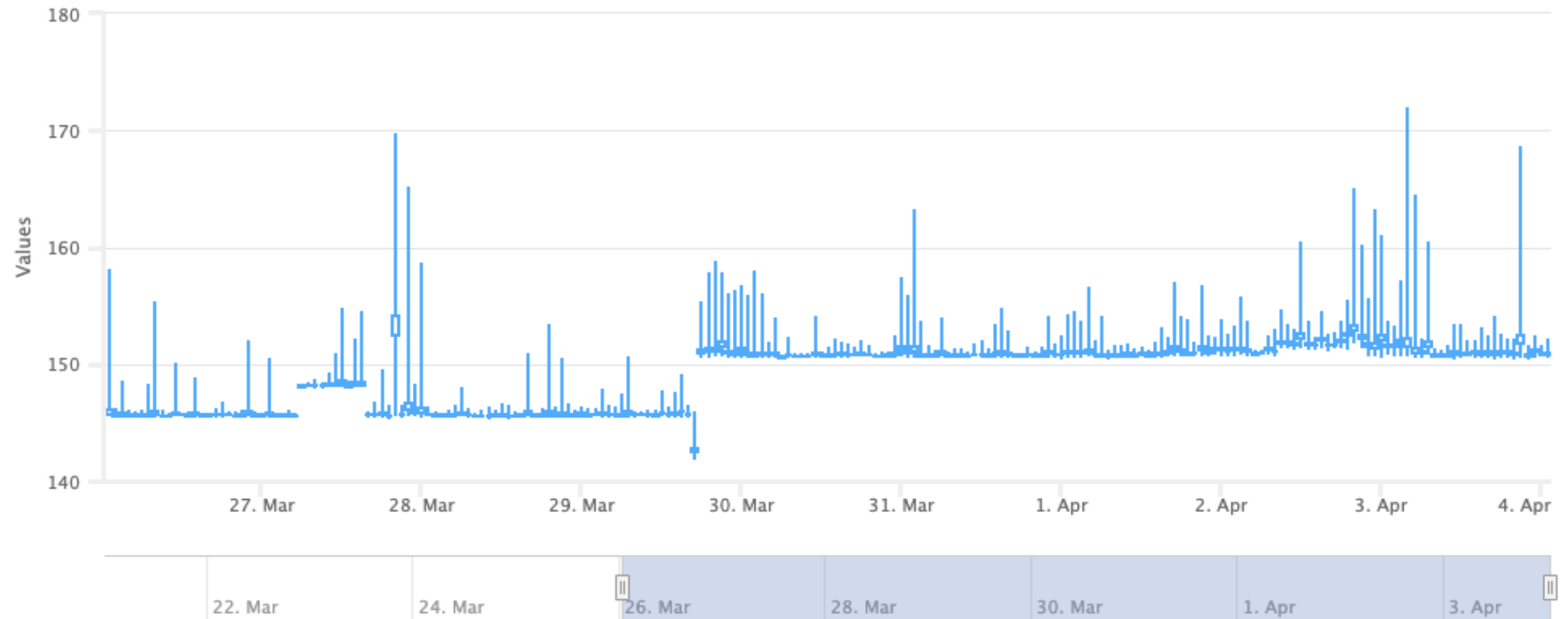
Timeline of sender

Latency

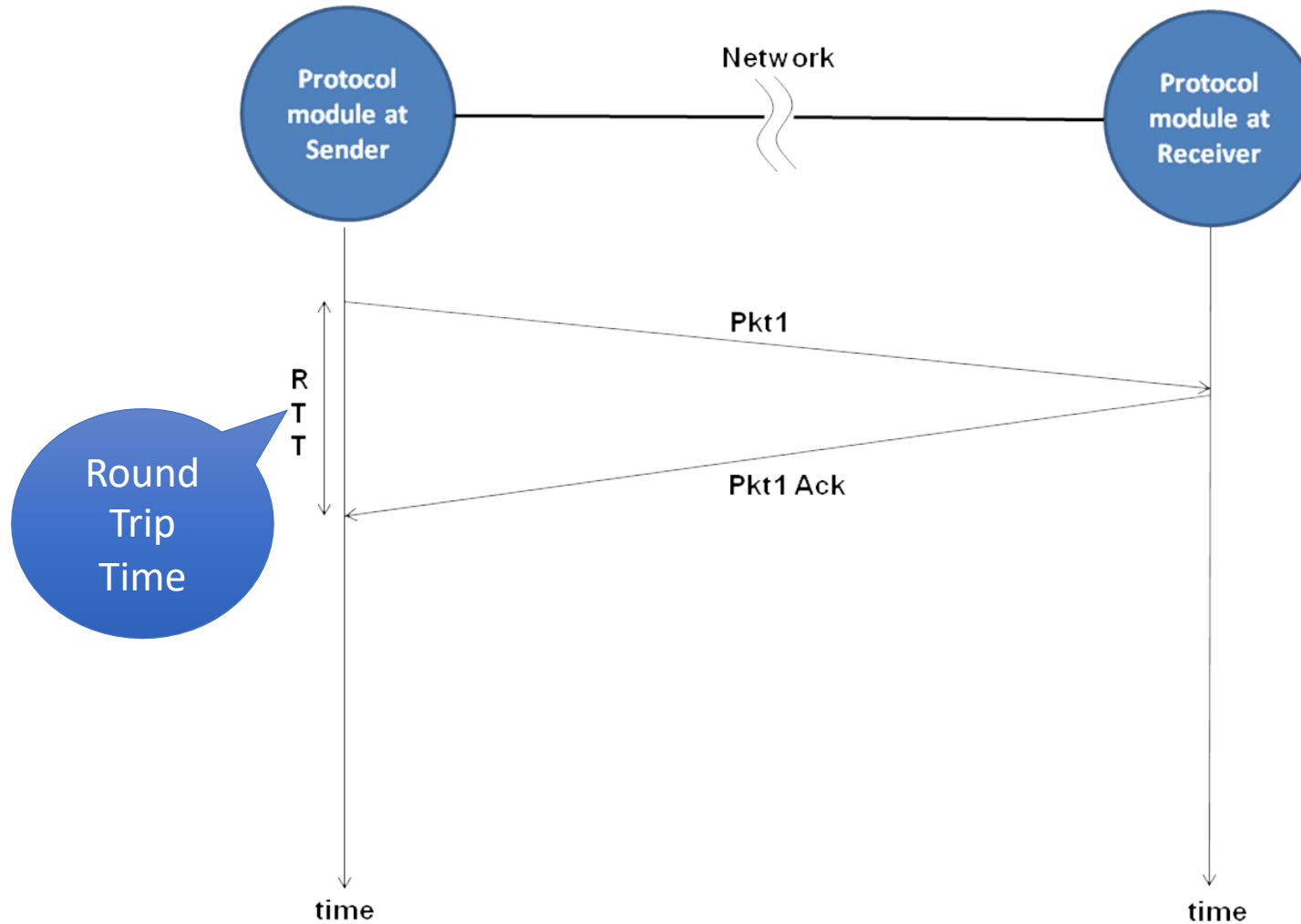
Global Ping Statistics → Atlanta and Athens

Greece

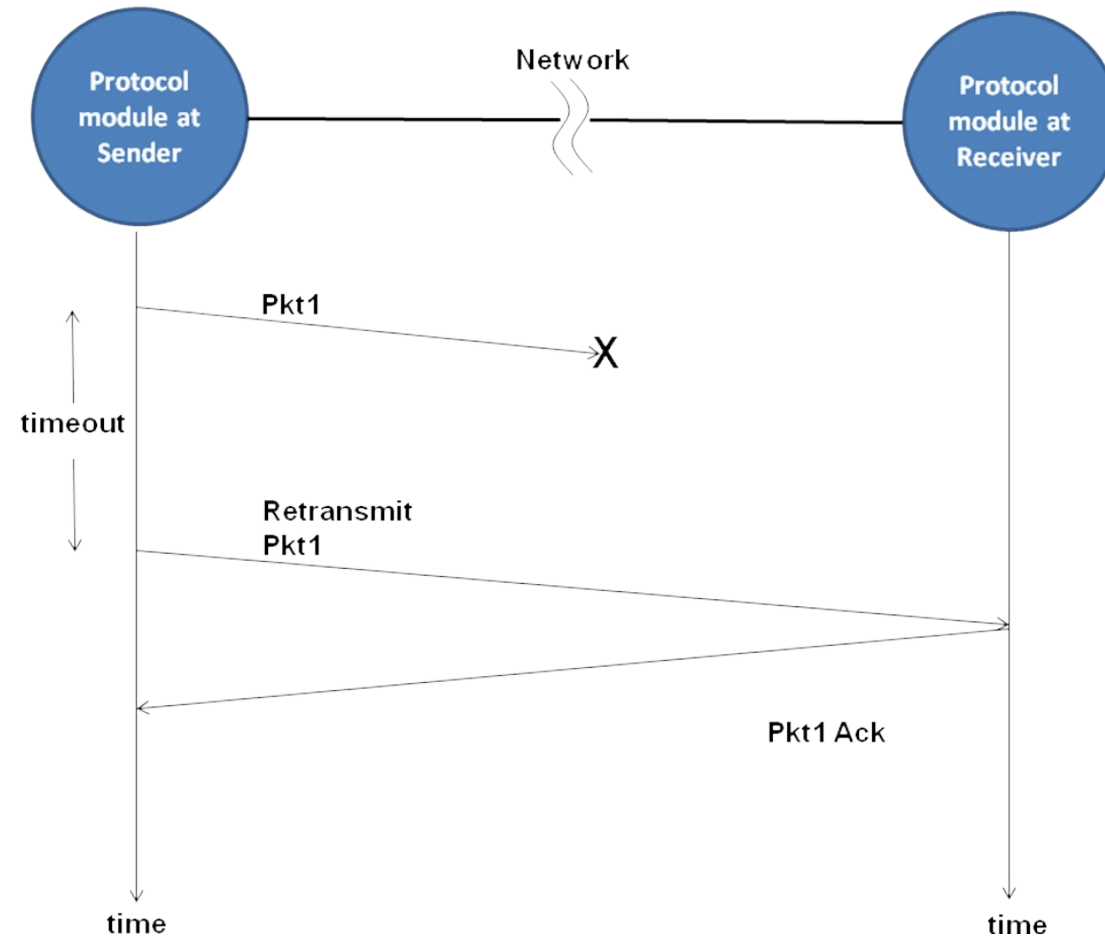
Showing historical ping data between major cities. Bar shows Average \pm Median Deviation.



