Georgia Tech Computer Science

CS2200
Systems and Networks
Spring 2024

Lecture 25: Networking

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

Lecture slides adapted from Bill Leahy and Charles Lively of Georgia Tech

Networking

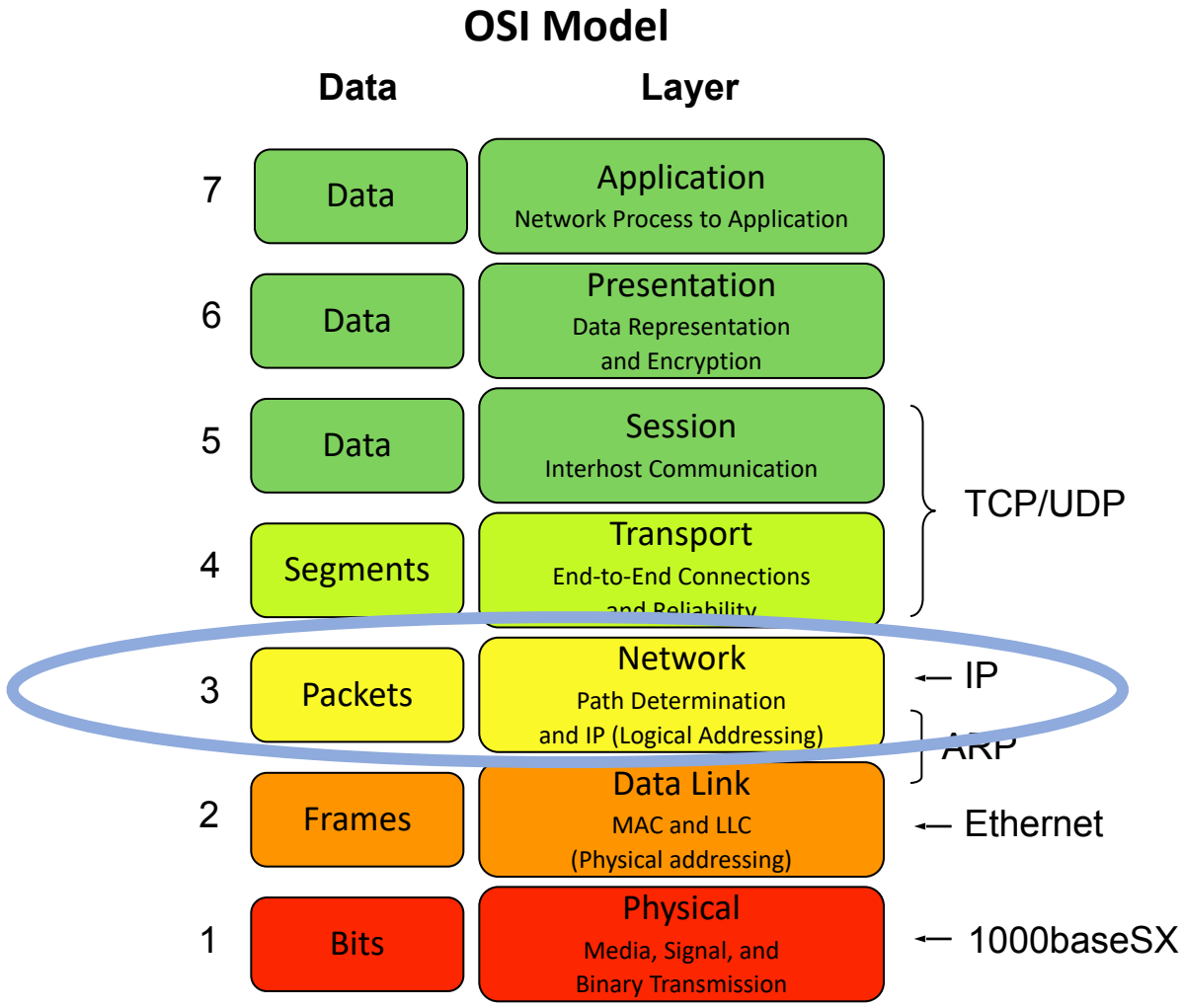
- Network stack overview – layer encapsulation
- Three transport protocol types
- Basic distinctions between TCP and UDP

Today + Thursday:

- Network layer
- Ports and connections
- Data Link layer (Ethernet)
- Performance metrics
- Routing and IP table construction

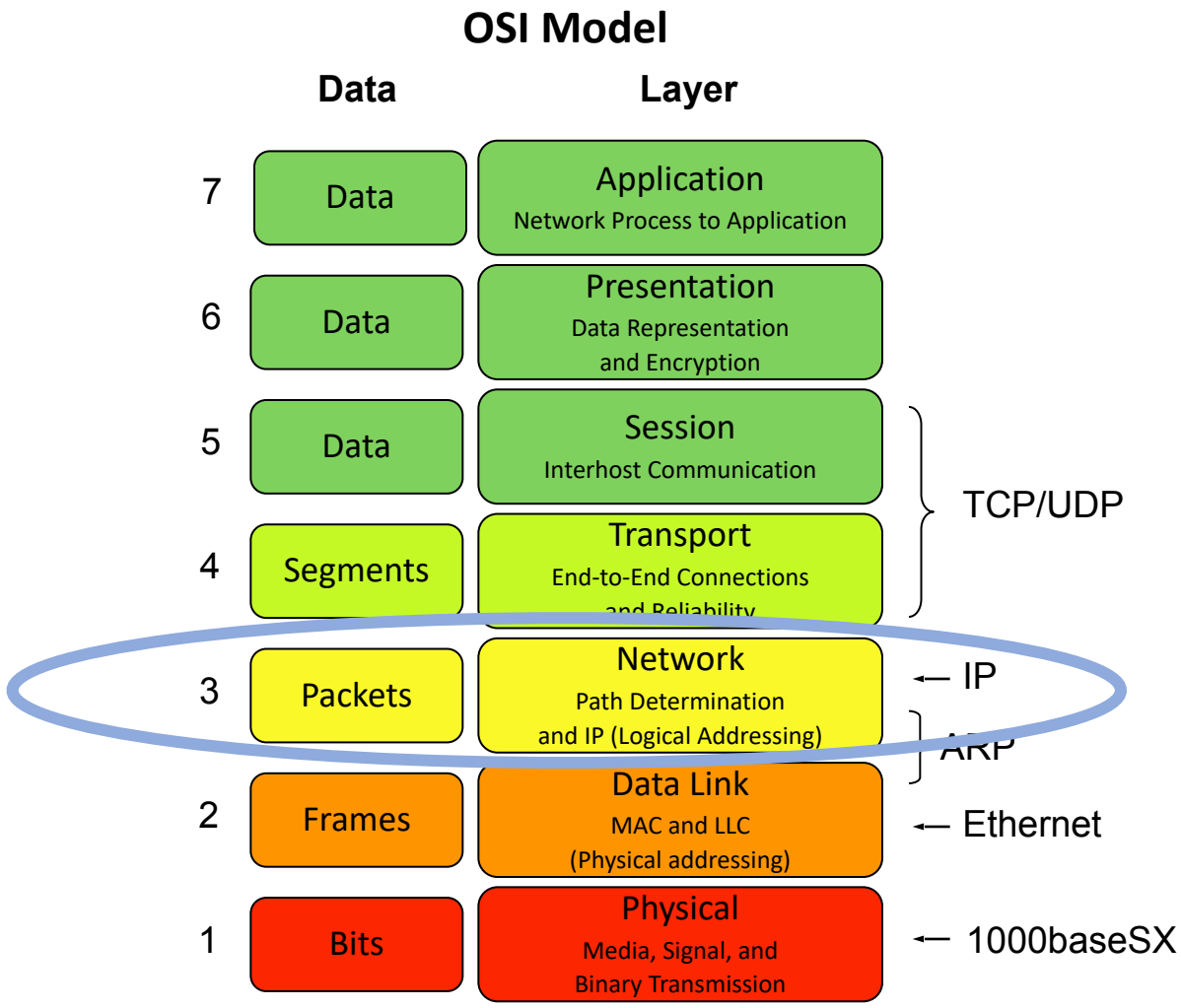
Now let's look lower

- IP lives at the network layer



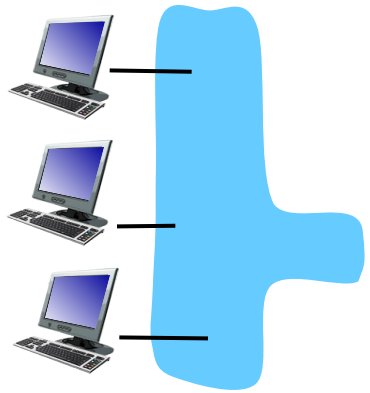
Now let's look lower

- 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

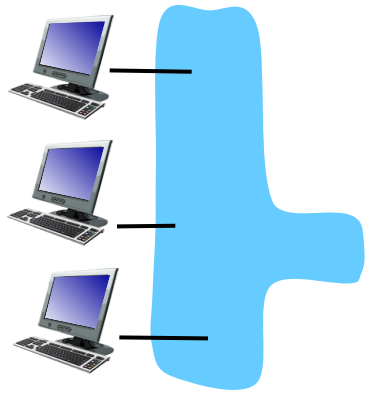


- No public internet

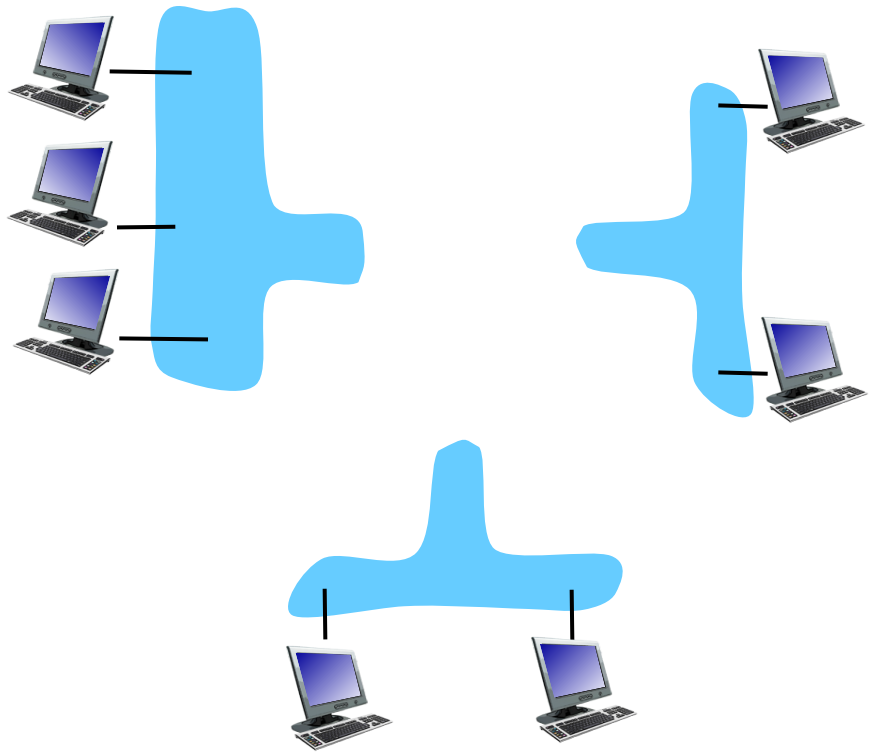
- No public internet



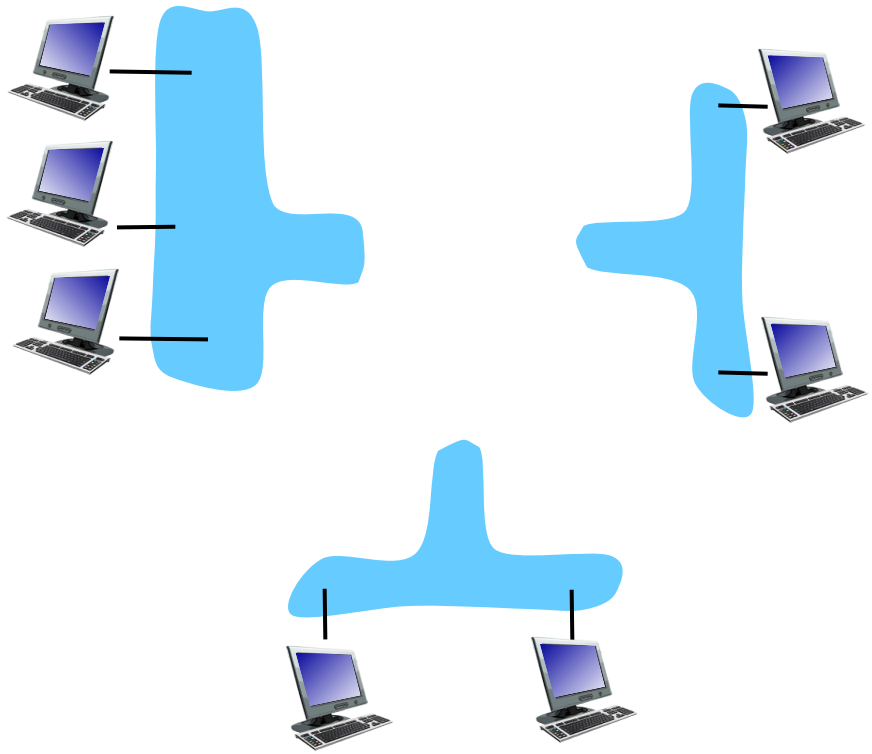
- No public internet
- Islands of connectivity
  - Data link layer networks all over, even in 1983
    - Ethernet
    - 802.11 wireless
    - Token Ring



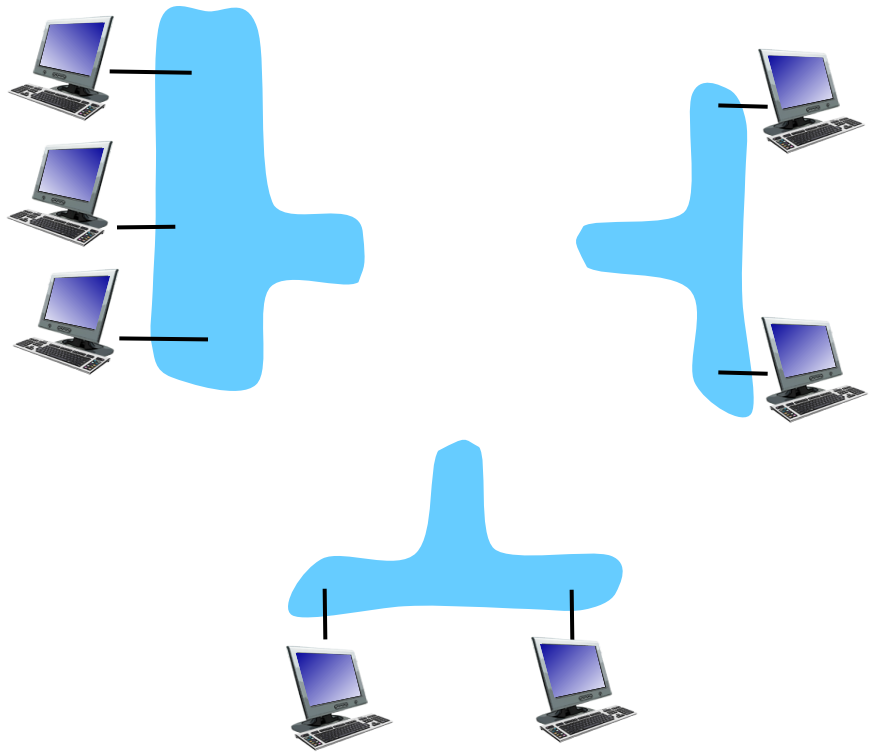
- No public internet
- Islands of connectivity
  - Data link layer networks all over, even in 1983
    - Ethernet
    - 802.11 wireless
    - Token Ring
    - AppleTalk
    - RS232 serial
    - ATM
    - And plenty more



- 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?

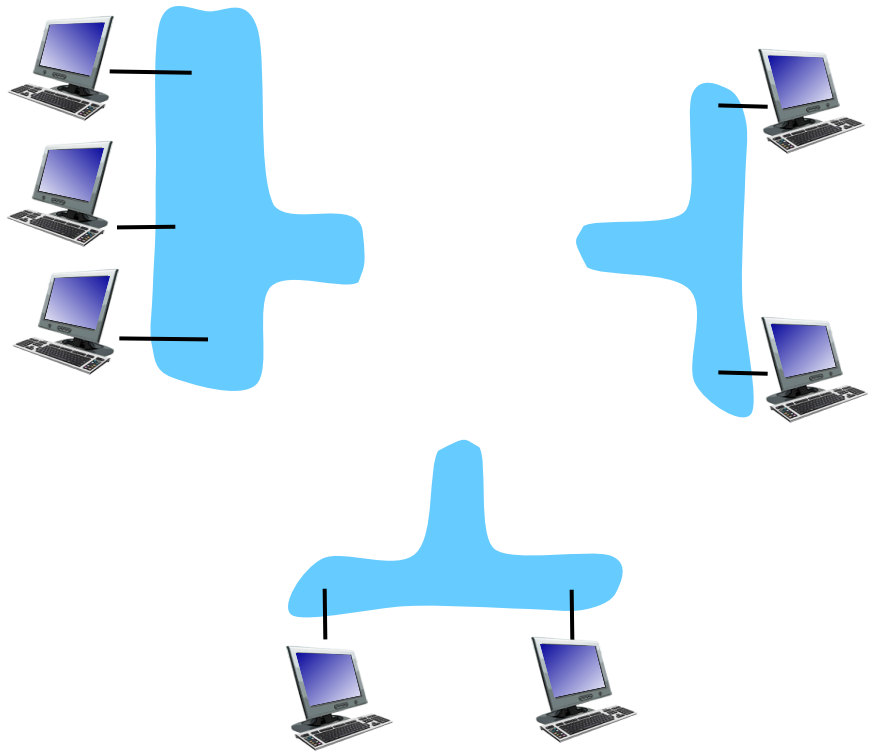


- Build a network layer that can address and interconnect all sorts of different data link networks

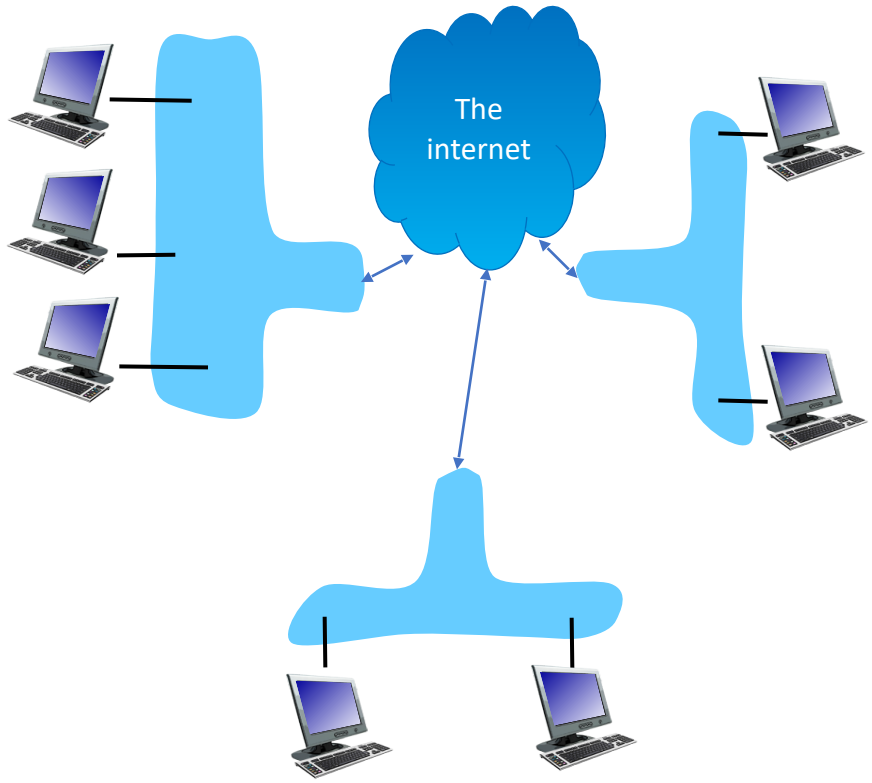


- Build a network layer that can address and interconnect all sorts of different data link networks

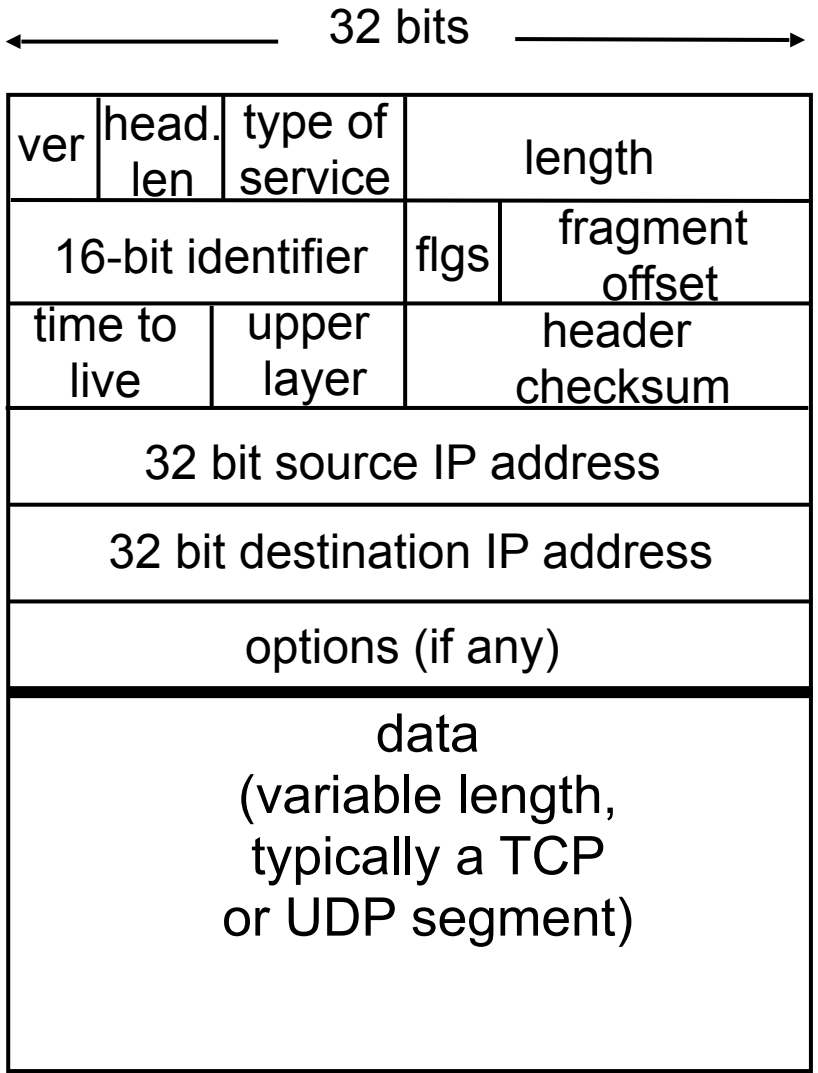
- After a decade of innovation and another decade of growth



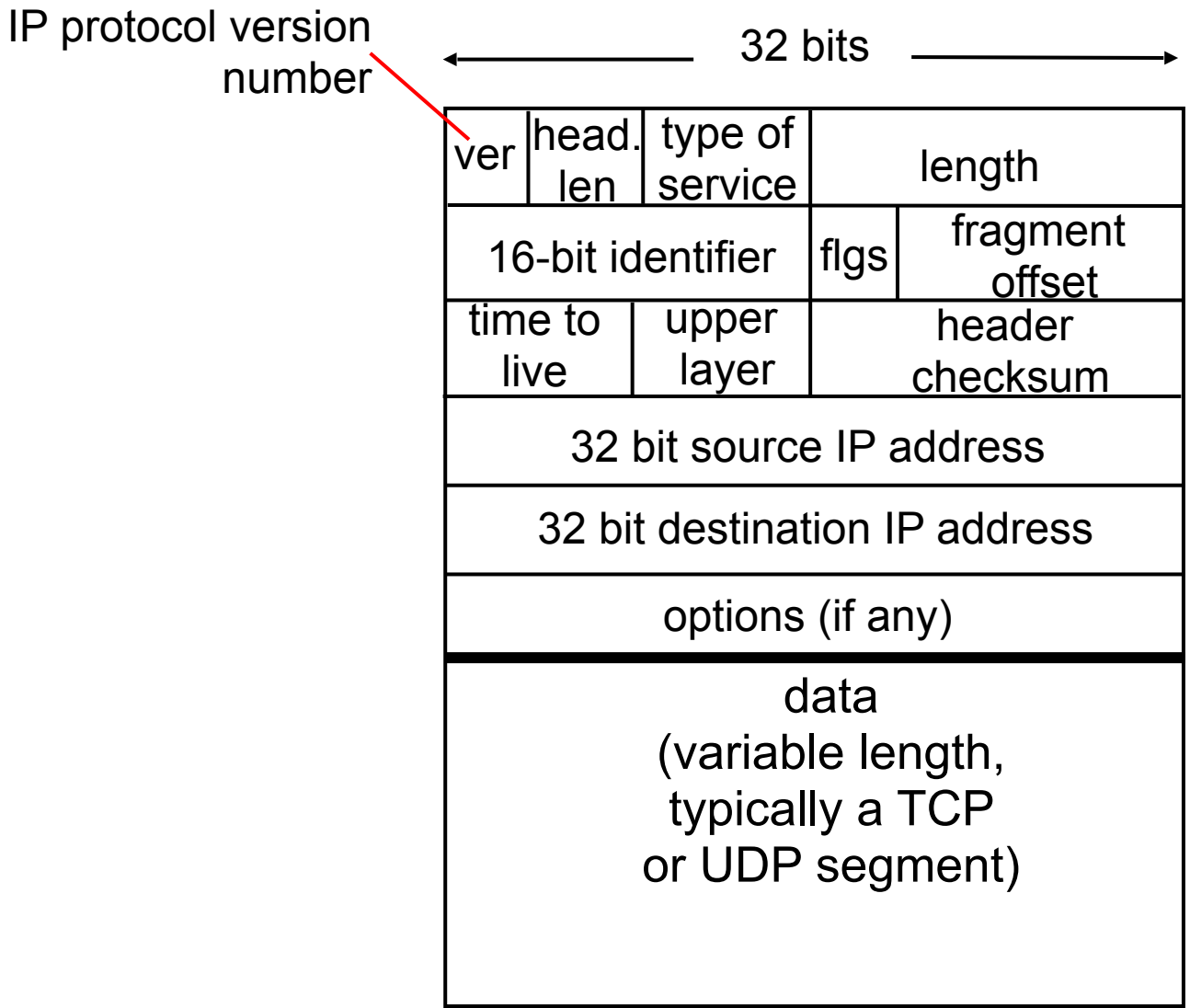
- Build a network layer that can address and interconnect all sorts of different data link networks
- After a decade of innovation and another decade of growth
- In 1980, IP for everybody (version 4 that is) with two built-in transport protocols, TCP and UDP