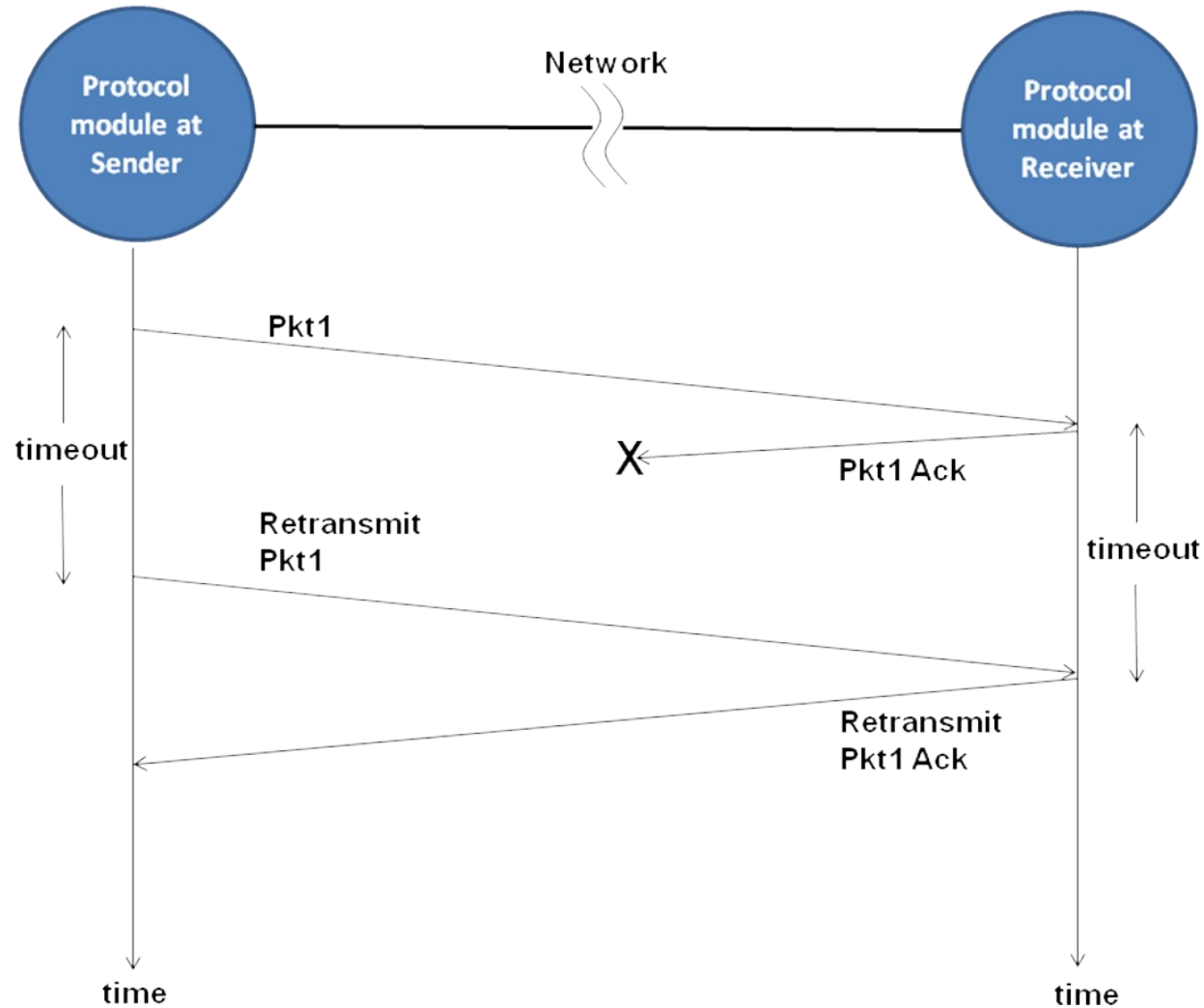
One round trip



Lost packet



Lost ack



More details

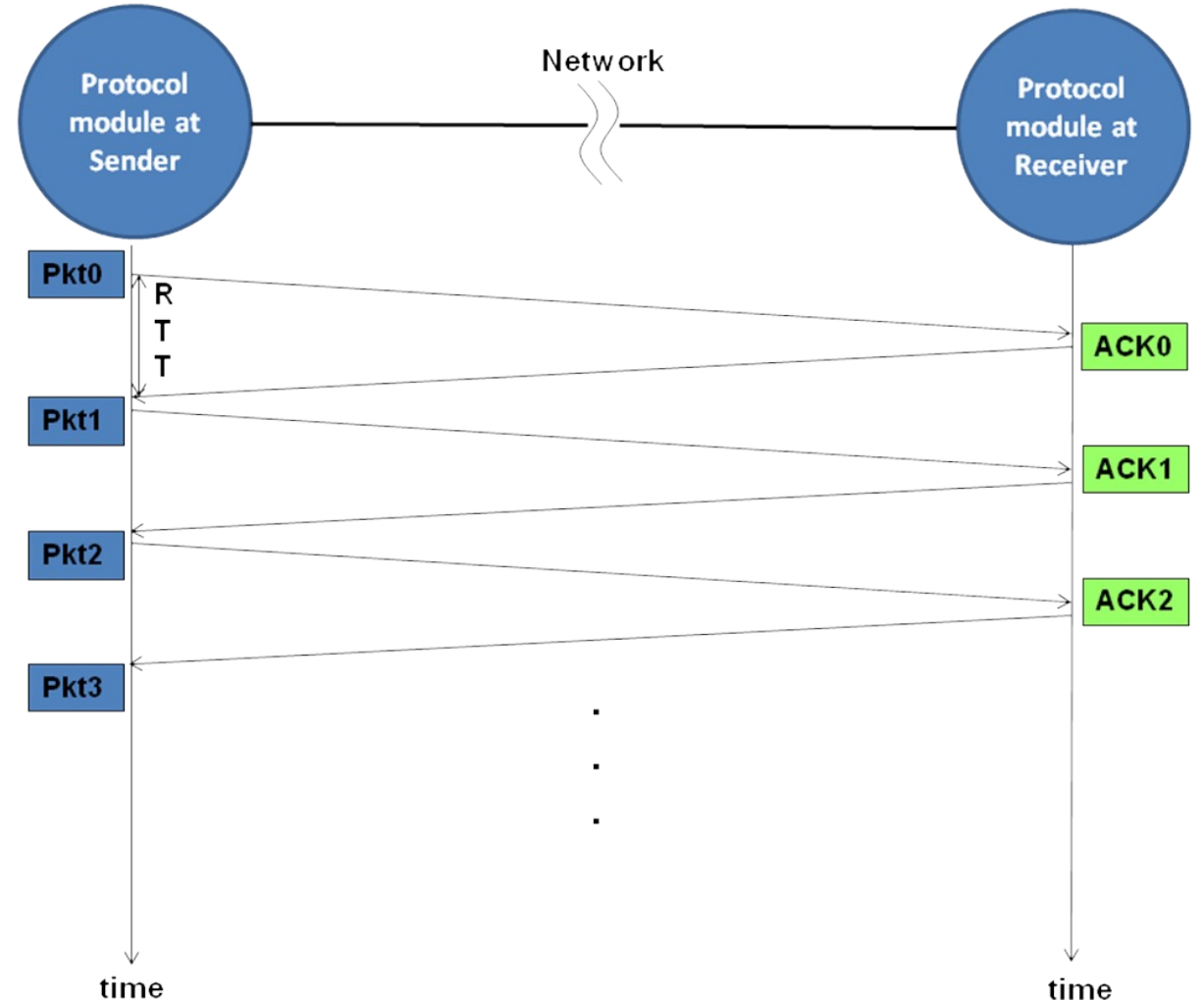
- How much buffer does the stack need at the sender?
 - Just 1
- How much buffer does the stack need at the receiver?
 - Just 1
- How do I know at the receiver that a packet I'm getting is not a duplicate
 - Include a sequence number
- How big should a sequence number be?
 - It might only need to be 1 bit if there's only one message in flight

Assumptions

- In our stop-and-wait protocol, we make some assumptions
 - Packet loss is possible
 - Packets don't get reordered or arbitrarily delayed (how can they if only one packet is in flight)
 - So we can get away with a 1 bit sequence number
- However, there are more fundamental assumptions in a packet-switched network
 - Packet loss is possible (i.e. best-effort delivery)
 - Packets may be reordered or arbitrarily delayed
 - Packets may be damaged in transit
- These additional assumptions mean that a 1-bit sequence number isn't going to hold up on the internet
 - So we use monotonically increasing sequence numbers

Working at top speed!

- Remember Latency?
- 150 ms to Athens
- $RTT = 150\text{ms}$
- ~7 packets per second throughput?
- No matter how fast the network?



Pipelined protocol



Pkt	Pkt	Pkt	Pkt
0	1	2	3

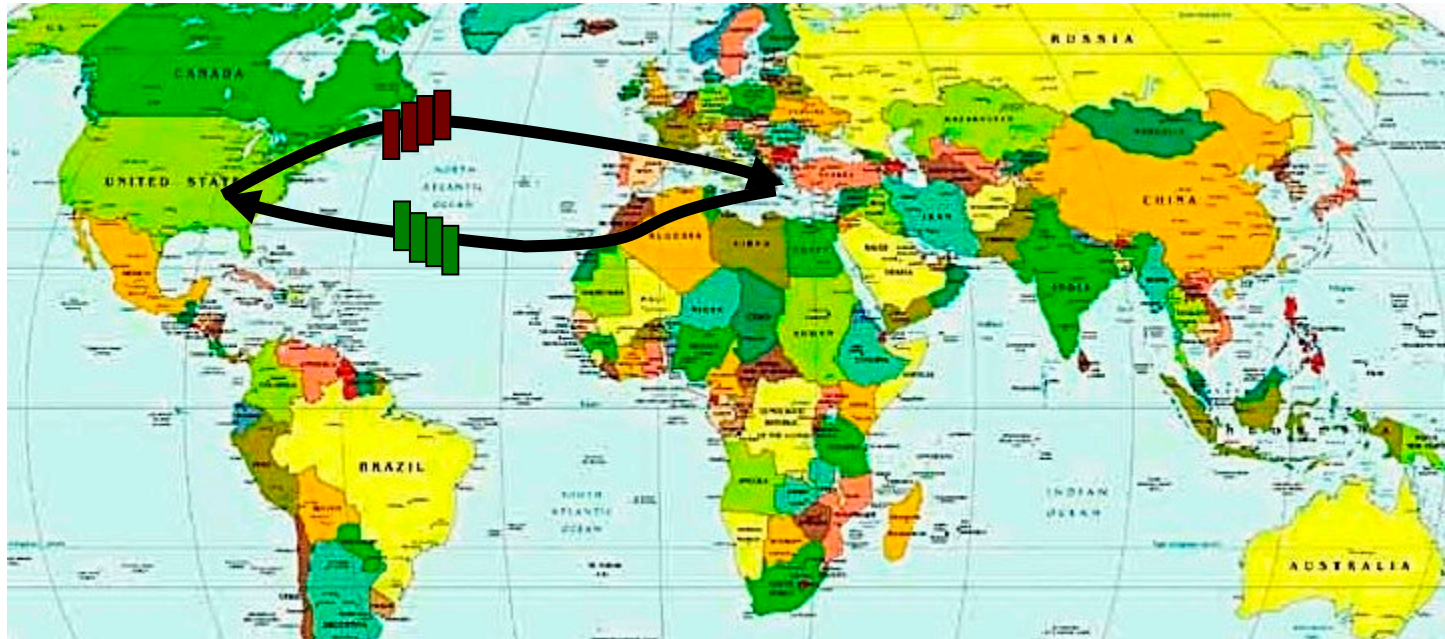
...

➔ Blast a bunch of packets one after another

What could go wrong?

Timeline of sender

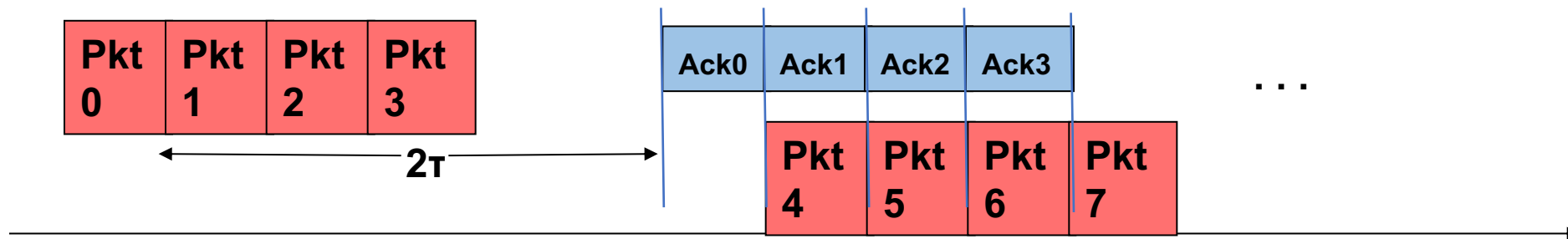
Reliable pipelined protocol



Sometimes some packets don't make it all the way!

We have to keep track

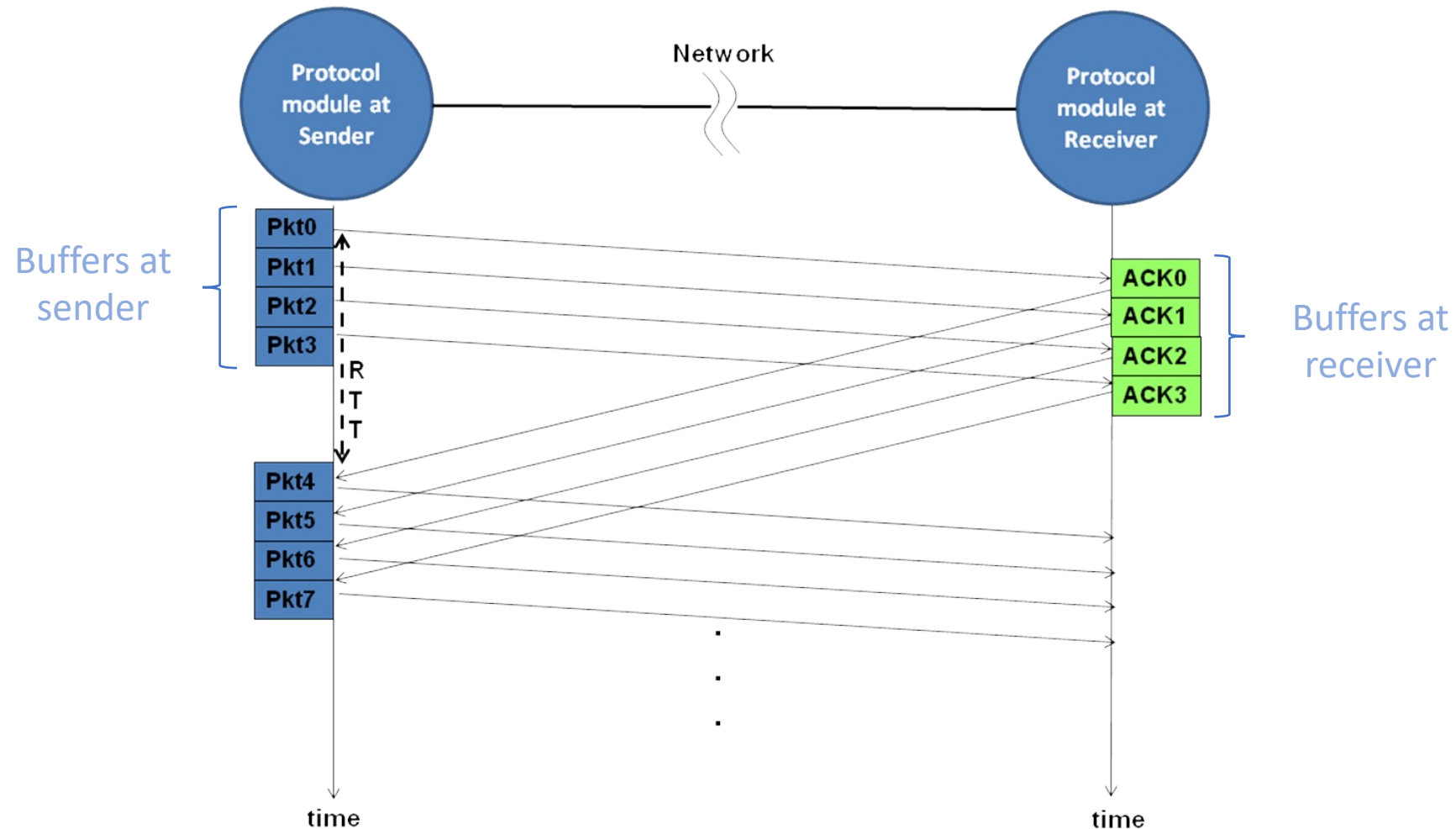
Reliable pipelined protocol



Now we get 4 packets every 150ms = 27 packets/sec

Timeline of sender

Reliable pipelined protocol





How many packet buffers does the sender need for stop-and-wait protocol?

- A. I just want the participation credit
- B. 1
- C. 2
- D. Designer's choice
- E. No way to predict in advance



How many?

How many packet buffers does the sender need to implement a reliable pipelined protocol?

- A. I just want the participation credit
- B. 1
- C. 2
- D. One buffer for each packet in the pipeline
- E. Protocol designer's choice

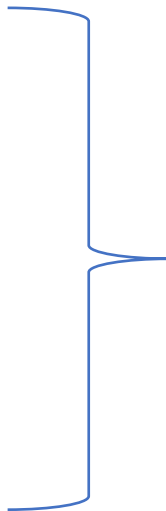
What might a packet look like?

```
struct header_t {
    int destination_address; /* destination address */
    int source_address;      /* source address */
    int num_packets;         /* total number of packets in the message */
    int sequence_number;     /* sequence number of this packet */
    int packet_size;         /* size of data contained in the packet */
    int checksum;            /* for integrity check of this packet */
};

struct packet_t {
    struct header_t header; /* packet header */
    char *data;             /* pointer to the memory buffer containing the data
                             of size packet_size */
};
```

Transport (& session) layer concerns

- Arbitrary message size
- Out of order delivery
- Packet loss
- Bit errors
- Queuing delays



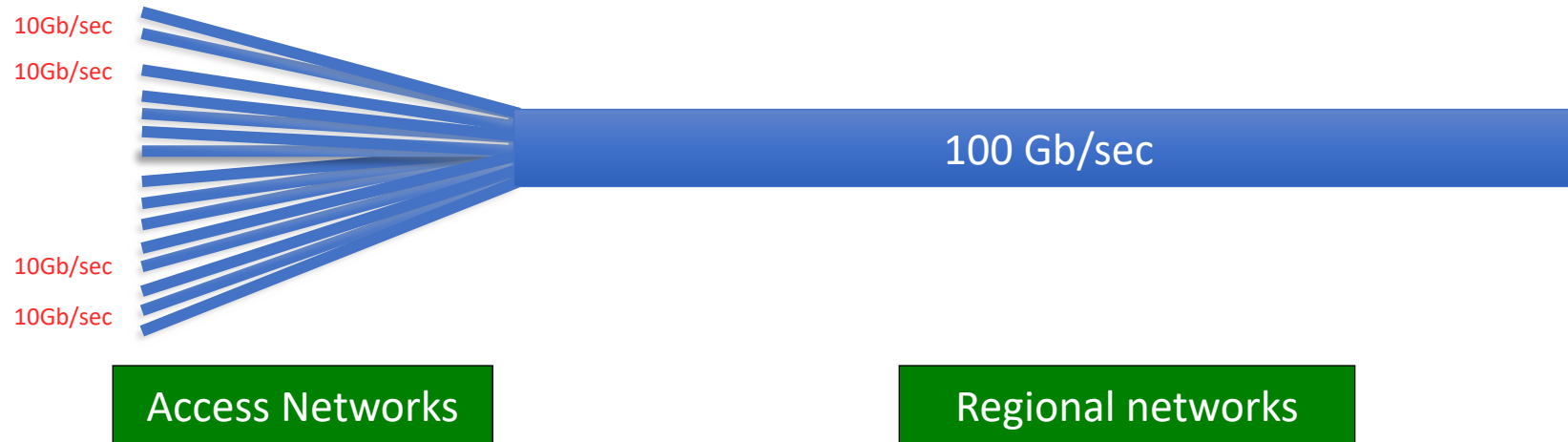
How do we
address these?

Transport (& session) layer concerns

- Arbitrary message size → Big messages use multiple packets
- Out of order delivery → Sequence numbers
- Packet loss → Positive ACK and buffers
- Bit errors → Checksum for packet
- Queuing delays → But how to handle this one?