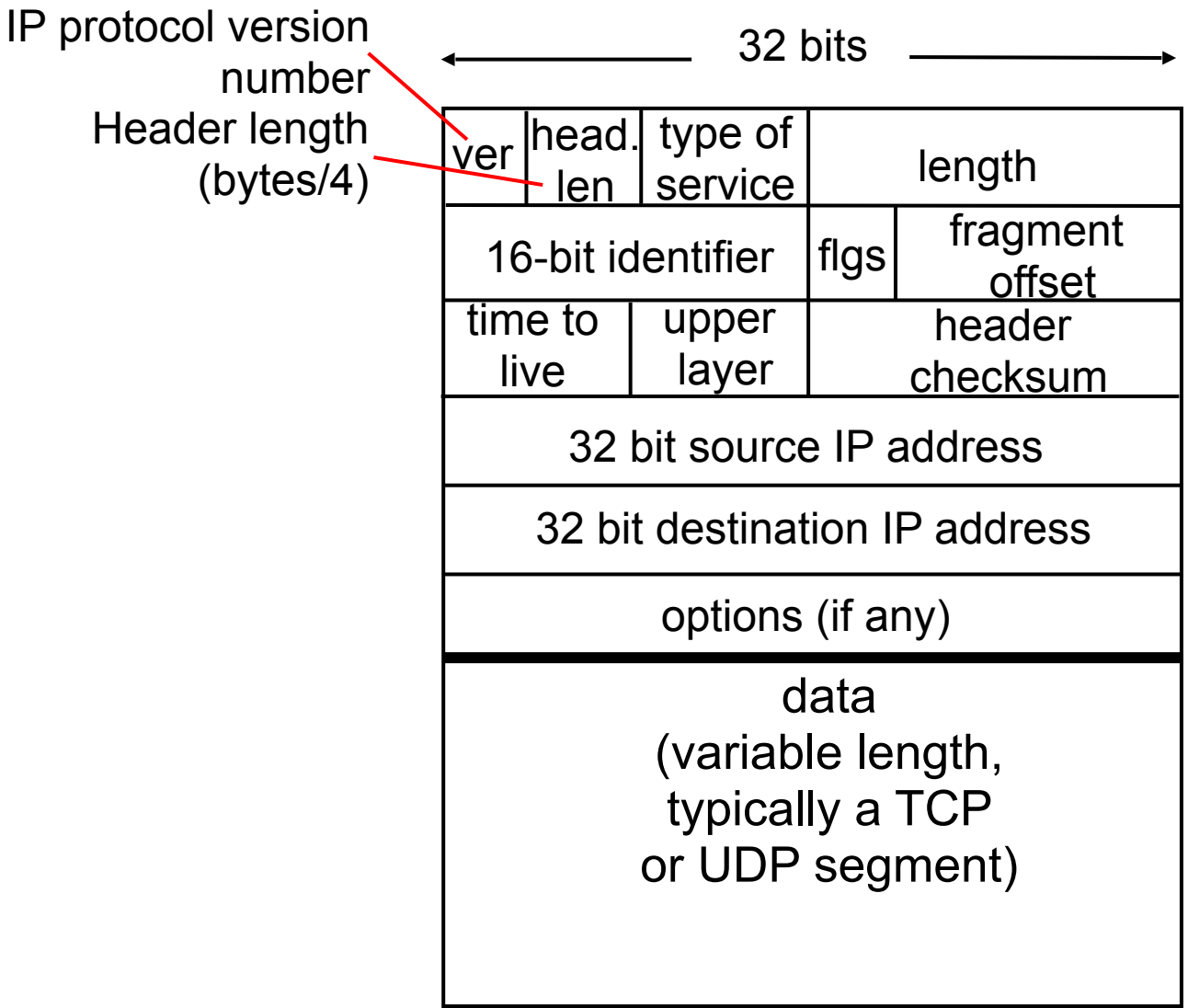
IP datagram (packet) format



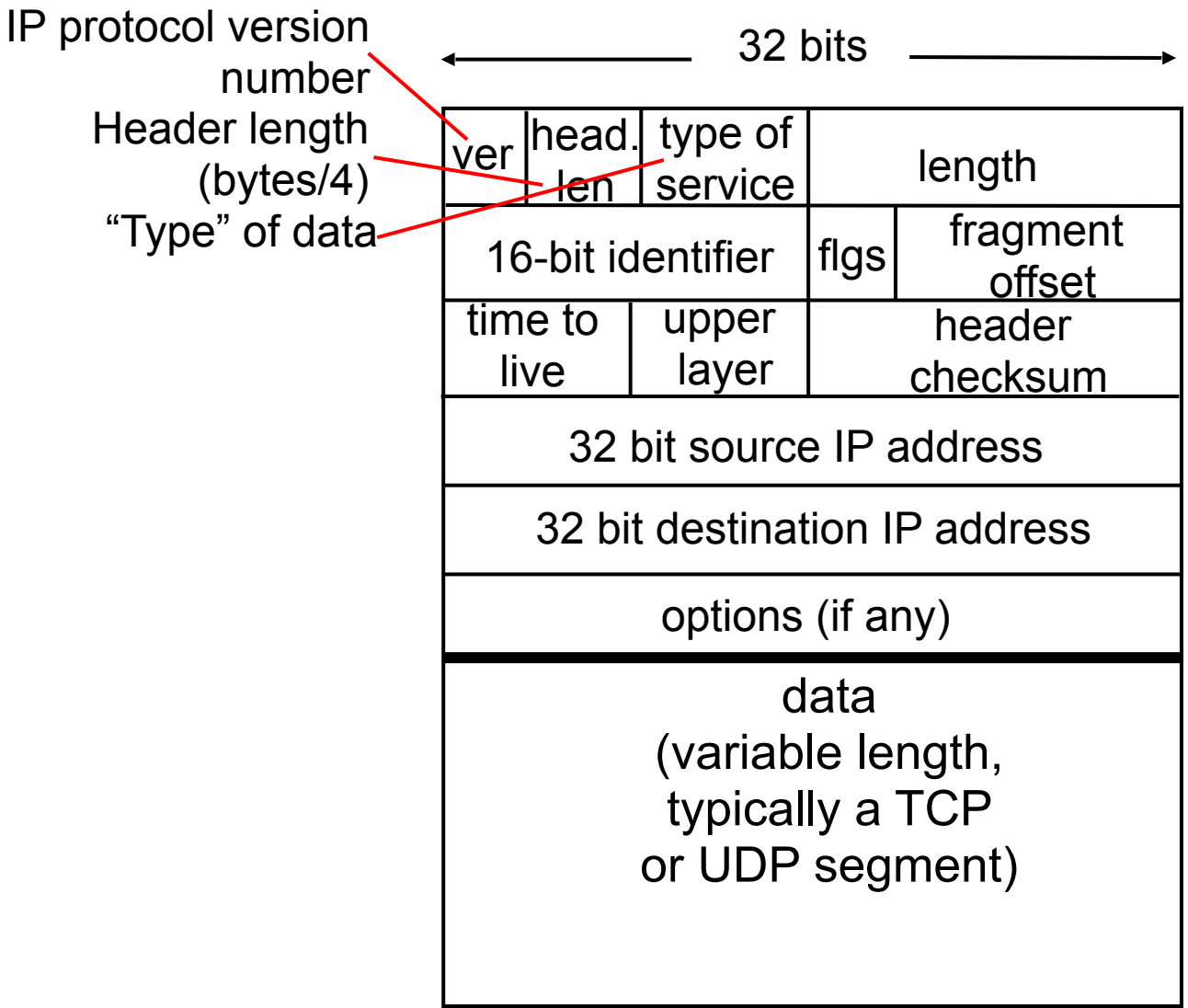
IP datagram (packet) format



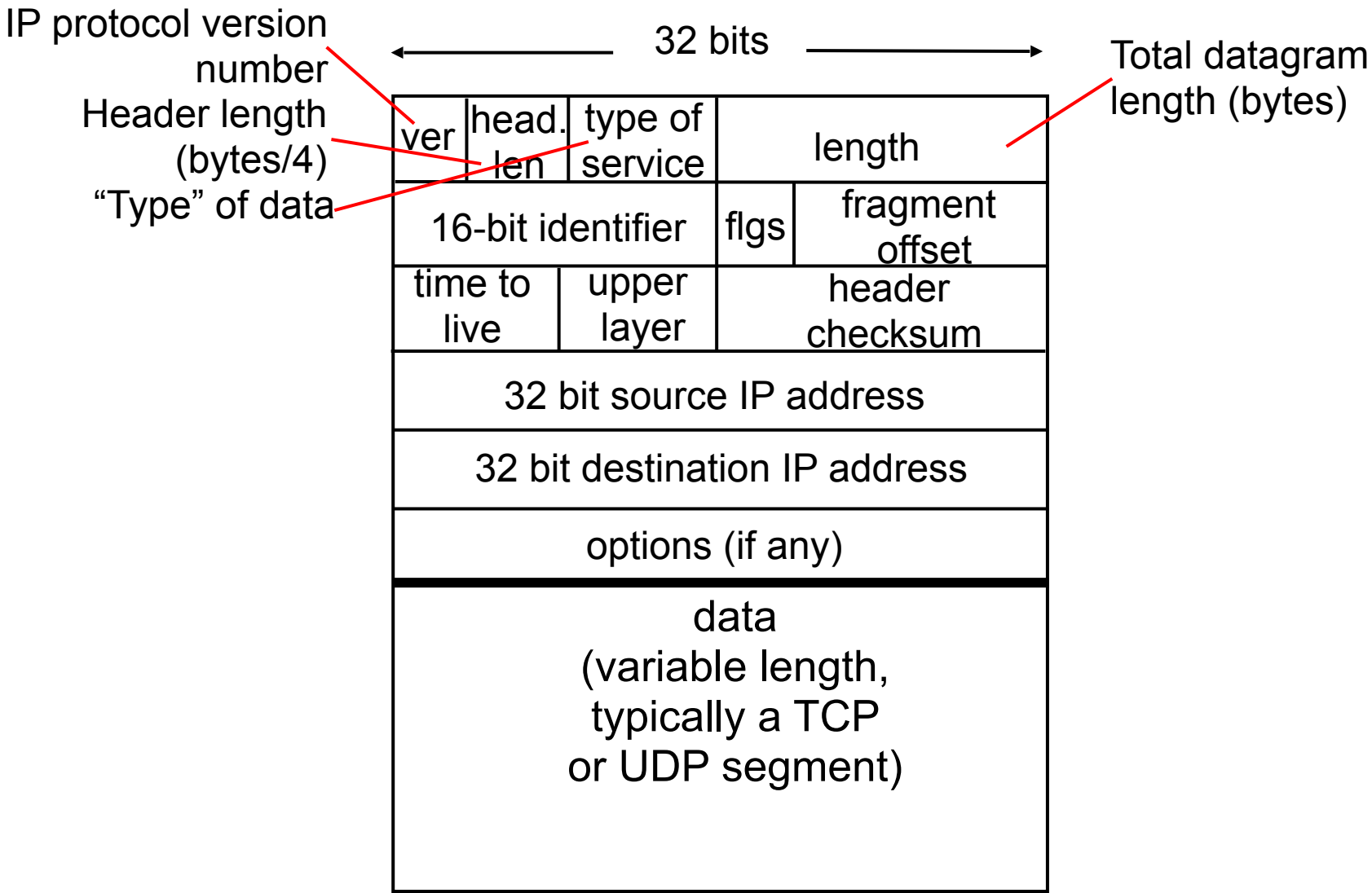
IP datagram (packet) format



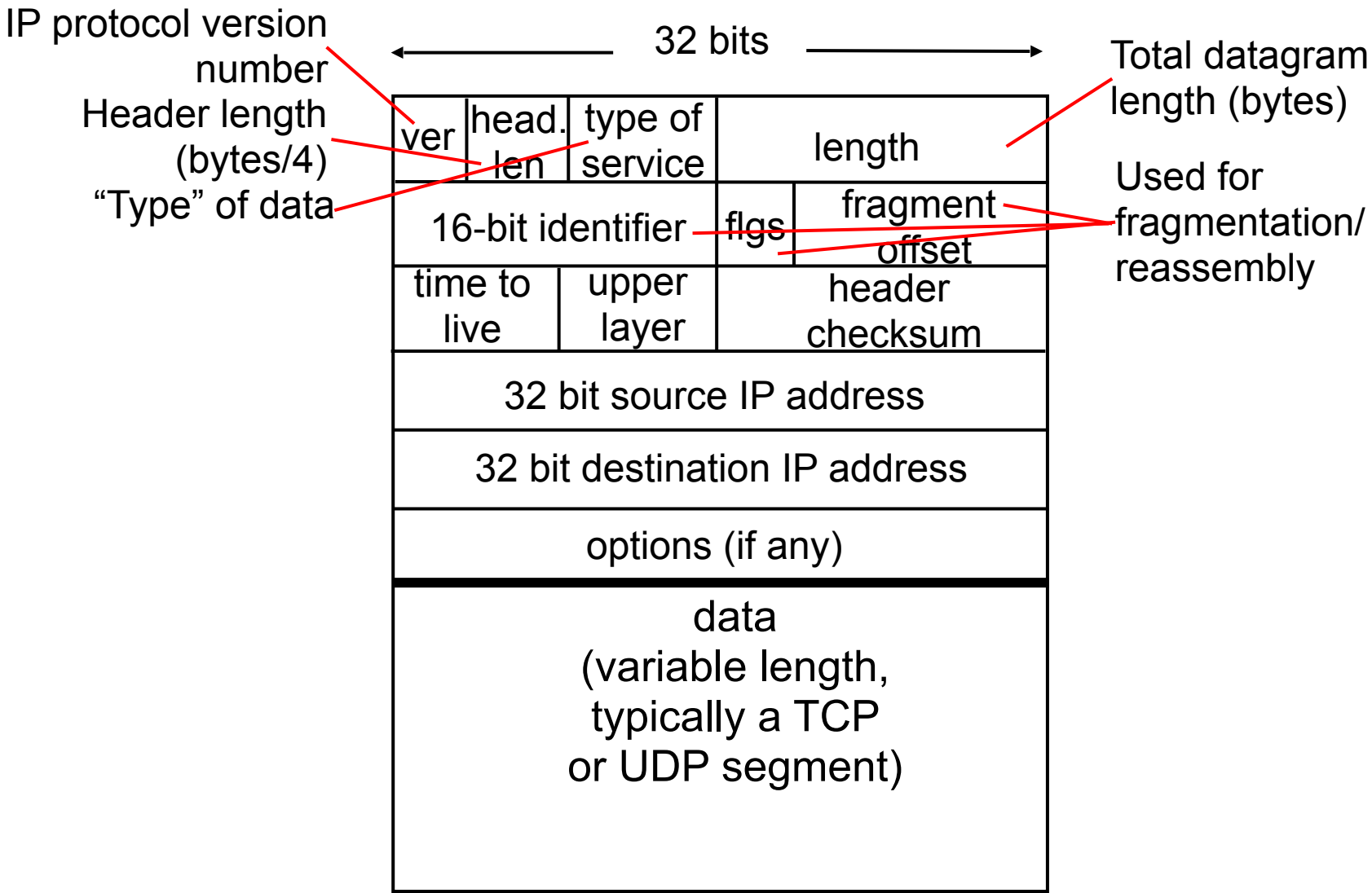
IP datagram (packet) format



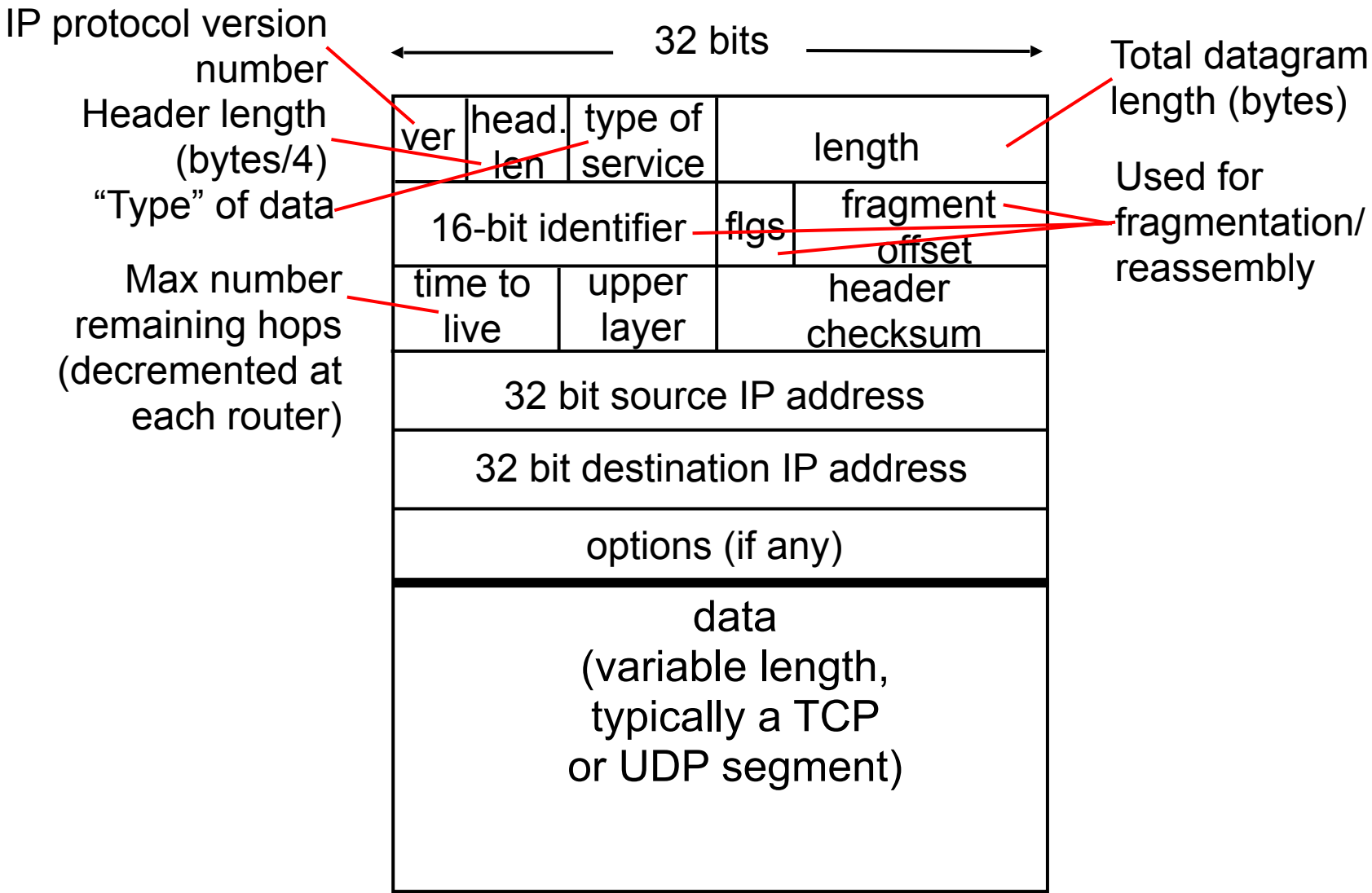
IP datagram (packet) format



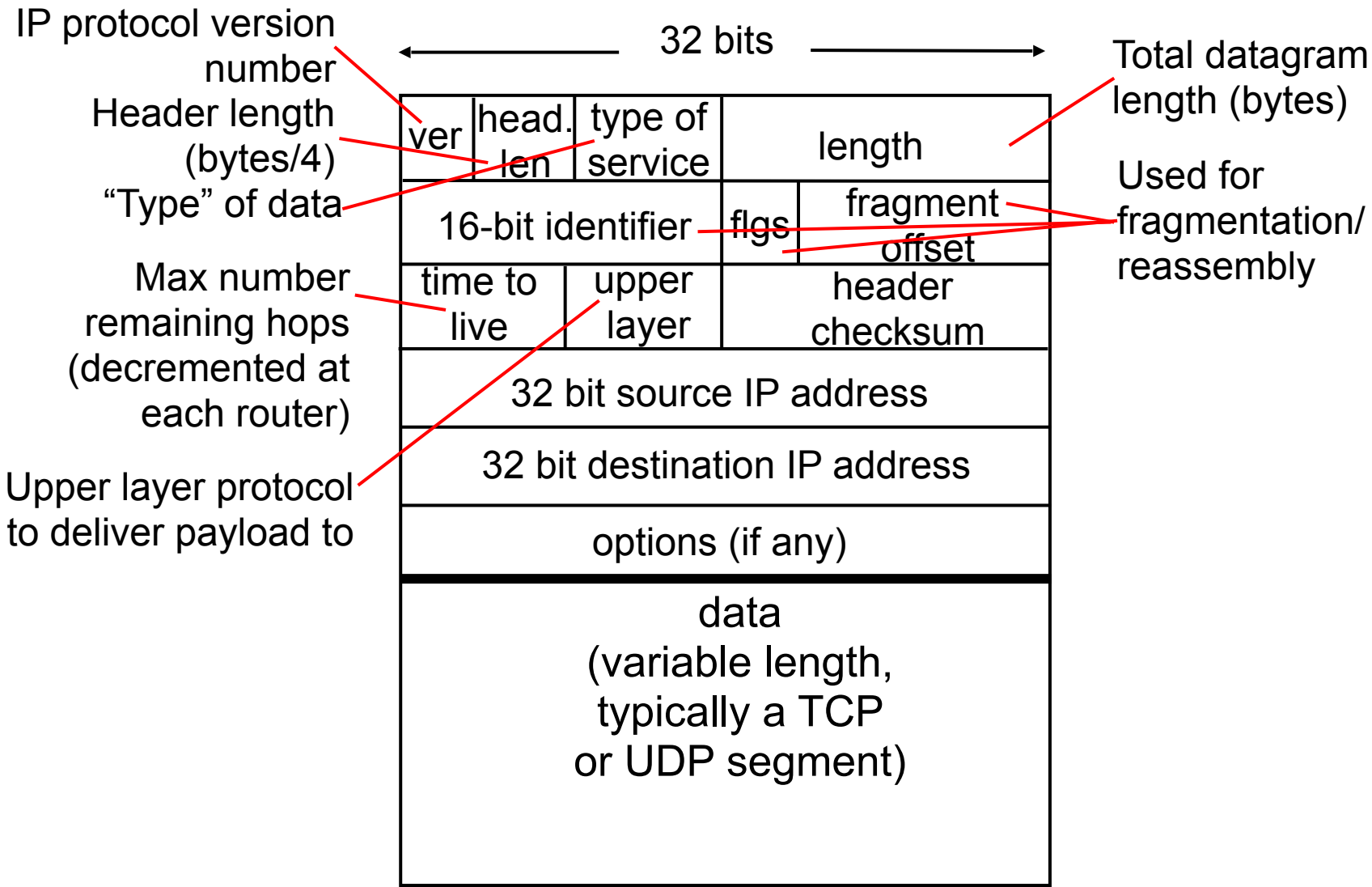
IP datagram (packet) format



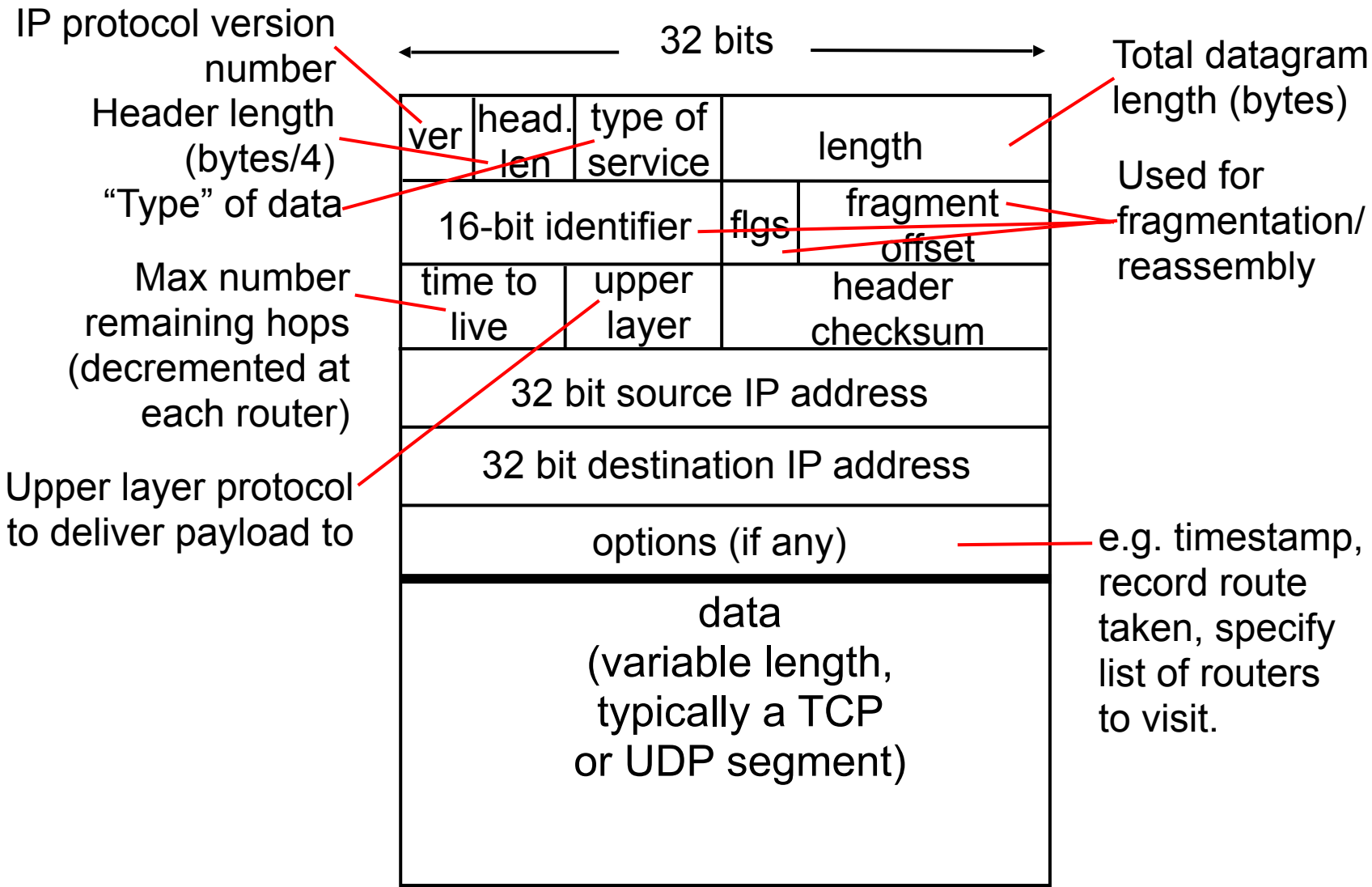
IP datagram (packet) format



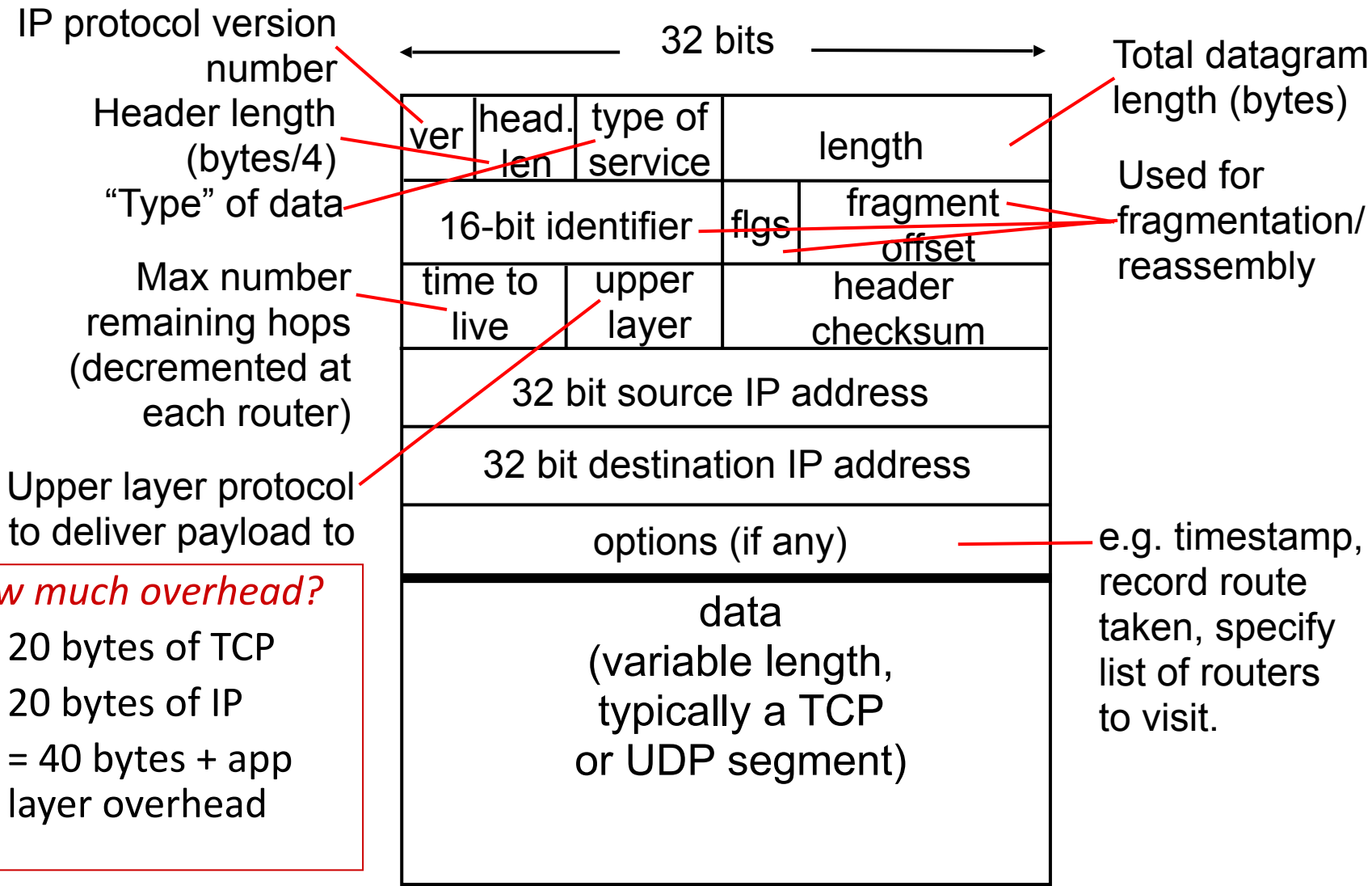
IP datagram (packet) format



IP datagram (packet) format



IP datagram (packet) format



how much overhead?

- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

IPv4 Header

IPv4 Header Format

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																																

TCP Segment Header

TCP segment header

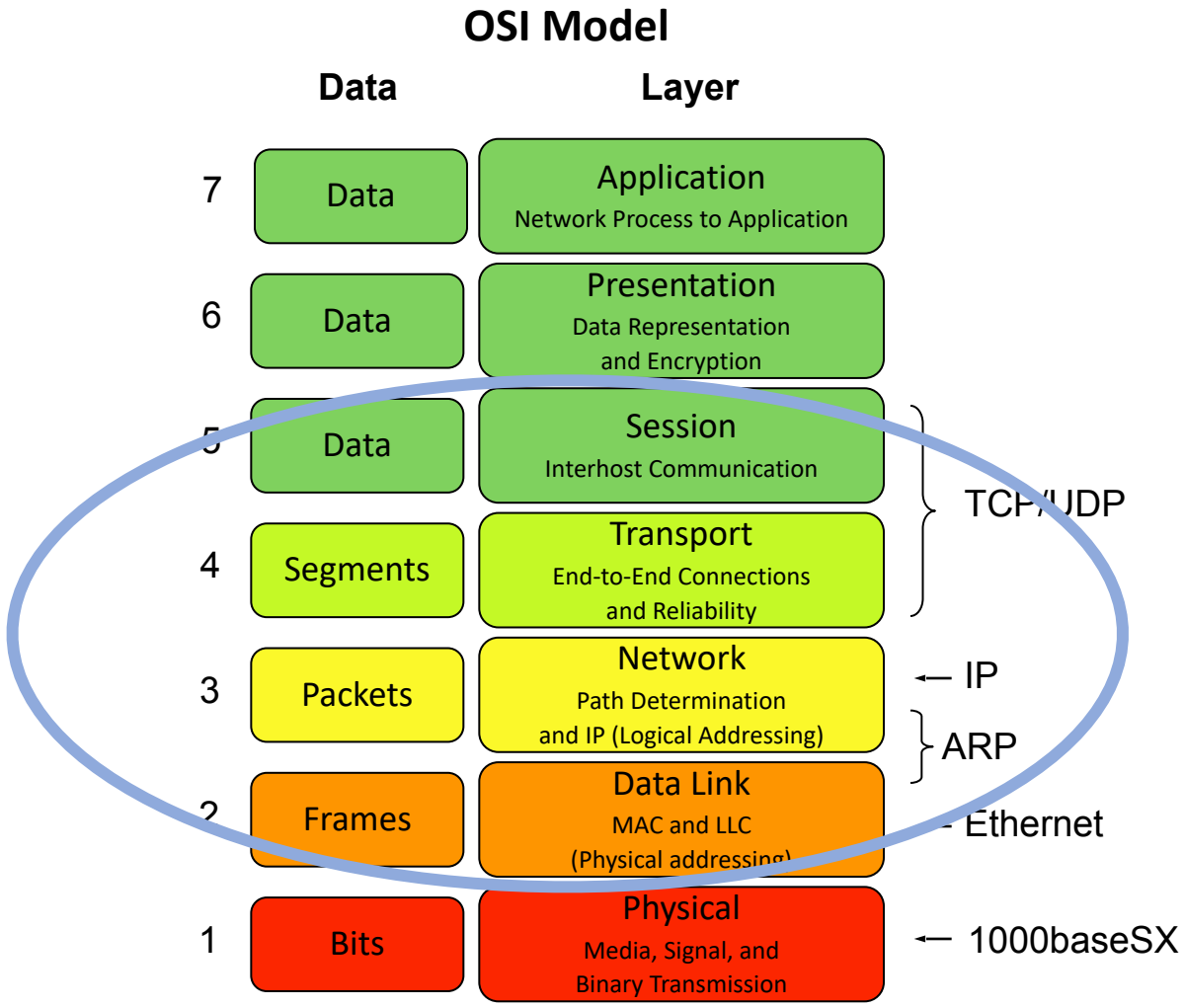
Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 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
  - i.e., segments larger than MTU – Maximum Transmission Unit
- Probably should be considered deprecated these days
- IPv6 doesn’t allow it

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…

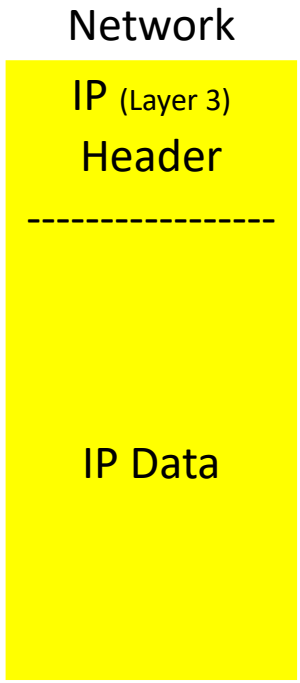


Encapsulated packets

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

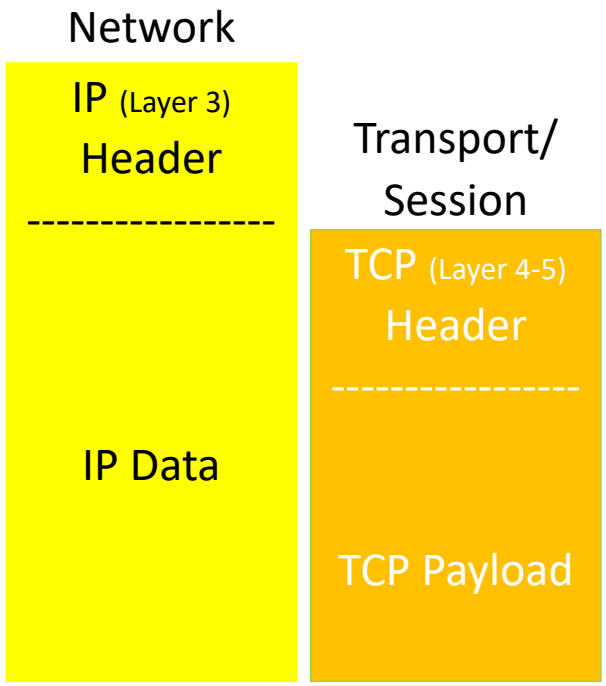
Encapsulated packets

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



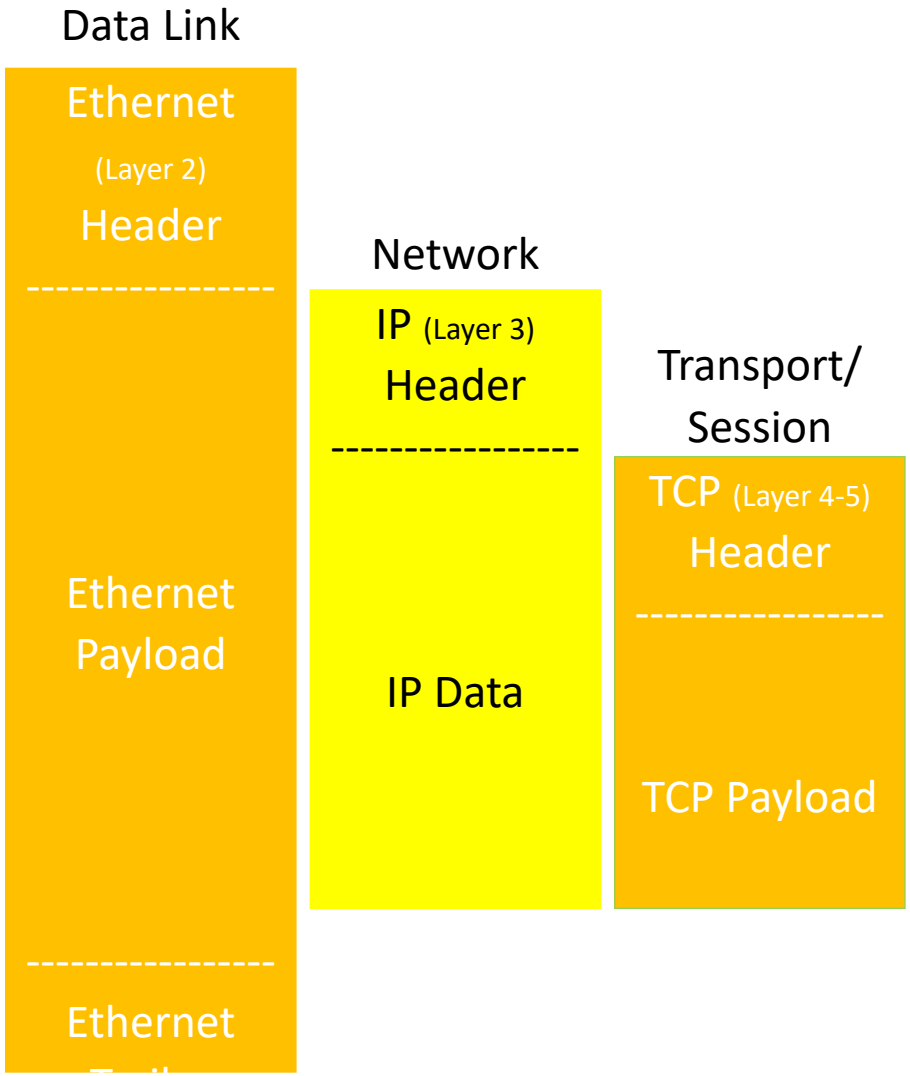
Encapsulated packets

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



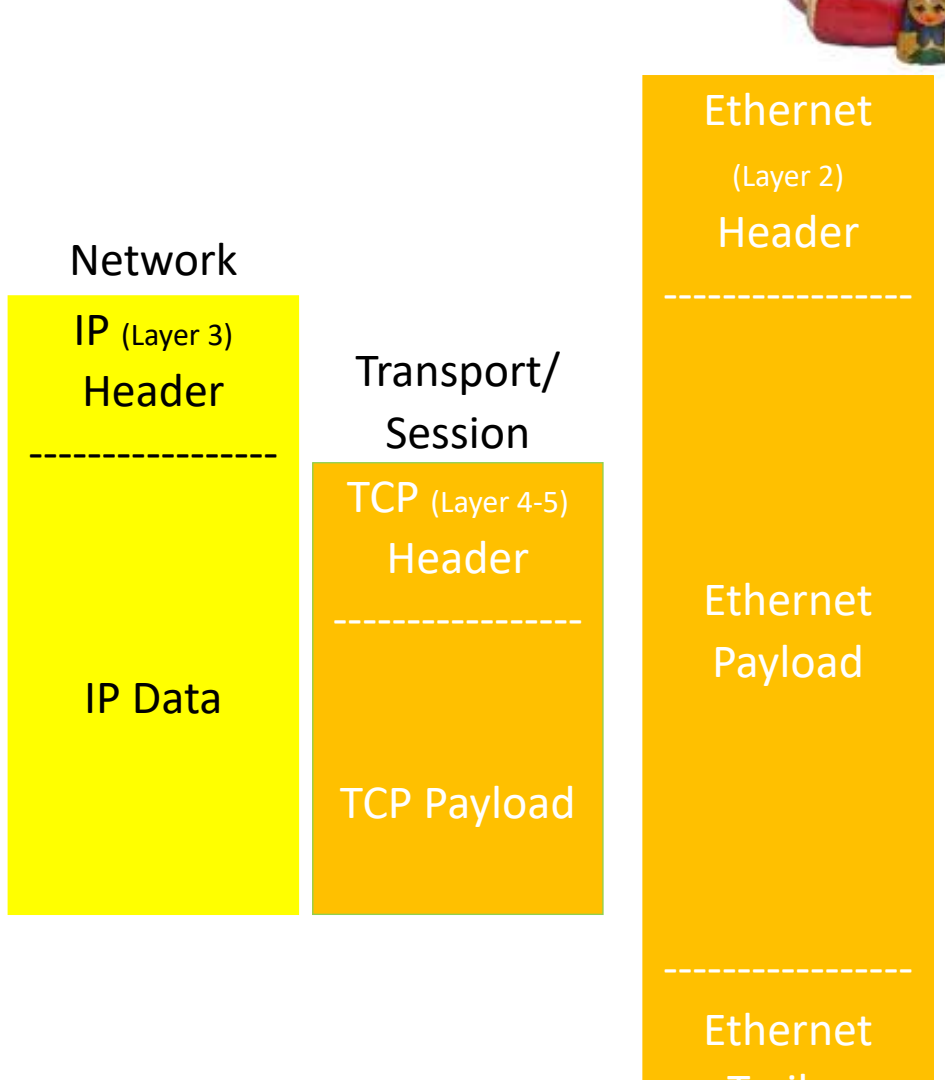
Encapsulated packets

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



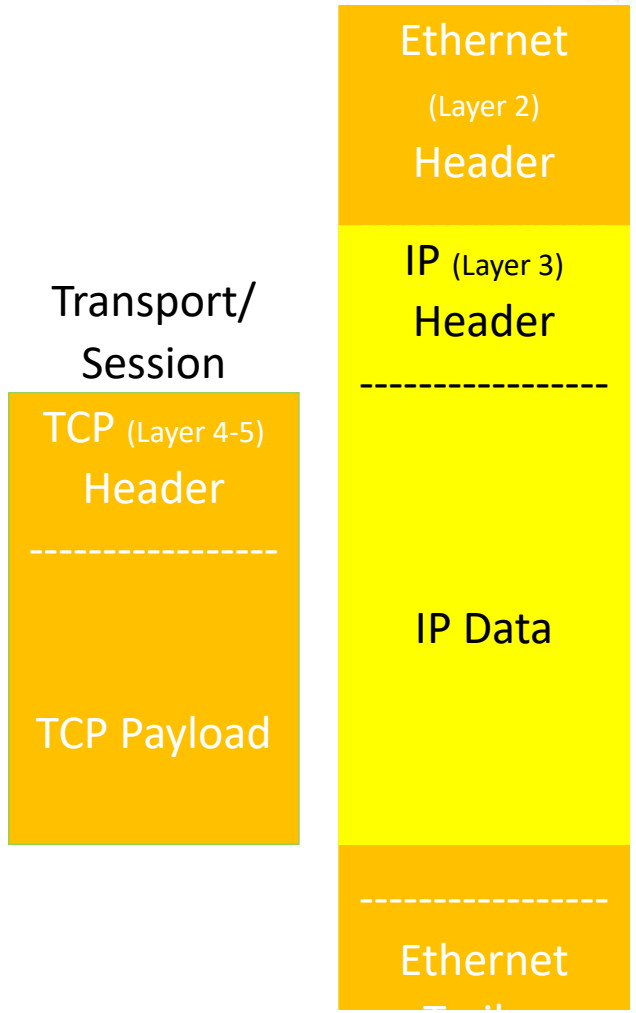
Encapsulated packets

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



Encapsulated packets

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



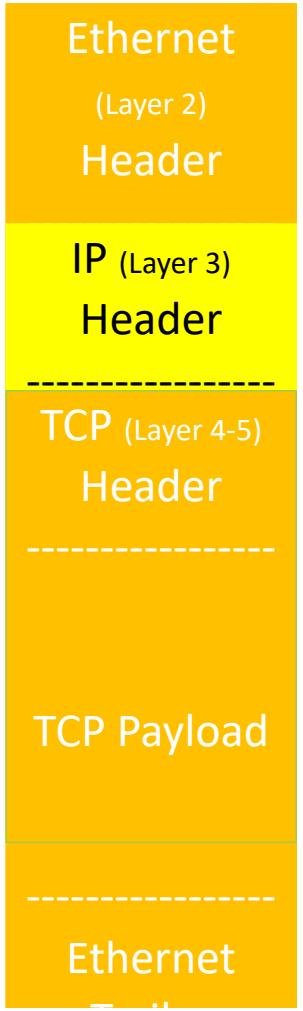
Encapsulated packets

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

Data Link

Network

Transport/
Session



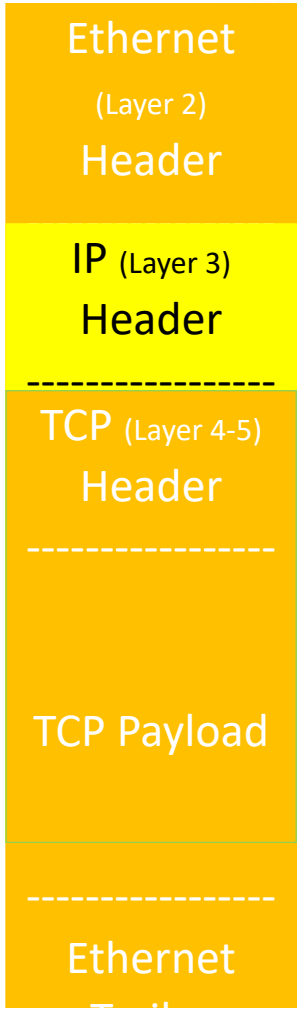
Encapsulated packets

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

Data Link

Network

Transport/
Session



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

IPv4 addresses

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

IPv4 addresses

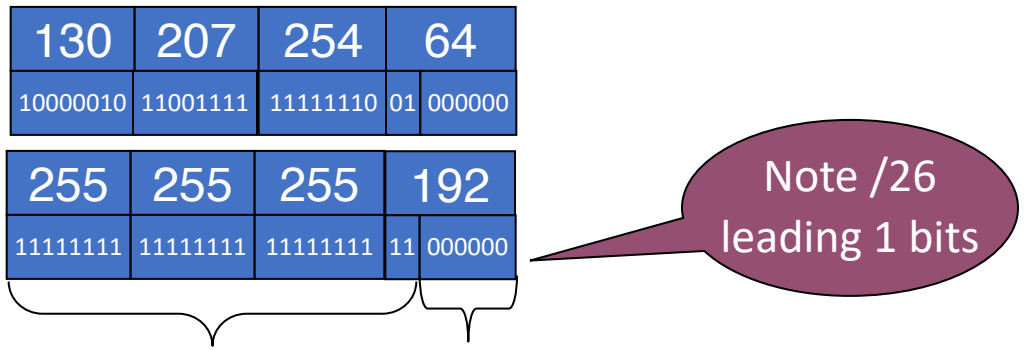
- 32 bits / 4 octets / four-decimal dotted quad
  - Did you know that 130.207.7.31 becomes 0x82CF071F?
  - Each of those four numbers is decimal for a single byte
- And we need a mask

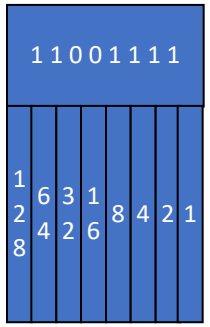
IPv4 addresses

- 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., IP address 130.207.7.31 with 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 ← CIDR notation
- 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





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
  - The means the data link layer will be called on to deliver the packet
    - The network layer's responsibility is only to inject the packet into the right network!

What does it mean to be on the same network?

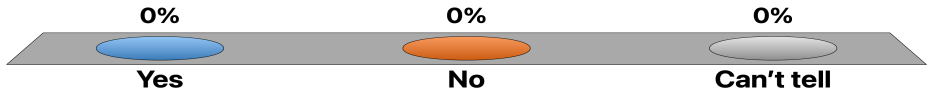
- In the IP world it's simple
  - If network bits between two addresses are equal, they are on the same network
  - The means the data link layer will be called on to deliver the packet
    - The network layer's responsibility is only to inject the packet into the right network!

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. Yes
B. No
C. Can't tell

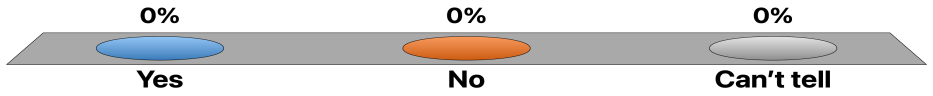


Are these two IP addresses on the same network?

143.215.14.243 and 143.214.14.242 if the mask for each is /24

A. Yes
B. No
C. Can't tell

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

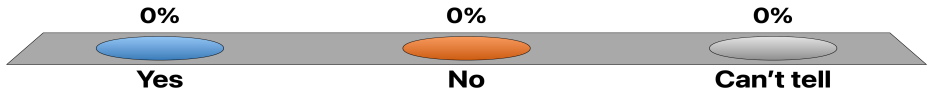


Are these two IP addresses on the same network?

143.215.14.243 and 143.214.14.242 if the mask for each is /24

A. Yes
B. No
C. Can't tell

Keeping the first 24 bits of the two addresses, we get
143.215.14.0



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

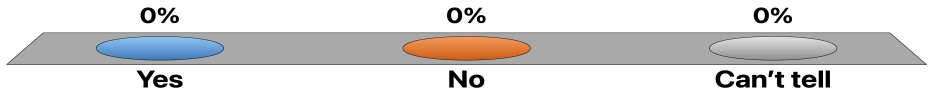
A. Yes
B. No
C. Can't tell

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

143.215.14.0

and

143.214.14.0



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. Yes
B. No
C. Can't tell

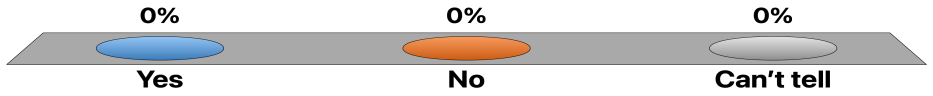
Keeping the first 24 bits of the two addresses, we get

143.215.14.0

and

143.214.14.0

Are they the same number?



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. Yes
B. No
C. Can't tell

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

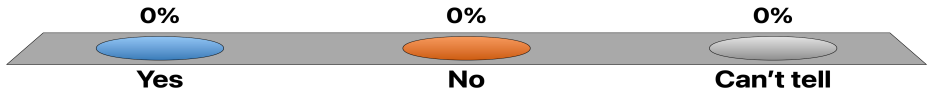
143.215.14.0

and

143.214.14.0

Are they the same number?

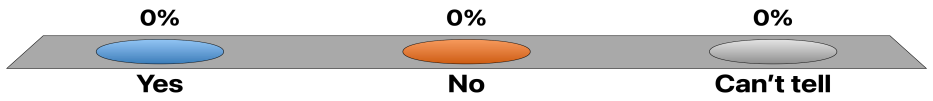
Nope.



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. Yes
B. No
C. Can't tell

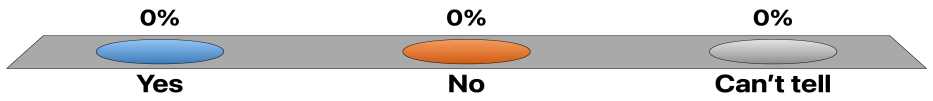


Are these two IP addresses on the same network?

143.215.14.243 and 143.214.14.242 if the mask for each is /15

A. Yes
B. No
C. Can't tell

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

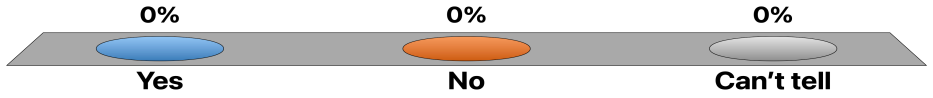


Are these two IP addresses on the same network?

143.215.14.243 and 143.214.14.242 if the mask for each is /15

A. Yes
B. No
C. Can't tell

Keeping the first 15 bits of the two addresses, we get
143.214.0.0 (215=11010111_2; first 7 bits are 11010110_2)



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

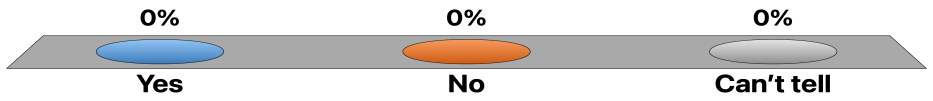
A. Yes
B. No
C. Can't tell

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

143.214.0.0 (215=11010111_2; first 7 bits are 1101011_2)

and

143.214.0.0 (214=11010110_2; first 7 bits are 1101011_2)



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. Yes
B. No
C. Can't tell

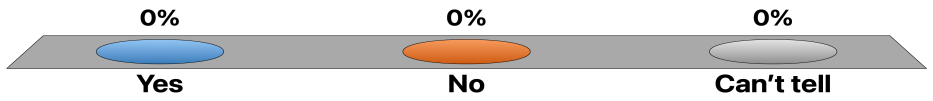
Keeping the first 15 bits of the two addresses, we get

143.214.0.0 (215=11010111_2; first 7 bits are 11010110_2)

and

143.214.0.0 (214=11010110_2; first 7 bits are 11010110_2)

Are they the same number?



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. Yes
B. No
C. Can't tell

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

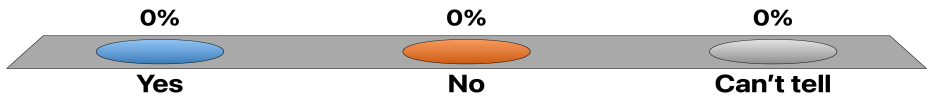
143.214.0.0 (215=11010111_2; first 7 bits are 11010110_2)

and

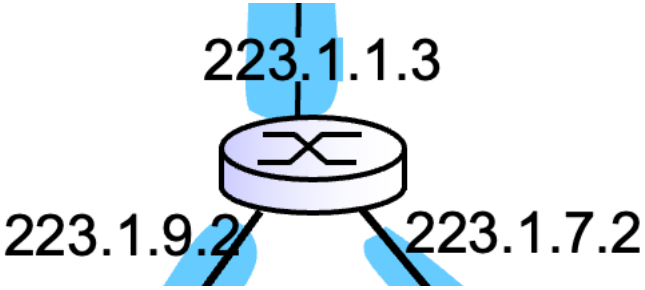
143.214.0.0 (214=11010110_2; first 7 bits are 11010110_2)

Are they the same number?

Yes!

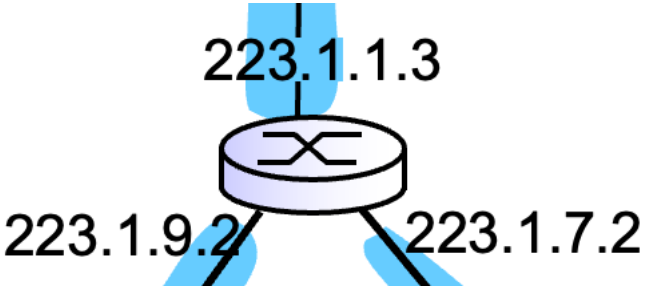


Subnets



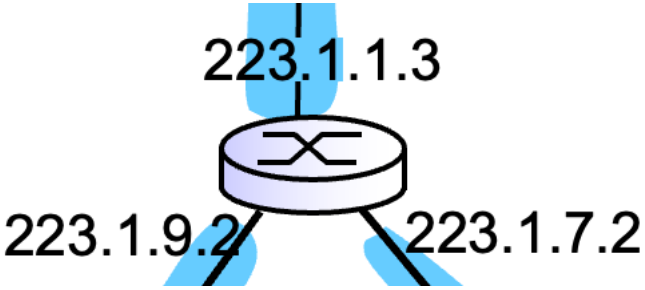
- Remember those Islands of Connectivity? And that mask?
- First rule of IP routing:
  - If you're on the same network, punt it to the data link layer to deliver
  - Otherwise, use your IP routing table to forward it to another IP host on your network

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, so masks don't need to be passed in packets

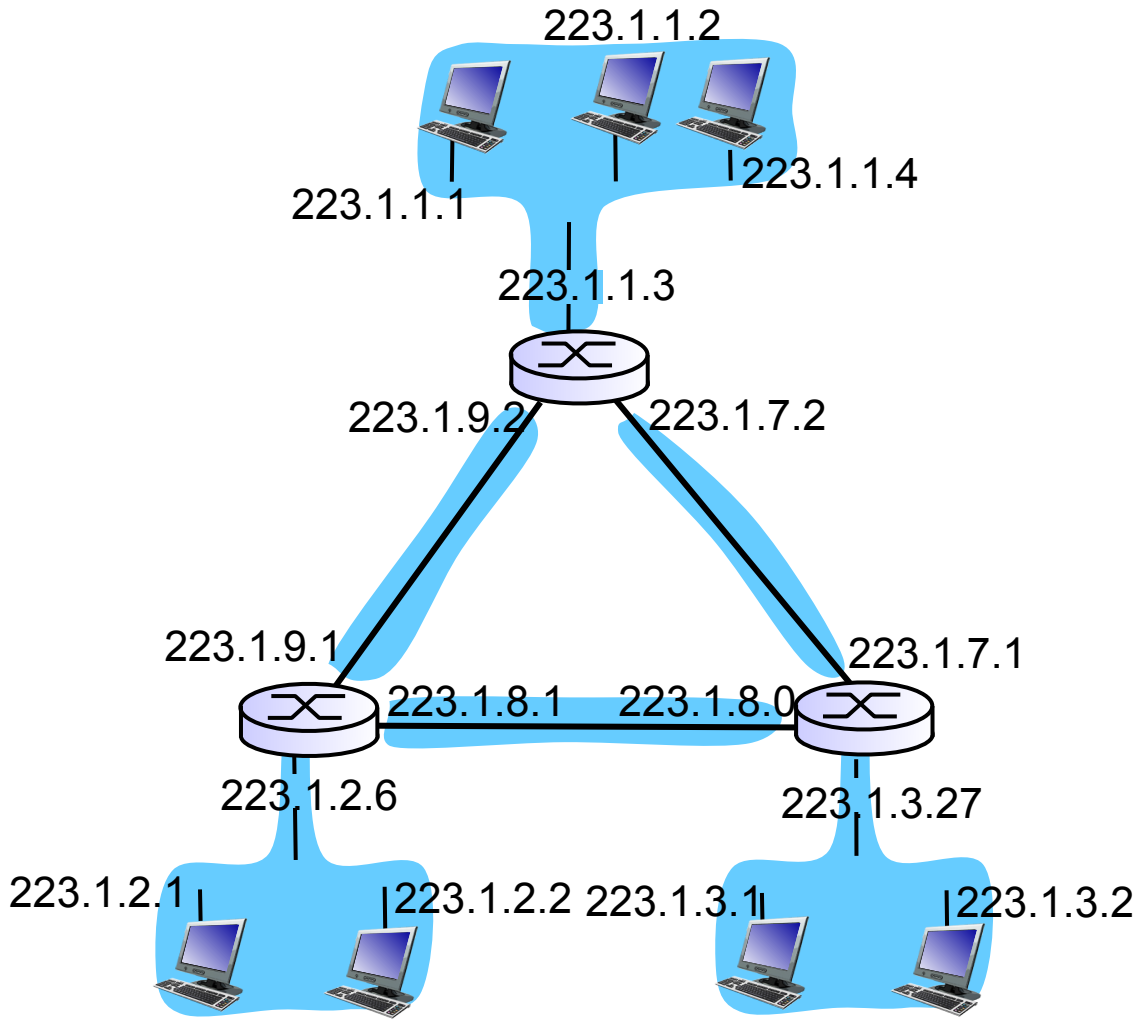
Subnets



- 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, 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).

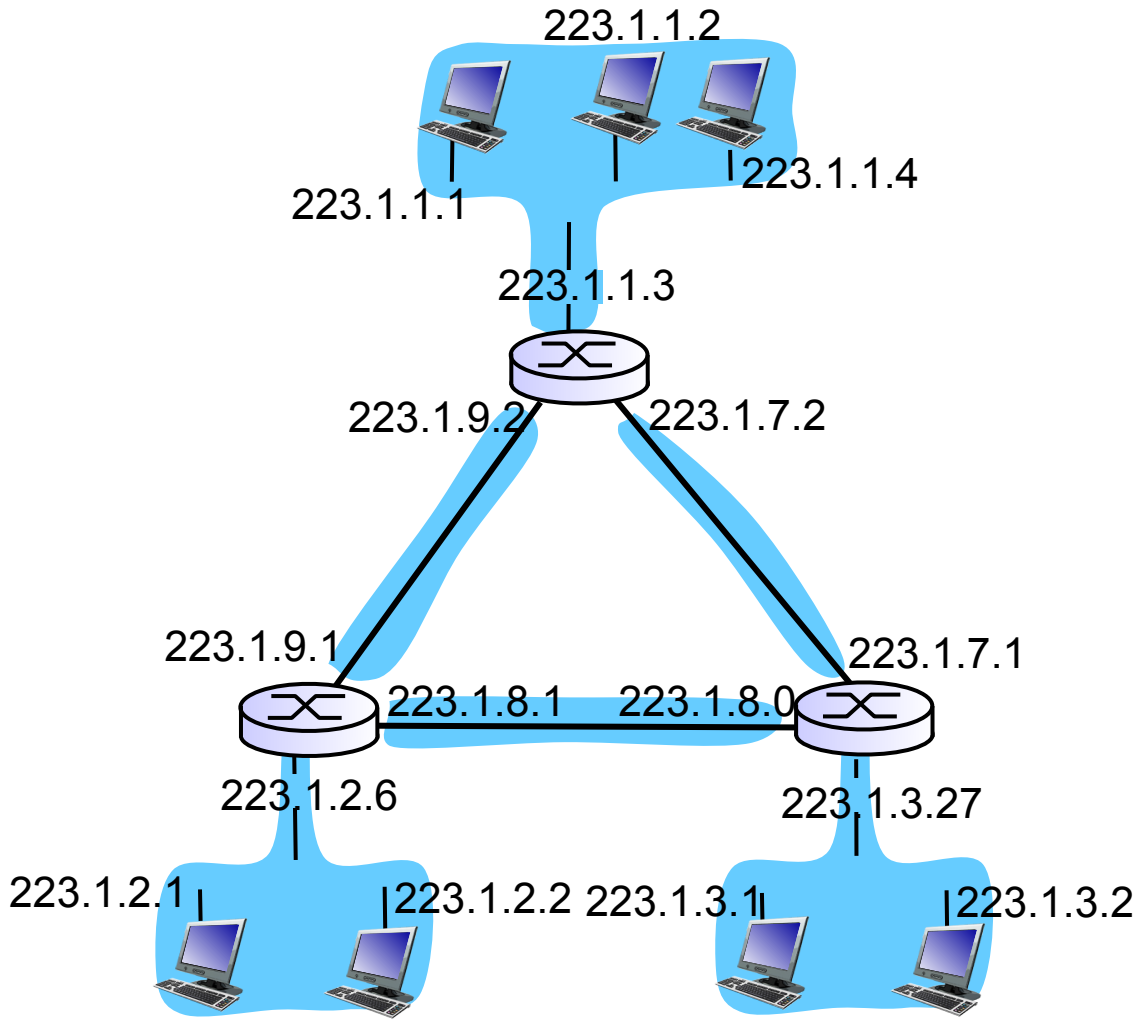
Connecting the Islands

- Data link layer networks connected by IP hosts



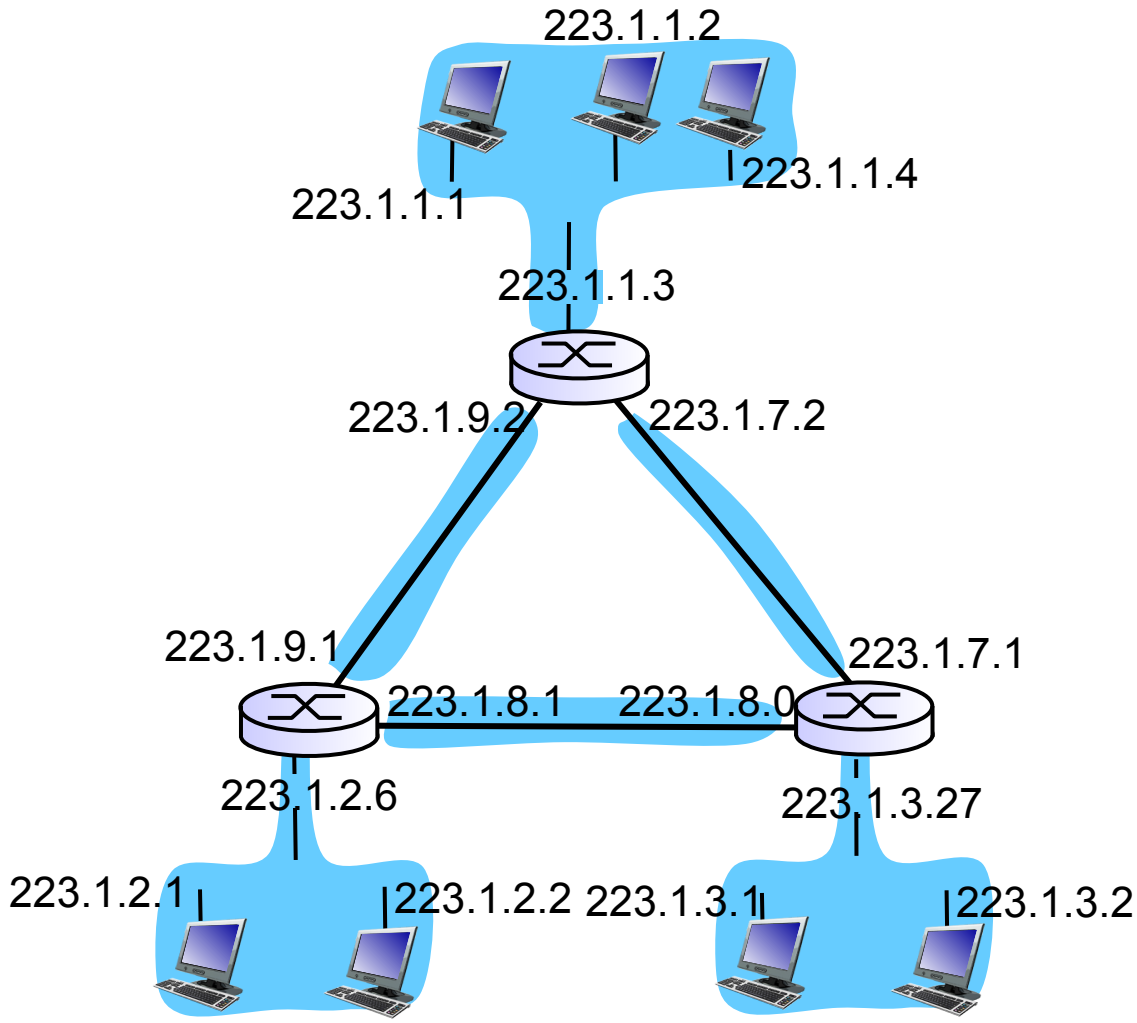
Connecting the Islands

- Data link layer networks connected by IP hosts
- Now where do the masks come in?



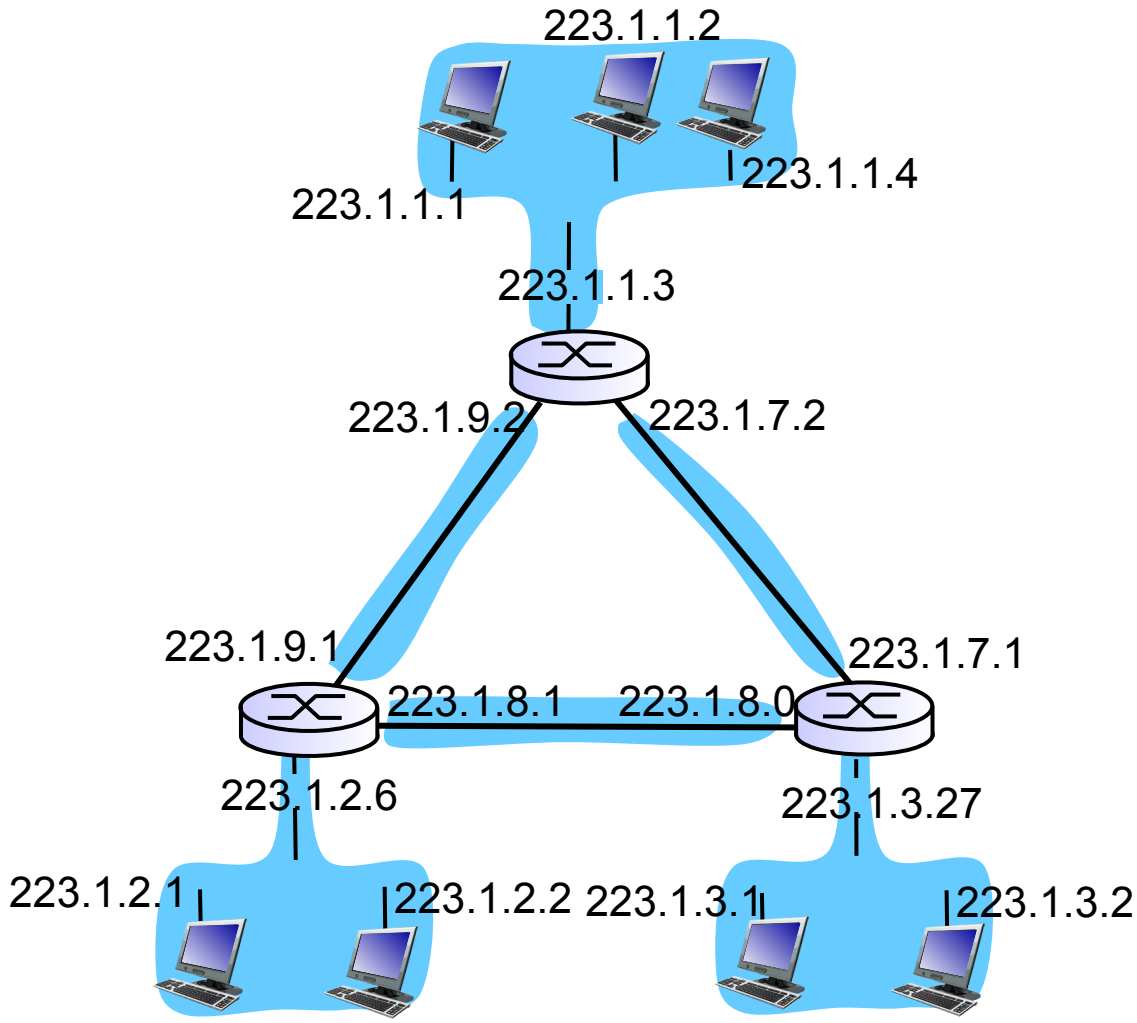
Connecting the Islands

- Data link layer networks connected by IP hosts
- Now where do the masks come in?
- Clue: All the masks here are /24.



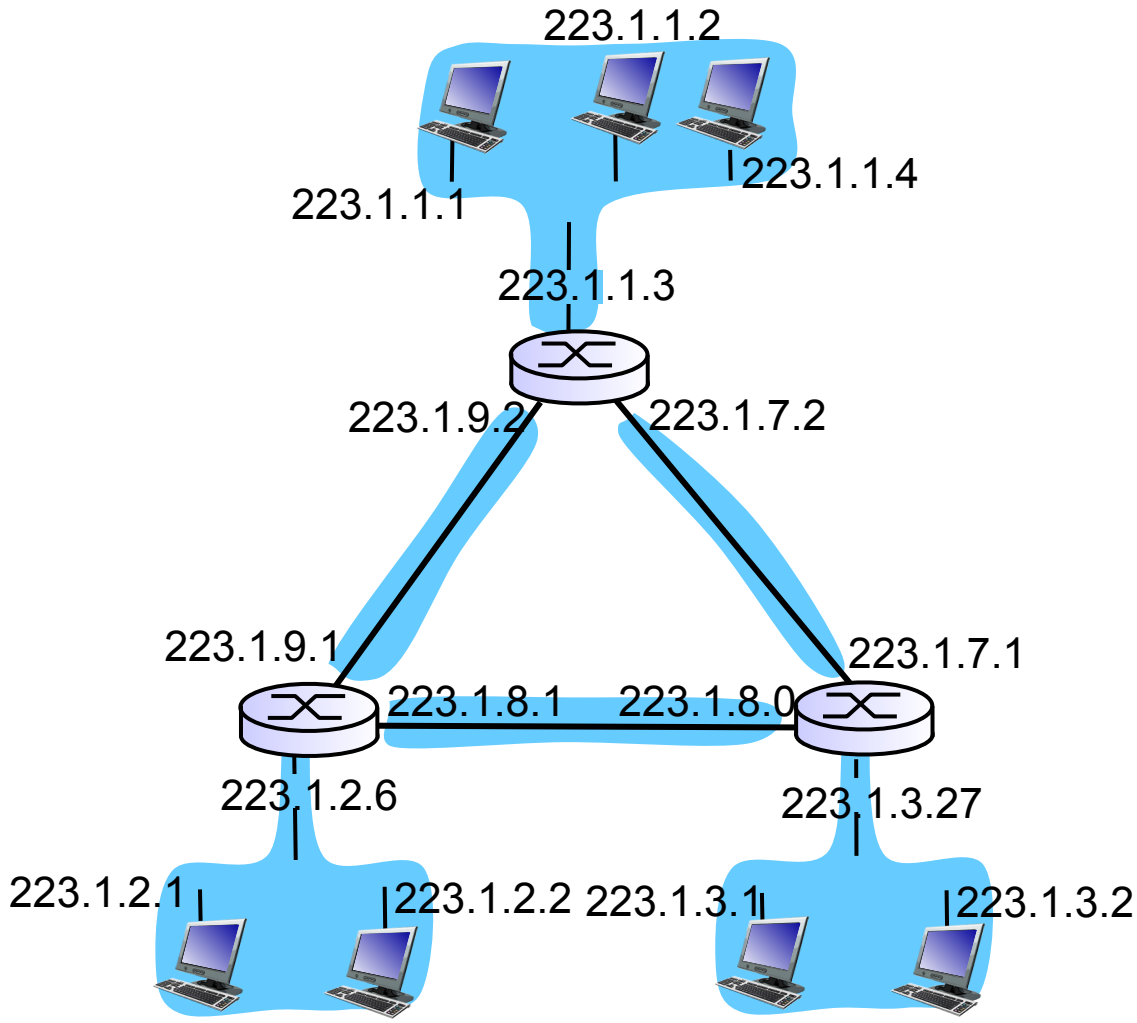
Connecting the Islands

- 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



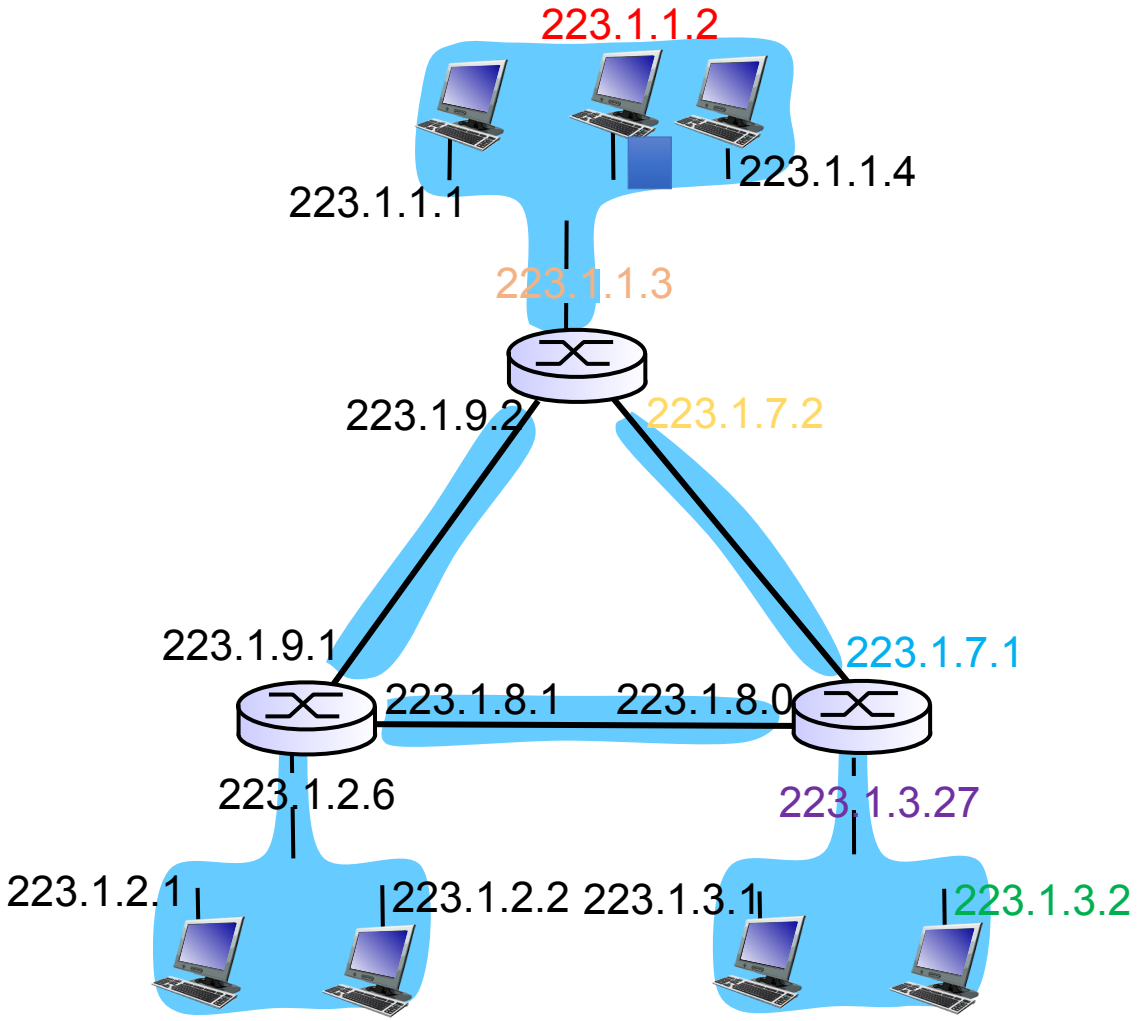
Connecting the Islands

- 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



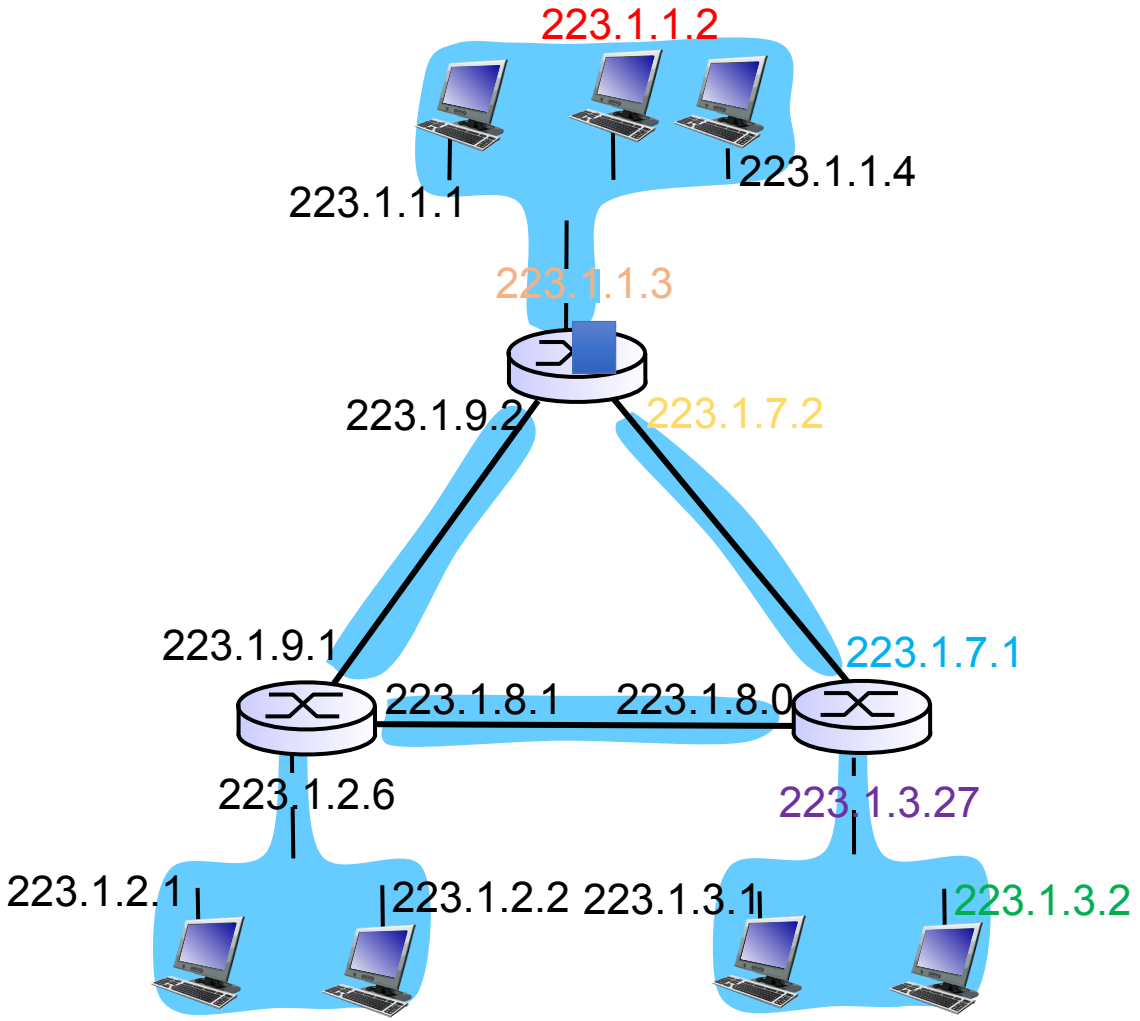
Traveling between the Islands

- For example, 223.1.1.2 can talk to 223.1.3.2:



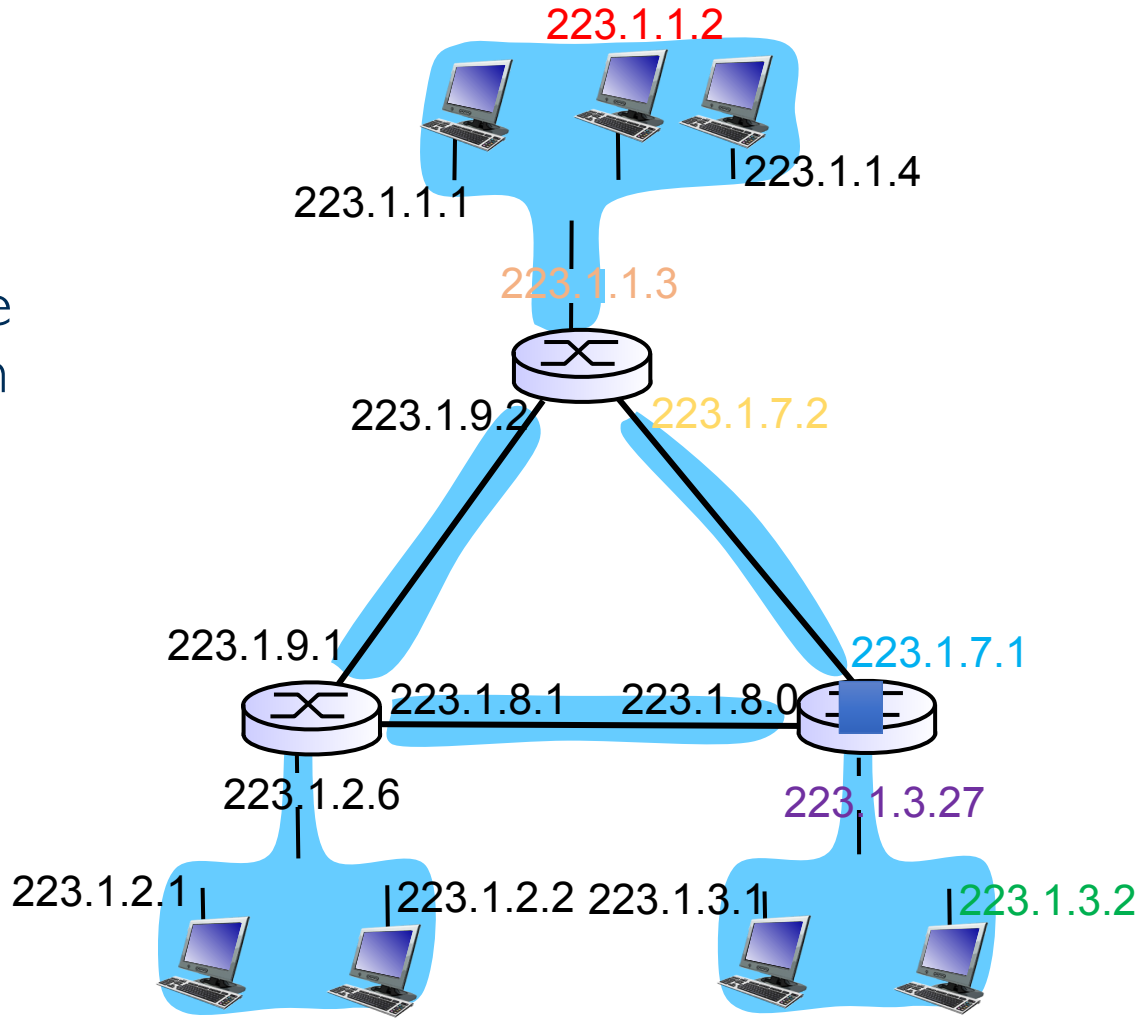
Traveling between the Islands

- 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



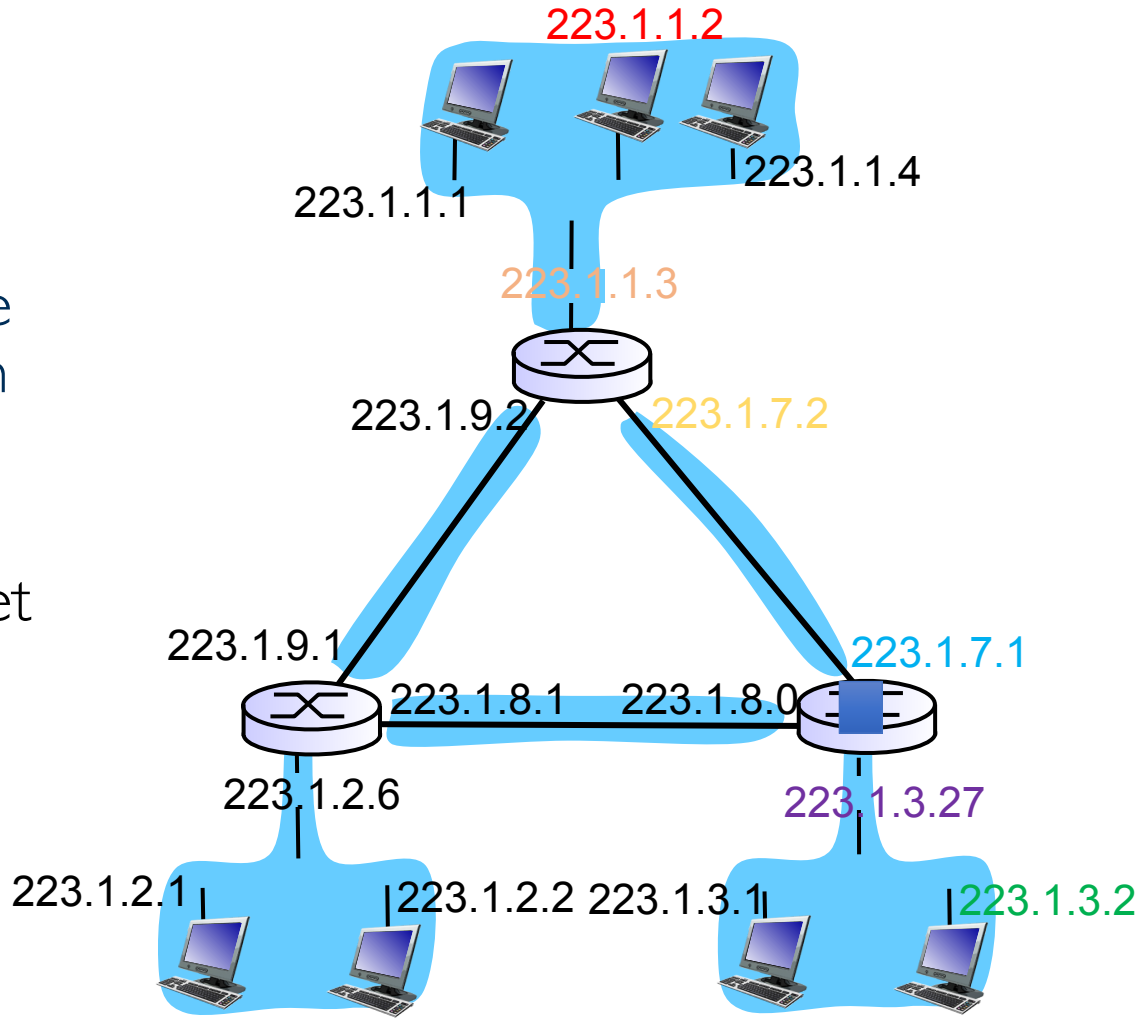
Traveling between the Islands

- 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



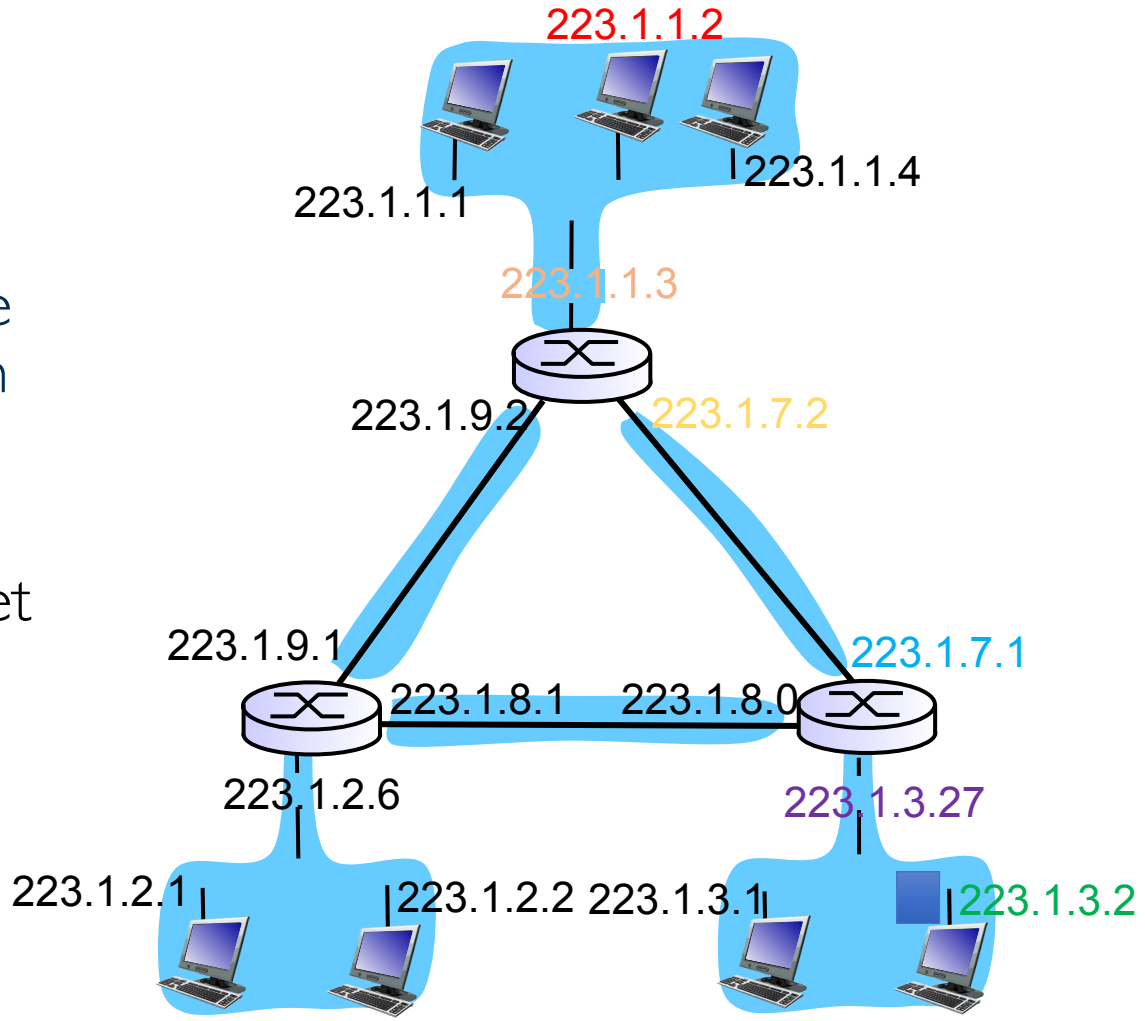
Traveling between the Islands

- 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



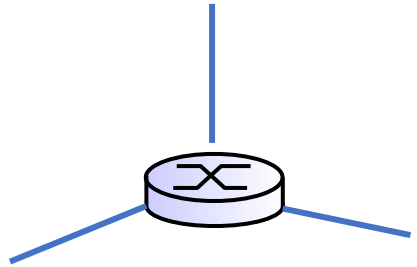
Traveling between the Islands

- 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 "3 hops"
  - 223.1.1.0/24 → 223.1.7.0/24 → 223.1.3.0/24



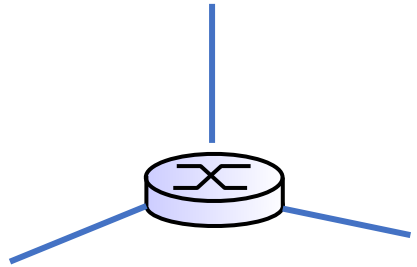
IP Routing Table

Destination	Gateway	Interface
0.0.0.0/0	128.61.5.1	a
128.61.5.0/24	128.61.5.166	b
128.61.5.21/32	128.61.5.166	b
127.0.0.1/32	127.0.0.1	c



IP Routing Table

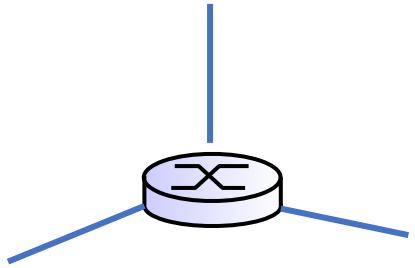
Destination	Gateway	Interface
0.0.0.0/0	128.61.5.1	a
128.61.5.0/24	128.61.5.166	b
128.61.5.21/32	128.61.5.166	b
127.0.0.1/32	127.0.0.1	c



- If the destination is on our network, send the packet to the data link layer

IP Routing Table

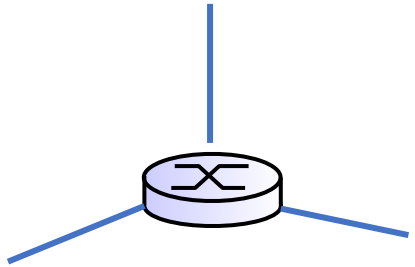
Destination	Gateway	Interface
0.0.0.0/0	128.61.5.1	a
128.61.5.0/24	128.61.5.166	b
128.61.5.21/32	128.61.5.166	b
127.0.0.1/32	127.0.0.1	c



- 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

IP Routing Table

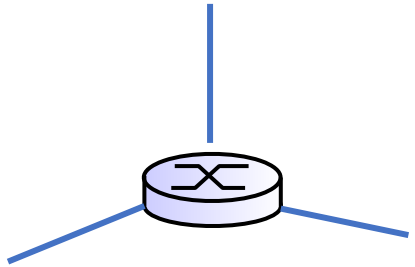
Destination	Gateway	Interface
0.0.0.0/0	128.61.5.1	a
128.61.5.0/24	128.61.5.166	b
128.61.5.21/32	128.61.5.166	b
127.0.0.1/32	127.0.0.1	c



- 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?

IP Routing Table

Destination	Gateway	Interface
0.0.0.0/0	128.61.5.1	a
128.61.5.0/24	128.61.5.166	b
128.61.5.21/32	128.61.5.166	b
127.0.0.1/32	127.0.0.1	c



- 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

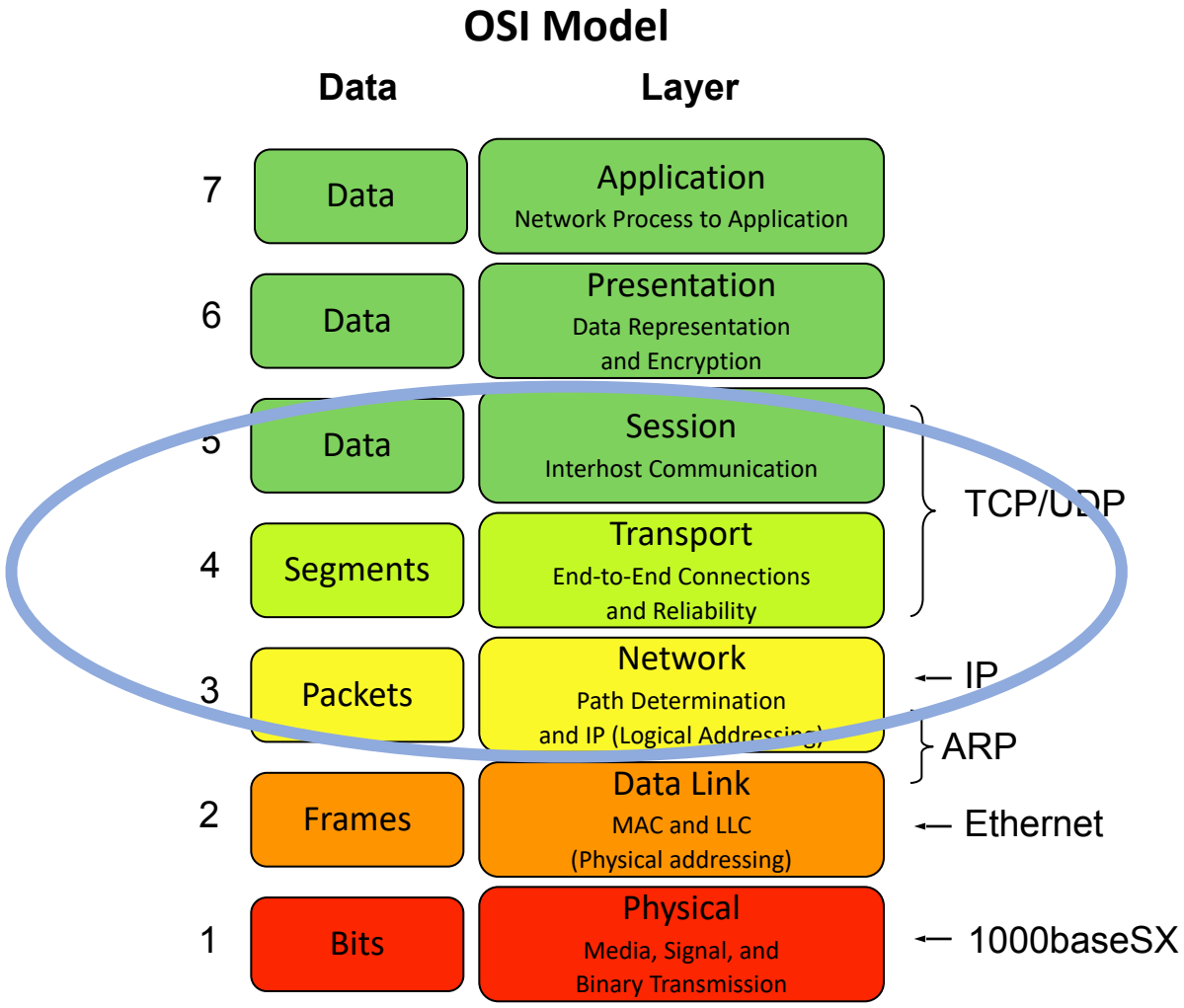
A. Pass the packet to layer 2 for delivery
B. Send the packet to the next-hop router based on the IP routing table
C. Return an Acknowledgement to the source address

Questions: IP Routing rules

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

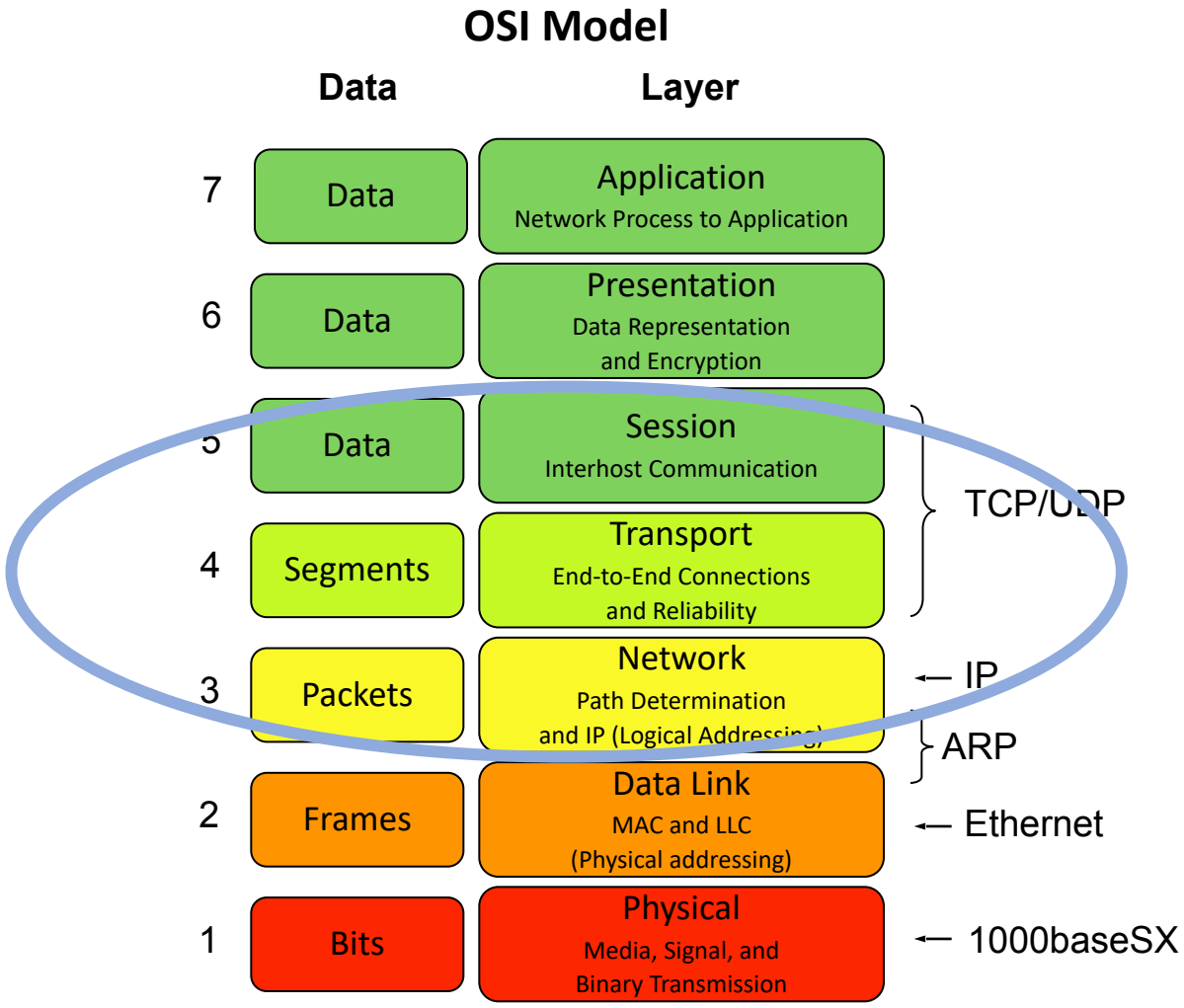
A. Pass the packet to layer 2 for delivery
B. Send the packet to the next-hop router by choosing the longest match in the IP routing table
C. Send the packet to the next-hop router by choosing the shortest match in the IP routing table
D. Return an ICMP "No route to host" message

Now let's include Layers 3-5 in IP



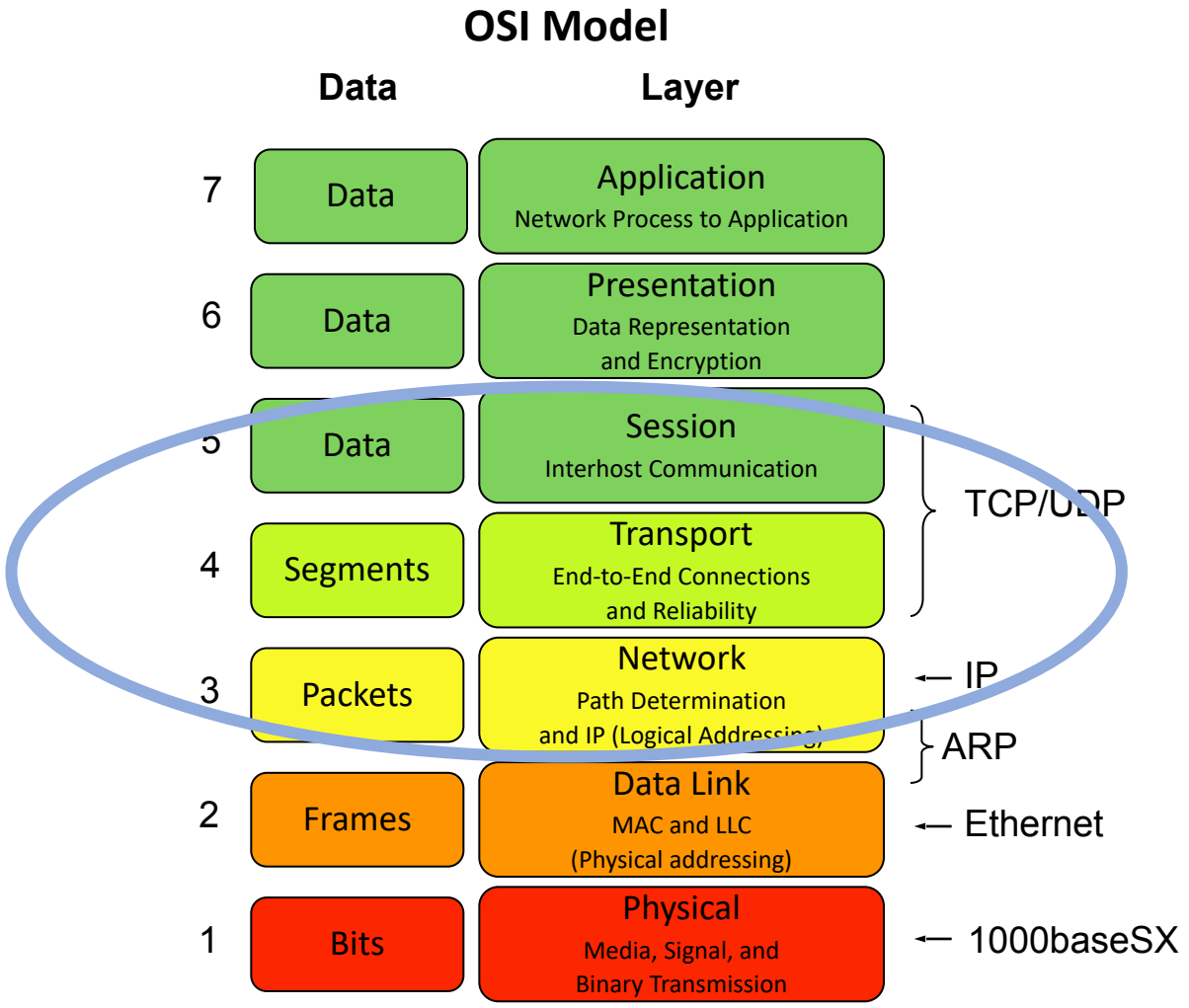
Now let's include Layers 3-5 in IP

- TCP and UDP are the two most common IP Transport protocols



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

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

Ports and connection addressing

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

Ports and connection addressing

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
- With TCP and UDP protocols, always consider the addressing as a **quadruple**: *(source IP, source port, destination IP, destination port)*
  - (Note that the fields are not actually ordered this way in the packet)

Ports and connection addressing

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
- With TCP and UDP protocols, always consider the addressing as a **quadruple**: *(source IP, source port, destination IP, destination port)*
  - (Note that the fields are not actually ordered this way in the packet)
- Let's look at where that ordered quad shows up in the segment...