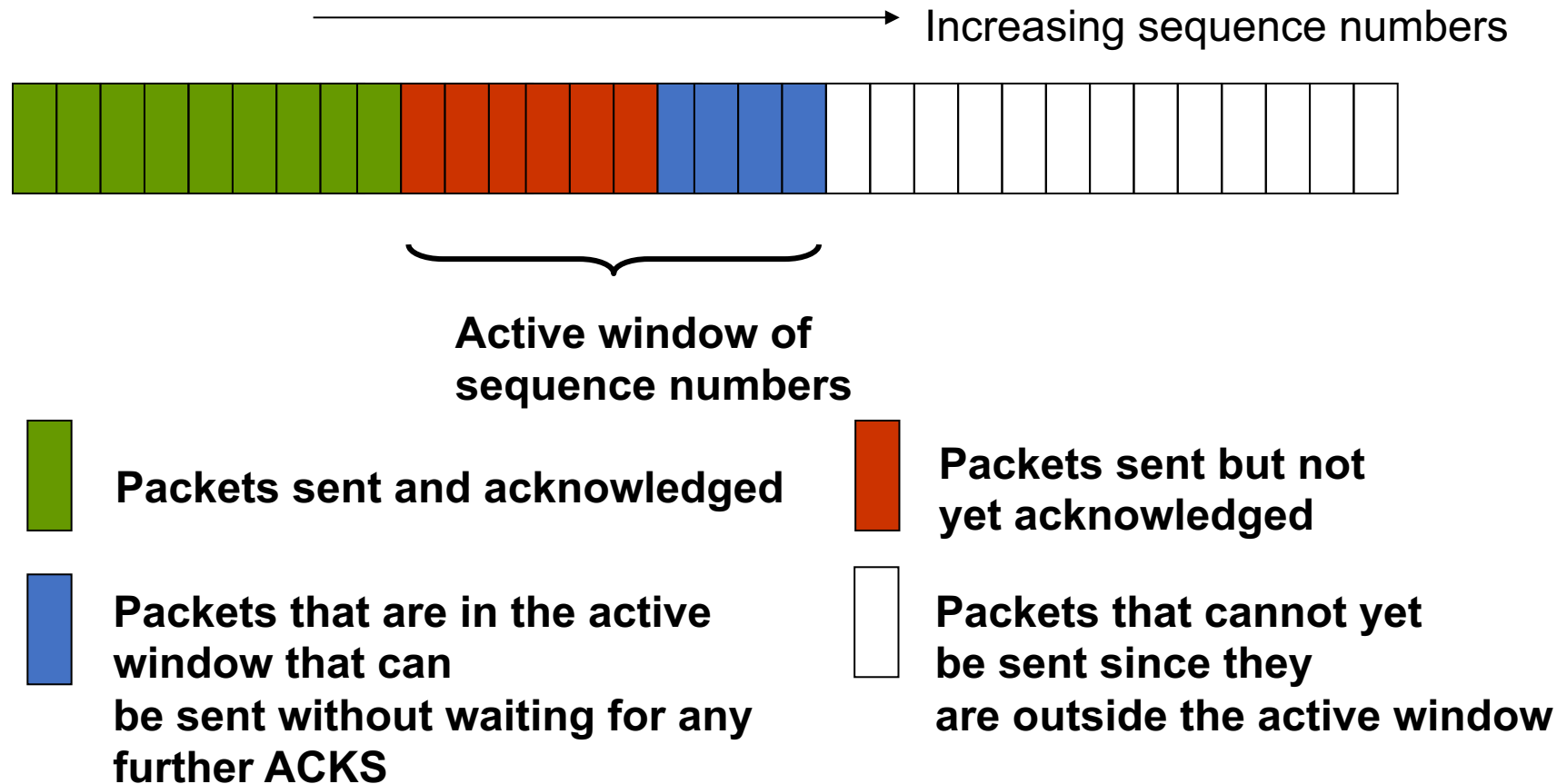
```
struct header_t {  
    int destination_address; /* destination address */  
    int source_address;      /* source address */  
    int num_packets;         /* total number of packets in the message */  
    int sequence_number;     /* sequence number of this packet */  
    int packet_size;         /* size of data contained in the packet */  
    int checksum;            /* for integrity check of this packet */  
};
```


Network congestion



- Networks tend to grow
- Network connections get overcommitted
- Packets are queued, but get so stale they must be dropped
- Because of the sequence number, we can ignore heavily delayed packets if they ever make it

Reliable protocol with windowing



Congratulations! You've now been introduced to TCP from the IPv4 protocol

Transport protocols on the Internet

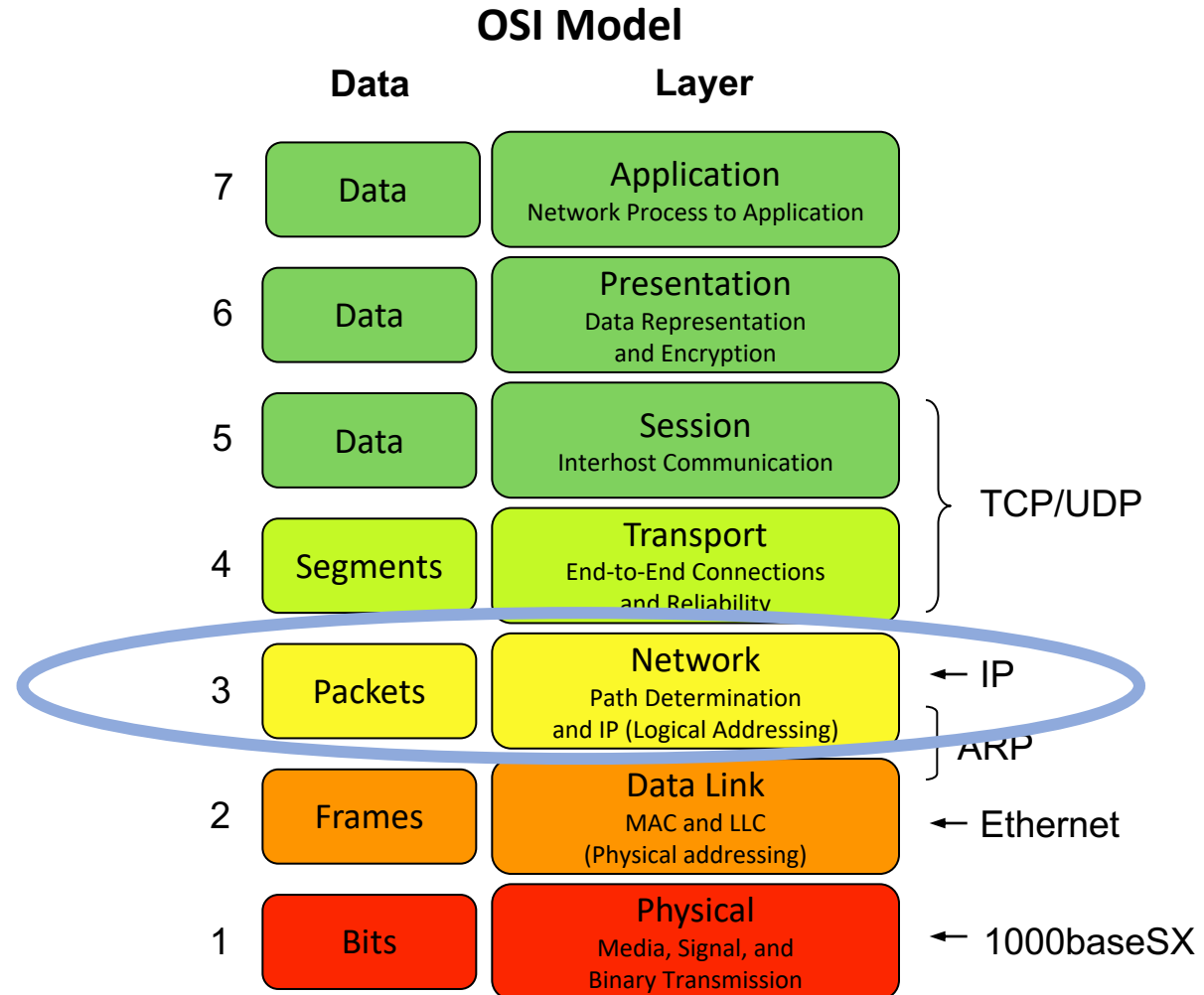
Transport protocol	Features	Pros	Cons
TCP	Connection-oriented; self-regulating; data flow as stream; supports windowing and ACKs	Reliable; messages arrive in order; well-behaved due to self-policing	Complexity and extra latency in connection setup and tear-down; at a disadvantage when mixed with unregulated flows; no guarantees on delay or transmission rate
UDP	Connection-less; unregulated; message as datagram; no ACKs or windowing	Simplicity; no frills; especially suited for environments with low chance of packet loss and applications tolerant to packet loss;	Unreliable; message may arrive out of order; may contribute to network congestion; no guarantees on delay or transmission rate

Choice of transport protocols

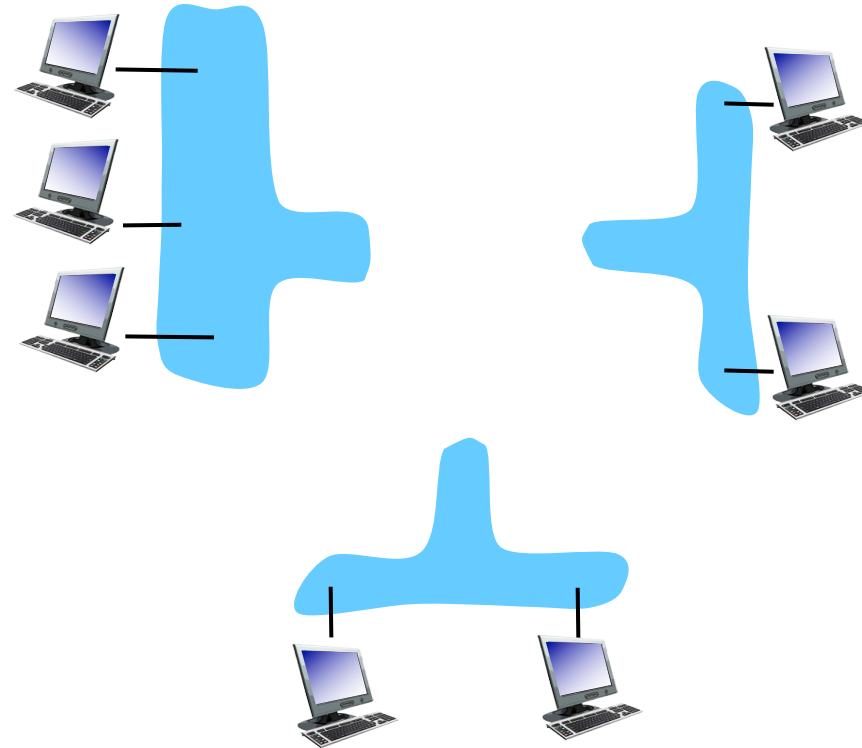
Application	Key requirement	Transport protocol
Web browser	Reliable messaging; in order arrival of messages	TCP
Instant messaging	Reliable messaging; in order arrival of messages	TCP
Voice over IP	Low latency	Usually UDP
Electronic Mail	Reliable messaging	TCP
Electronic file transfer	Reliable messaging; in order delivery	TCP
Video over Internet	Low latency	Usually UDP; may be TCP
File download on P2P networks	Reliable messaging; in order arrival of messages	TCP
Network file service on LAN	Reliable messaging; in order arrival of messages	TCP; or reliable messaging on top of UDP
Remote terminal access	Reliable messaging; in order arrival of messages	TCP

Now let's look lower

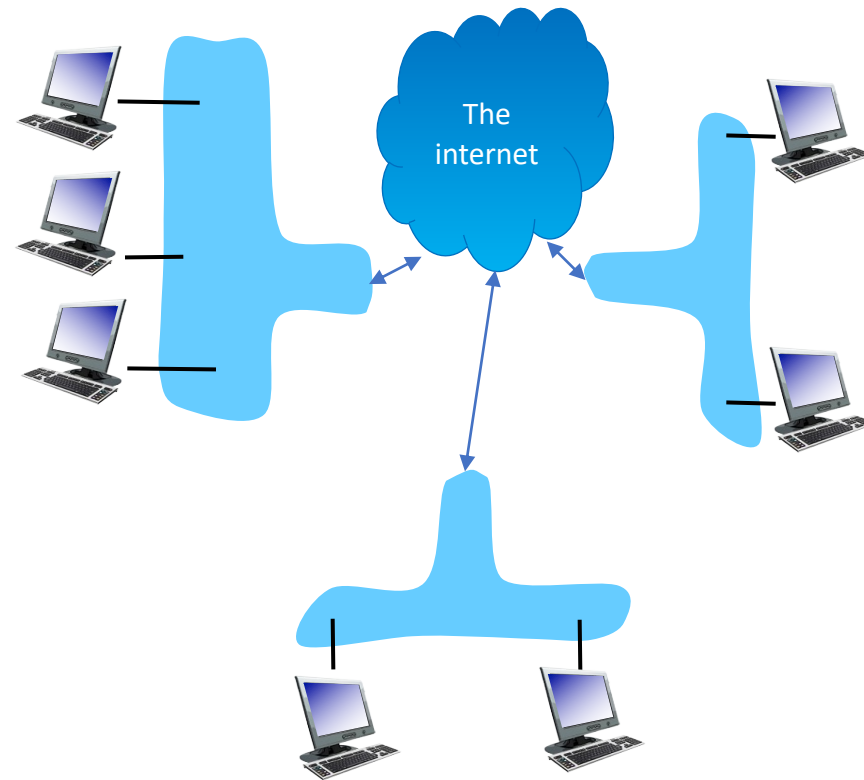
- IP lives at the network layer
- Let's look specifically at IP as a layer 3 as well as its relationship to the Transport (Session) layers



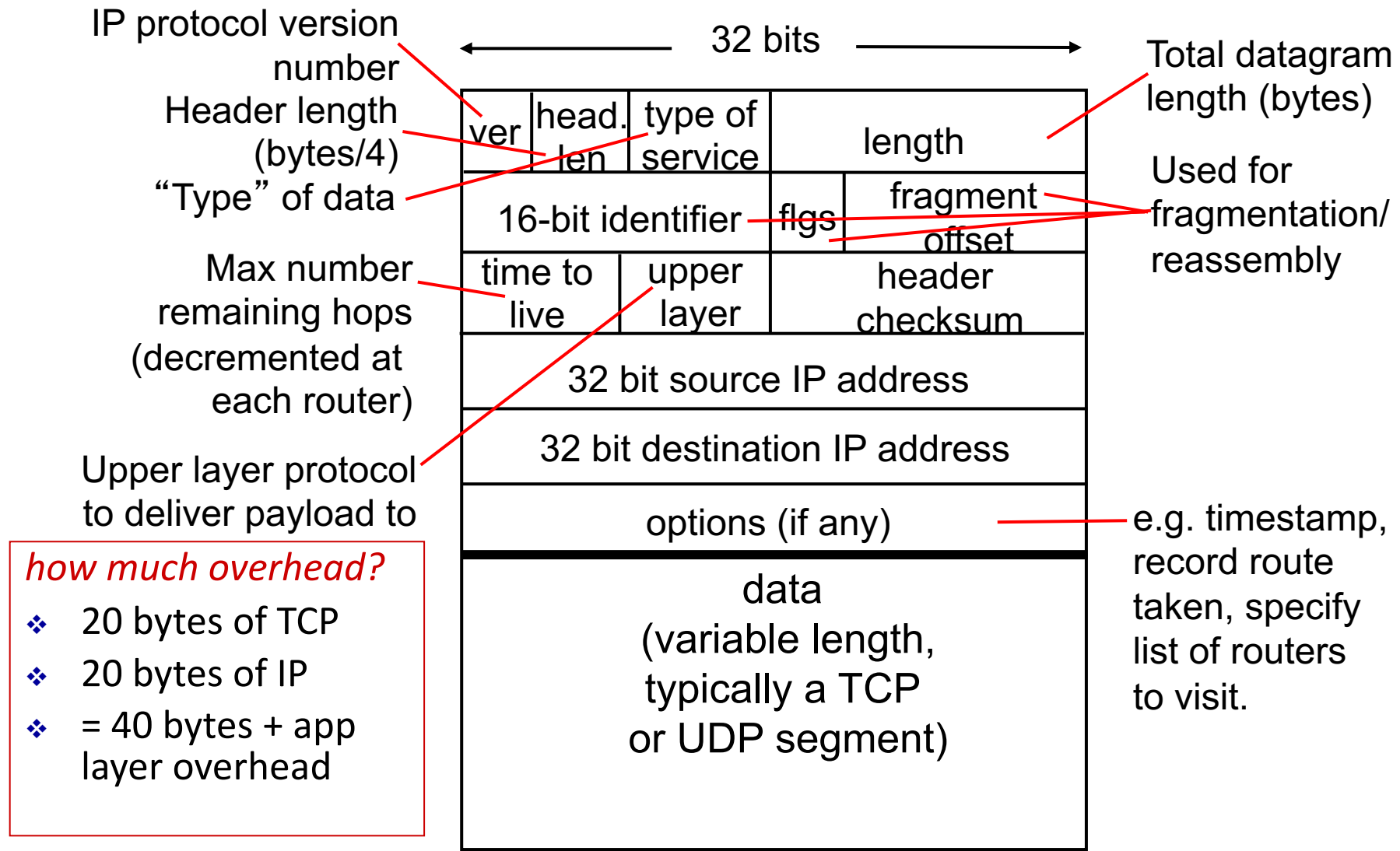
- No public internet
- Islands of connectivity
 - Data link layer networks all over, even in 1983
 - Ethernet
 - 802.11 wireless
 - Token Ring
 - AppleTalk
 - RS232 serial
 - ATM
 - And plenty more
- So how do we build a network of networks?



- Build a network layer that can address and interconnect all sorts of different data link networks
- After a decade of innovation and another decade of growth
- In 1980, IP for everybody (version 4 that is) with two built-in transport protocols, TCP and UDP