TCP (and UDP) port addressing

IPv4 Header Format

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																																

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 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)														
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

TCP (and UDP) port addressing

IPv4 Header Format

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																																

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 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)														
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

TCP (and UDP) port addressing

IPv4 Header Format

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																																

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 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)														
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

TCP (and UDP) port addressing

IPv4 Header Format

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																																

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 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)														
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

TCP (and UDP) port addressing

IPv4 Header Format

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																																

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 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)														
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

TCP (and UDP) port addressing

IPv4 Header Format

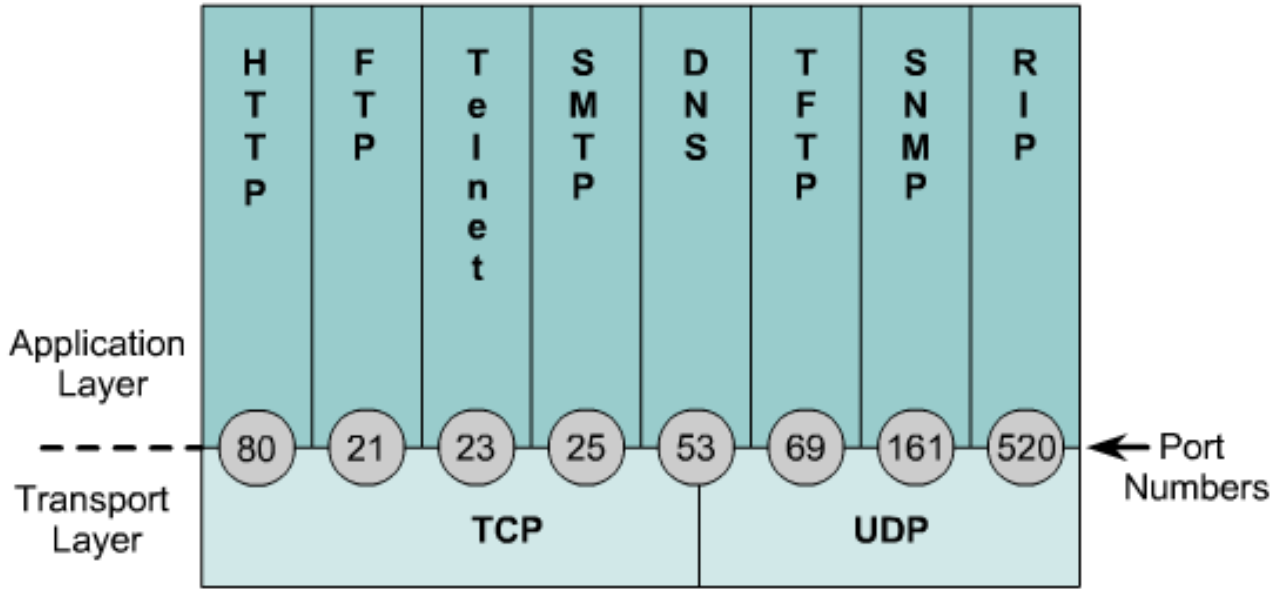
Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								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																																

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 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)														
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

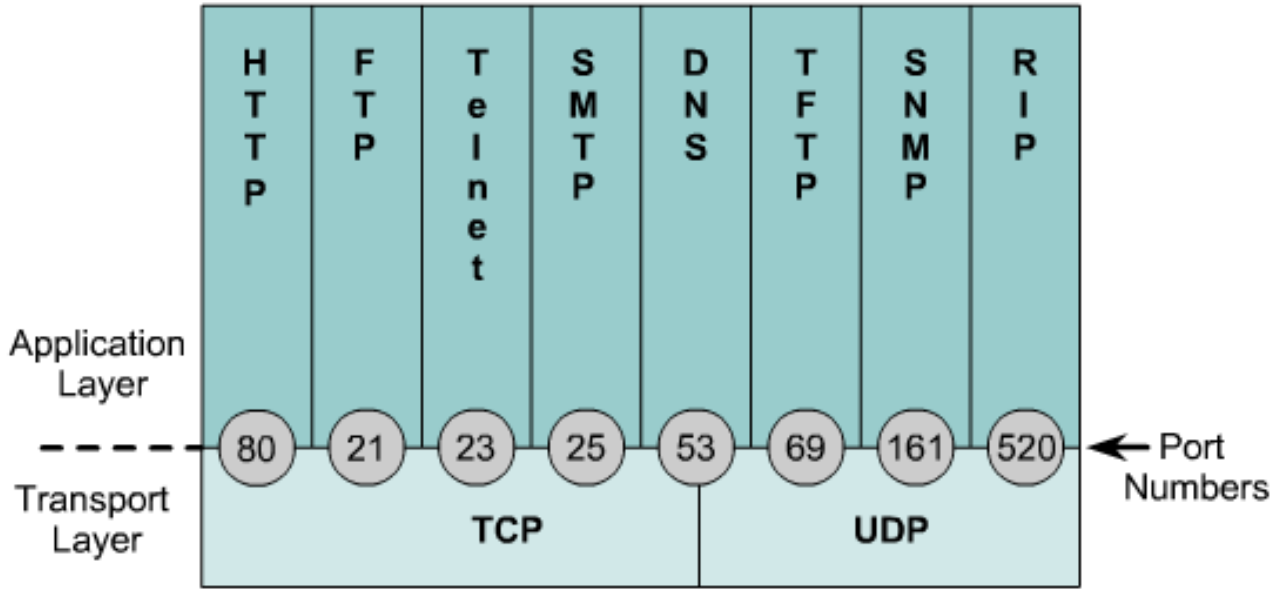
TCP and UDP port numbers

- Port numbers are used to keep track of different conversations.



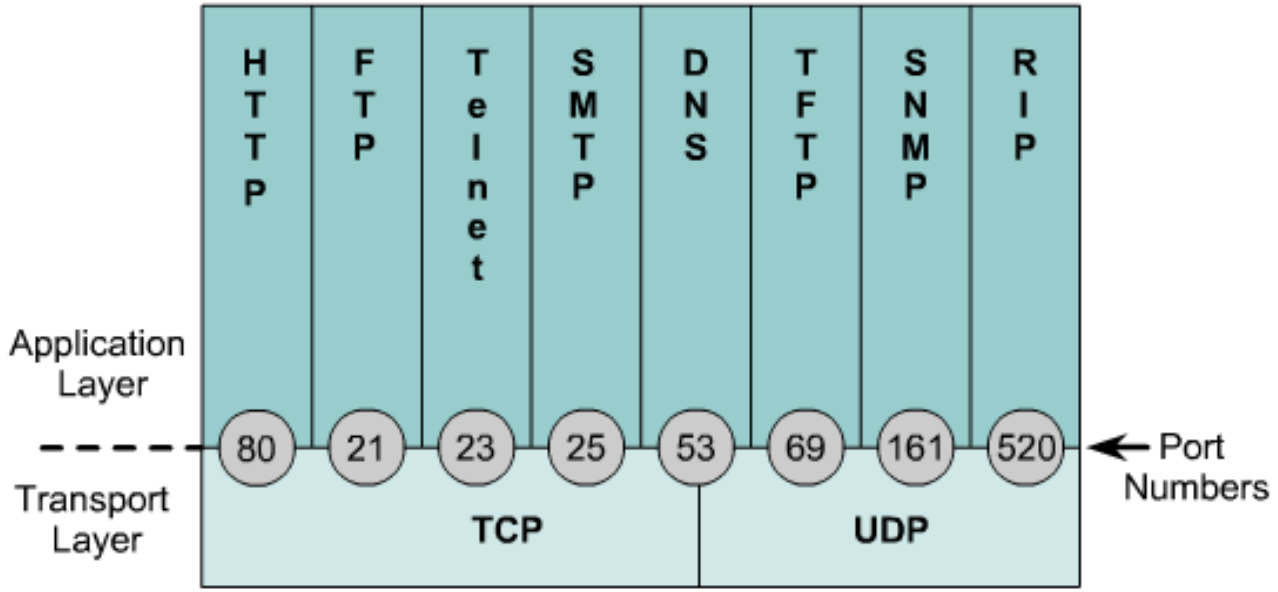
TCP and UDP port numbers

- Port numbers are used to keep track of different conversations.
- Well-known ports numbers are traditionally from 1-1023.



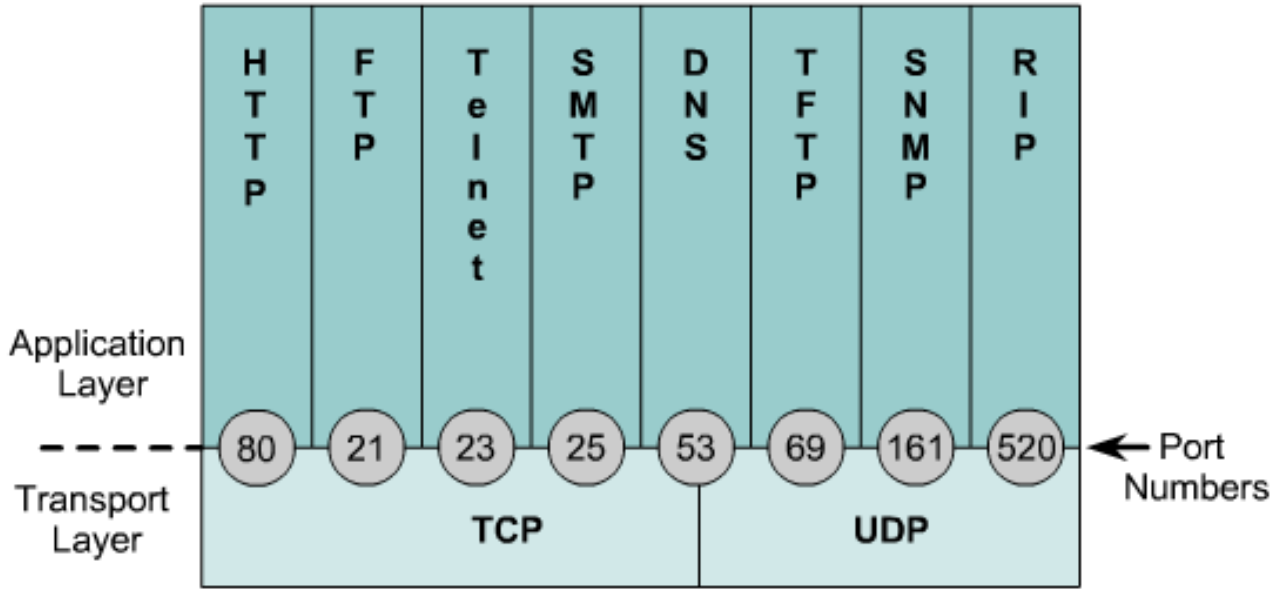
TCP and UDP port numbers

- 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.



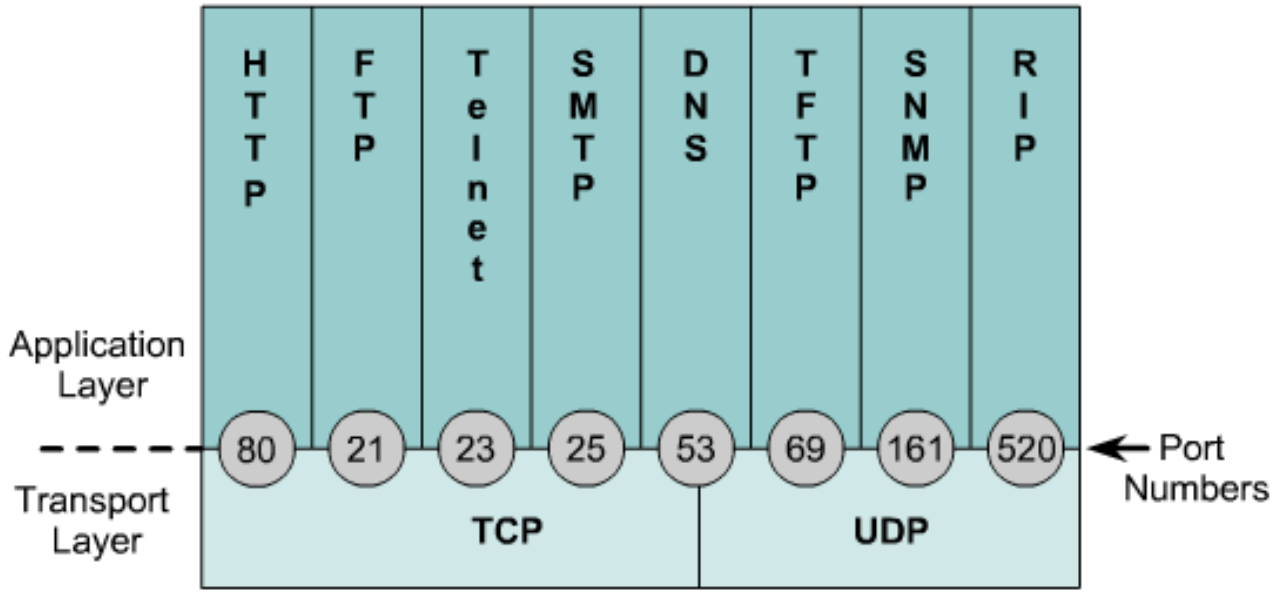
TCP and UDP port numbers

- 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.



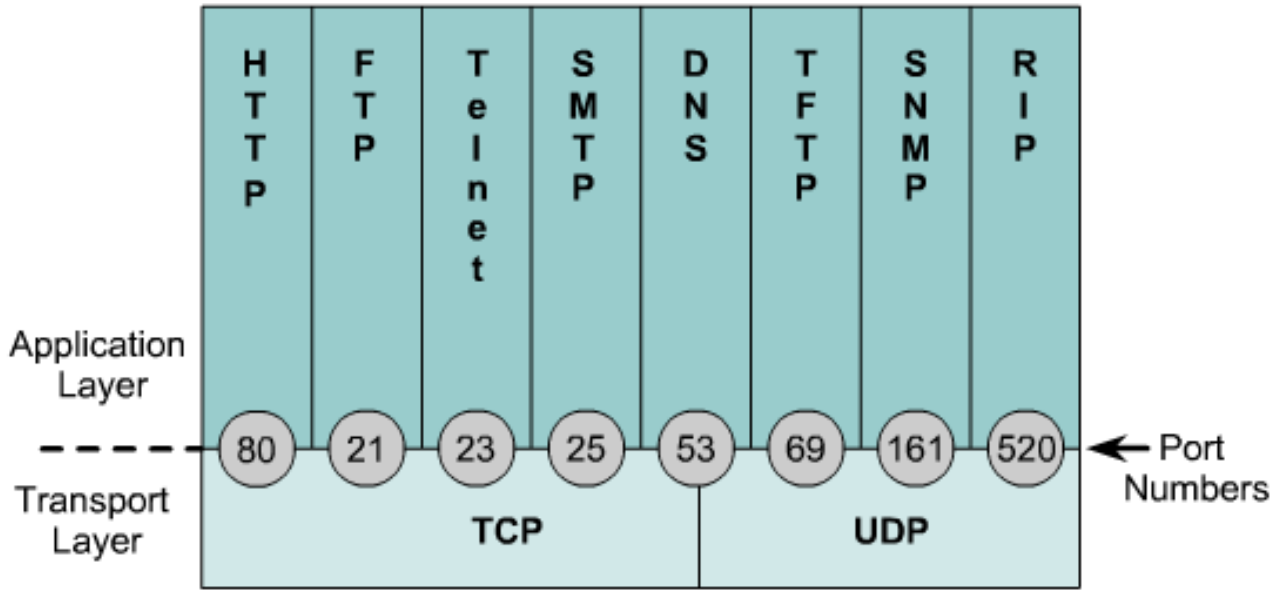
TCP and UDP port numbers

- Destination ports are chosen based on the desired service



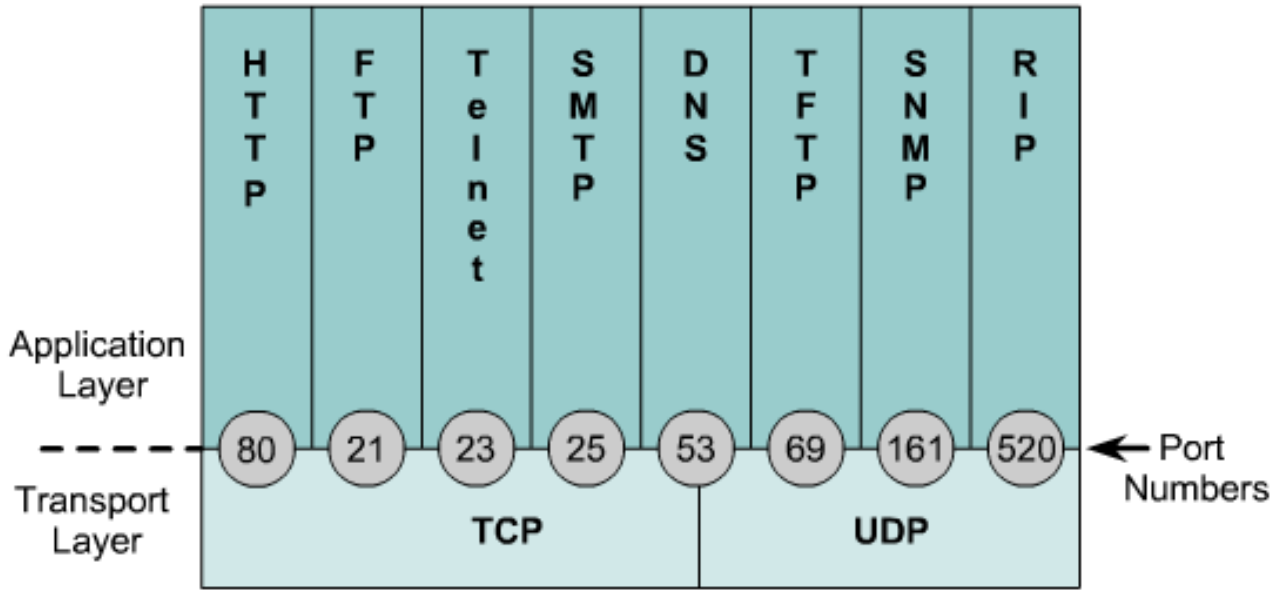
TCP and UDP port numbers

- 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



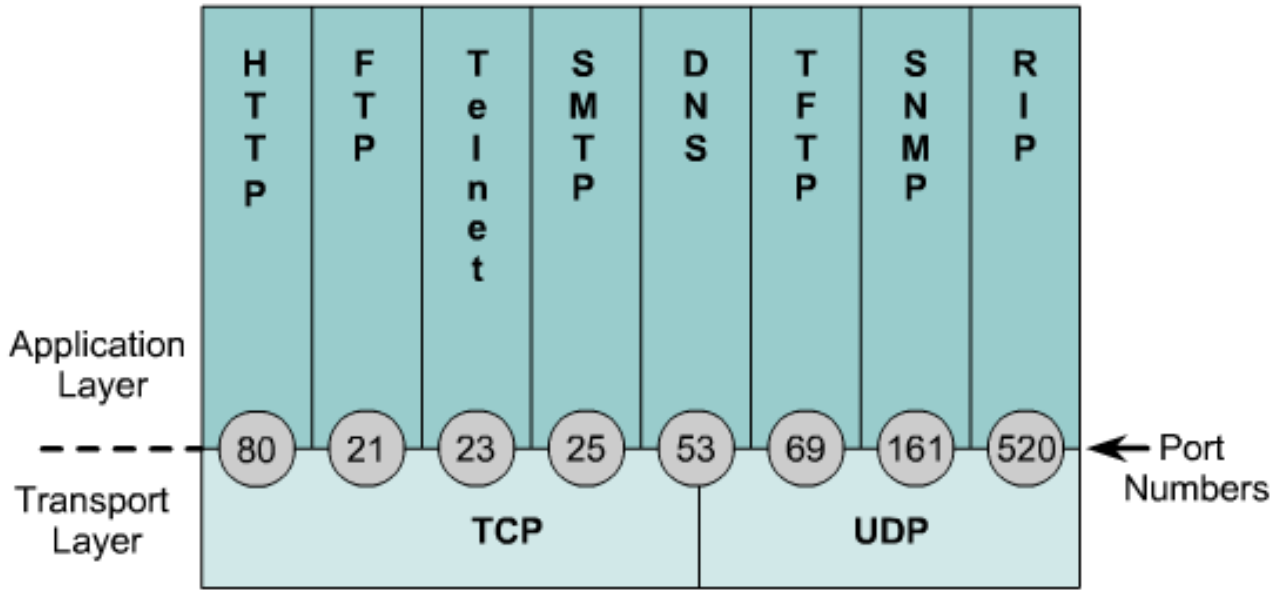
TCP and UDP port numbers

- 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 on the host



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 on the host
- That makes the ordered quad unique among open connections
  - This is how we keep connections separate even when the IPs are the same



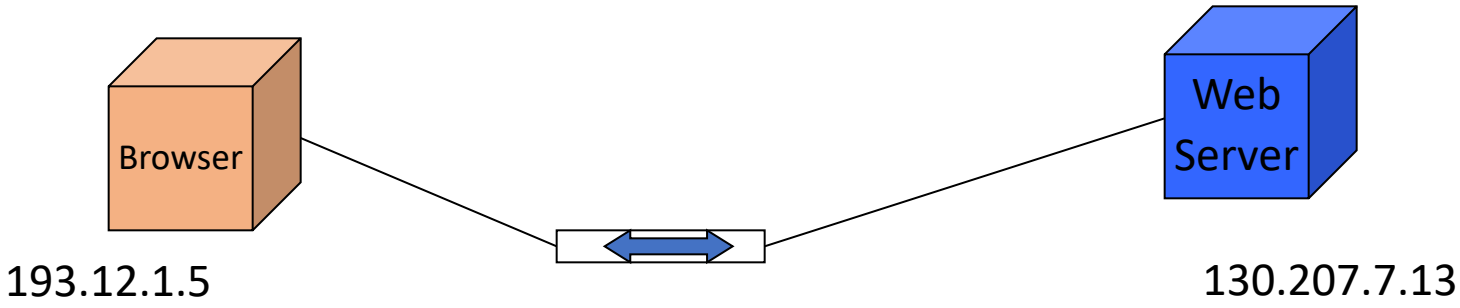
A browser opens two TCP connections to a web server

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?

A. IP does this automatically
B. The conversations use independent sequence numbers to disambiguate them
C. The conversations include a special session-identifier that disambiguates them
D. The browser lets the TCP stack choose a unique source port number for each connection

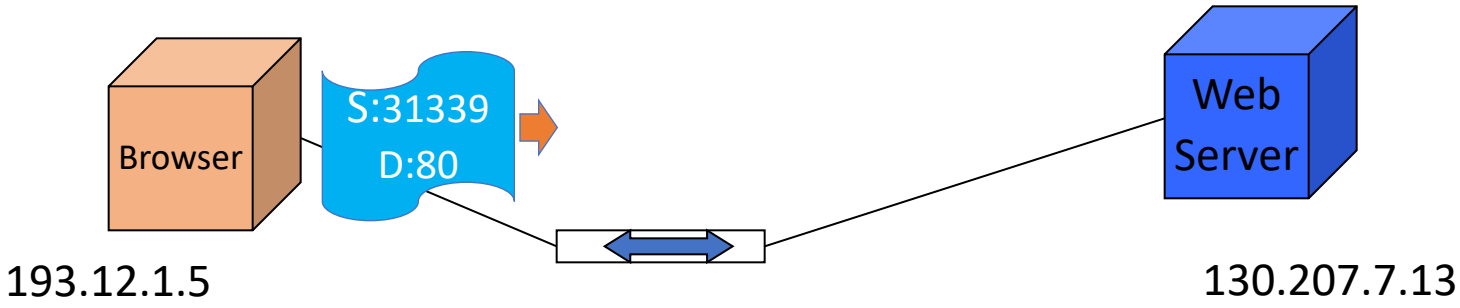
Conversations

- 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.



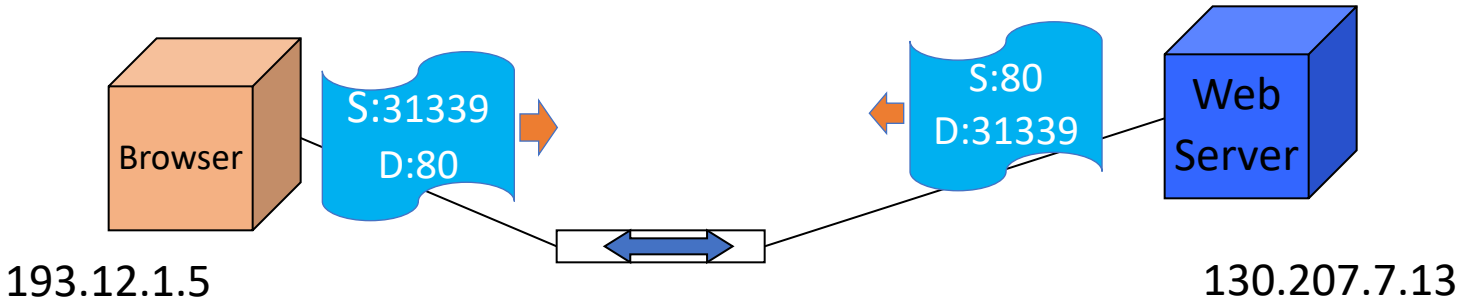
Conversations

- 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.



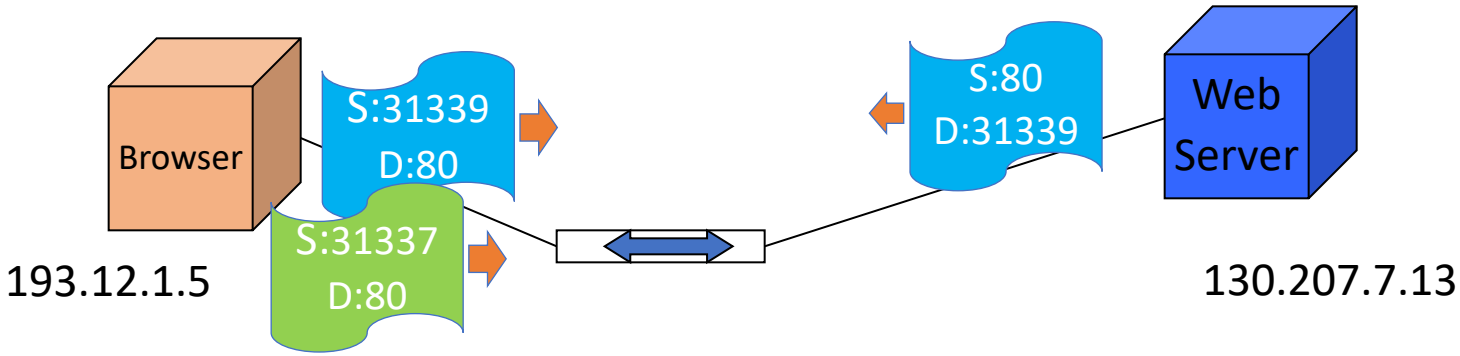
Conversations

- 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.



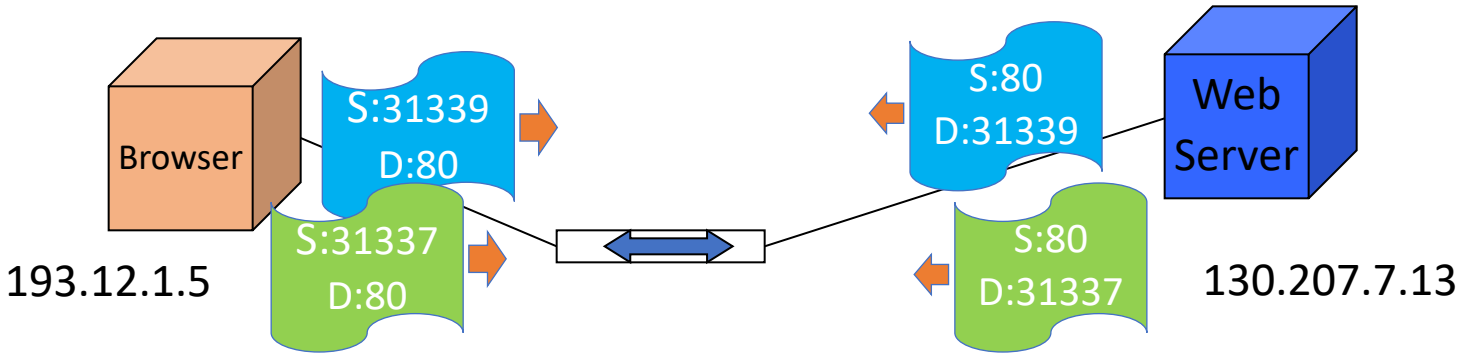
Conversations

- 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.



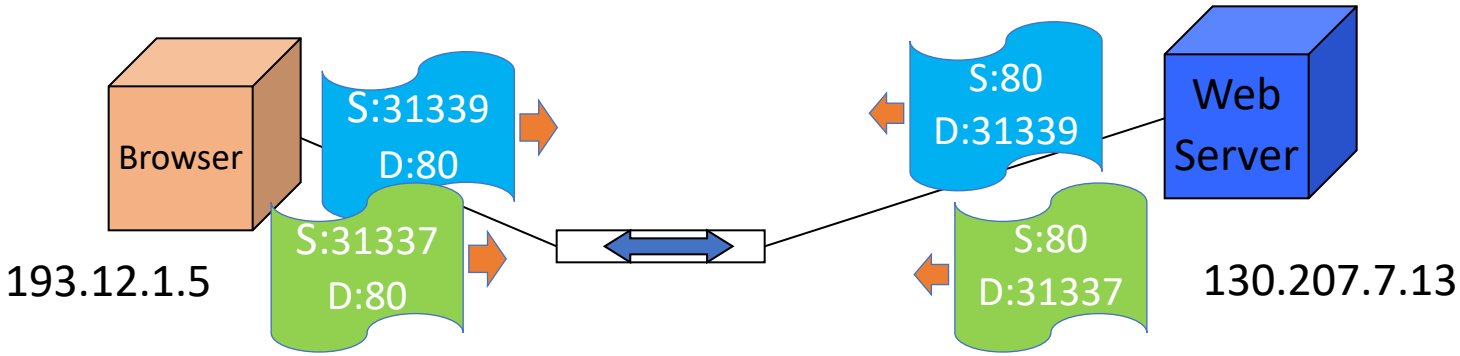
Conversations

- 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.



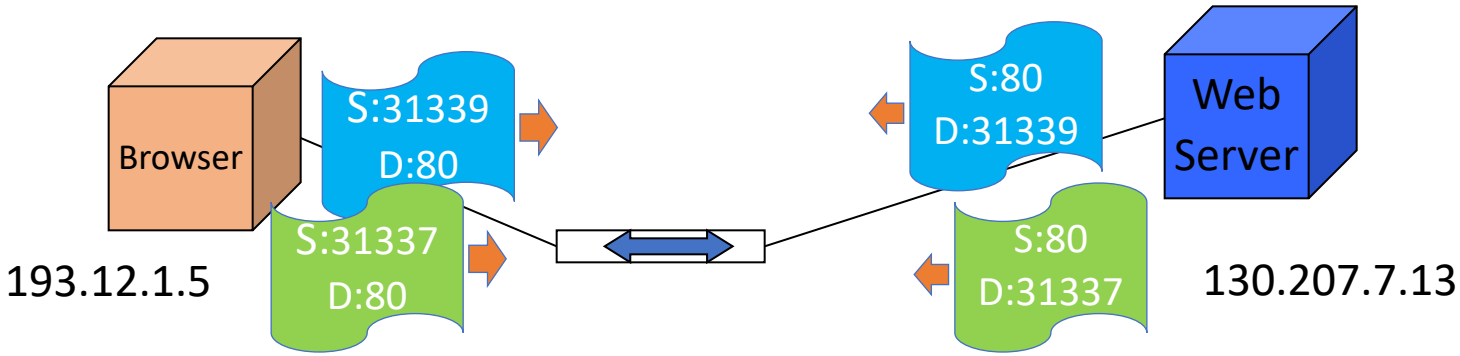
Conversations

- 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



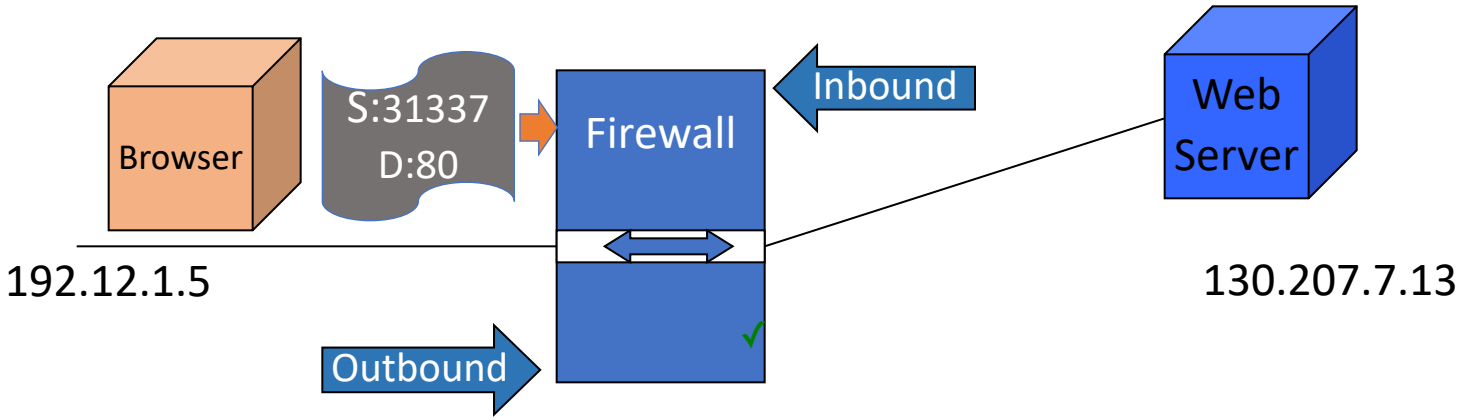
Conversations

- 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

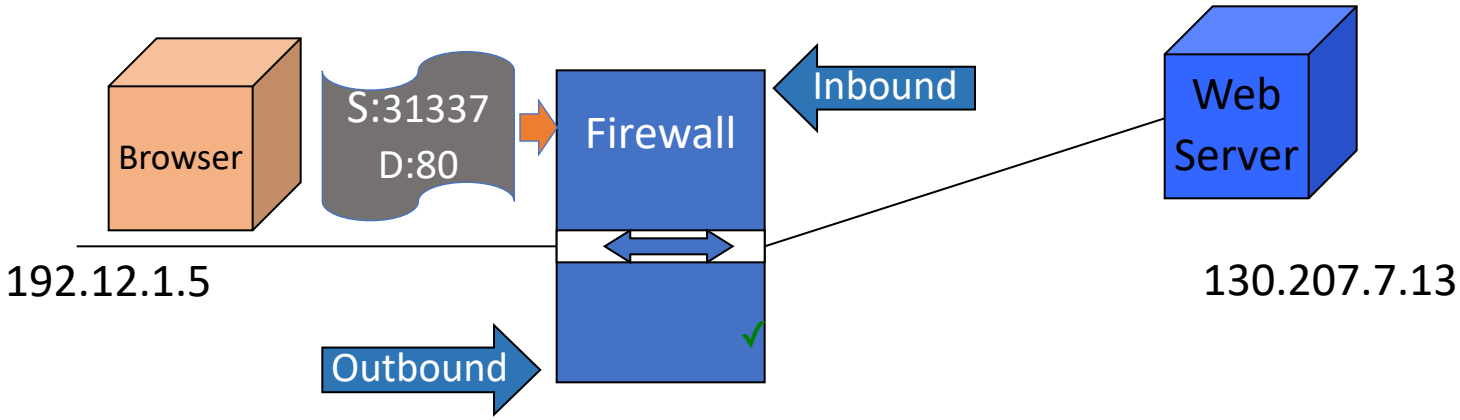
- Follow a predefined policy on which ordered quads are allowed from which direction
  - Example: Allow no inbound packets; Allow all outbound packets



Stateful Firewalls

- Follow a predefined policy on which ordered quads are allowed from which direction
  - Example: Allow no inbound packets; Allow all outbound packets

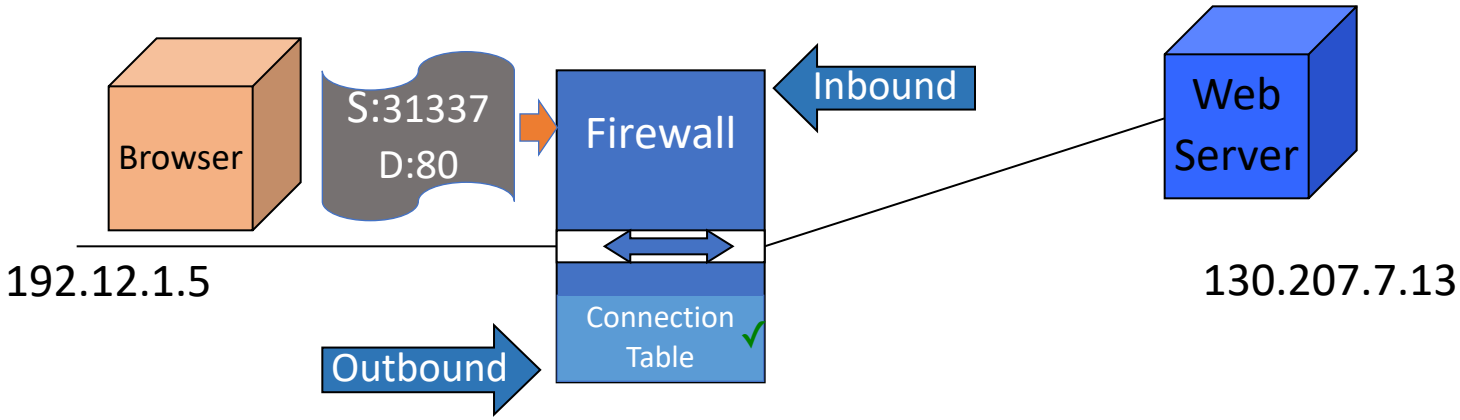
Makes sense if we don't offer any services



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)

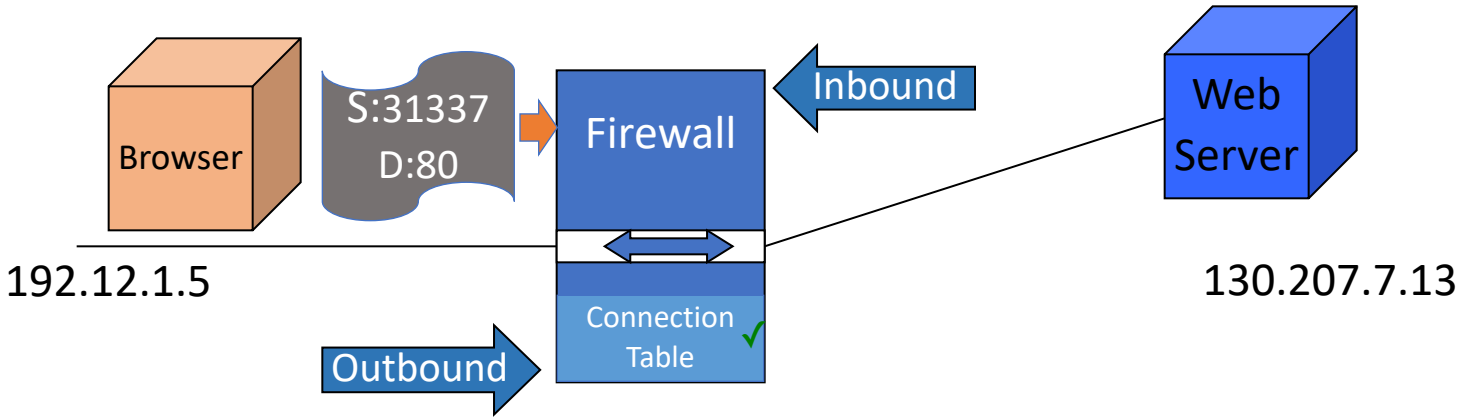
Makes sense if we don't offer any services