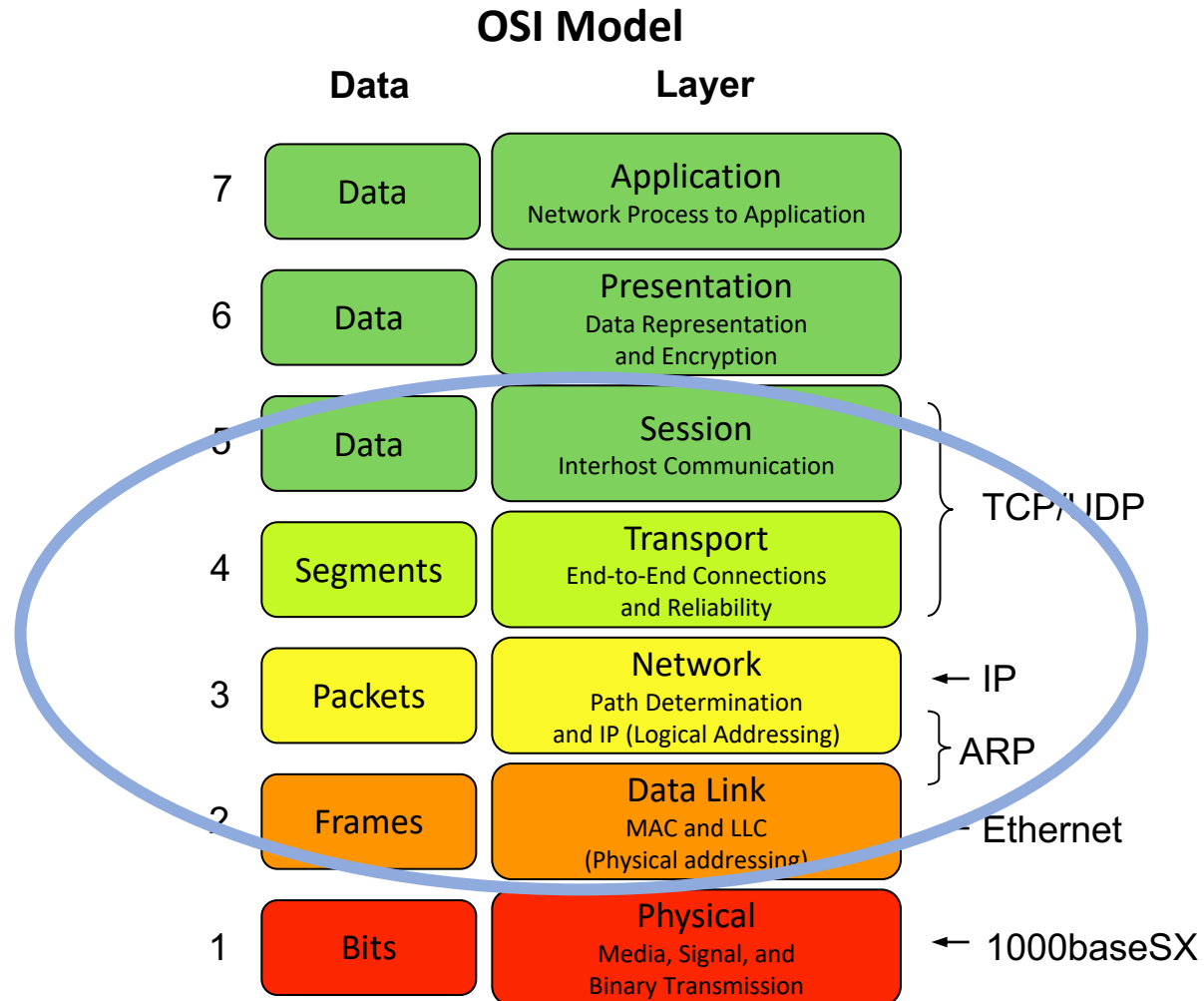


IP datagram (packet) format



Visible relations between layers

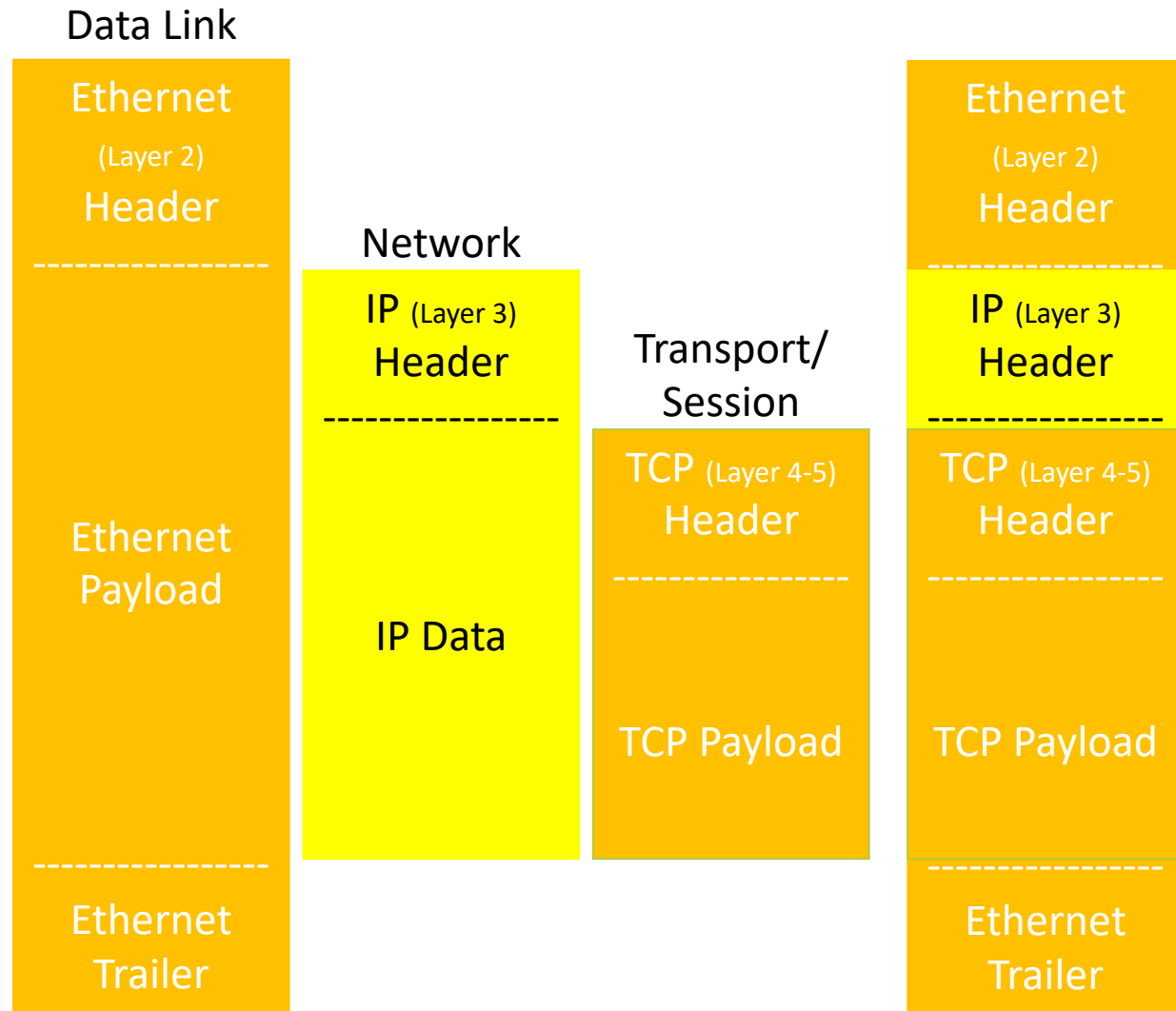
- Layers 2 through 5 are manifested in the packets we send on the network
- (Layers 1, 6, and 7 are too, but showing them makes this too complicated)
- Let's see how...



Encapsulated packets

- The network protocol is designed in layers
- Thus, the lower-level packets encapsulate (enclose) the higher-level packets

Remember: Layer 2 doesn't have to be ethernet and layer 4-5 doesn't have to be TCP



Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags				Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags		Fragment Offset													
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			N S	C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I N	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if <i>data offset</i> > 5. Padded at the end with "0" bytes if necessary.)																															
...																															

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			N S	C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I N	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if <i>data offset</i> > 5. Padded at the end with "0" bytes if necessary.)																															
...																															

Fragmentation

- One way to deal with segments that are too big to fit in a single packet
- Probably should be considered deprecated these days
- IPv6 doesn't allow it

IPv4 addresses

- 32 bits / 4 octets / four-decimal dotted quad
 - Did you know that **130.207.7.31** becomes 0x82CF071F?
 - Each of those four numbers is decimal for a single byte
- And we need a mask
- We use it to divide the IP address into a "network" part and a "host" part
 - It's a 32-bit number that starts with 1s and ends with 0s
 - The network part is "under" the 1s in the mask
 - E.g., mask 255.255.255.0 indicates we are looking for host **31** in network **130.207.7.0**

Subnet and CIDR Notation

- Two different notations, same meaning:
 - 130.207.254.64 255.255.255.192
 - 130.207.254.64/26
- In this example, the network part is the first 26 bits, host is last 6
- If you're doing this every day, you learn to do it in your head; if not, use a subnet calculator, e.g. <http://www.subnet-calculator.com/cidr.php>
- Cisco equipment often uses the former notation; many other modern tools use the latter

130	207	254	64
10000010	11001111	11111110	01 000000
255	255	255	192
11111111	11111111	11111111	11 000000

Note /26
leading 1 bits

11001111							
1	6	3	1	8	4	2	1
2	4	2	6				
8							

What does it mean to be on the same network?

- In the IP world it's simple
 - If network bits between two addresses are equal, they are on the same network
 - This means the data link layer will be called on to deliver the packet

Source IP address	Destination IP address	Mask	On the same network?
130.207.7.23	130.207.254.16	/16 (255.255.0.0)	Yes
192.168.1.2	192.168.2.2	/24 (255.255.255.0)	No
0x81880507	0x818910F1	/15 (255.254.0.0)	Yes
172.16.32.12	10.0.5.2	/16 (255.255.0.0)	No



Are these two IP addresses on the same network?

143.215.14.243 and 143.214.14.242 if the mask for each is /24

- A. I just want the participation credit
- B. Yes
- C. No
- D. Can't tell
- E. All of the above

Keeping the first 24 bits of the two addresses, we get

143.215.14.0

and

143.214.14.0

Are they the same number?

Nope.



Are these two IP addresses on the same network?

143.215.14.243 and 143.214.14.242 if the mask for each is /15

- A. I just want the participation credit
- B. Yes
- C. No
- D. Can't tell
- E. None of the above (B to D)

Keeping the first 15 bits of the two addresses, we get

143.214.0.0 (215=11010111₂; first 7 bits are 11010110₂)

and

143.214.0.0 (214=11010110₂; first 7 bits are 11010110₂)

Are they the same number?

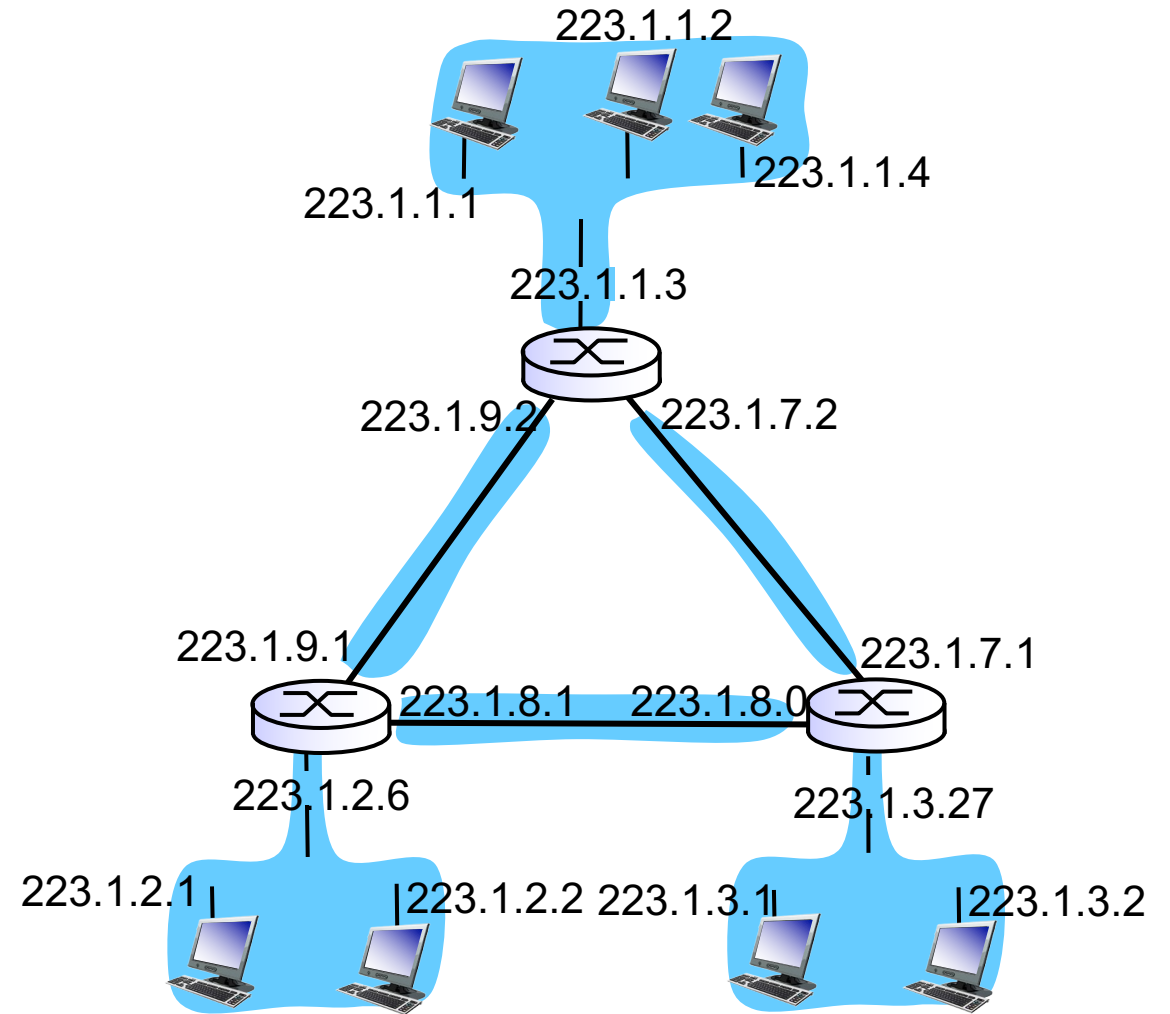
Yes!

Subnets

- Remember those Islands of Connectivity? And that mask?
- First rule of IP routing:
 - If you're on the same network, punt it to the data link layer to deliver
 - Otherwise, use your IP routing table to forward it to another IP host on your network
- How can you tell if you're on the same network?
 - The mask tells you how many leading bits are the “network part” of the IP address.
 - If the network part of the destination address matches the network part of (one of) your interface address(es), you're on the same network.
 - Each node on the network only needs to know the mask for its own IP address (think about it), so masks don't need to be passed in packets
- So all hosts on a data link level network must be assigned IP addresses that have the same network part (and hence the same mask).

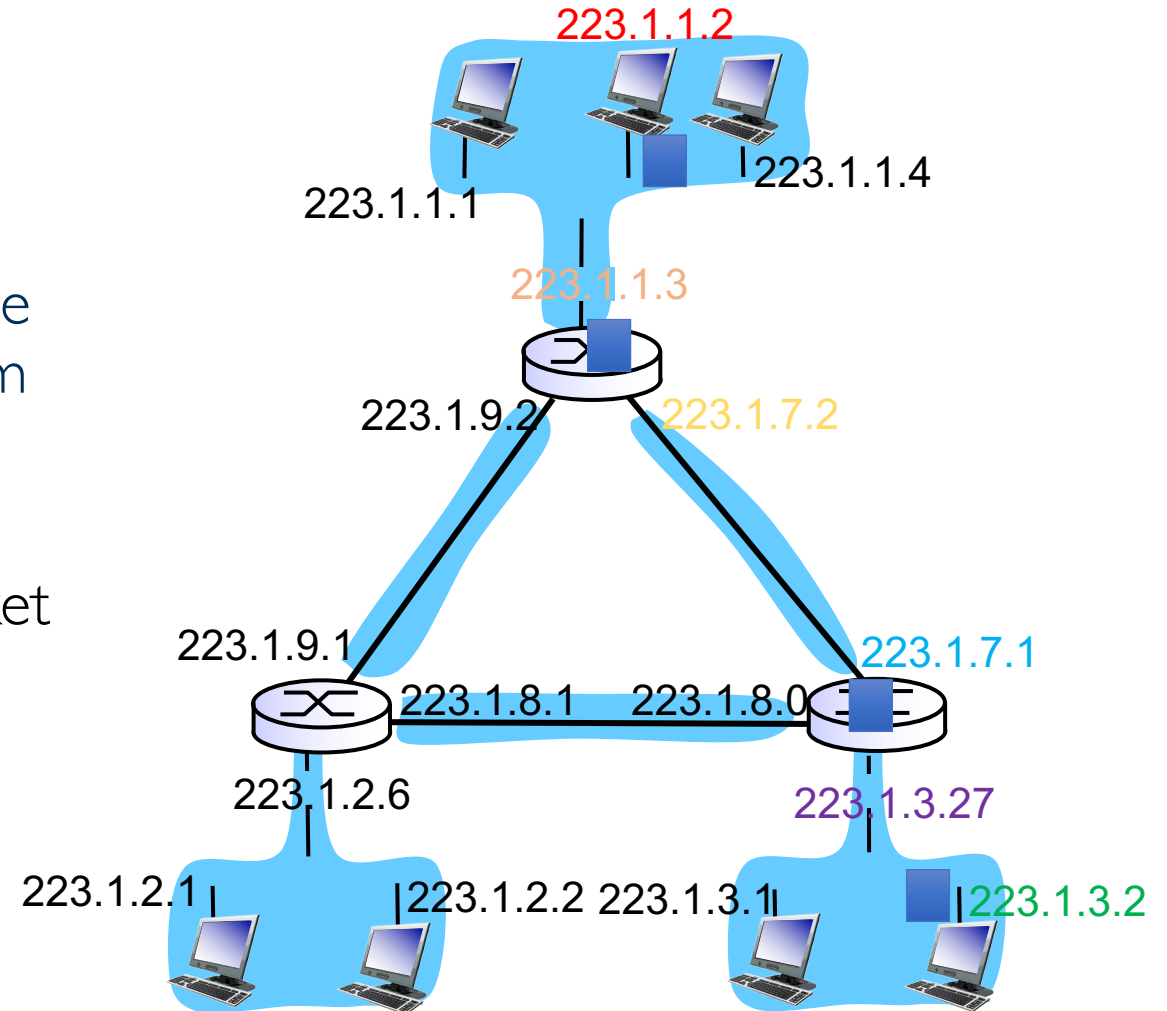
Connecting the Islands

- Data link layer networks connected by IP hosts
- Now where do the masks come in?
- Clue: All the masks here are /24.
- The blue blobs are indeed different IP networks
- The three routers pass traffic between the three outer networks



Traveling between the Islands

- For example, 223.1.1.2 can talk to 223.1.3.2:
- 223.1.1.2 sends its packet for 223.1.3.2 to 223.1.1.3 via layer 2
- The top router knows how it's connected to the other two routers and so sends the packet from 223.1.1.2 to 223.1.7.1
- The bottom right router knows it has a connection to 223.1.3.0/24 so it sends the packet from 223.1.1.2 to 223.1.3.2.
- We'd say this route has "2 hops"



-
- In the dark ages, subnet masks could be derived from the network address
 - Now that's called "classful" addressing
 - These can still show up as a default mask if you don't specify one

Class	Prefix	Range	Mask	Purpose
A	0...	0.0.0.0-127.255.255.255	8	
B	01...	128.0.0.0-191.255.255.255	16	
C	011...	192.0.0.0-223.255.255.255	24	
D	0111...	224.0.0.0-239.255.255.255	-	multicast
E	1111...	240.0.0.0-255.255.255.255	-	reserved

CIDR: Classless Inter-Domain Routing

- We ran out of IP addresses in 1995.
- This was the very clever solution
- Turns out you only really need the network mask if you're connected to that network!
- Now we simply specify the mask for the local subnets on each host

IP Routing Table

Destination	Gateway	Flags	Interface
0.0.0.0/0	128.61.5.1	UG	
128.61.5.0/24	128.61.5.166	U	Eth0
128.61.5.21/32	128.61.5.166	U	Eth0
127.0.0.1/32	127.0.0.1	UH	

- If the destination is on our network, send the packet to the data link layer
- Look through the routing table and choose the match with the *longest prefix* (largest mask); send the packet to that gateway
- Question: Which line does 128.61.5.82 match?
How about 130.207.5.9?
- The interesting part comes later: Creating the routing table



Questions: IP Routing rules

If the network address part of the interface and destination addresses are the **same**, then

- A. I just want the participation credit
- B. Pass the packet to layer 2 for delivery
- C. Send the packet to the next-hop router based on the IP routing table
- D. Return an Acknowledgement to the source address



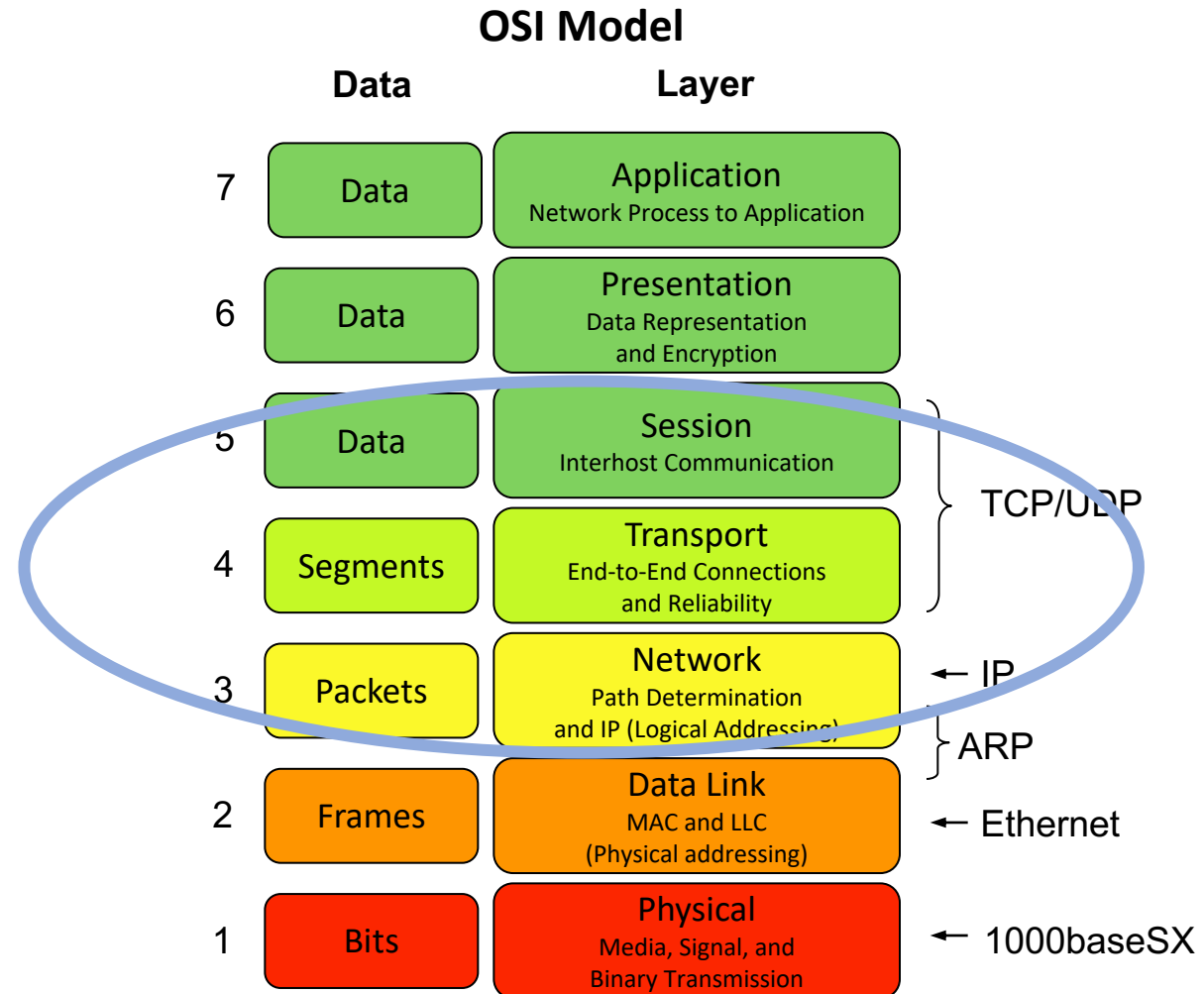
Questions: IP Routing rules

If the network address part of the interface and destination addresses are **different**, then

- 27% A. I just want the participation credit
- 20% B. Pass the packet to layer 2 for delivery
- 20% C. Send the packet to the next-hop router by choosing the longest match in the IP routing table
- 20% D. Send the packet to the next-hop router by choosing the shortest match in the IP routing table
- 13% E. Return an ICMP "No route to host" message

Now let's include Layers 3-5 in IP

- TCP and UDP are the two most common IP Transport protocols
- In IP, recall the division between L4 and L5 is not well defined



Ports and connection addressing

Source IP	Source Port	Destination IP	Destination Port	Rest of the packet
--------------	----------------	-------------------	---------------------	-----------------------

We know how to get packets from one host to another, but how do we identify conversations?

- How does one distinguish UDP and TCP traffic from different services that share the same network interface?
 - On each side of a conversation, port numbers (0-65535) are used to define unique connections
 - Often the destination port on the first packet is a Well-known or Registered port number
 - Note that the fields are not actually ordered this way in the packet
- With TCP and UDP protocols, always consider the addressing as a **quadruple**: (*source IP, source port, destination IP, destination port*)
- Let's look at where that ordered quad shows up in the segment...