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

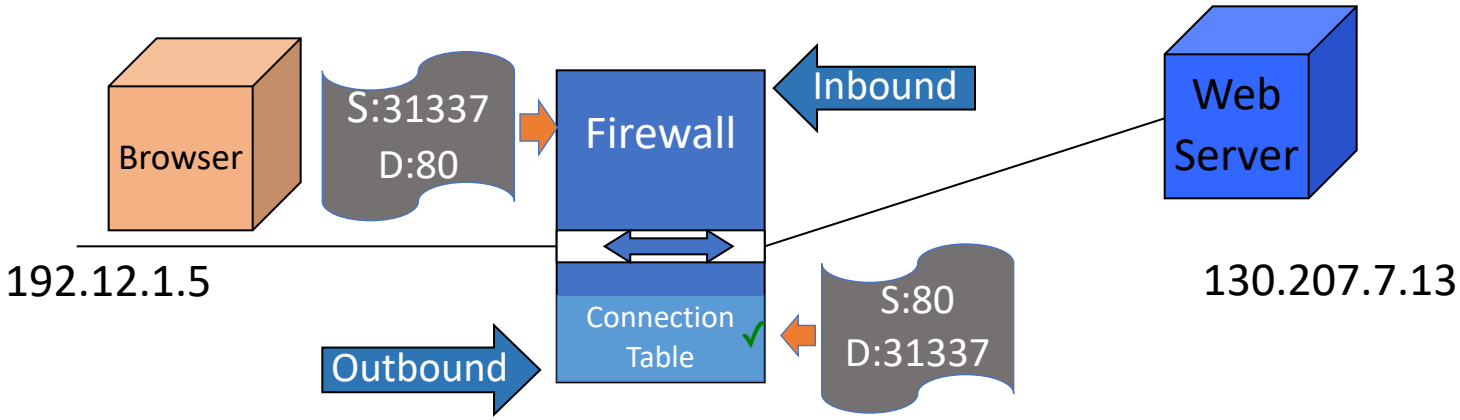
Makes sense if we don't offer any services



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

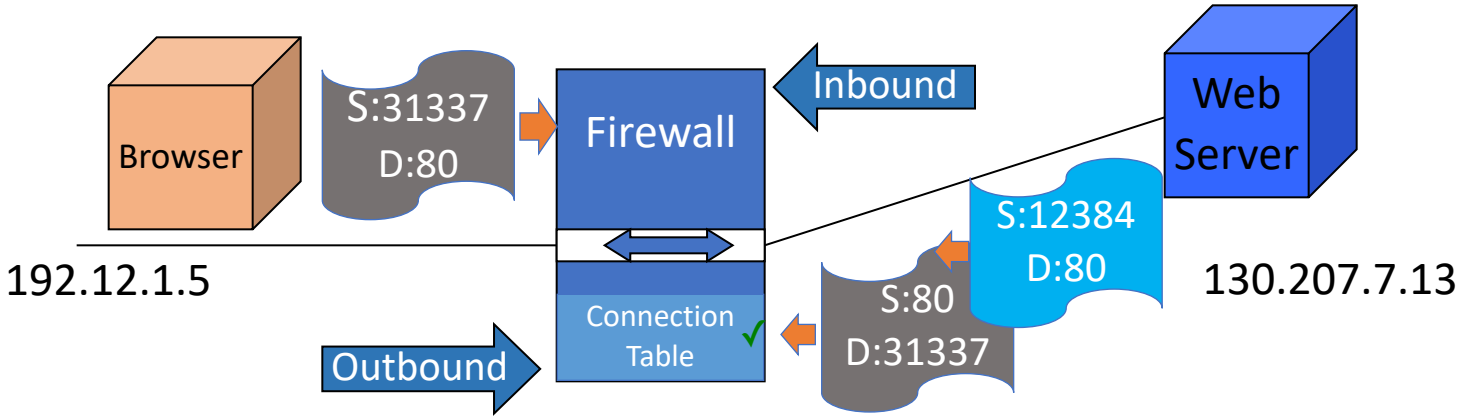
Makes sense if we don't offer any services



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

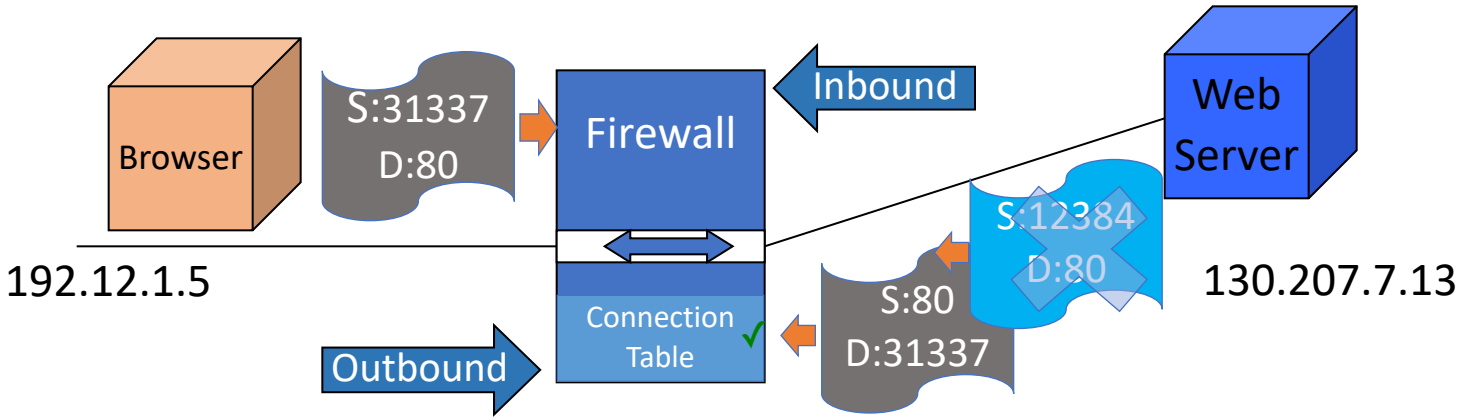
Makes sense if we don't offer any services



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

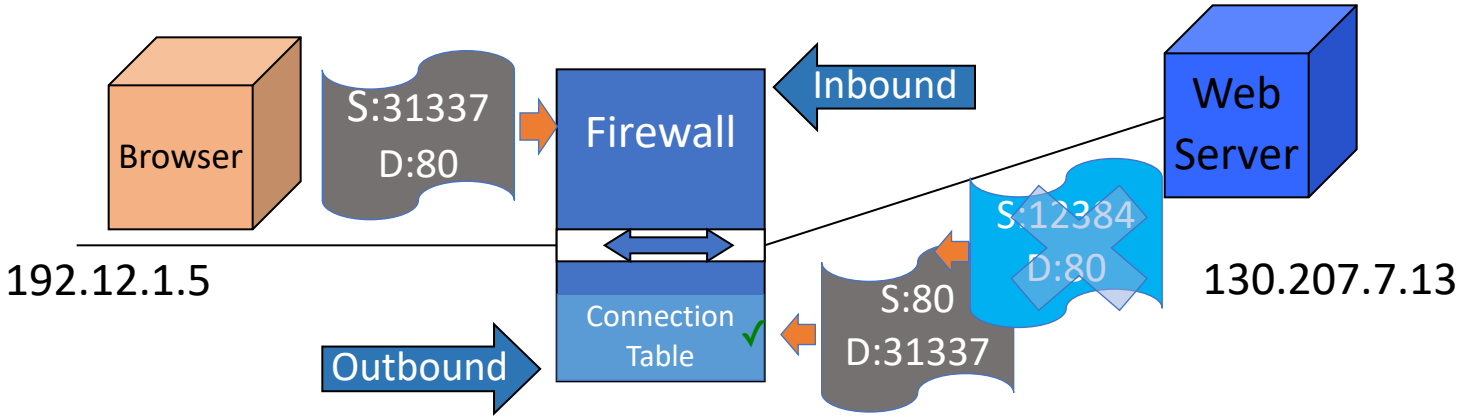
Makes sense if we don't offer any services



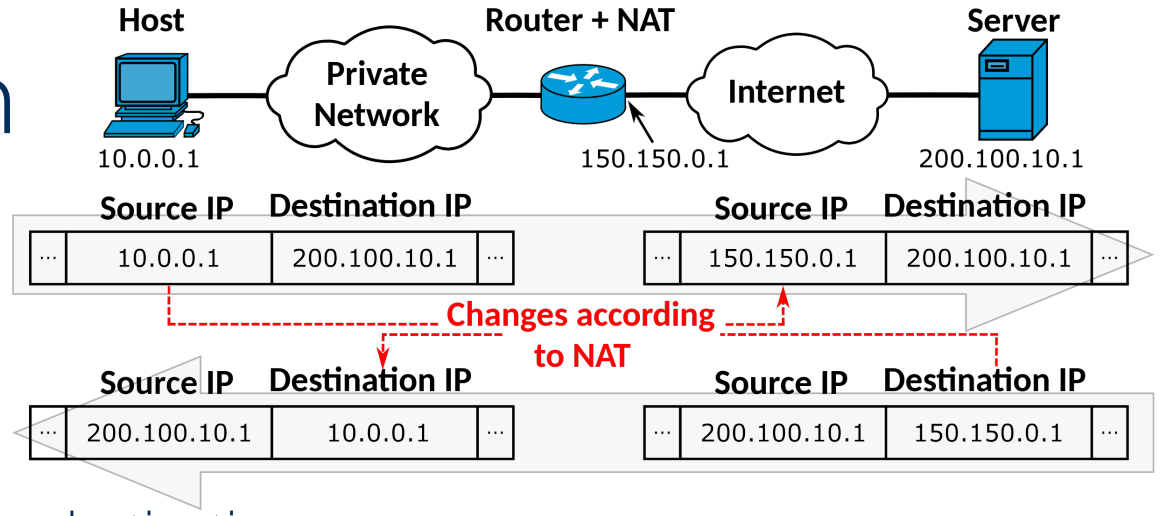
Stateful Firewalls

- 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

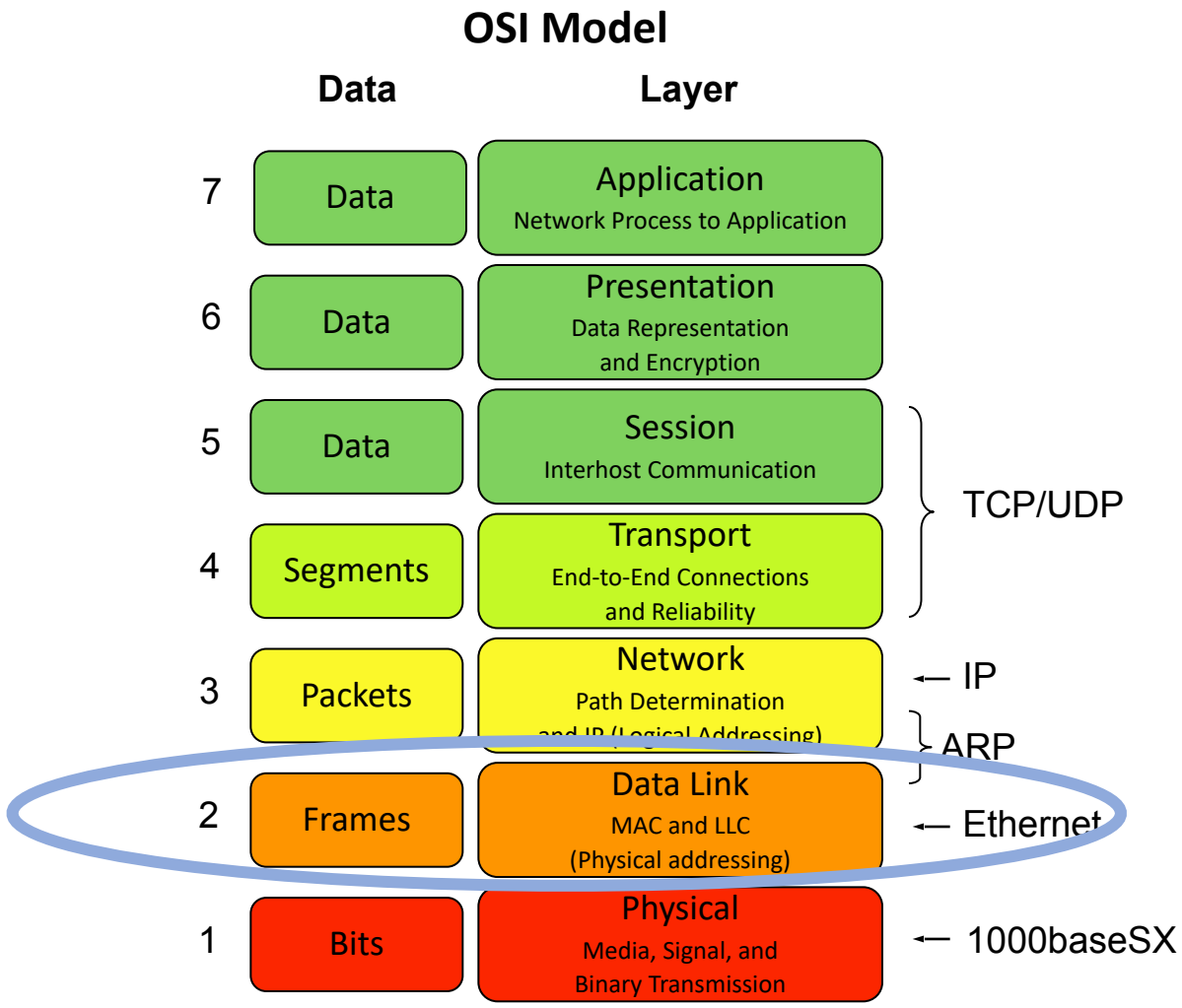


Network Address Translation



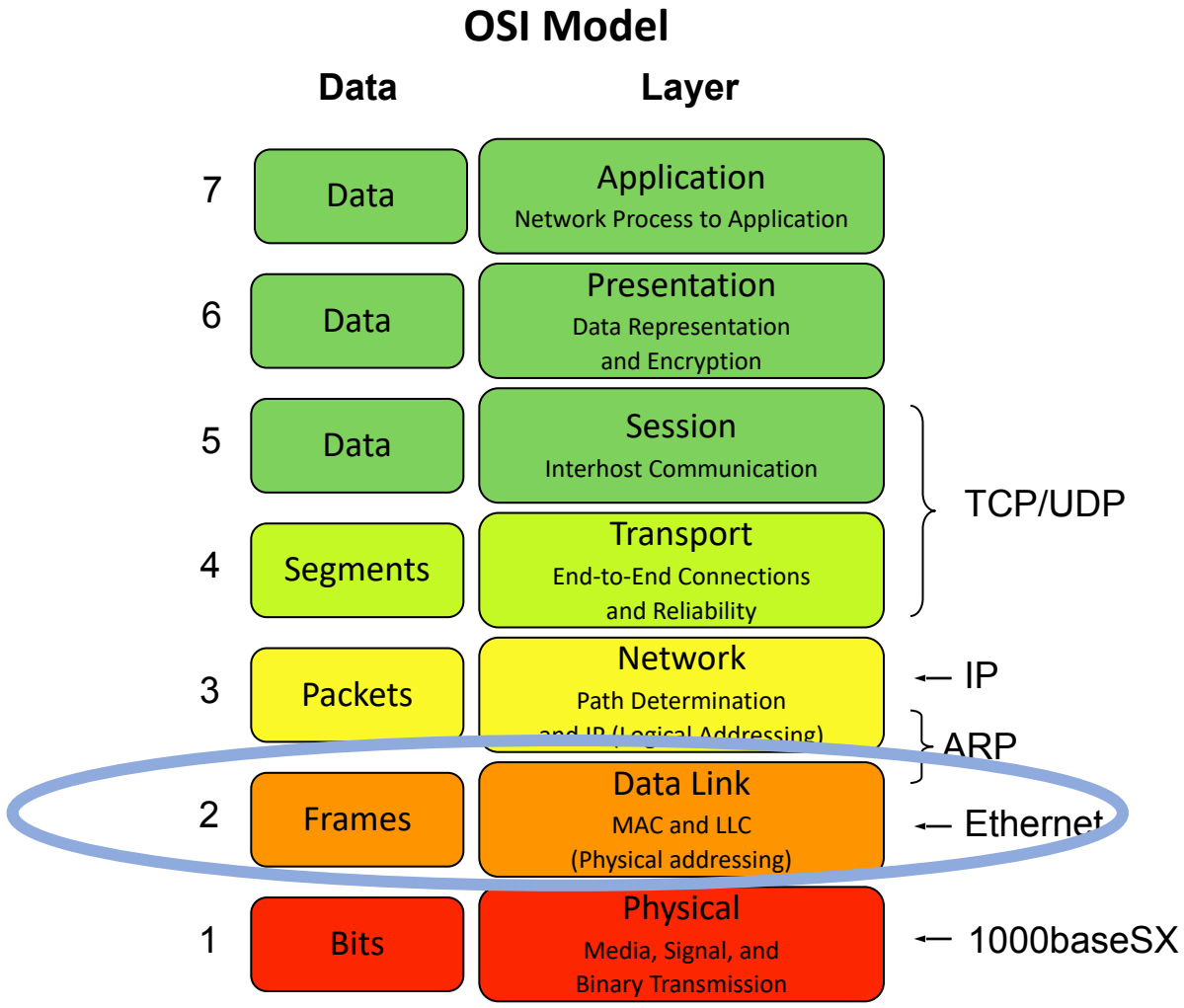
- Note there is confusing terminology
- Types
  - Static one-to-one NAT
    - A mapping is configured so that source address or destination address is changed from/to the mapped address on transit; checksums are recalculated
  - Dynamic one-to-one NAT
    - Same as Static, but a public address from a 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 NAPT (network address and port 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
- NAPT 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

Now let's look at the data link layer



Now let's look at the data link layer

- Ethernet is a very common data link network



Let's talk about Ethernet

- Ethernet is a particular type of data link network
- It was designed for relatively small areas, like a building, hence the term Local Area Network (LAN)
- It has its very own standard (IEEE 802.3)
- It's evolved a great deal since the 1980s

Terminology

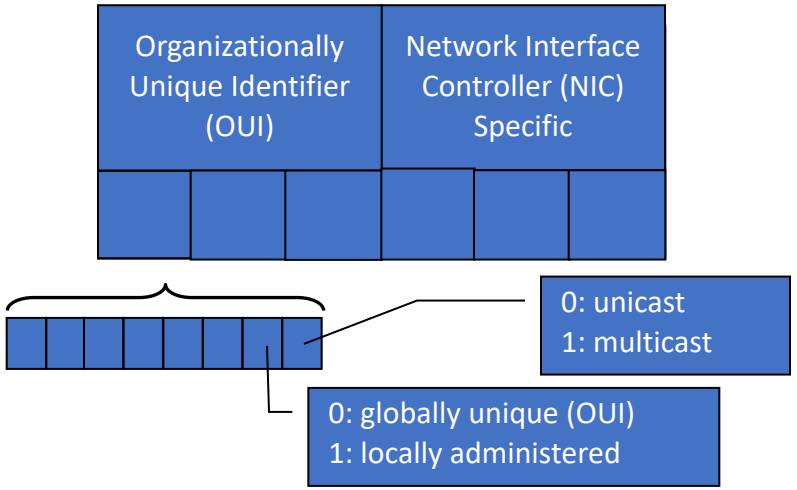
- Collision domain - All the NICs that share a physical ethernet; only one can transmit at a time
- Broadcast domain - All the NICs that can hear an ethernet broadcast frame

---

- Repeater/hub - layer 1 replication/amplification of bits (both sides stay in one collision domain)
- Bridge/switch - layer 2 re-transmission of ethernet frames (both sides stay in one broadcast domain)
- Router - layer 3 forwarding of packets based on a layer-3 (IP) routing table
- Host – a computer with one or more NICs; if it forwards traffic using multiple NICs, we're going to call it a router, not a host.

Ethernet Addressing

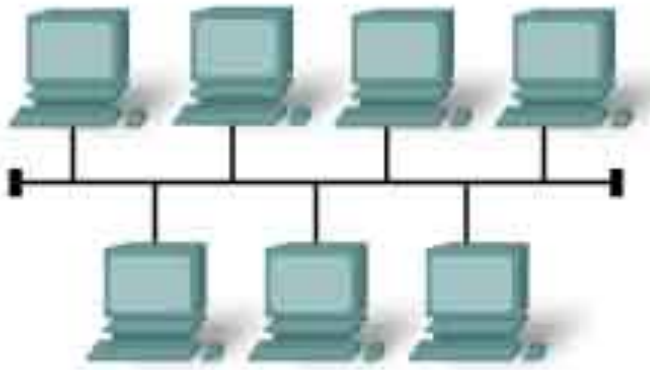
Every ethernet device gets a unique burned-in 48-bit MAC address



- 6 bytes/48 bits
  - 3 bytes OUI
  - 3 bytes NIC specific
- Examples
  - Apple: 00:25:00:f4:6d:c2
  - Oracle: 00-03-ba-a1-02-df
- To identify the OUI
  - https://www.wireshark.org/tools/oui-lookup.html

Ethernet Frames

Preamble	Destination MAC	Source MAC	802.1Q tag	Ether type/ Length	Payload	CRC	Interframe Gap
8	6	6	0-4	2	46-1500	4	12

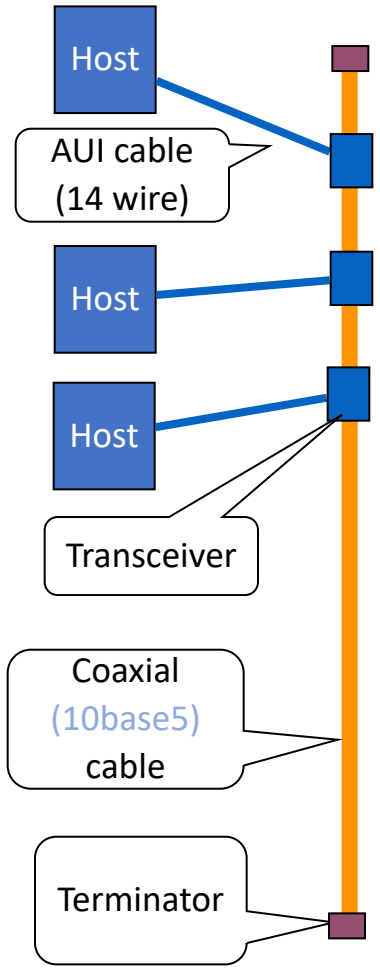


- To send a frame
  - Wait until no one using the cable (carrier sense)
  - Assert carrier signal on the cable
  - Broadcast the frame to all stations on the cable
  - If you detect a collision while transmitting, stop and requeue the frame for retransmission
- If you hear a frame on the cable
  - Accept it if the destination MAC belongs to us (or if it has a broadcast or multicast destination MAC)
  - Drop it if you see a collision during its receipt or its CRC doesn't match

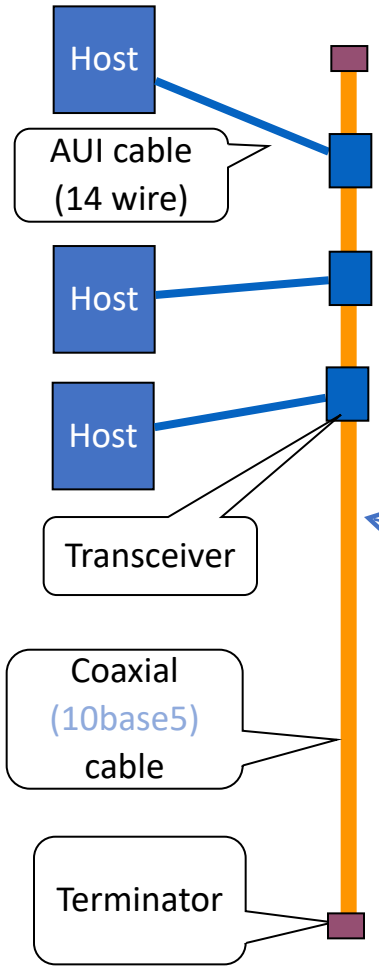
Something is Missing…

- Only one station can send at a time?
- Where are the IP addresses?
- We'll get back to this…

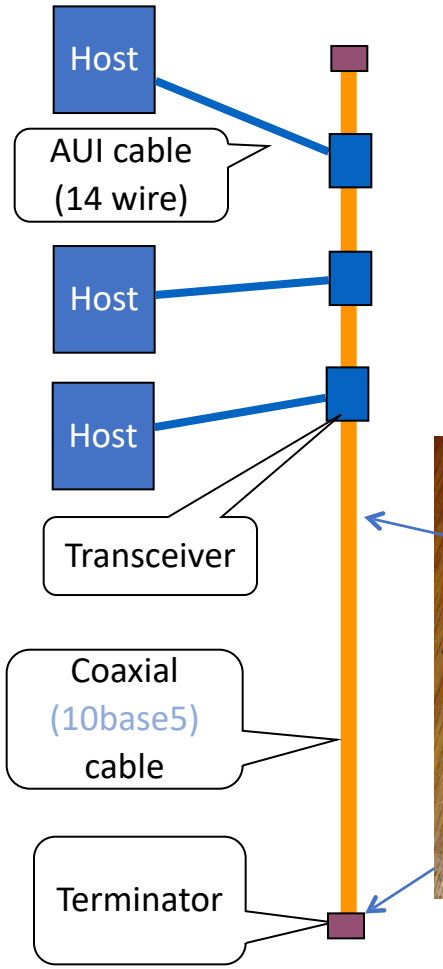
Ethernet Topology (some history)

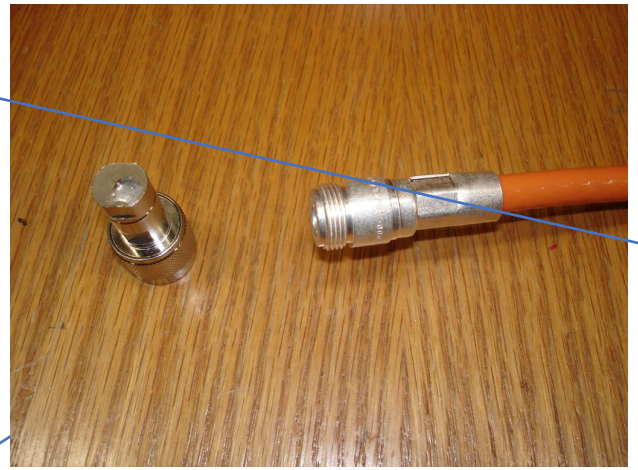


Ethernet Topology (some history)

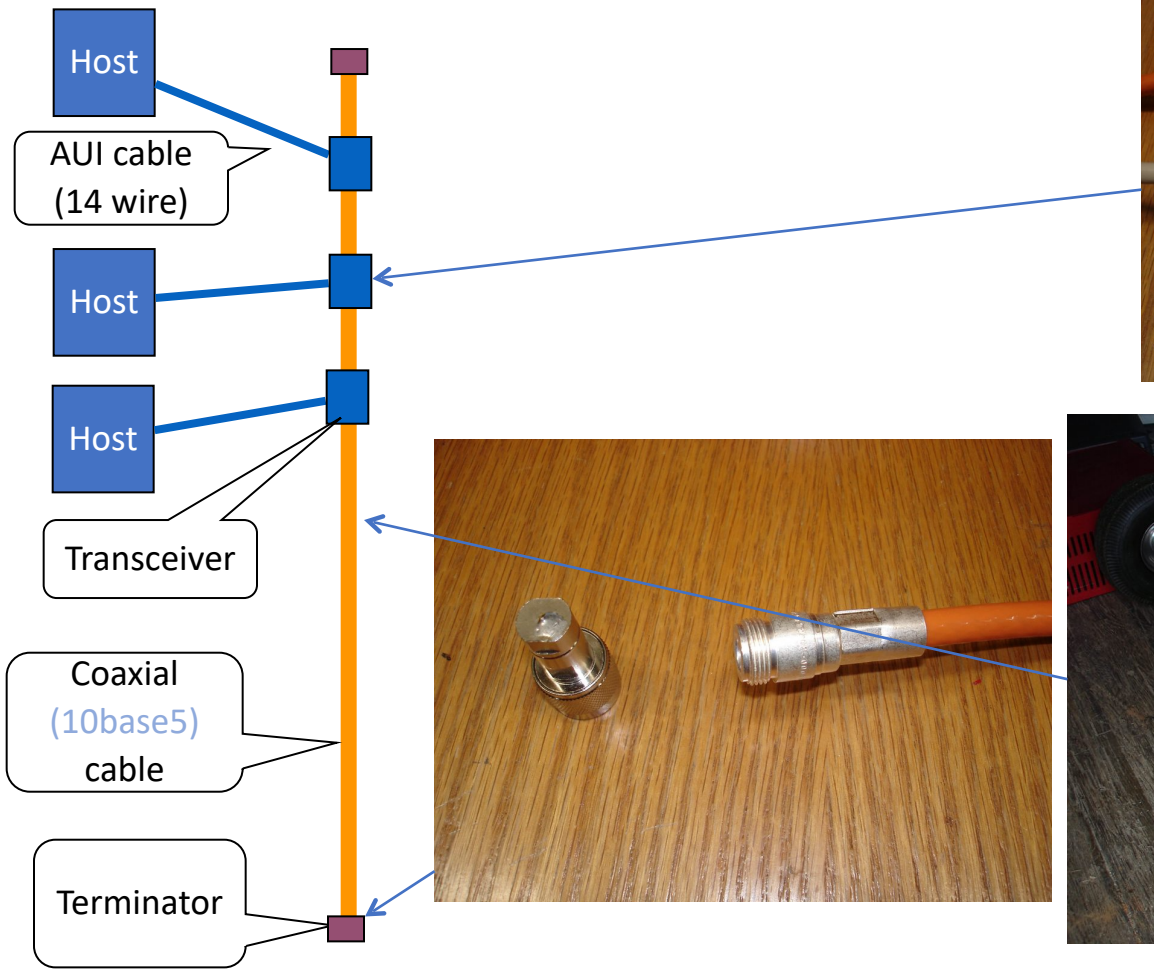


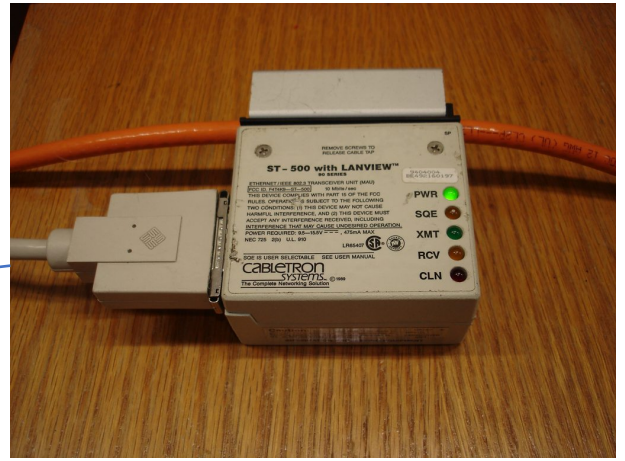
Ethernet Topology (some history)



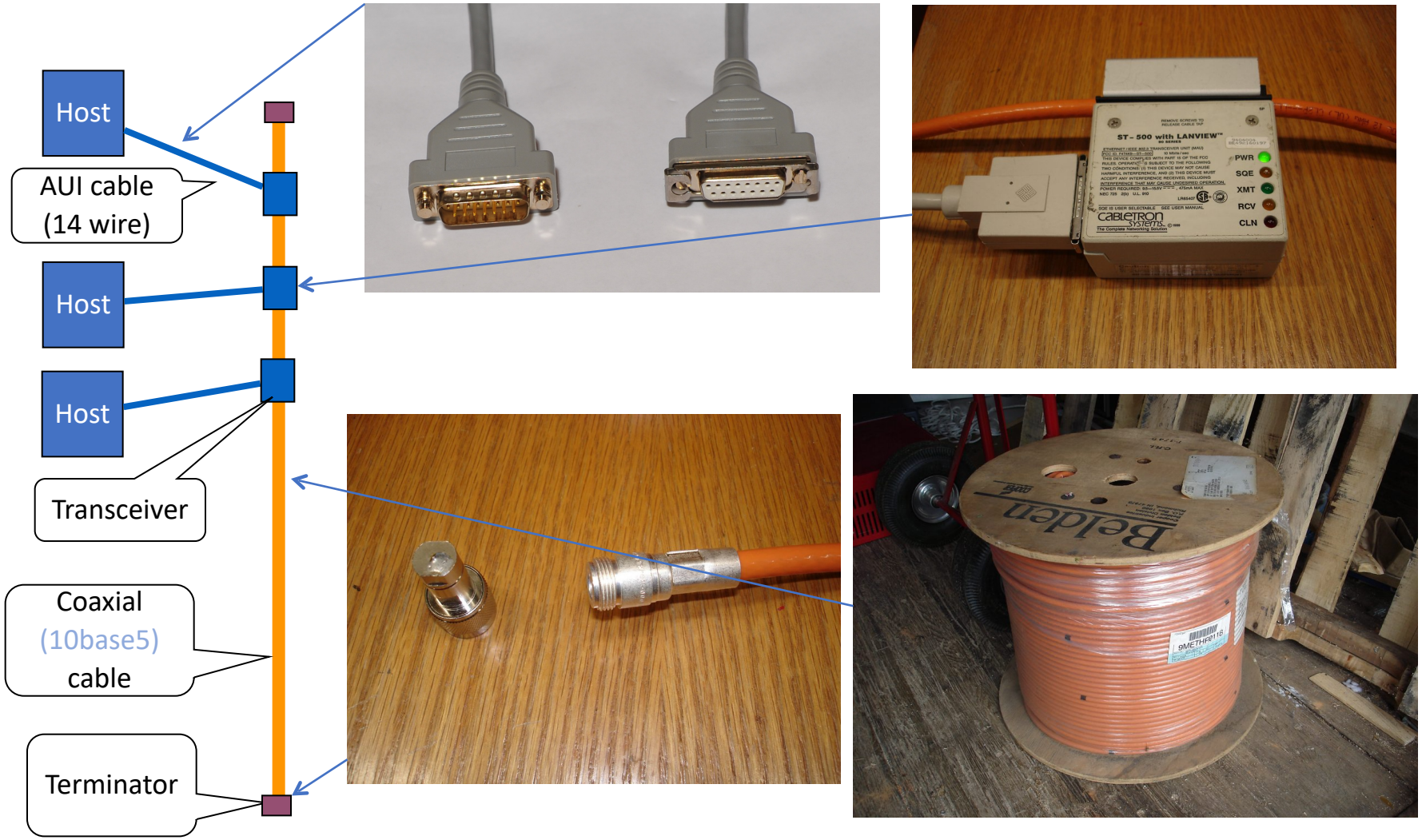


Ethernet Topology (some history)



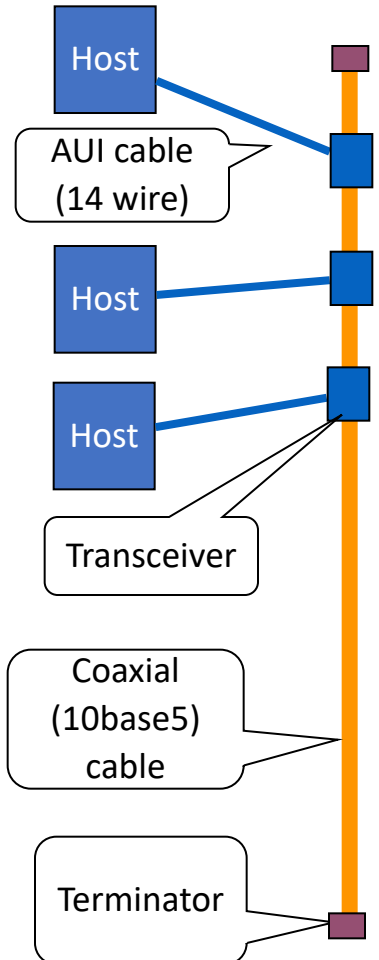


Ethernet Topology (some history)

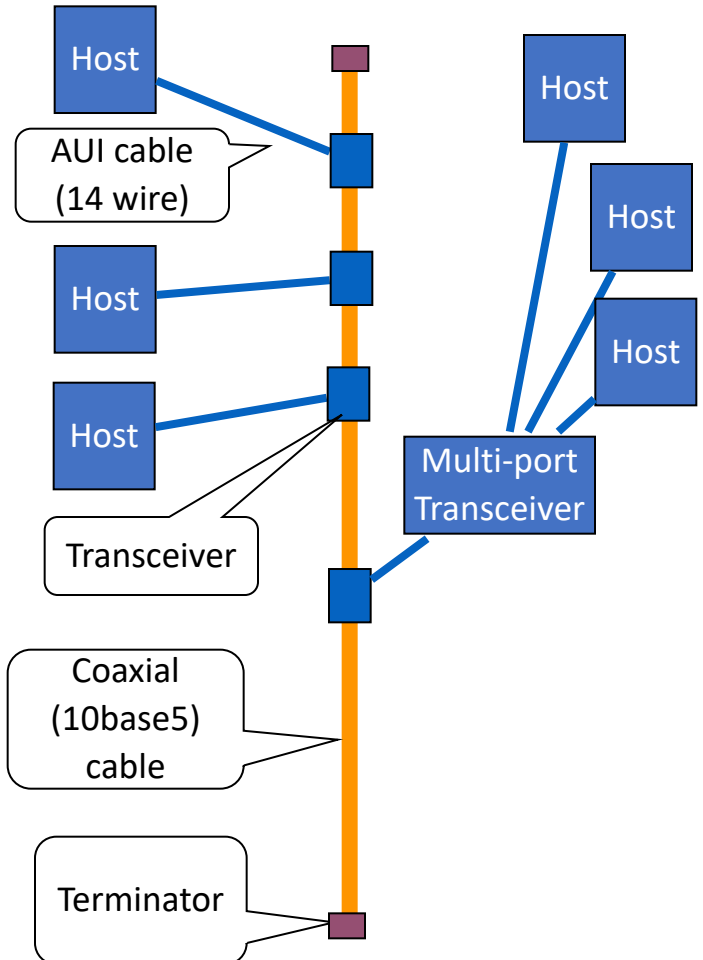


Ethernet Topology (some history)

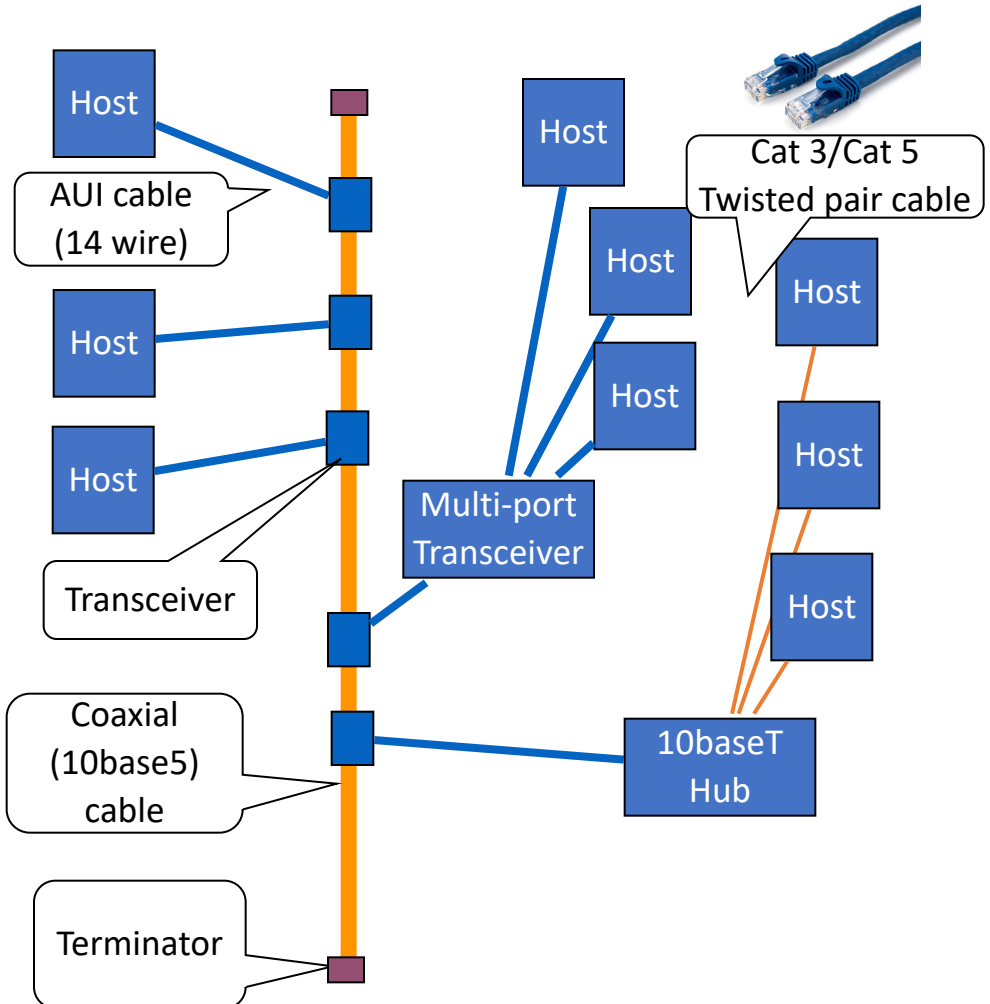
Ethernet Topology (some history)



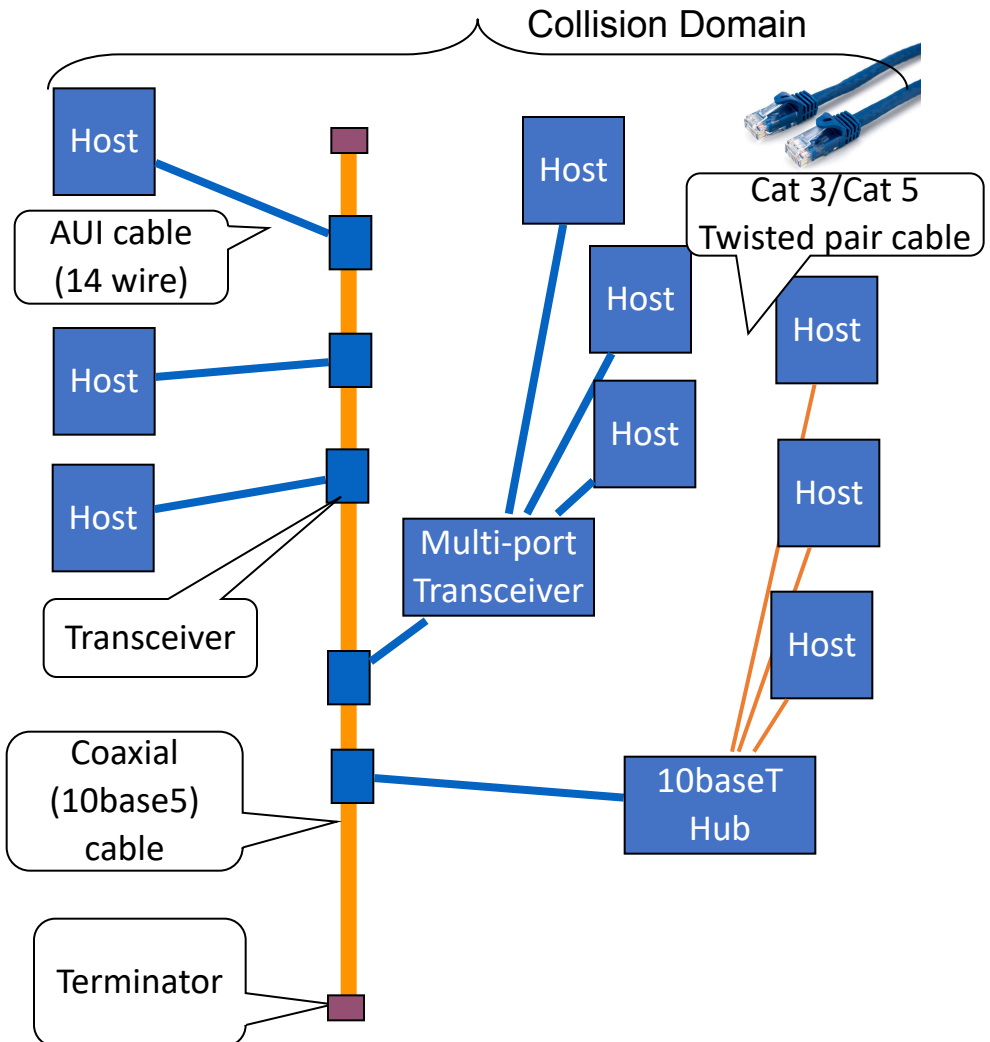
Ethernet Topology (some history)



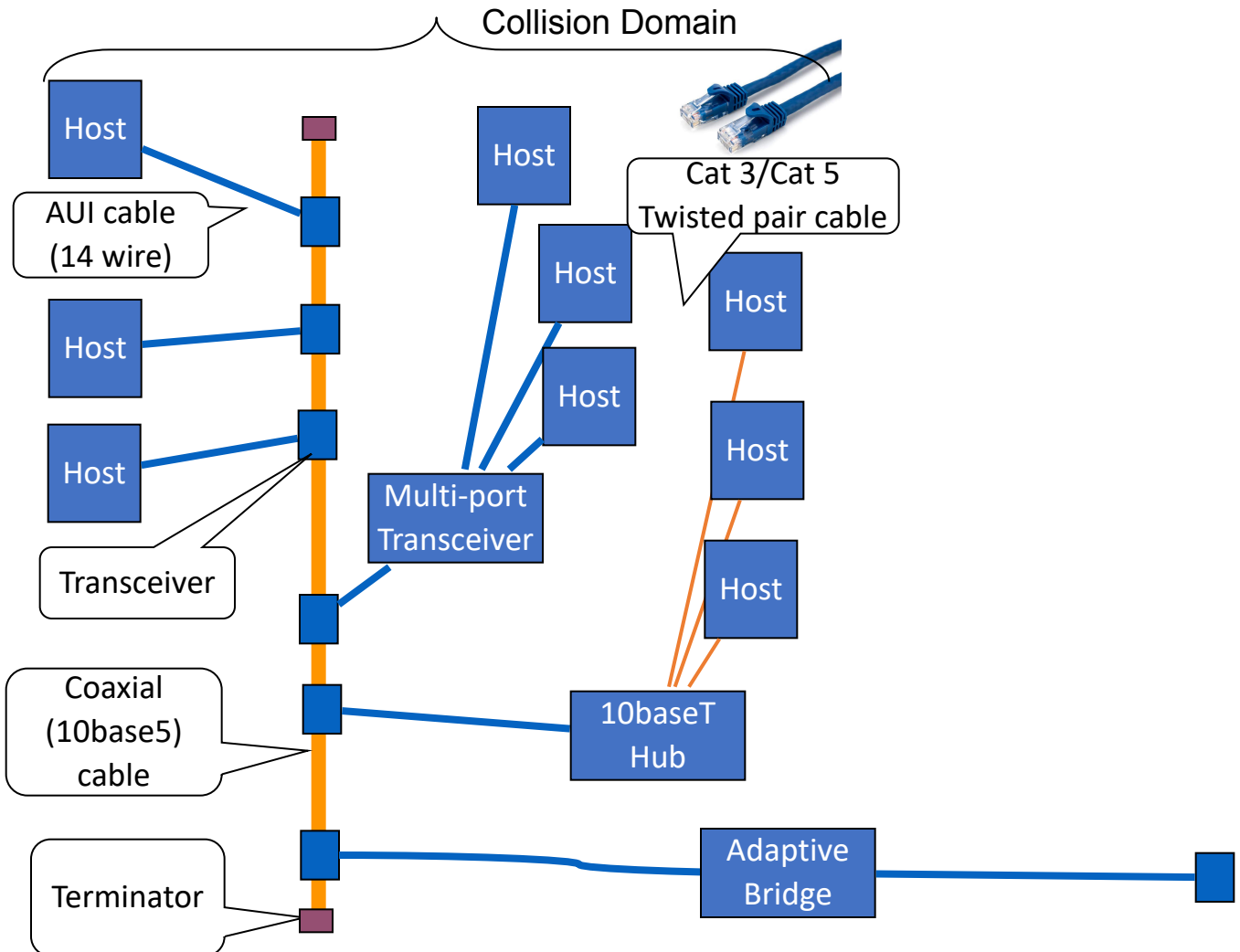
Ethernet Topology (some history)



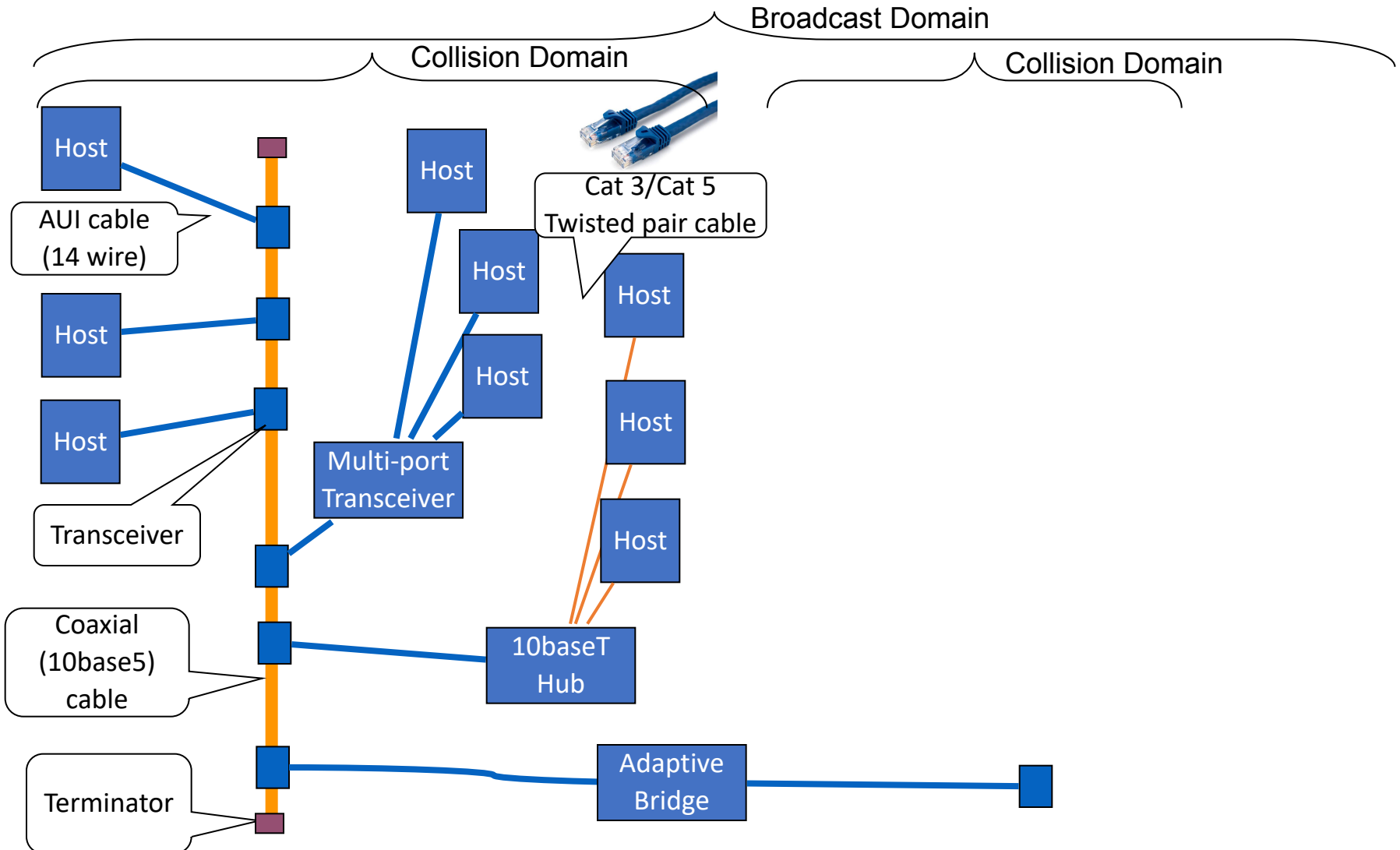
Ethernet Topology (some history)



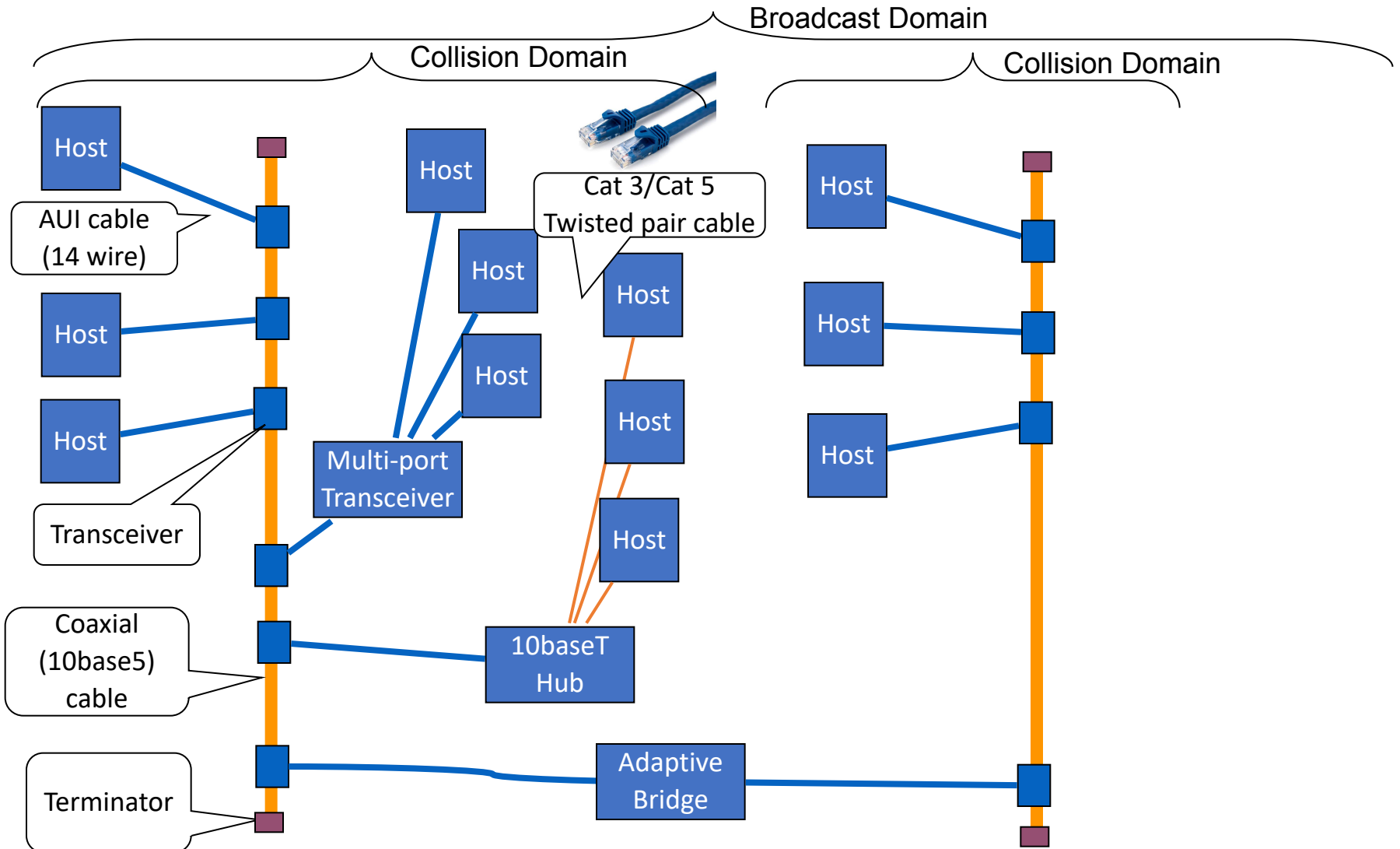
Ethernet Topology (some history)



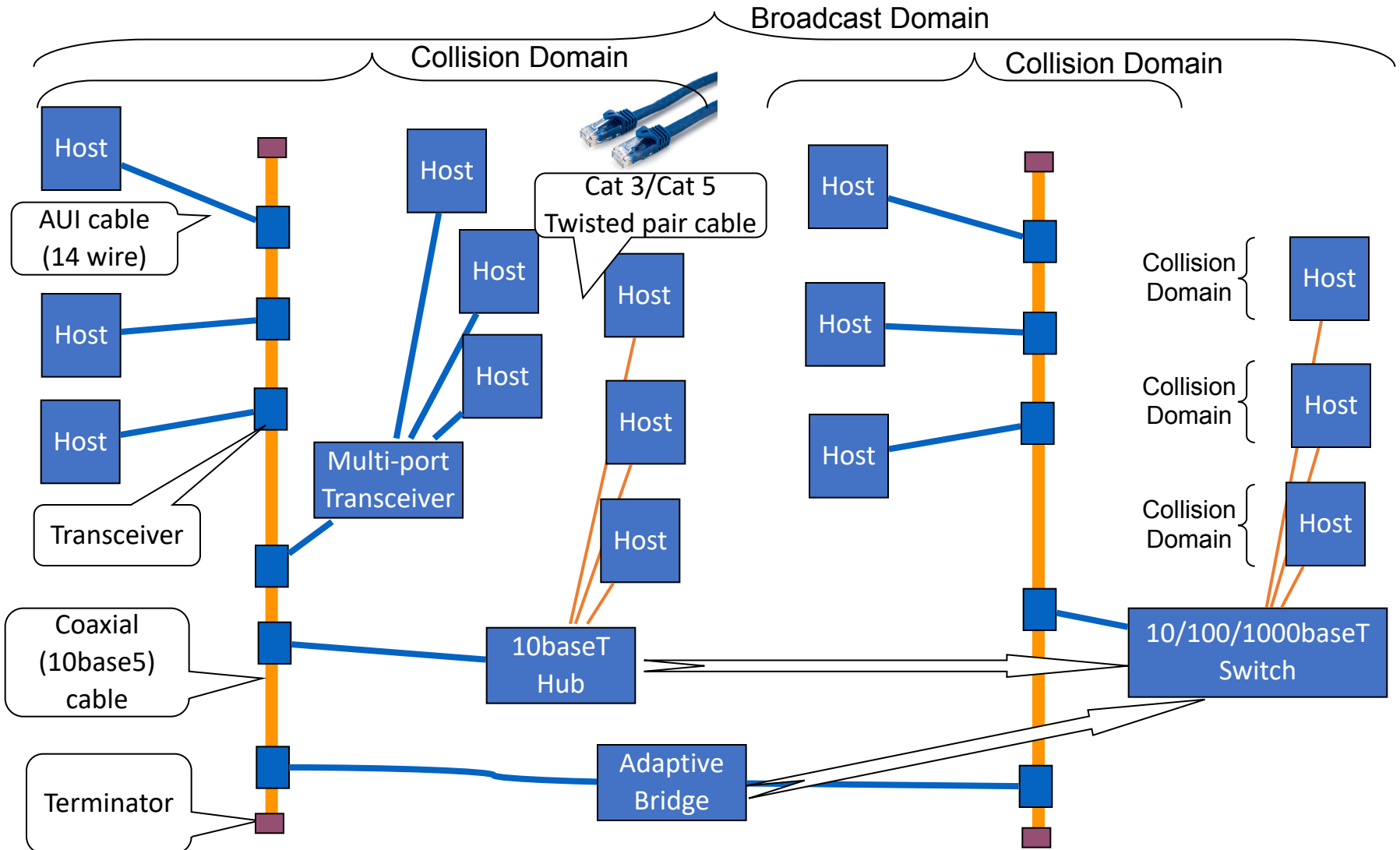
Ethernet Topology (some history)



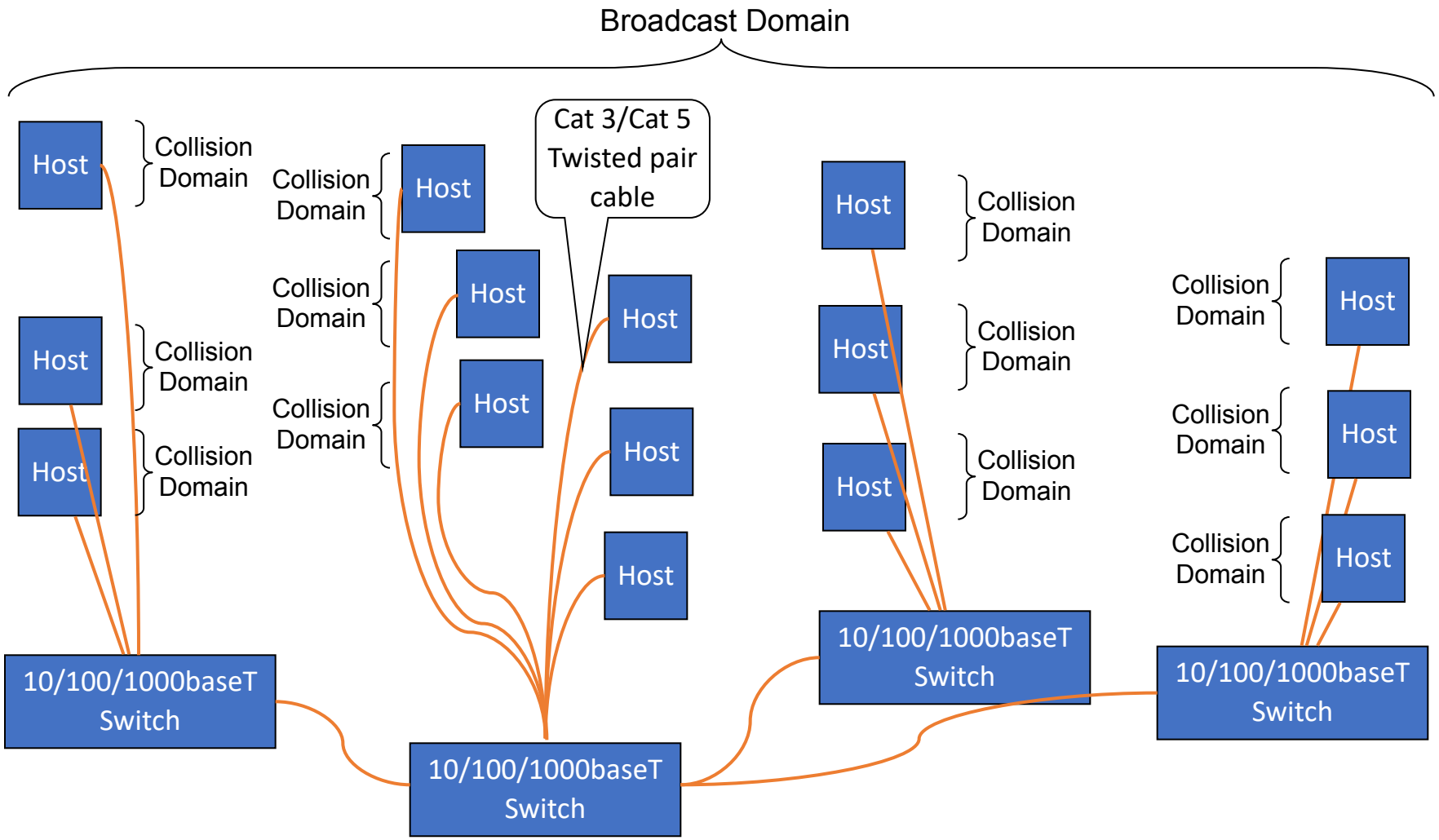
Ethernet Topology (some history)



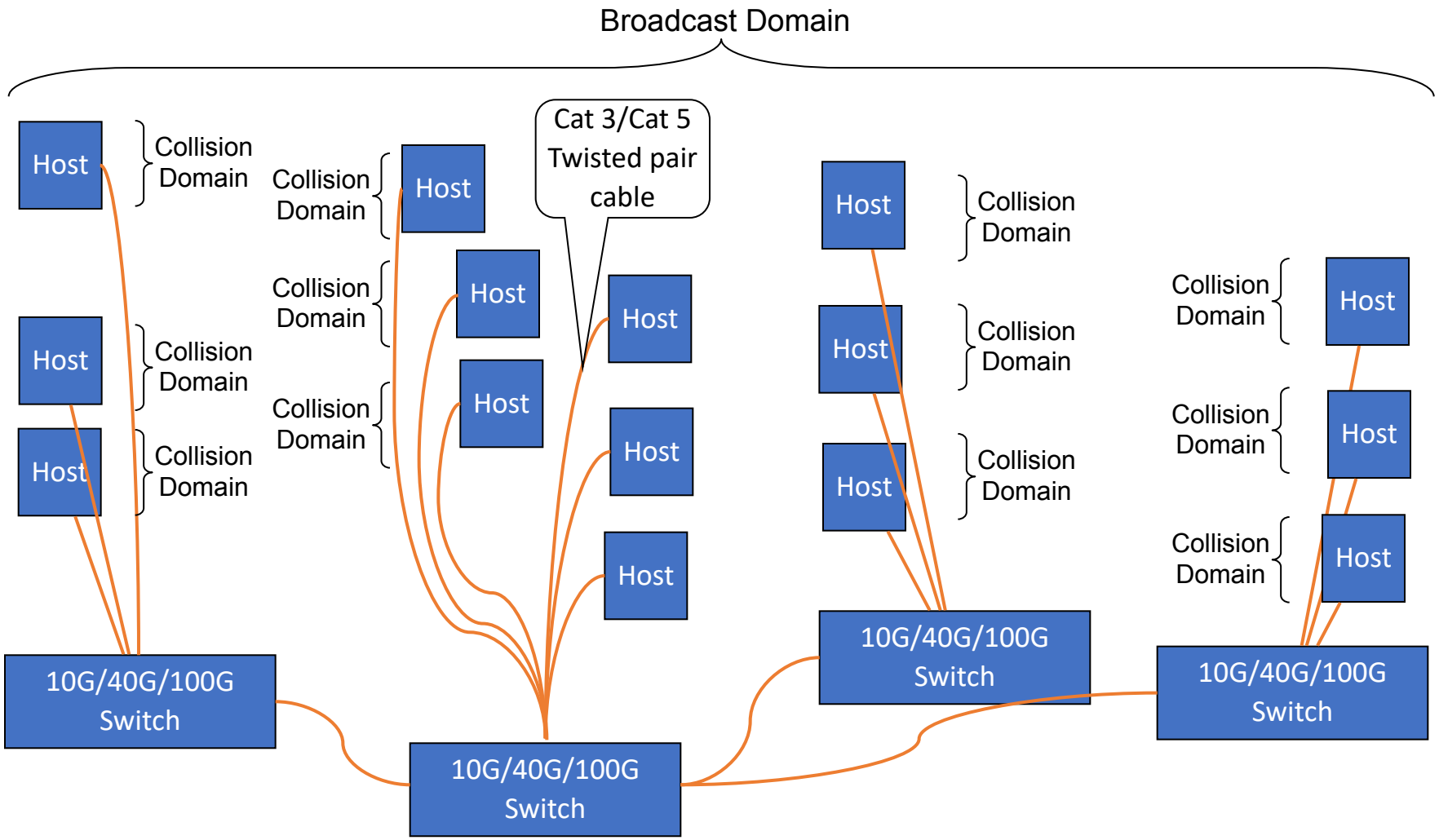
Ethernet Topology (some history)



Ethernet Topology (some history)



Ethernet Topology (some history)



Ethernet Bridging Algorithm (simplified)

Bridge Table		
MAC Addr	Last Seen on Port	Last Used
002080 00324a	*FastEthernet 1/4*	*12 sec ago*
0003ba 51733a	*FastEthernet 2/9*	*72 sec ago*
. . .	. . .	. . .

- When a frame enters the bridge (switch):

Ethernet Bridging Algorithm (simplified)

Bridge Table		
MAC Addr	Last Seen on Port	Last Used
002080 00324a	*FastEthernet 1/4*	*12 sec ago*
0003ba 51733a	*FastEthernet 2/9*	*72 sec ago*
. . .	. . .	. . .

- When a frame enters the bridge (switch):
  - If the **destination** MAC address is in the bridge table
    - Send the frame out the port listed in the bridge table entry
  - Otherwise
    - Flood the frame out all ports except the ingress port
  - Put the **source** MAC address in the bridge table, if necessary, along with the frame ingress port and mark that entry “recently used”

Ethernet Bridging Algorithm (simplified)

Bridge Table		
MAC Addr	Last Seen on Port	Last Used
002080 00324a	*FastEthernet 1/4*	*12 sec ago*
0003ba 51733a	*FastEthernet 2/9*	*72 sec ago*
. . .	. . .	. . .

- When a frame enters the bridge (switch):
  - If the **destination** MAC address is in the bridge table
    - Send the frame out the port listed in the bridge table entry
  - Otherwise
    - Flood the frame out all ports except the ingress port
  - Put the **source** MAC address in the bridge table, if necessary, along with the frame ingress port and mark that entry “recently used”
  - Remove any “stale” entries in the bridge table

Something Is Still Missing

Something Is Still Missing

- Now more than one message can be in transit at a time!

Something Is Still Missing

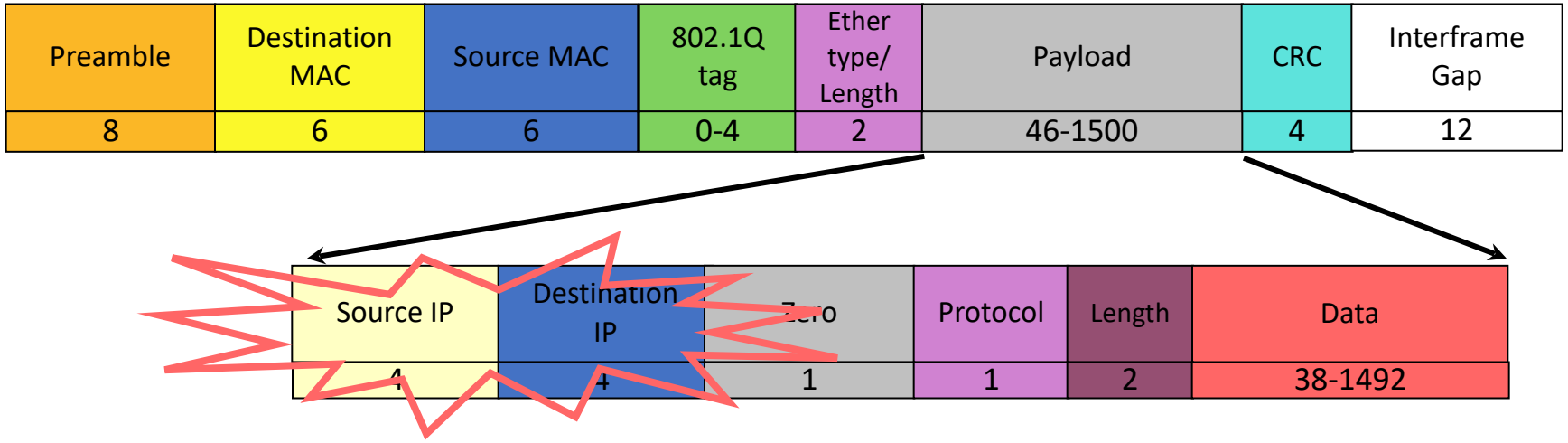
- Now more than one message can be in transit at a time!
- We've only talked about MAC addresses.

Something Is Still Missing

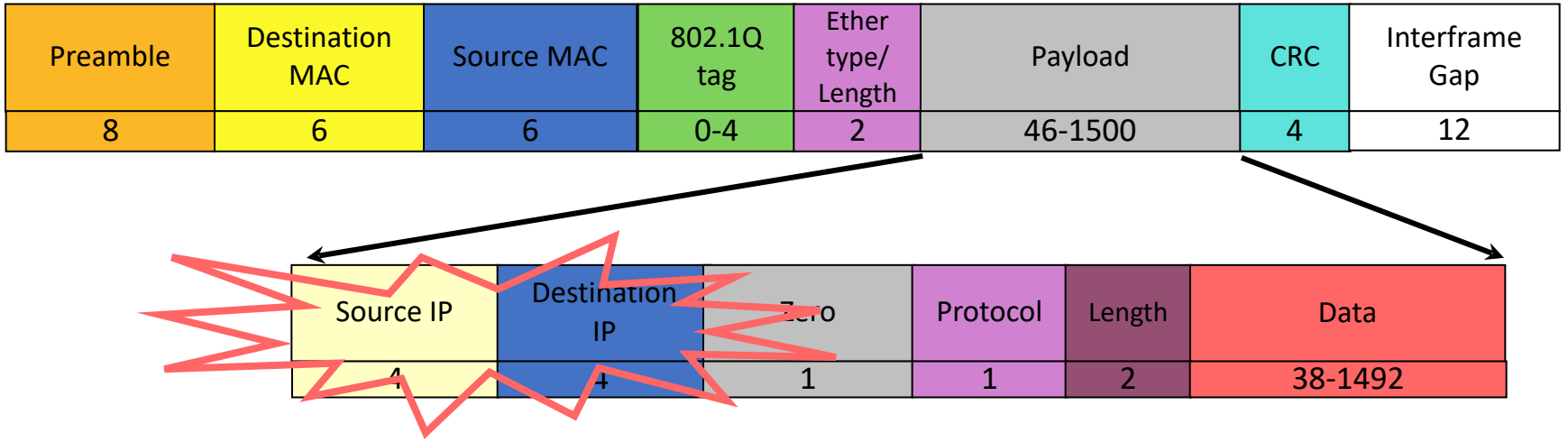
- Now more than one message can be in transit at a time!

- We've only talked about MAC addresses.
- Where are the IP addresses?

Layer 3 (IP) packets

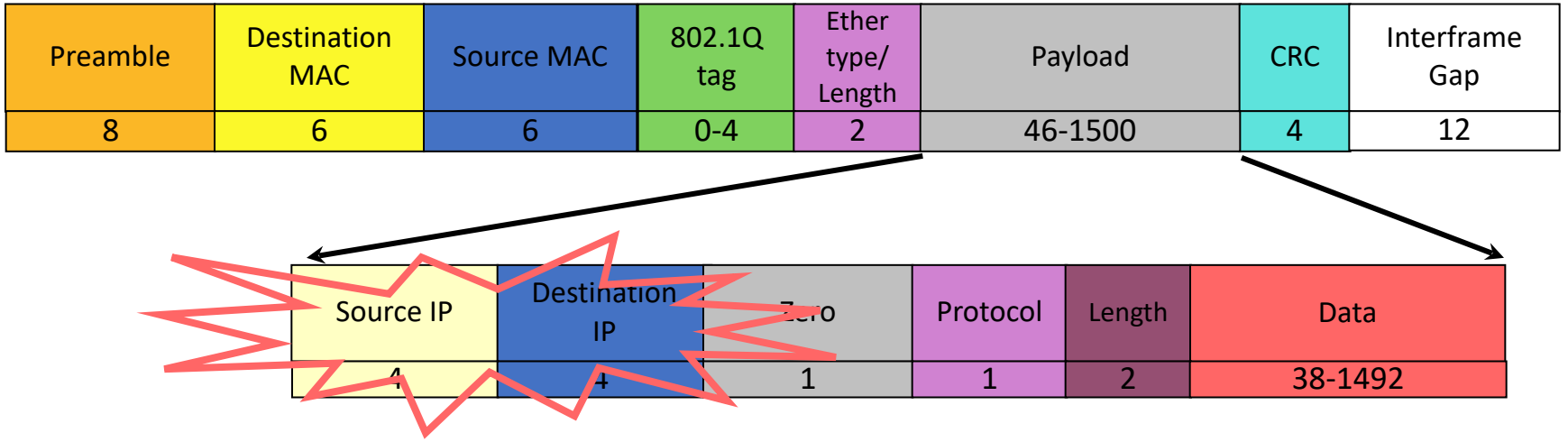


Layer 3 (IP) packets



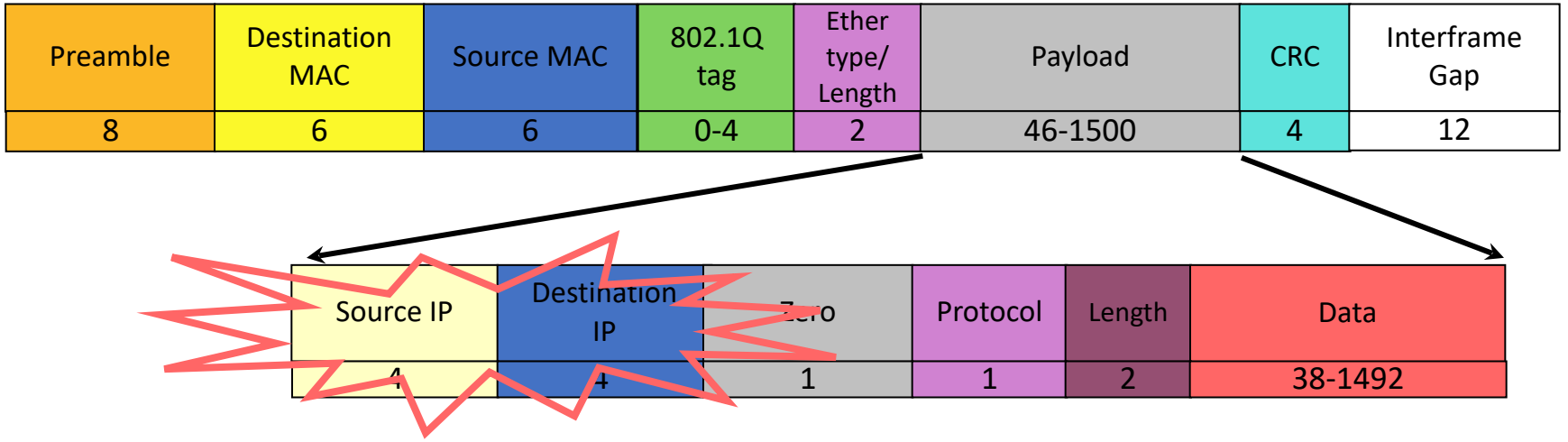
- IP addresses appear in the Ethernet frame payload, not the header

Layer 3 (IP) packets



- IP addresses appear in the Ethernet frame payload, not the header
- But Ethernet hardware only examines the MAC addresses in the Ethernet frame(!)

Layer 3 (IP) packets



- IP addresses appear in the Ethernet frame payload, not the header
- But Ethernet hardware only examines the MAC addresses in the Ethernet frame(!)
- How do we connect these two layers?

ARP

- Address Resolution Protocol

ARP

- Address Resolution Protocol
- Used with Ethernet to discover IP addresses

ARP

- Address Resolution Protocol
- Used with Ethernet to discover IP addresses
- How?

ARP

- Address Resolution Protocol
- Used with Ethernet to discover IP addresses
- How?
  - IP stack needs to send to IP address *a*, but doesn't know what destination MAC address to use.

ARP

- Address Resolution Protocol
- Used with Ethernet to discover IP addresses
- How?
  - IP stack needs to send to IP address *a*, but doesn't know what destination MAC address to use.
  - Broadcast an ARP-request message: "Who has IP address *a*?"

ARP

- Address Resolution Protocol
- Used with Ethernet to discover IP addresses
- How?
  - IP stack needs to send to IP address *a*, but doesn't know what destination MAC address to use.
  - Broadcast an ARP-request message: "Who has IP address *a*?"
  - Wait on unicast ARP-response: "I have IP address *a*."

ARP

- Address Resolution Protocol
- Used with Ethernet to discover IP addresses
- How?
  - IP stack needs to send to IP address *a*, but doesn't know what destination MAC address to use.
  - Broadcast an ARP-request message: "Who has IP address *a*?"
  - Wait on unicast ARP-response: "I have IP address *a*."
  - Store the response in a local *ARP Table*, discarding entries when they become stale.

ARP

- Address Resolution Protocol
- Used with Ethernet to discover IP addresses
- How?
  - IP stack needs to send to IP address *a*, but doesn't know what destination MAC address to use.
  - Broadcast an ARP-request message: "Who has IP address *a*?"
  - Wait on unicast ARP-response: "I have IP address *a*."
  - Store the response in a local *ARP Table*, discarding entries when they become stale.
  - Use the corresponding entry in the ARP table to set the destination MAC address of the frame.

ARP Example

```
bash-3.00# arp -an
Net to Media Table: IPv4
Device   IP Address               Mask      Flags      Phys Addr
------ -------------------- --------------- -------- ---------------
bge0   130.207.165.229      255.255.255.255 o        00:50:56:91:4a:2b
bge0   130.207.165.248      255.255.255.255 o        00:14:4f:f2:8c:ce
bge0   130.207.165.240      255.255.255.255 o        00:50:56:91:62:be
bge0   130.207.165.242      255.255.255.255 o        00:14:4f:21:14:f6
bge0   130.207.165.245      255.255.255.255 o        00:14:4f:94:d7:0c
bge0   130.207.165.194      255.255.255.255 o        00:50:56:91:59:25
bge0   130.207.165.197      255.255.255.255 o        00:50:56:91:46:a3
bge0   130.207.165.221      255.255.255.255 o        00:50:56:91:4c:af
bge0   130.207.165.163      255.255.255.255 o        00:14:4f:7d:7f:58
bge0   130.207.165.137      255.255.255.255          00:03:ba:1a:28:a7
bge0   130.207.165.138      255.255.255.255          00:03:ba:1a:2a:ff
bge0   130.207.165.158      255.255.255.255 o        00:14:4f:1e:26:ec
bge0   130.207.165.96       255.255.255.255 o        00:1b:24:1d:eb:7c
bge0   130.207.165.103      255.255.255.255 o        00:50:56:a4:7e:df
bge0   130.207.165.122      255.255.255.255 o        00:50:56:91:43:f8
```

VLANs

VLANs

- Virtual LAN, a.k.a. VLAN
  - A group of hosts that can communicate as if they were attached to the same **broadcast domain**, regardless of their physical location.
  - A VLAN has the same attributes as a physical LAN, but it allows for hosts to be grouped together even if they are not connected to the same network switch.
  - Network reconfiguration can be done through software instead of physically relocating devices and wires.