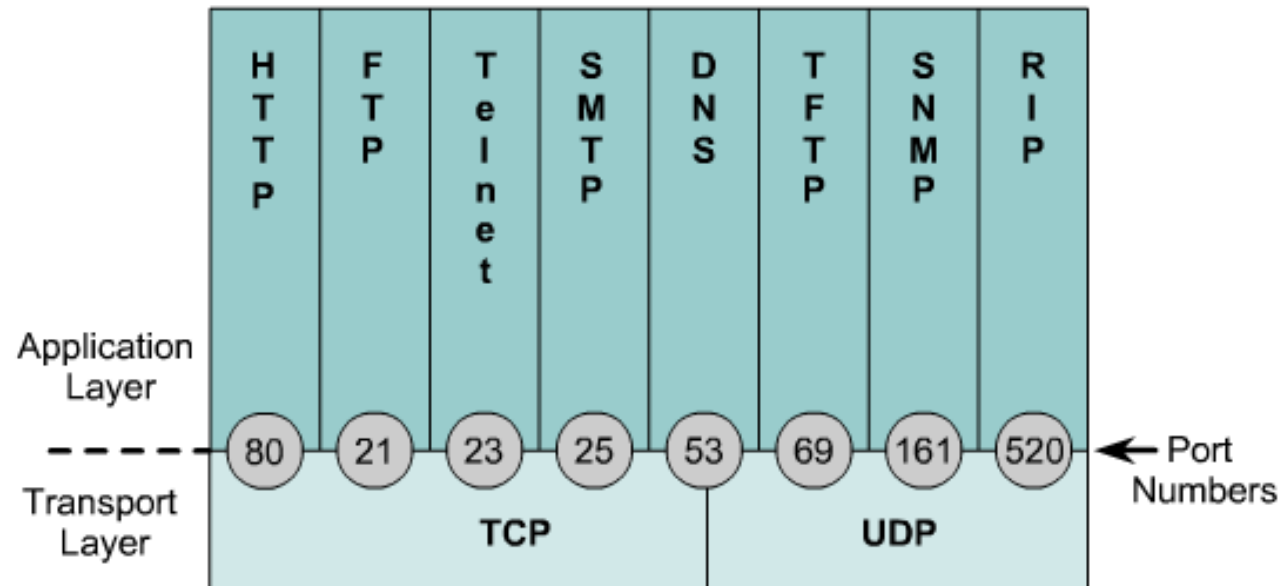
TCP (and UDP) port addressing

Offsets	Octet	IPv4 Header Format																															
Octet	Bit	0								1								2								3							
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP				ECN				Total Length															
4	32	Identification																Flags				Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																

32	256	TCP segment header																															
Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset		Reserved 000		NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size																		
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if <i>data offset</i> > 5. Padded at the end with "0" bytes if necessary.)																															
...																															

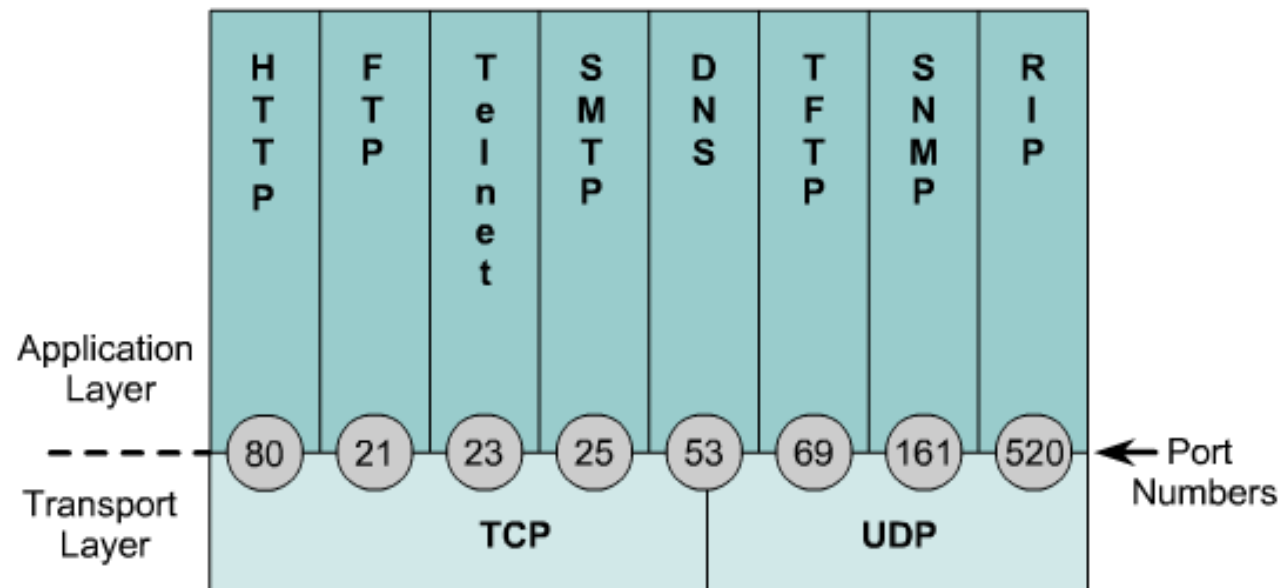
TCP and UDP port numbers

- Port numbers are used to keep track of different conversations.
- Well-known ports numbers are traditionally from 1-1023.
- Numbers 1024 and above are dynamic (ephemeral) port numbers.
- Registered port numbers for vendor-specific applications are now allowed above 1024.



TCP and UDP port numbers

- Destination ports are chosen based on the desired service
 - Often chosen based on the URL prefix, e.g. http: goes to 80, https: to 443
- Source ports are usually chosen by the IP stack to be unique among all open connections
- That makes the ordered quad unique among open connections
 - This is how we keep connections separate even when the IPs are the same





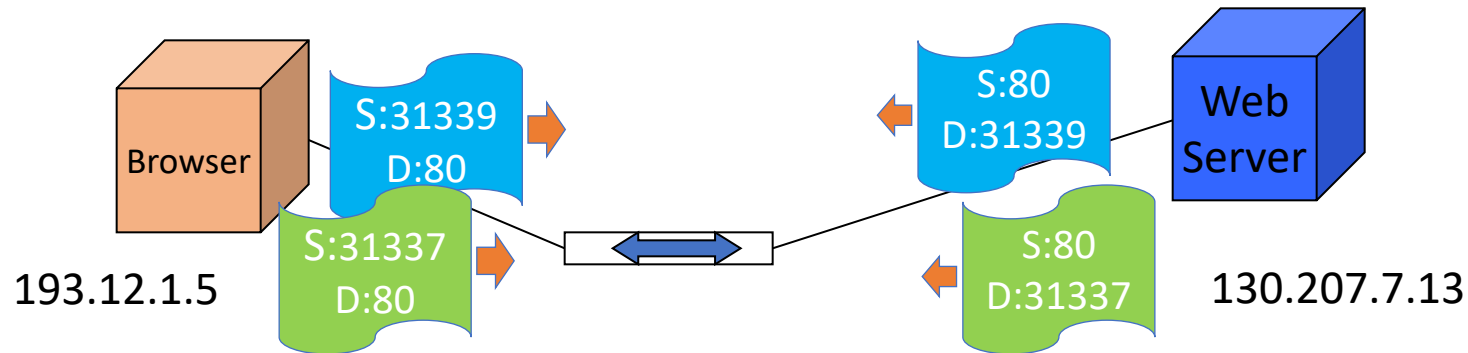
A browser opens two TCP connections to a web server

If the browser is running at 193.12.1.5, and the web server is offering service on port 80 at 130.207.7.13, how does TCP keep the conversations separate?

- 8% A. I just want the participation credit
- 17% B. IP does this automatically
- 42% C. The conversations use independent sequence numbers to disambiguate them
- 8% D. The conversations include a special session-identifier that disambiguates them
- 25% E. The browser lets the TCP stack choose a unique source port number for each connection

Conversations

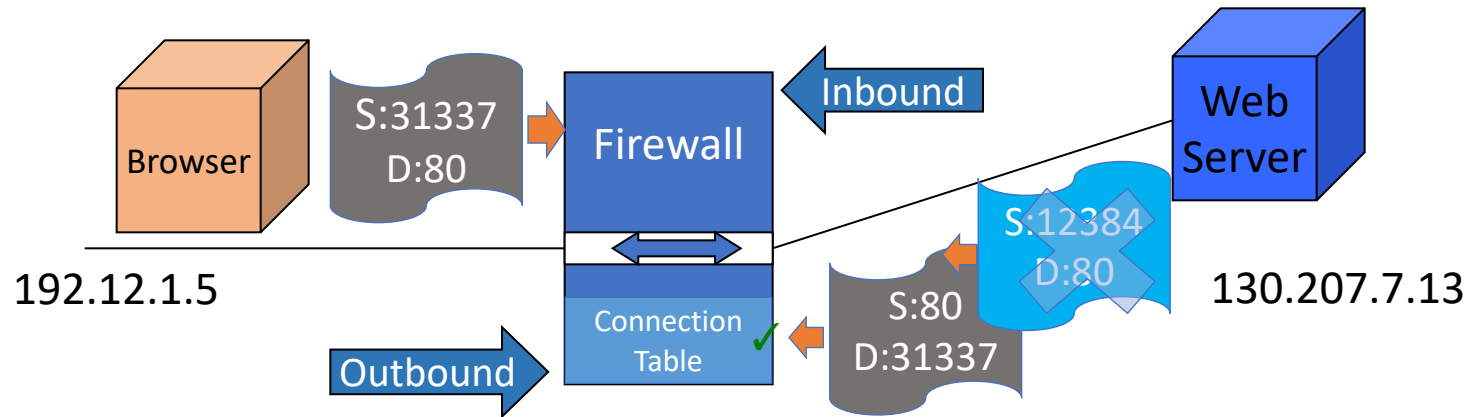
- When the web browser opens each connection, it doesn't specify a source port, allowing the TCP stack to pick a source port that isn't in use by any conversation on that host, say 31339 or 31337.
- The TCP stack uses the quads (193.12.1.5, 31337 or 31339, 130.207.7.13, 80) and its reverse to distinguish the two separate conversations
- Each connection gets its own sequence number, window size, etc. and is treated as a completely separate conversation by both the browser's and the web server's IP stacks



Stateful Firewalls

- Follow a predefined policy on which ordered quads are allowed from which direction
 - Example: Allow no inbound packets; Allow all outbound packets
- Records state on connections as it is opened
 - Requires a local state table (called a connection table)
- Compares packets to the connection table first to automatically allow return traffic
 - Example: If (192.12.1.5, 31337, 130.207.7.13, 80) is put in the connection table, then return traffic from (130.207.7.13, 80, 192.12.1.5, 31337) is automatically allowed
 - Any inbound traffic to other ports at 192.12.1.5 is dropped since no policy rule allows it
- Can do other protocol verification to drop malicious or badly-formed traffic

Makes sense if we don't offer any services



Network Address Translation

- Note there is confusing terminology
- Types
 - Static one-to-one NAT
 - A mapping is configured so that source address or destination address is changed from/to the mapped address on transit; checksums are recalculated
 - Dynamic one-to-one NAT
 - Same as Static, but a public address from a pool isn't mapped until a private address attempts to cross the NAT device; the public address is returned to the pool when the conversation ends
 - PAT or NPAT (network and port address translation)
 - Building on Dynamic NAT, both the source address and source port are mapped to an address and port from the public pool (often of size 1); return traffic is mapped by destination address and port.
 - And a lot of other variants
- NPAT is typically the way we share a single public IP address among multiple hosts -- another way we've managed the scarcity of public IP addresses