Geographic Addressing

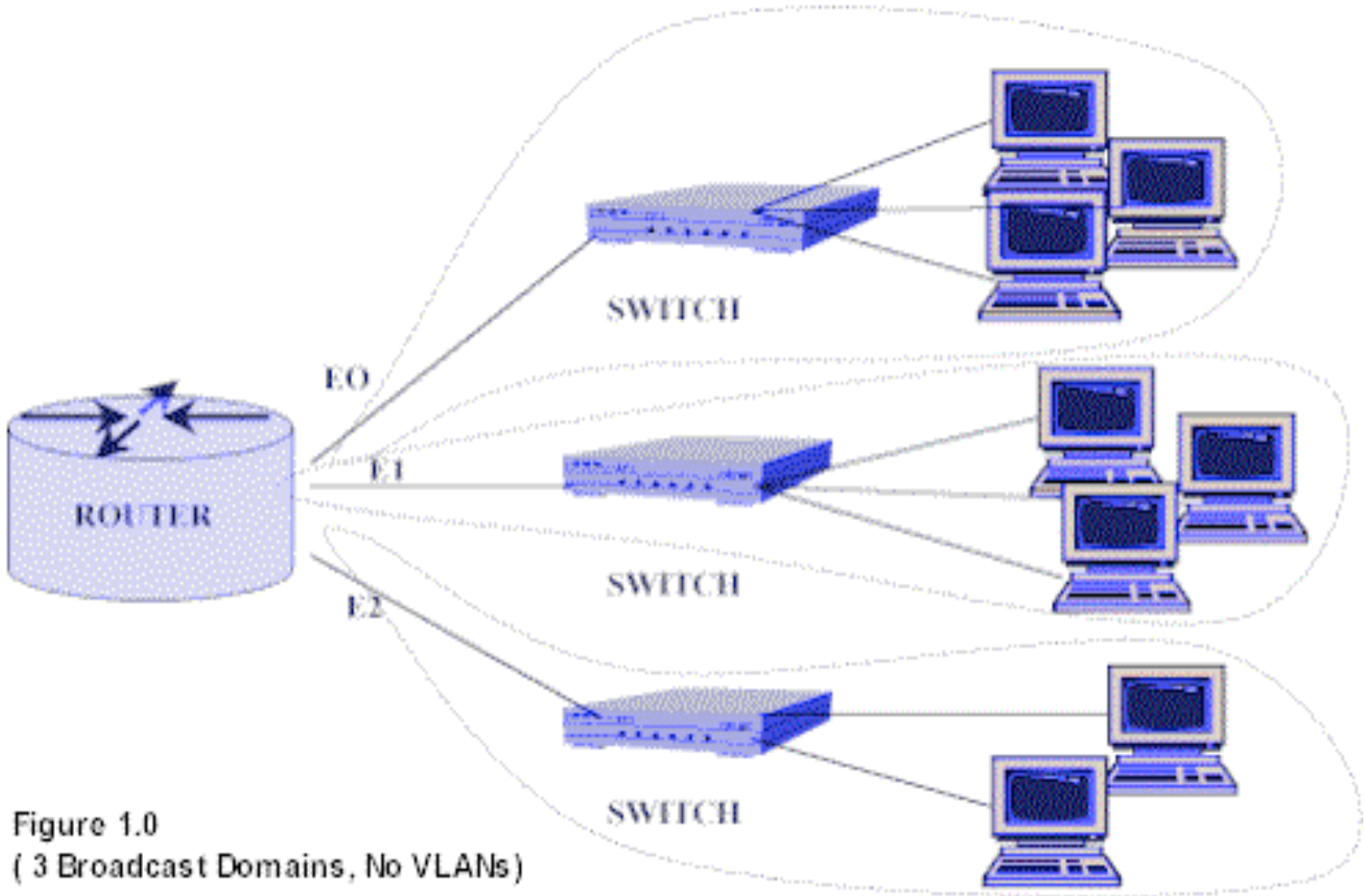
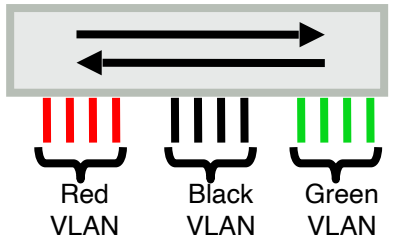


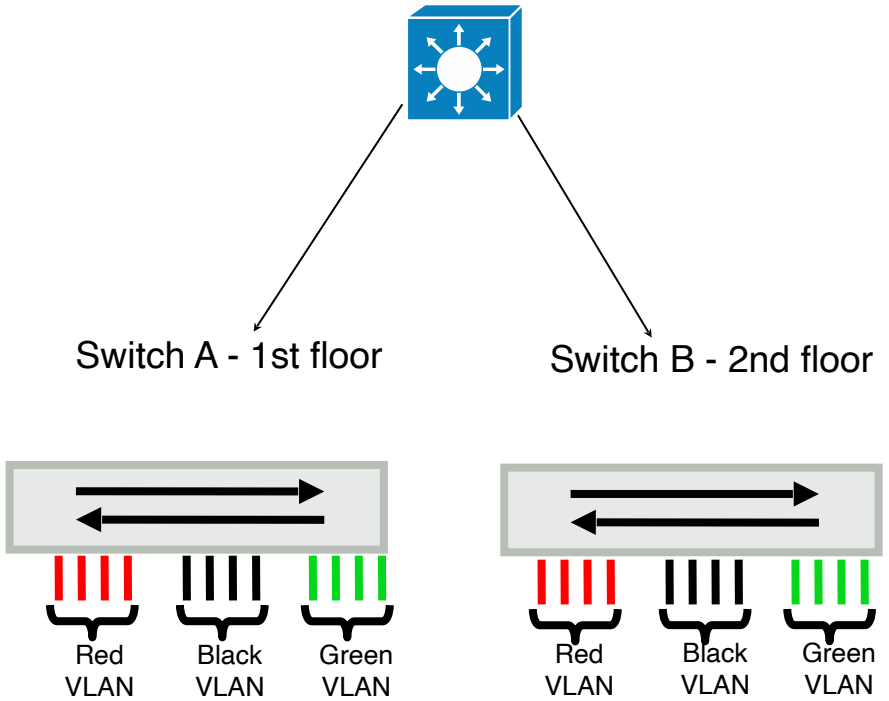
Figure 1.0
(3 Broadcast Domains, No VLANs)

After VLANs

Switch A



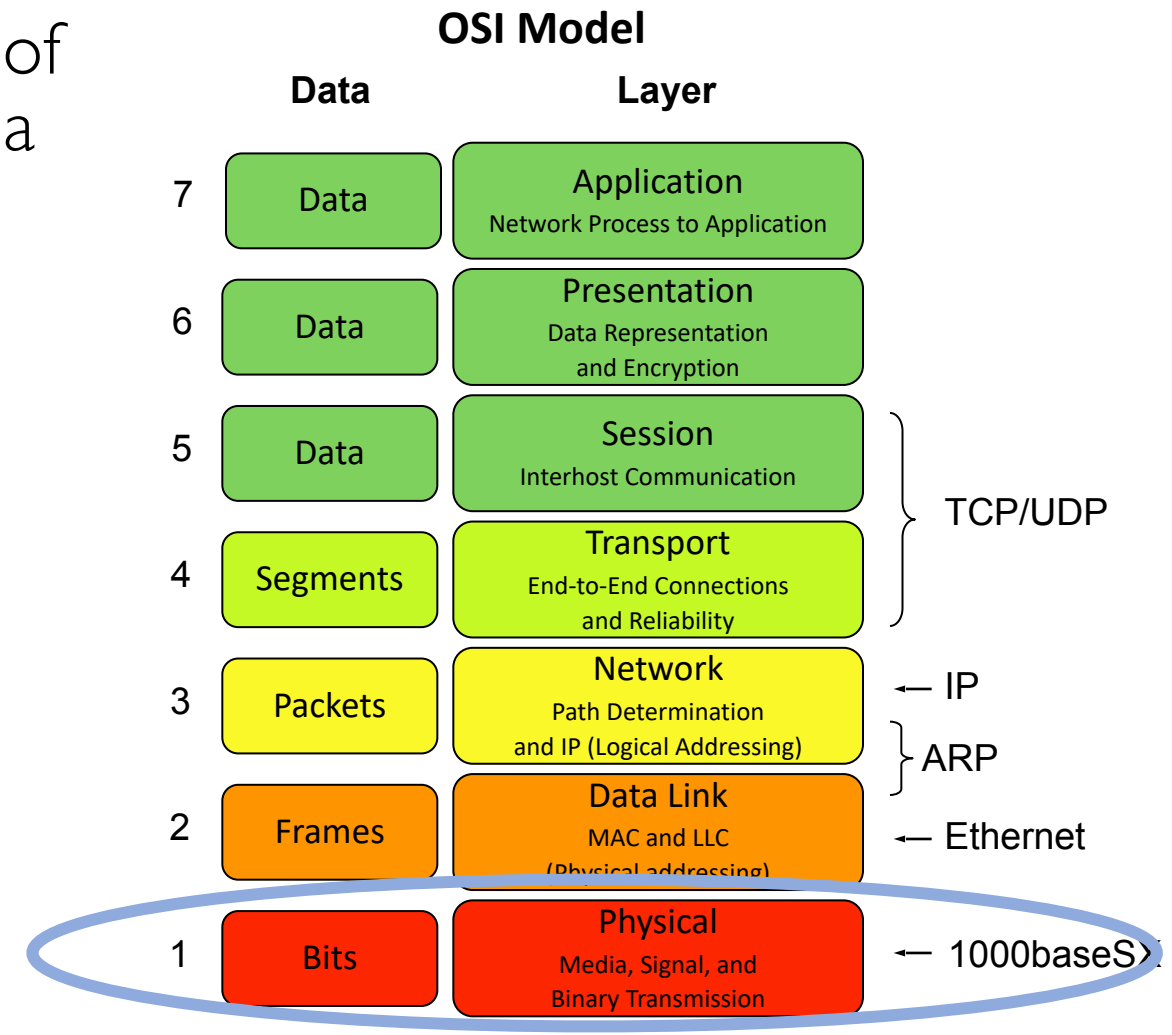
- Each logical VLAN is like a separate physical bridge
  - College of Engineering (red)
  - College of Computing (black)
  - Classroom network (green)



- Each logical VLAN is like a separate physical bridge
- VLANs can span across multiple switches

Now let's look at the last layer

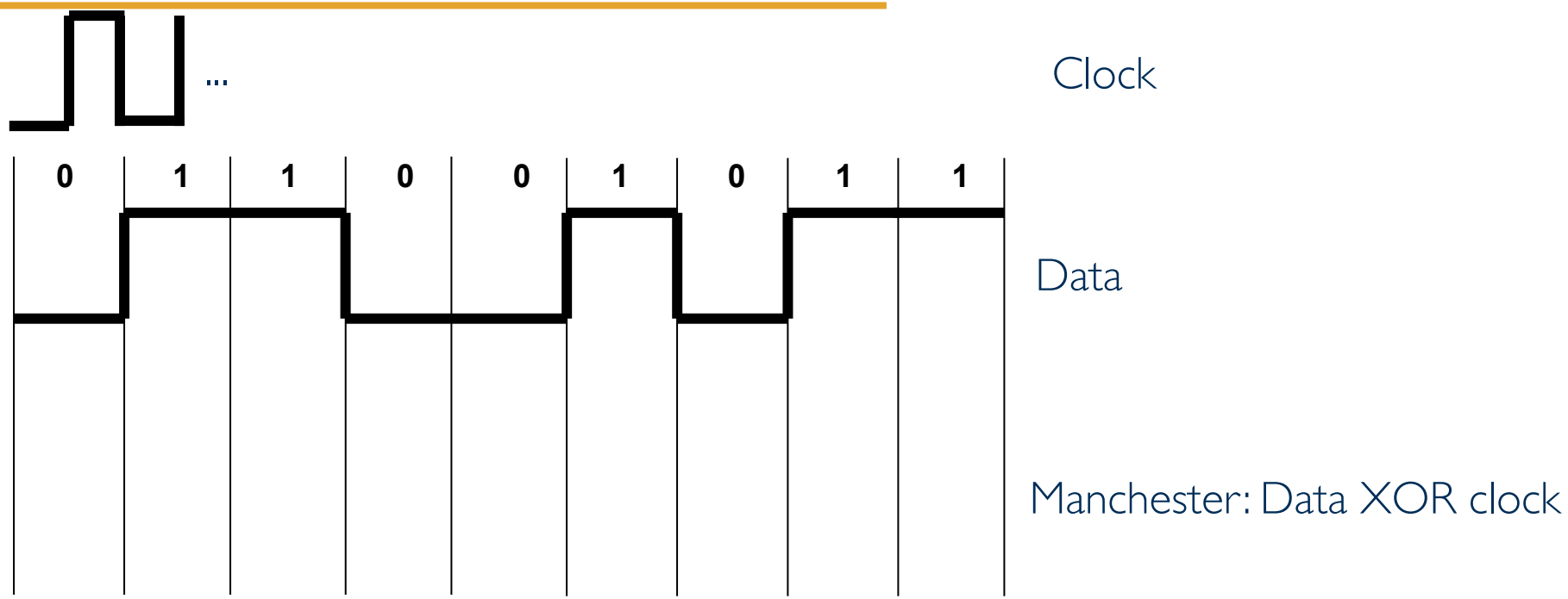
- Ethernet standards allow use of several types of physical media



Ethernet physical media standards

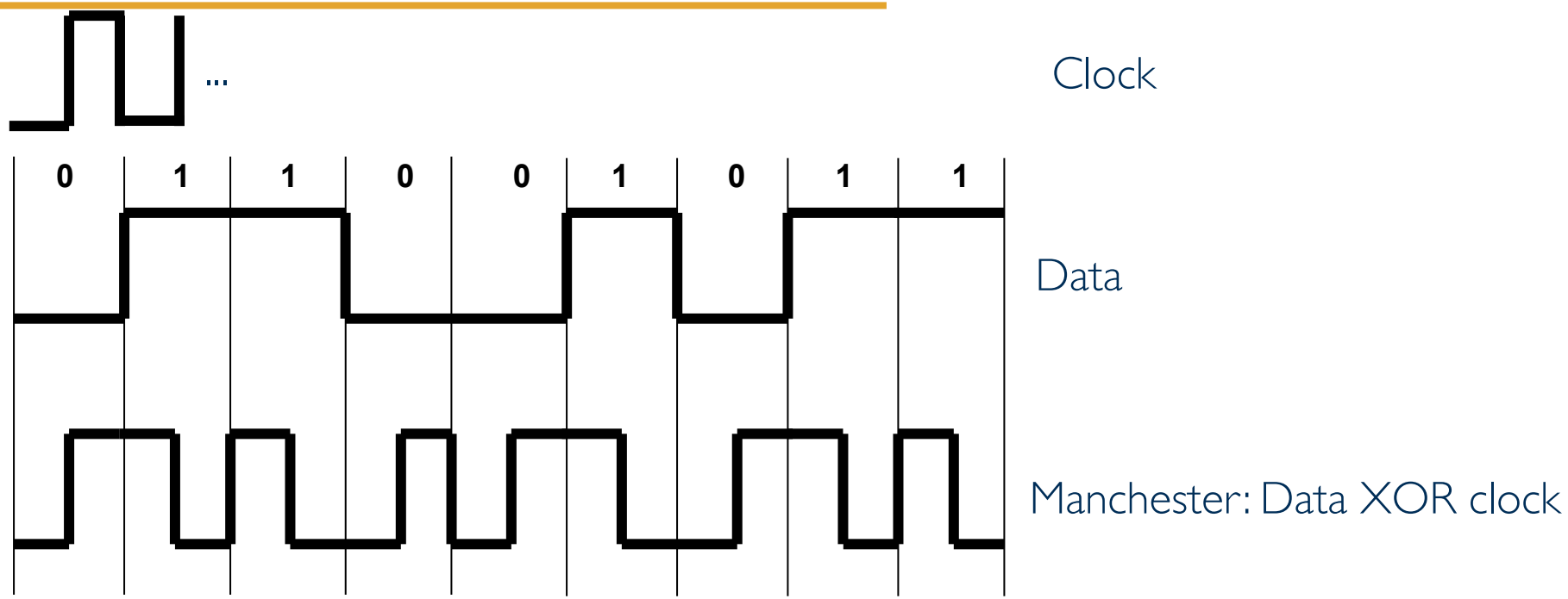
- Twisted pair copper (1m – 100m reach)
  - 10base-T
    100base-T
    1000base-TX
    10Gbase-T
- Fiber optic (100m – 40km reach)
  - 1000base-SX, 1000base-LX
    10Gbase-SR, 10Gbase-LR
    40Gbase-SR4, 40Gbase-LR4
    100Gbase-SR4, 100Gbase-SR10, 100Gbase-LR4
    and many more

Manchester encoding

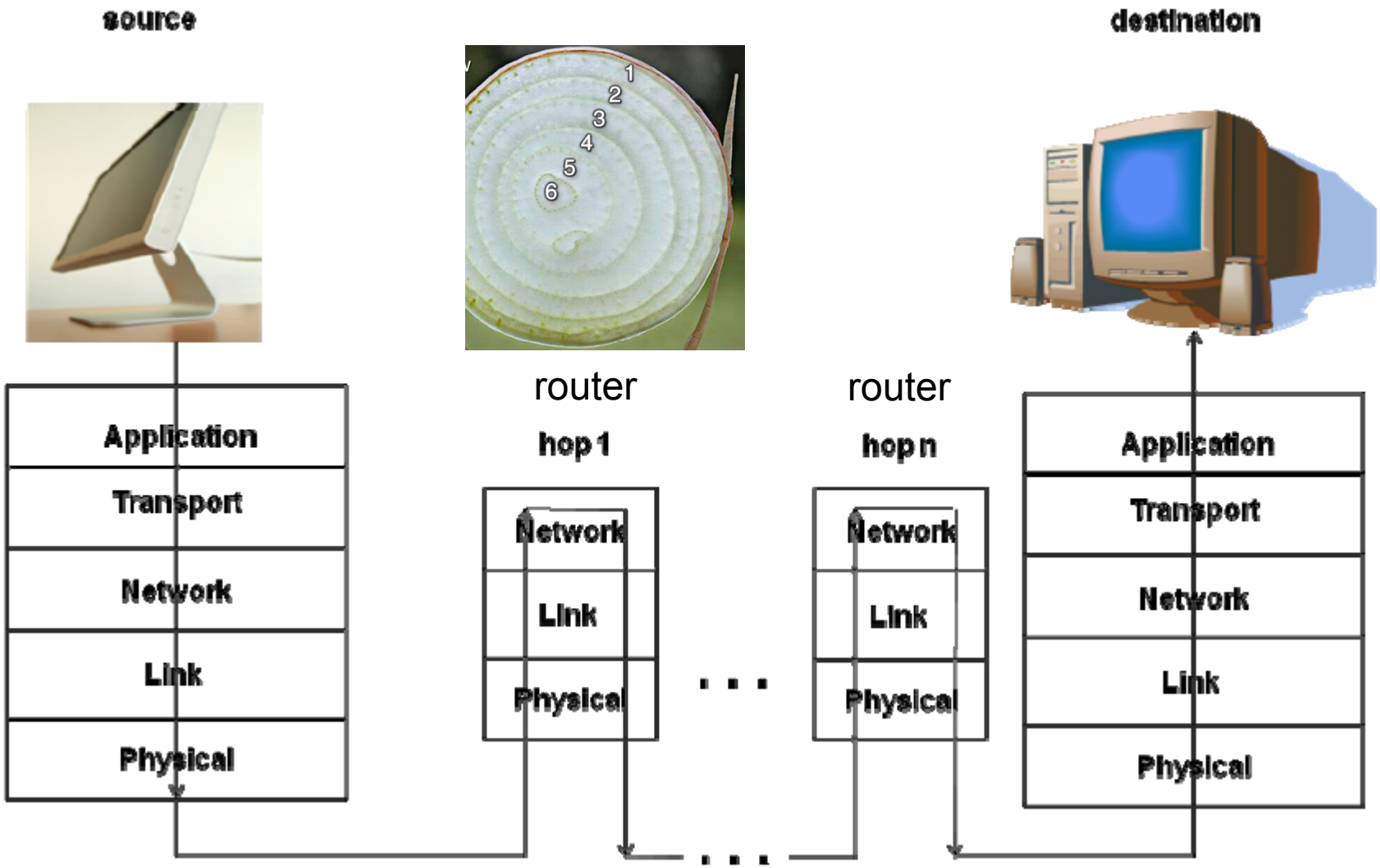


- Ethernet coax signaling uses Manchester encoding which is a scheme that guarantees at least one voltage transition during each clock tick

Manchester encoding



- Ethernet coax signaling uses Manchester encoding which is a scheme that guarantees at least one voltage transition during each clock tick



PHY (Layer 1)
Header

Ethernet
(Layer 2)
Header

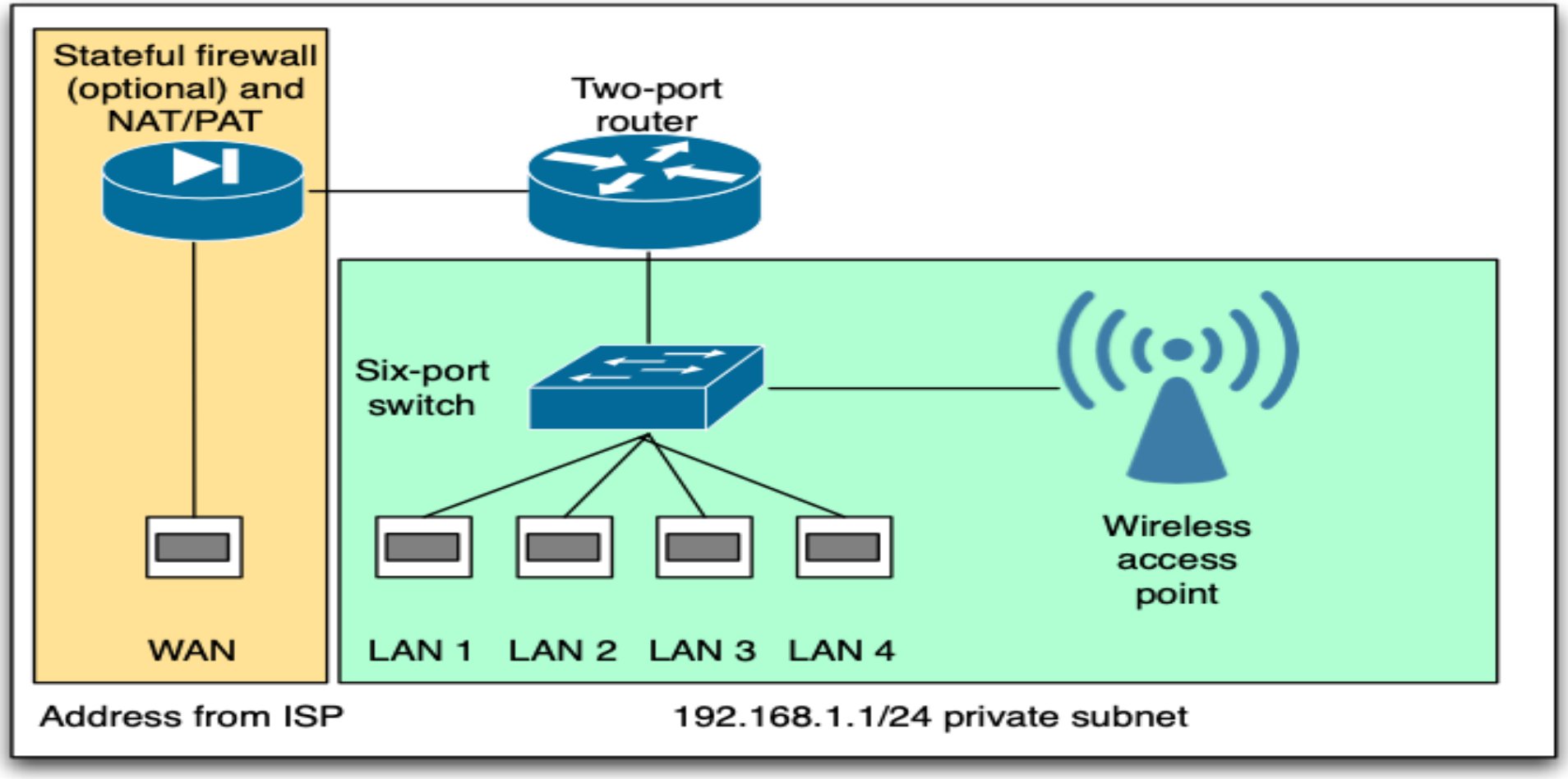
IP (Layer 3)
Header

TCP (Layer 4-5)
Header

TCP Payload

Ethernet
Trailer

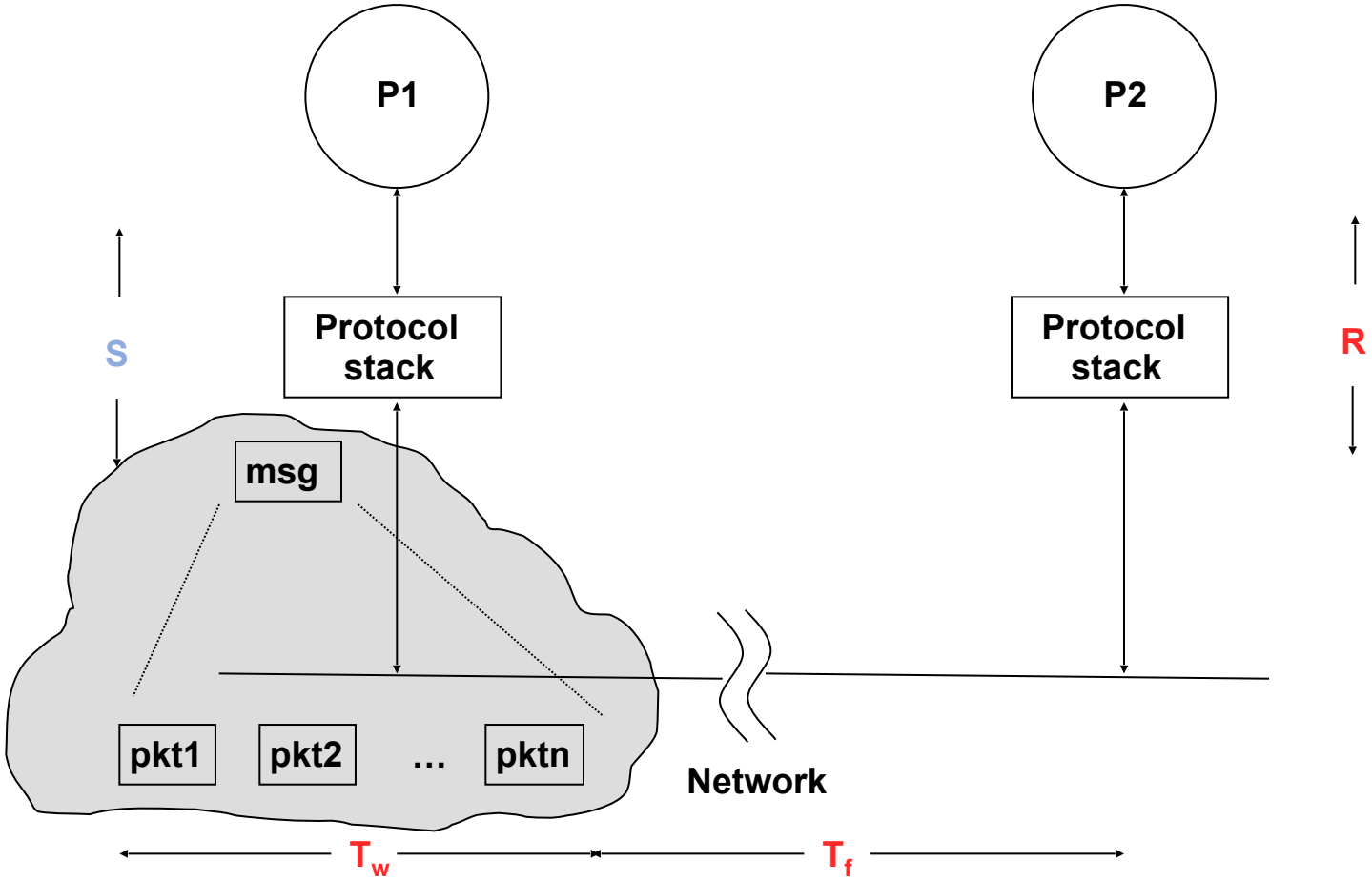
What's in a Home Router?



Performance metrics

Transmission time or end-to-end latency = $S+T_W+T_f+R$

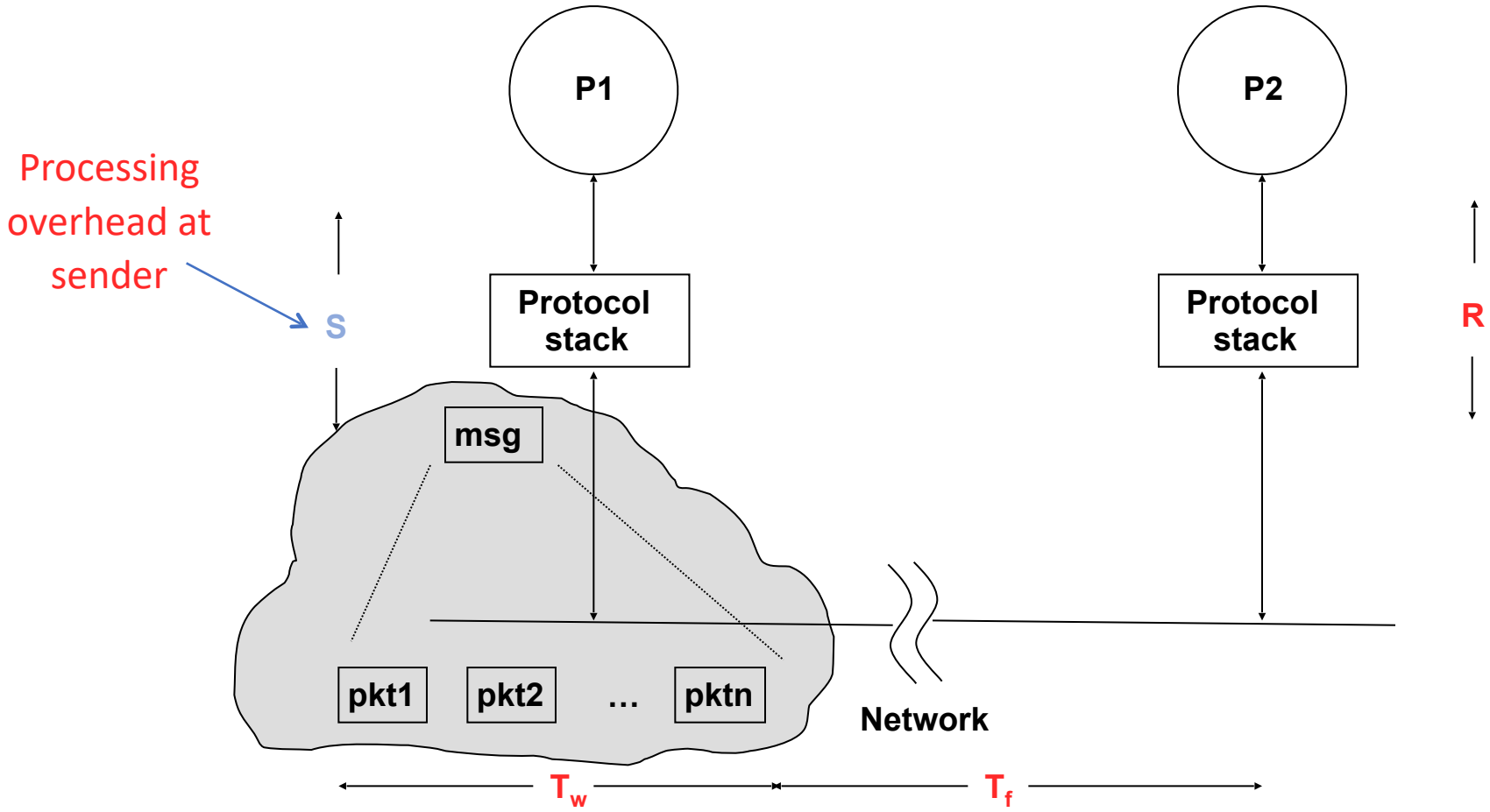
Message throughput = message-size/end-to-end-latency



Performance metrics

Transmission time or end-to-end latency = $S+T_W+T_f+R$

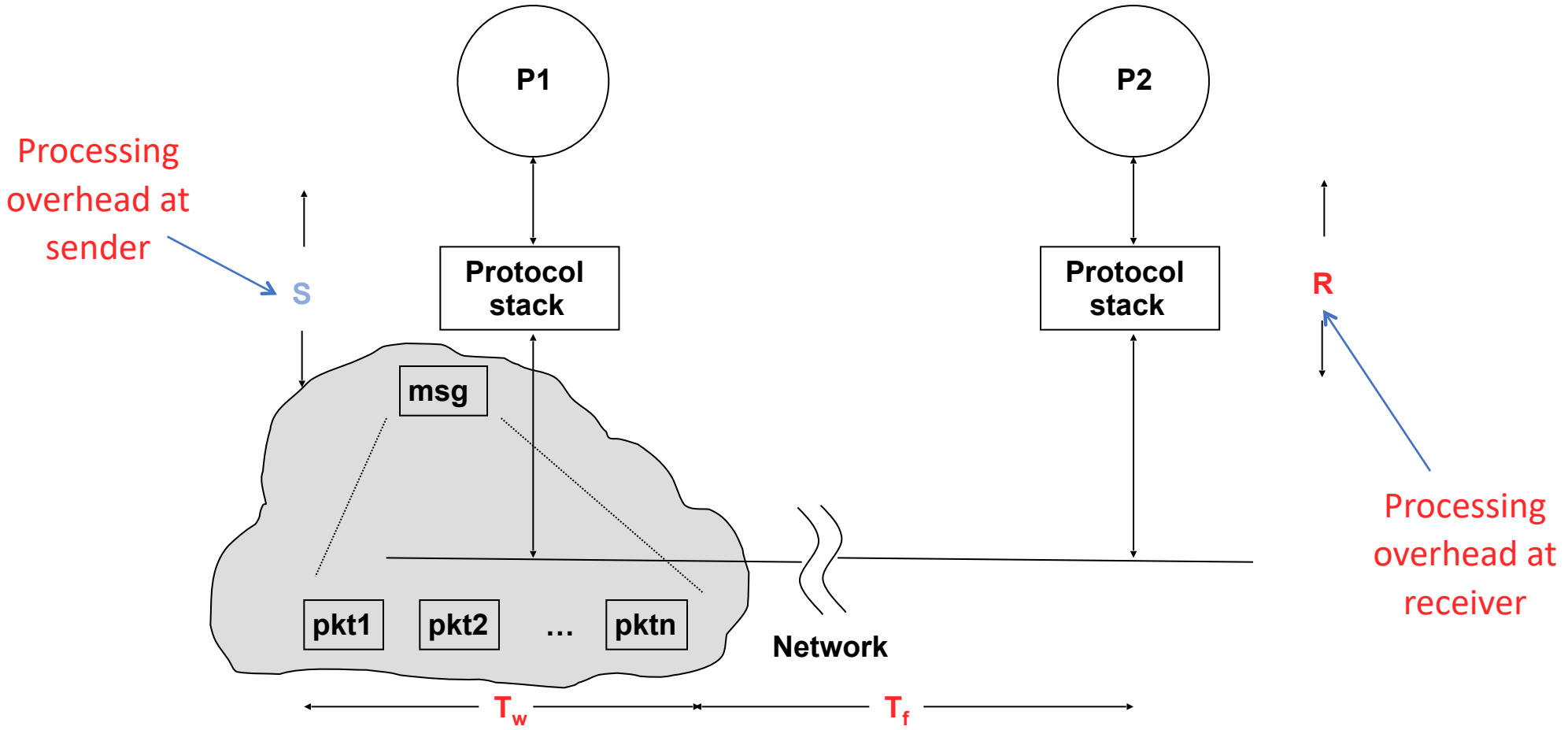
Message throughput = message-size/end-to-end-latency



Performance metrics

Transmission time or end-to-end latency = $S+T_W+T_f+R$

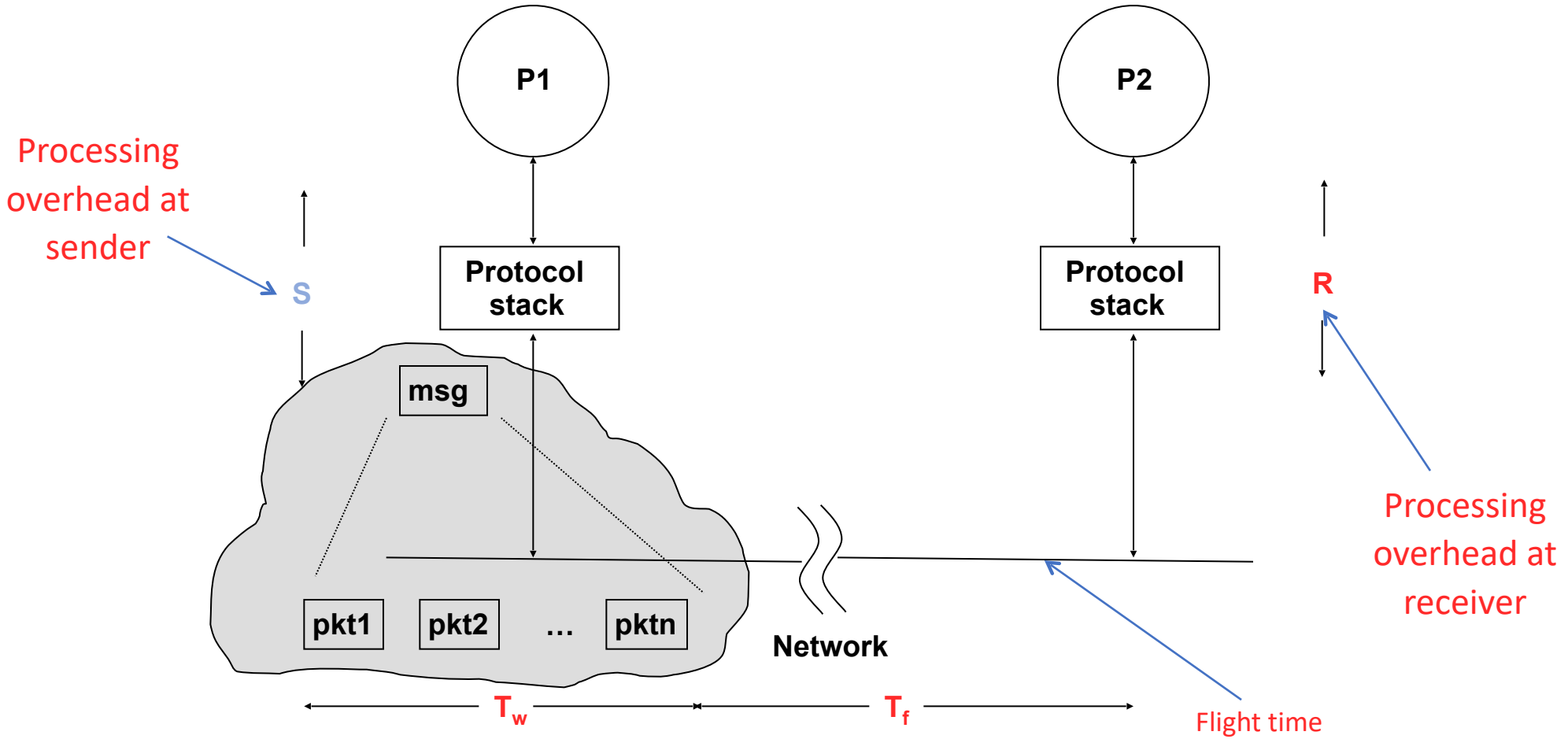
Message throughput = message-size/end-to-end-latency



Performance metrics

Transmission time or end-to-end latency = $S+T_W+T_f+R$

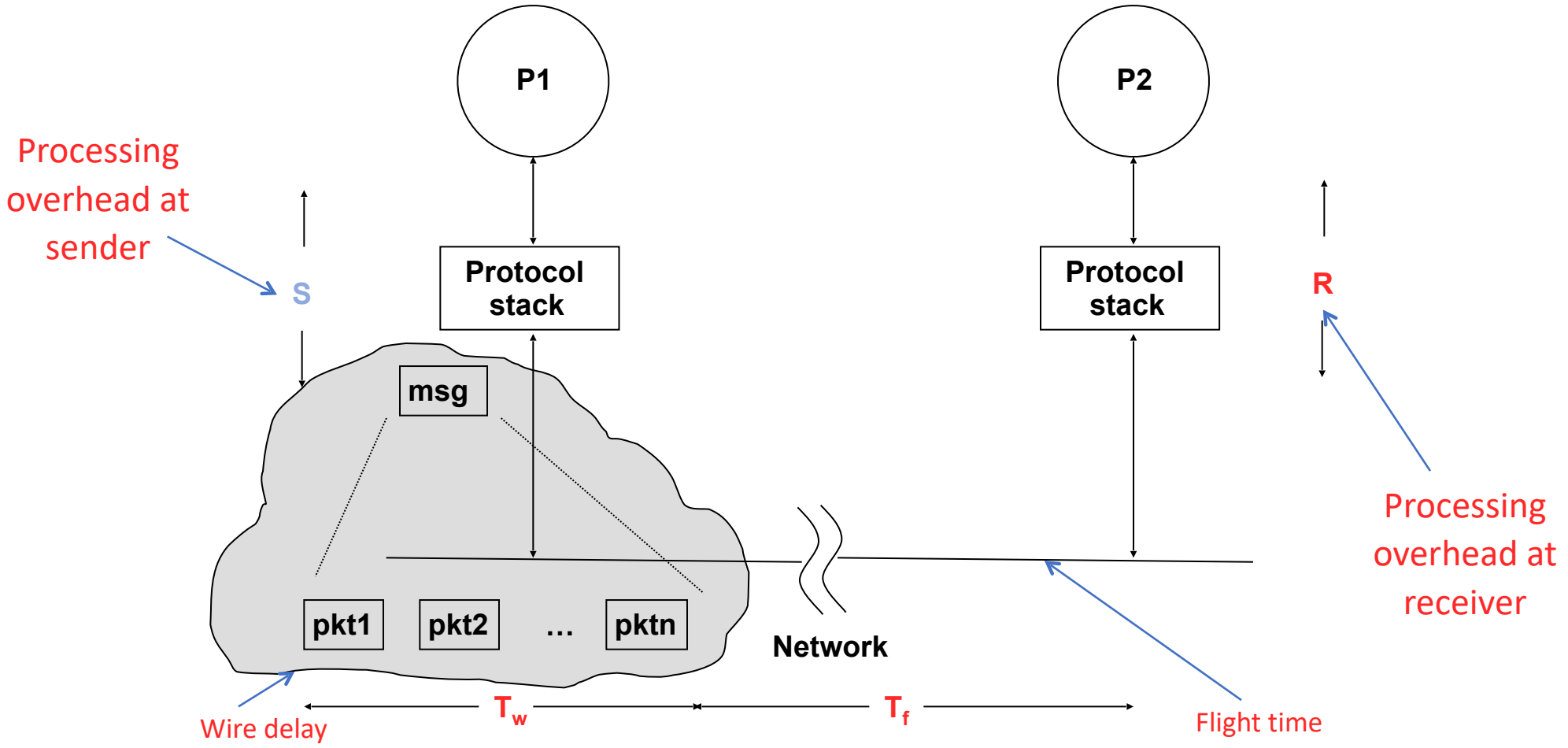
Message throughput = message-size/end-to-end-latency



Performance metrics

Transmission time or end-to-end latency = $S+T_W+T_f+R$

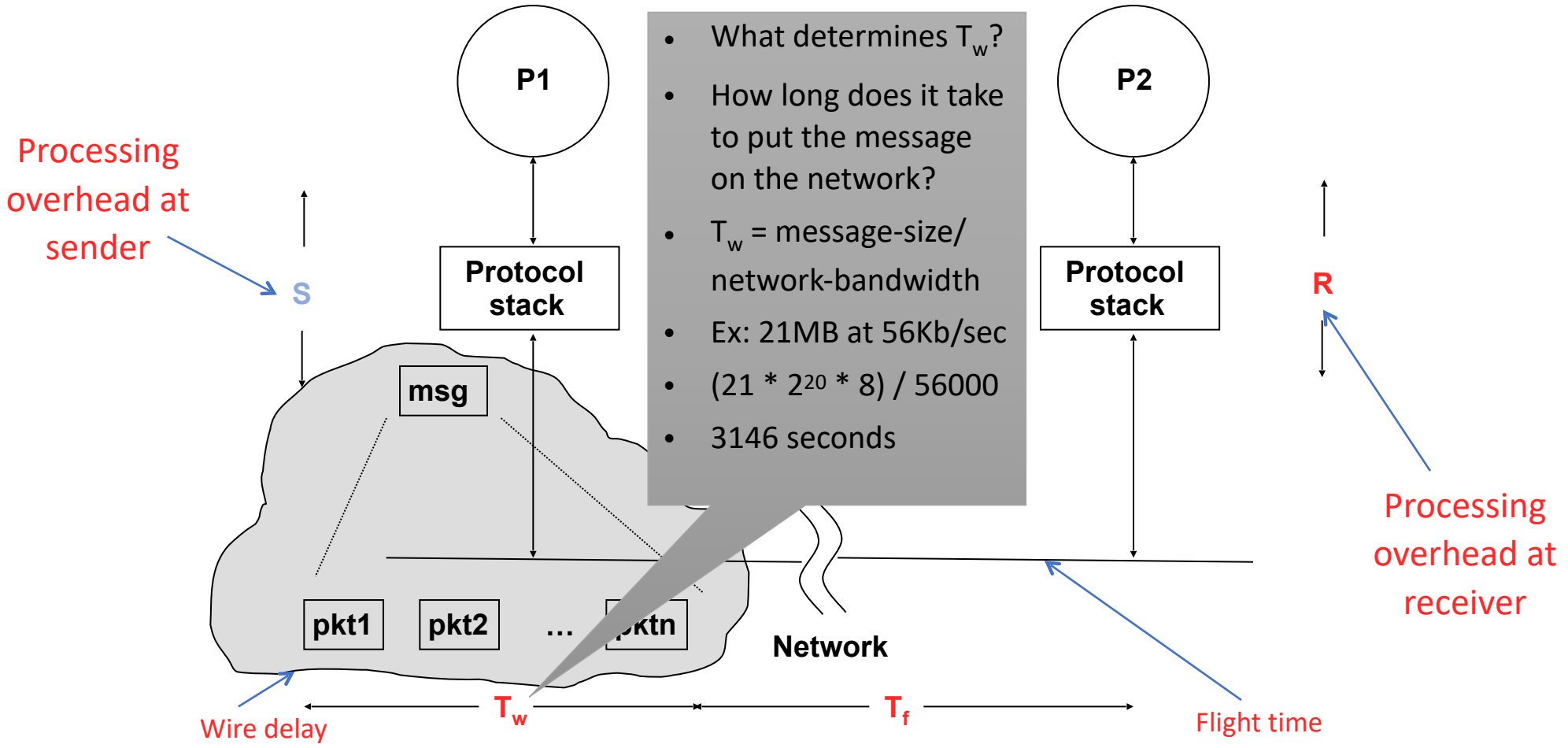
Message throughput = message-size/end-to-end-latency



Performance metrics

Transmission time or end-to-end latency = $S+T_W+T_f+R$

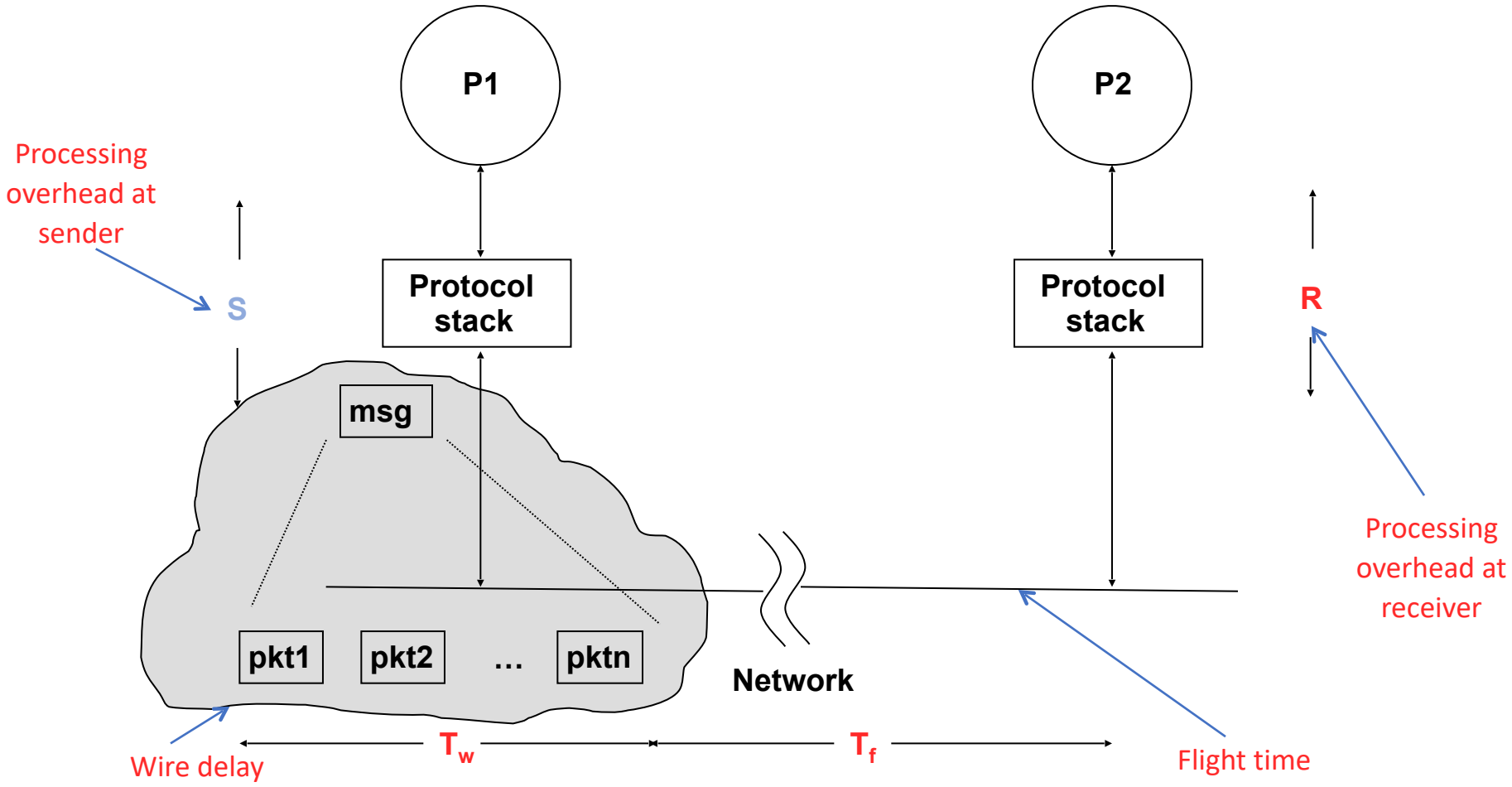
Message throughput = message-size/end-to-end-latency



Performance metrics

Transmission time or end-to-end latency = $S+T_W+T_f+R$

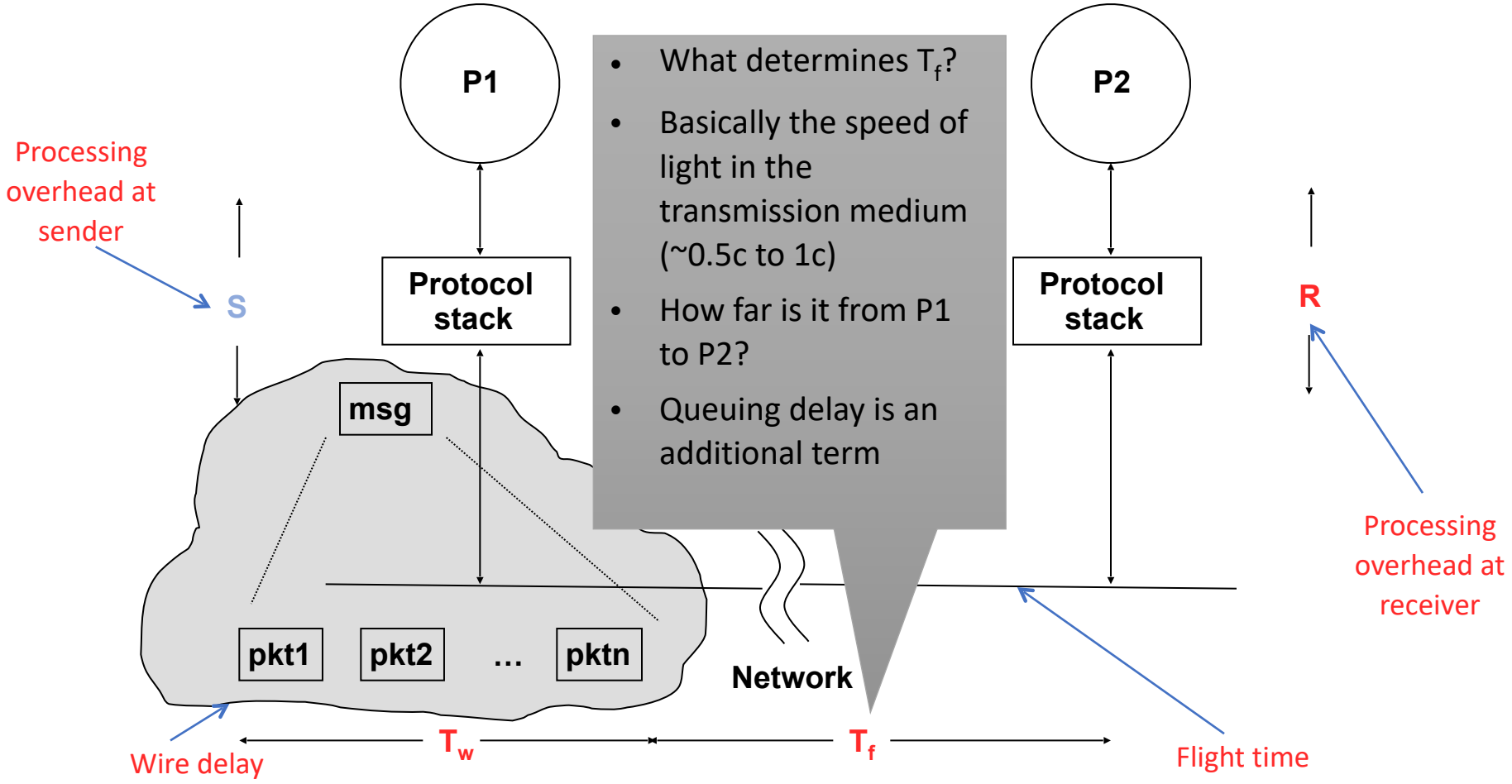
Message throughput = message-size/end-to-end-latency



Performance metrics

Transmission time or end-to-end latency = $S+T_W+T_f+R$

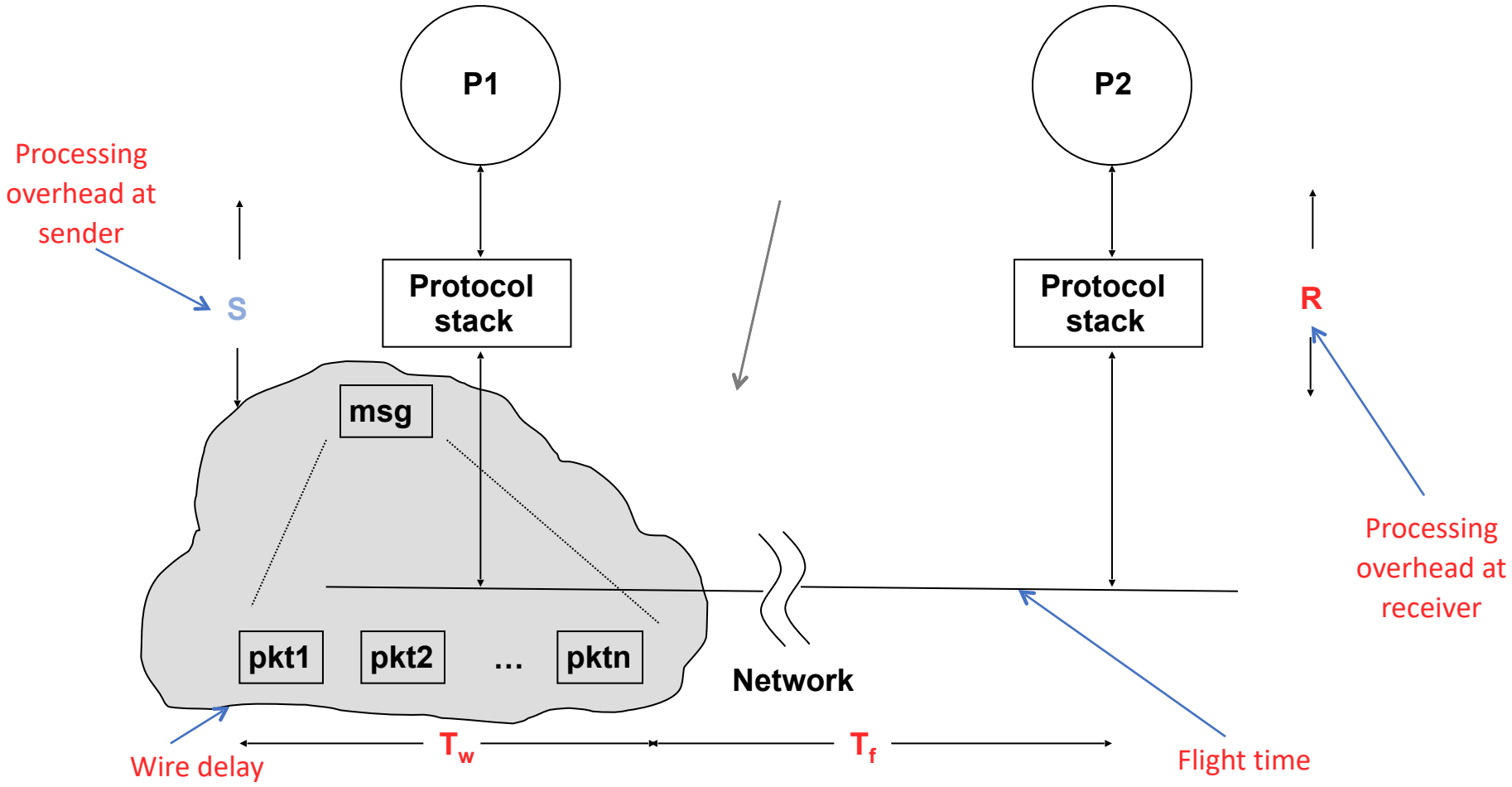
Message throughput = message-size/end-to-end-latency



Performance metrics

Transmission time or end-to-end latency = $S+T_W+T_f+R$

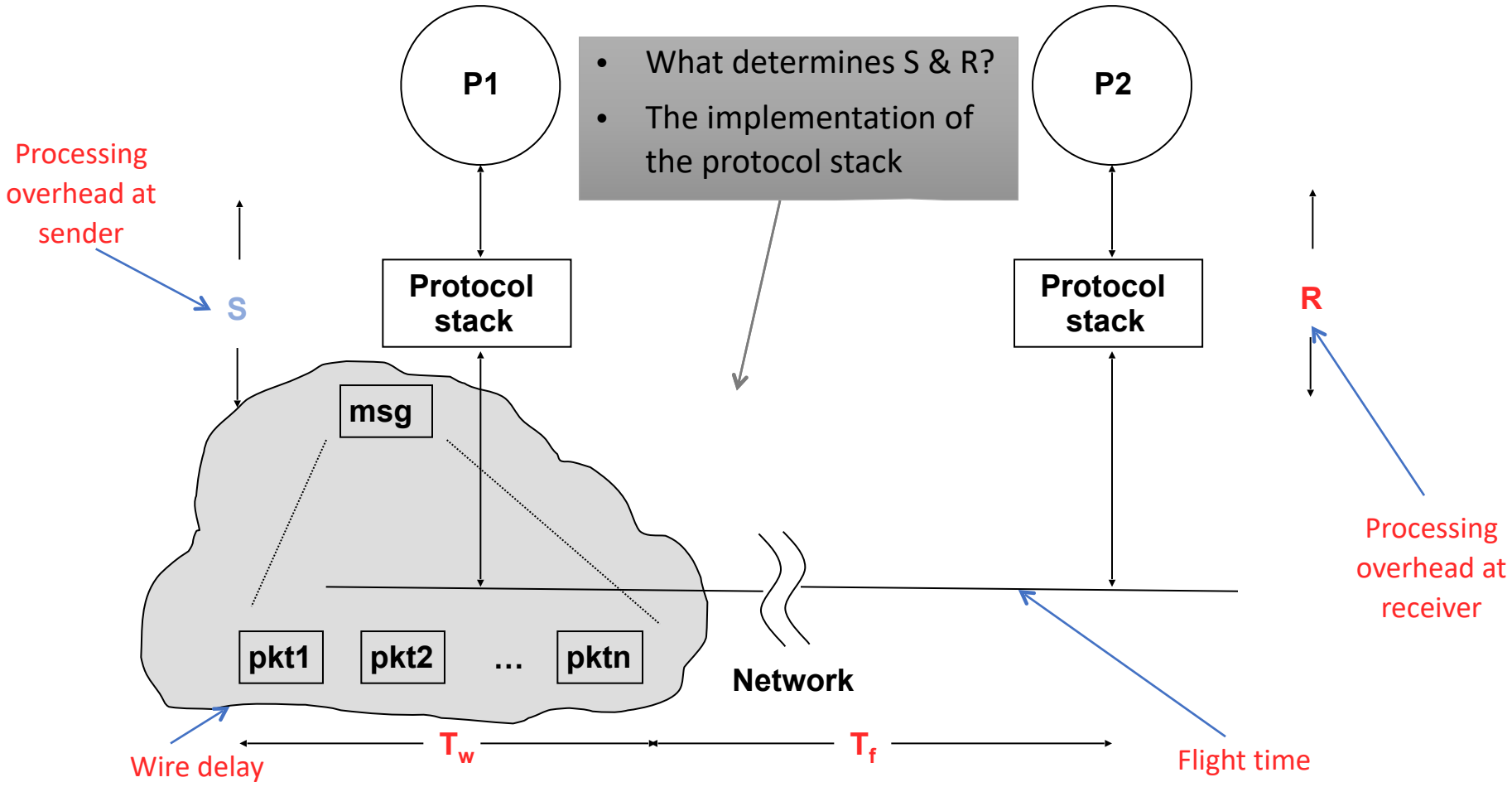
Message throughput = message-size/end-to-end-latency



Performance metrics

Transmission time or end-to-end latency = $S+T_W+T_f+R$

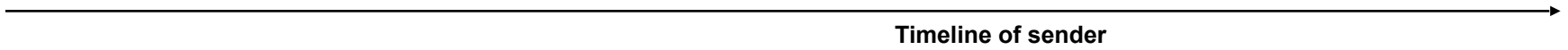
Message throughput = message-size/end-to-end-latency



Timeline example

100ms latency (flight time)
10 packets
Window size = 5

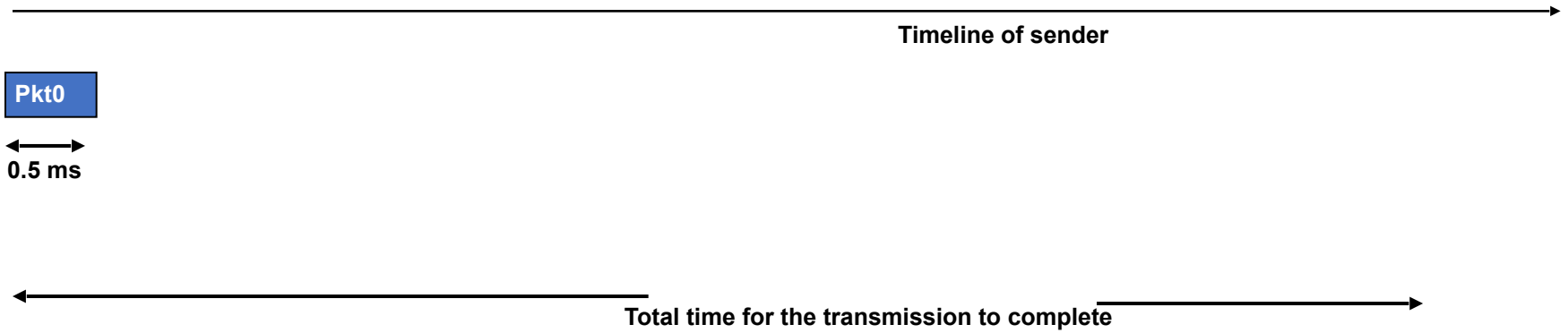
Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



Timeline example

100ms latency (flight time)
10 packets
Window size = 5

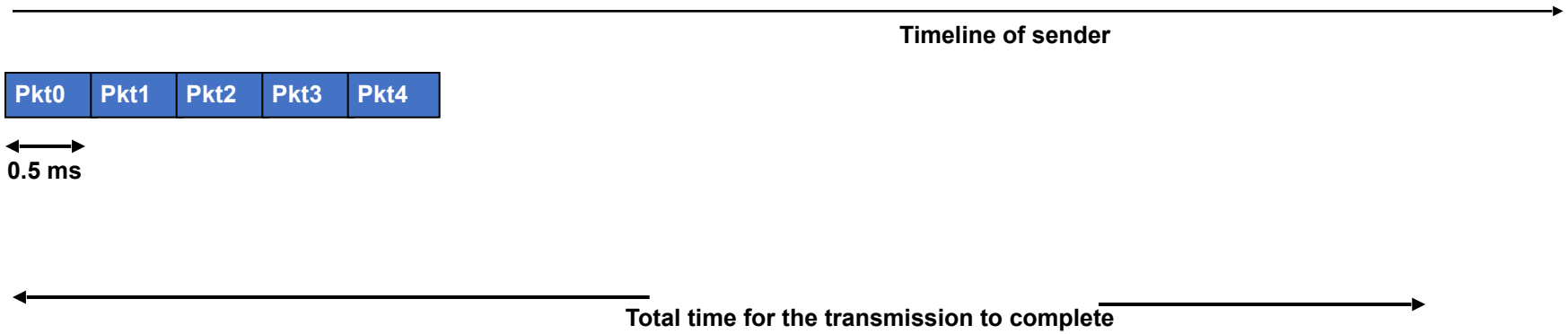
Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



Timeline example

100ms latency (flight time)
10 packets
Window size = 5

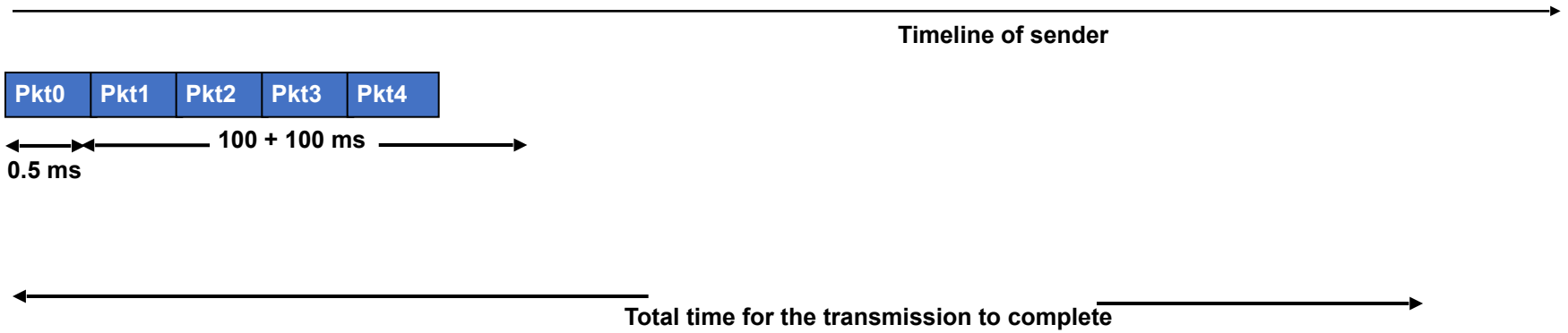
Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



Timeline example

100ms latency (flight time)
10 packets
Window size = 5

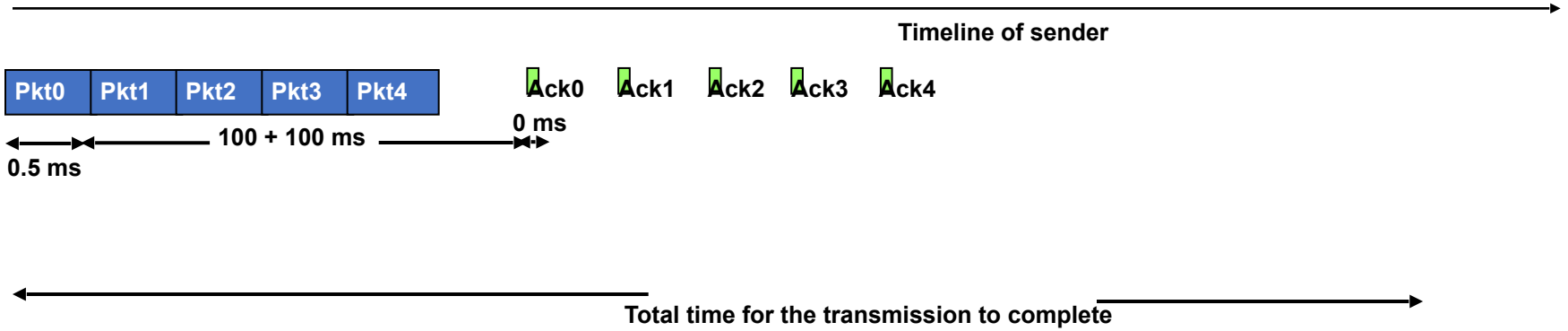
Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



Timeline example

100ms latency (flight time)
10 packets
Window size = 5

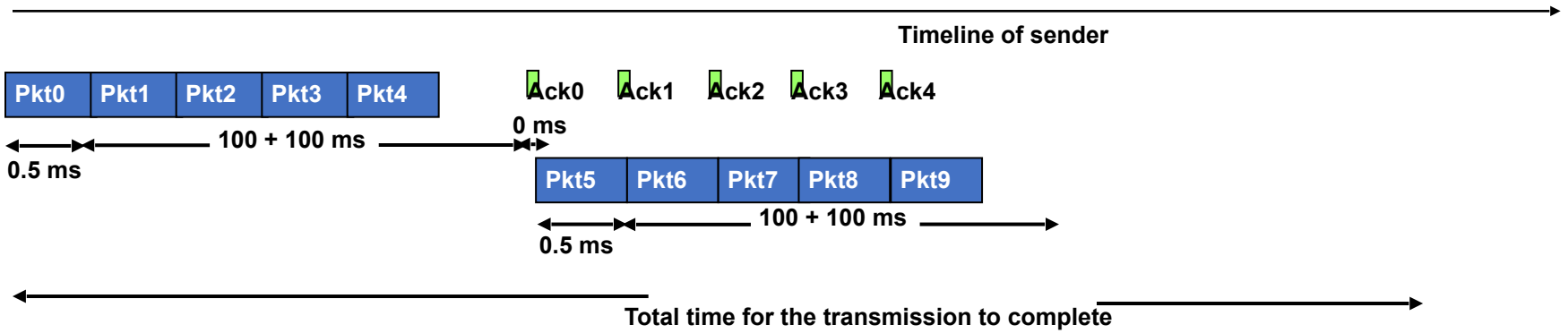
Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



Timeline example

100ms latency (flight time)
10 packets
Window size = 5

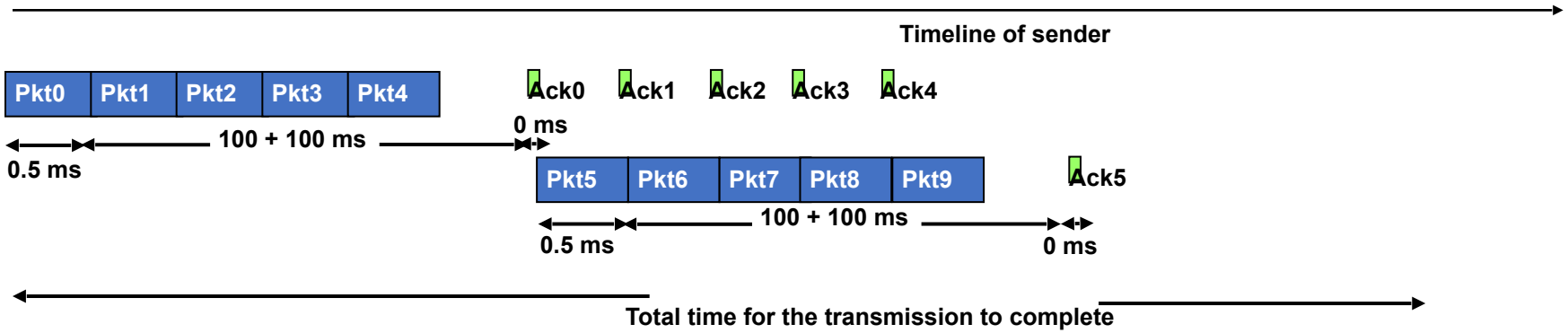
Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



Timeline example

100ms latency (flight time)
10 packets
Window size = 5

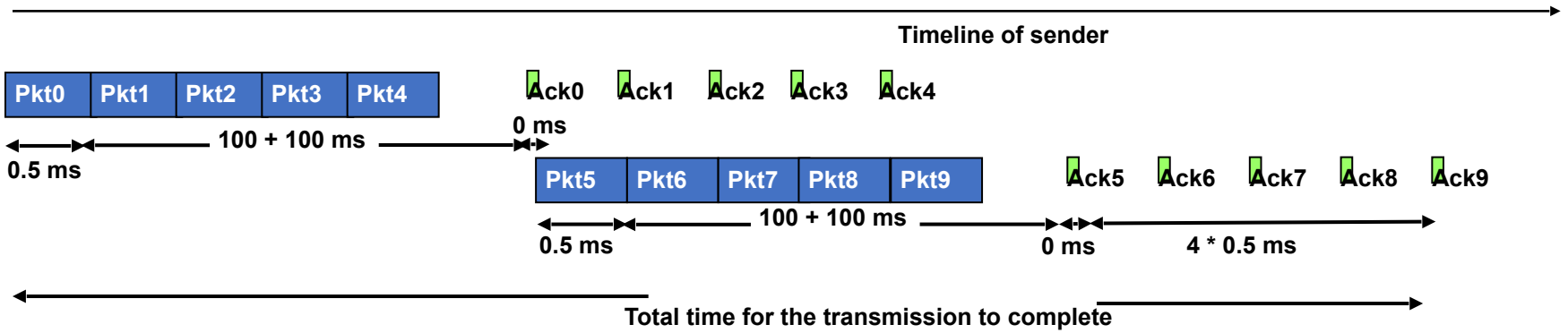
Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



Timeline example

100ms latency (flight time)
10 packets
Window size = 5

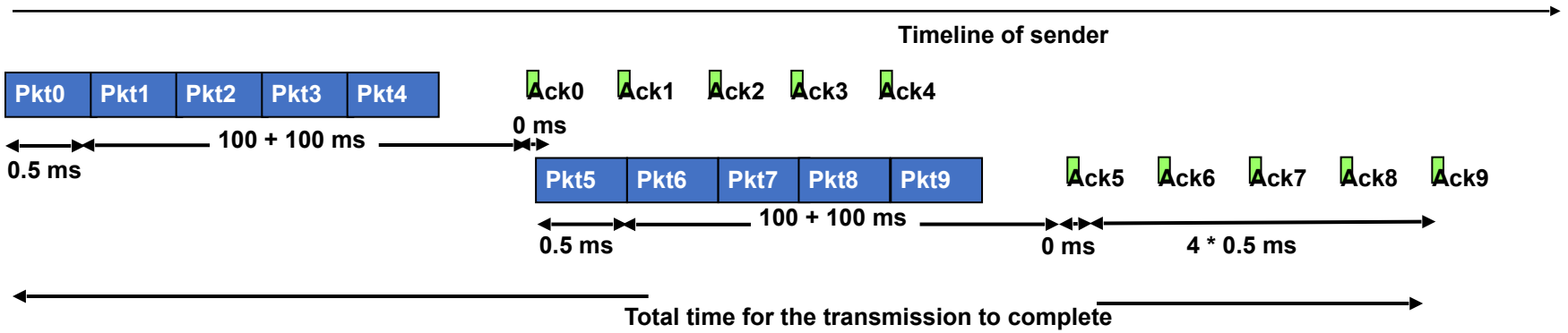
Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



Timeline example

100ms latency (flight time)
10 packets
Window size = 5

Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0

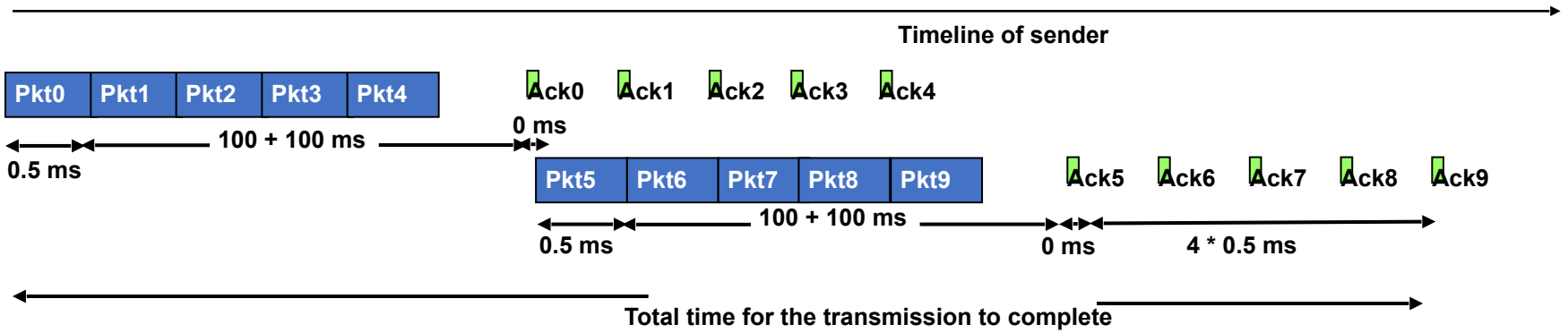


End-to-end latency for Pkt0
$= S + T_w + T_f + R$

Timeline example

100ms latency (flight time)
10 packets
Window size = 5

Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



End-to-end latency for Pkt0

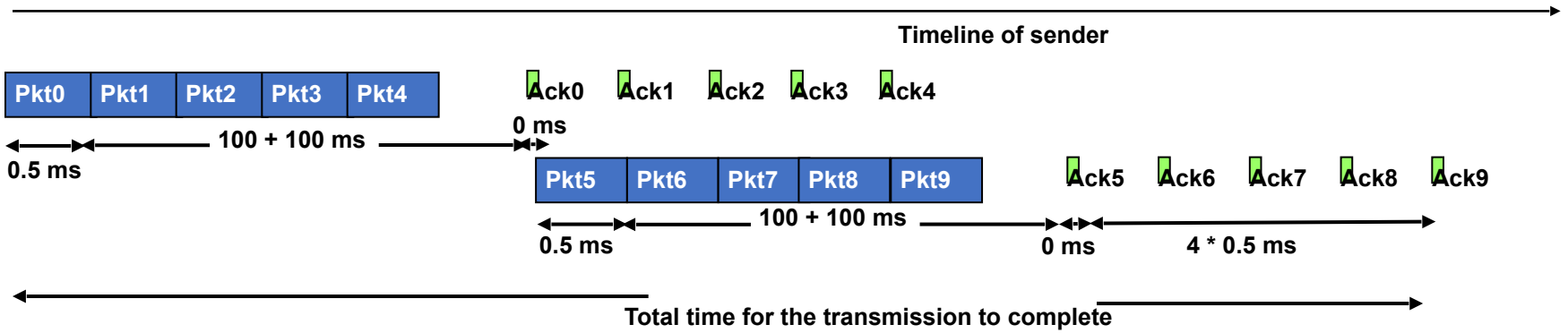
$= S + T_w + T_f + R$

$0 + 0.5 + 100 + 0$

Timeline example

100ms latency (flight time)
10 packets
Window size = 5

Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



End-to-end latency for Pkt0

$= S + T_w + T_f + R$

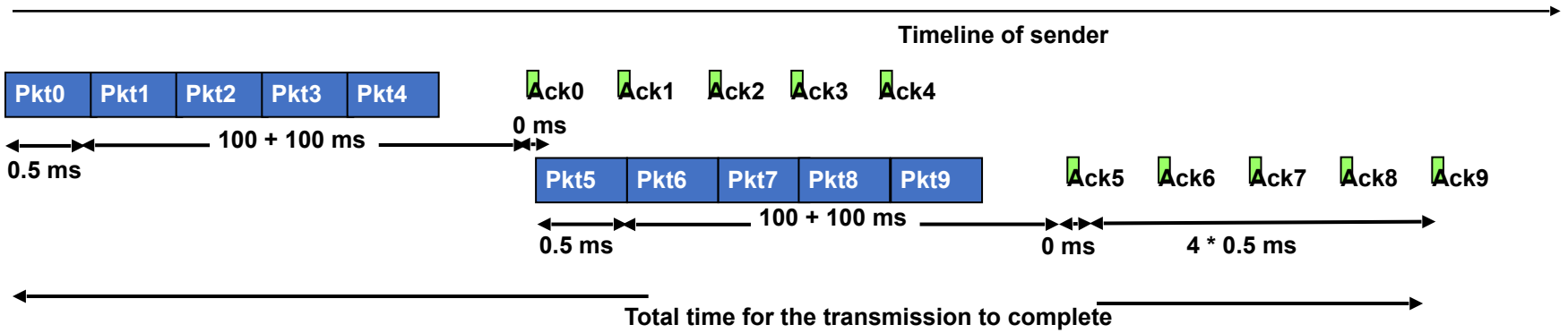
$0 + 0.5 + 100 + 0$

100.5 ms

Timeline example

100ms latency (flight time)
10 packets
Window size = 5

Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



End-to-end latency for Pkt0

$= S + T_w + T_f + R$

$0 + 0.5 + 100 + 0$

100.5 ms

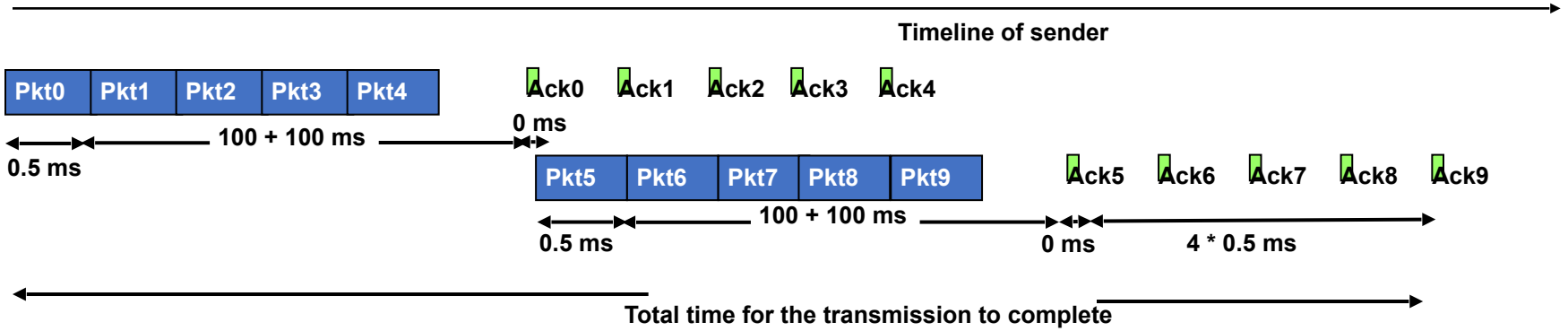
End-to-end latency for Ack0

$= S + T_w + T_f + R$

Timeline example

100ms latency (flight time)
10 packets
Window size = 5

Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



End-to-end latency for Pkt0

$= S + T_w + T_f + R$

$0 + 0.5 + 100 + 0$

100.5 ms

End-to-end latency for Ack0

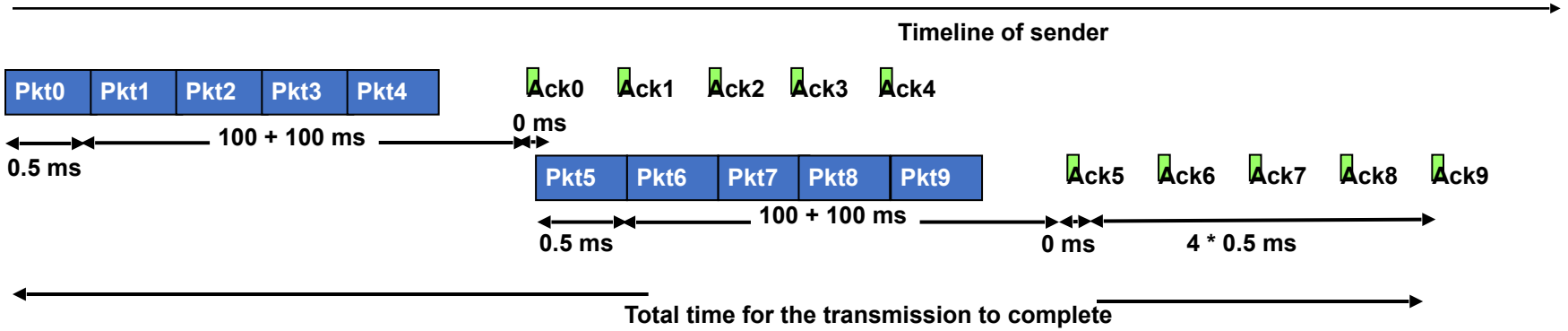
$= S + T_w + T_f + R$

$0 + 0.0 + 100 + 0$

Timeline example

100ms latency (flight time)
10 packets
Window size = 5

Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



End-to-end latency for Pkt0

$= S + T_w + T_f + R$

$0 + 0.5 + 100 + 0$

100.5 ms

End-to-end latency for Ack0

$= S + T_w + T_f + R$

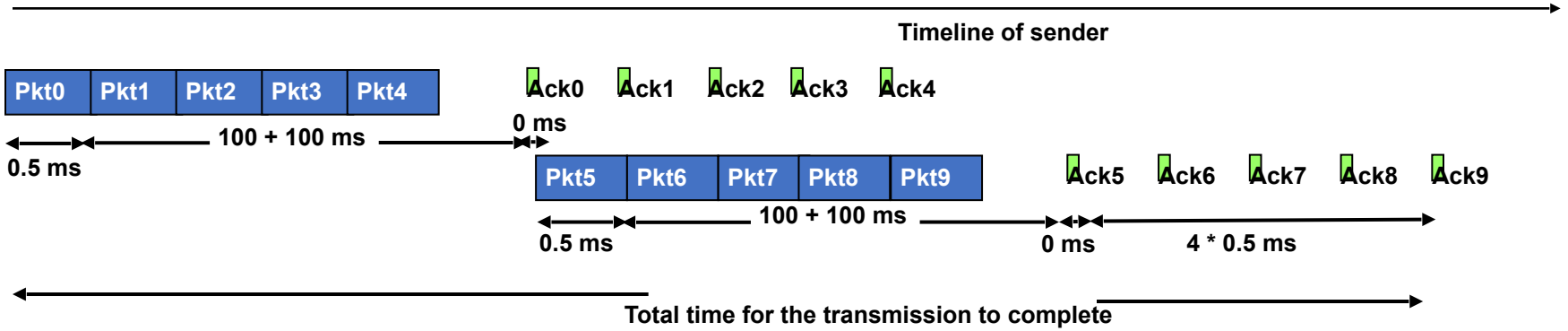
$0 + 0.0 + 100 + 0$

100.0 ms

Timeline example

100ms latency (flight time)
10 packets
Window size = 5

Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



End-to-end latency for Pkt0
= $S + T_w + T_f + R$
0 + 0.5 + 100 + 0
100.5 ms

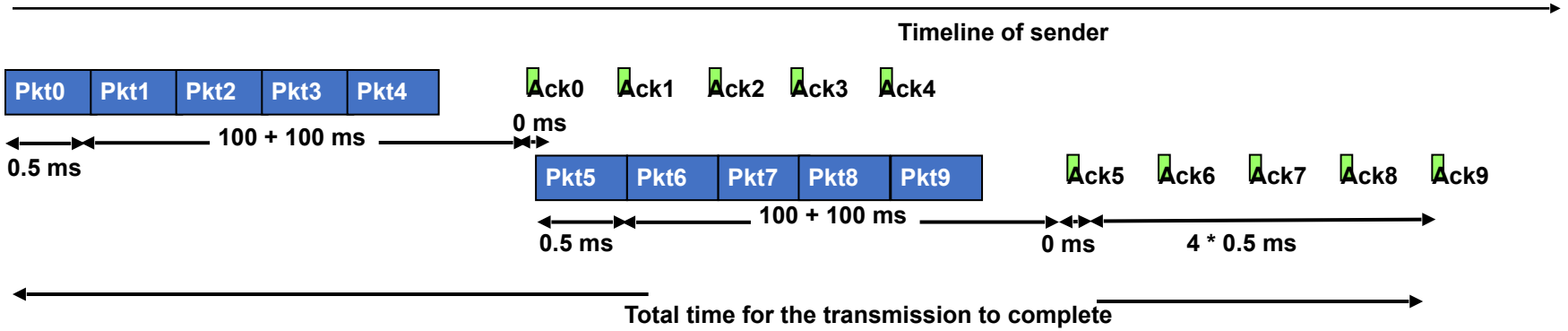
End-to-end latency for Ack0
= $S + T_w + T_f + R$
0 + 0.0 + 100 + 0
100.0 ms

Total transmission time

Timeline example

100ms latency (flight time)
10 packets
Window size = 5

Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



End-to-end latency for Pkt0
= $S + T_w + T_f + R$
0 + 0.5 + 100 + 0
100.5 ms

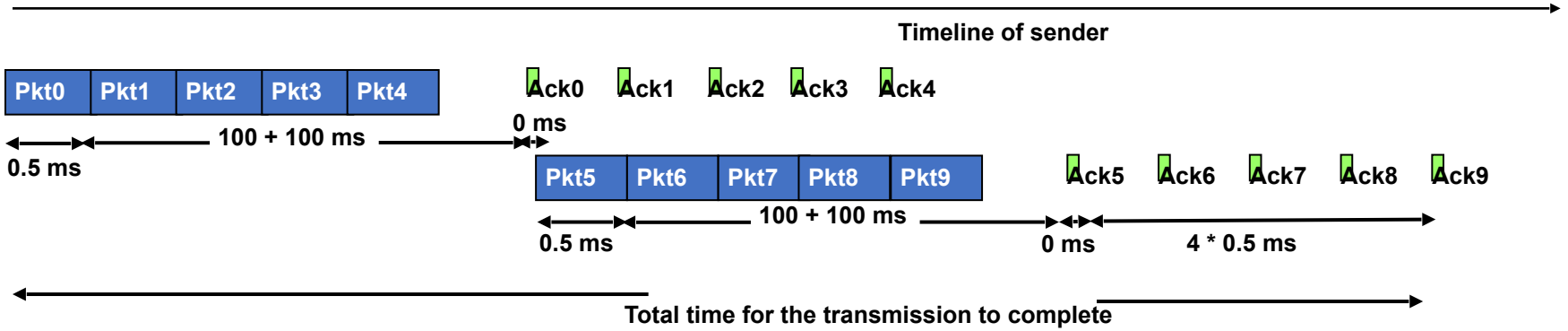
End-to-end latency for Ack0
= $S + T_w + T_f + R$
0 + 0.0 + 100 + 0
100.0 ms

Total transmission time
= 0.5 + 100 + 100 + 0 + 0.5 + 100 + 100 + 0.5 + 0.5 + 0.5 + 0.5

Timeline example

100ms latency (flight time)
10 packets
Window size = 5

Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



End-to-end latency for Pkt0
= $S + T_w + T_f + R$
0 + 0.5 + 100 + 0
100.5 ms

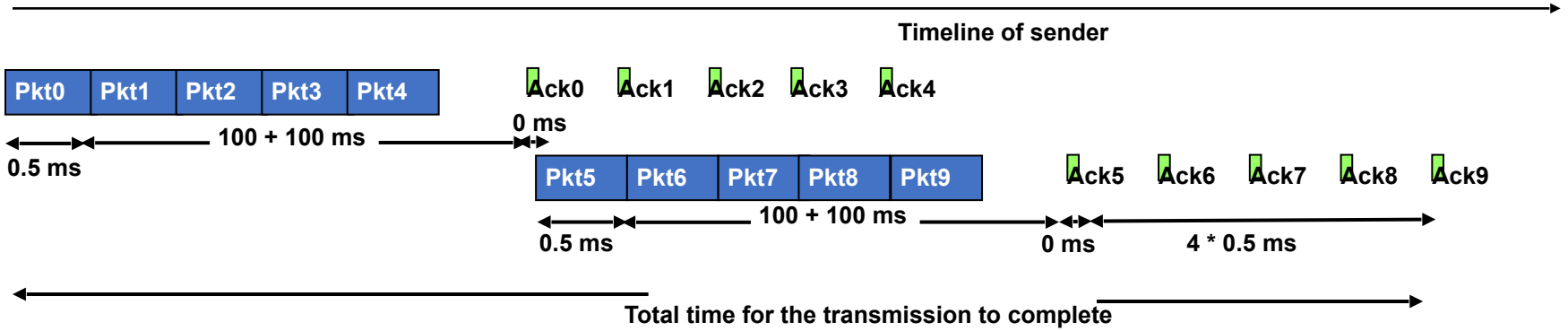
End-to-end latency for Ack0
= $S + T_w + T_f + R$
0 + 0.0 + 100 + 0
100.0 ms

Total transmission time
= 0.5 + 100 + 100 + 0 + 0.5 + 100 + 100 + 0.5 + 0.5 + 0.5 + 0.5
0.5 + 200 + 0.5 + 200 + 4 * 0.5

Timeline example

100ms latency (flight time)
10 packets
Window size = 5

Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



End-to-end latency for Pkt0
= $S + T_w + T_f + R$
0 + 0.5 + 100 + 0
100.5 ms

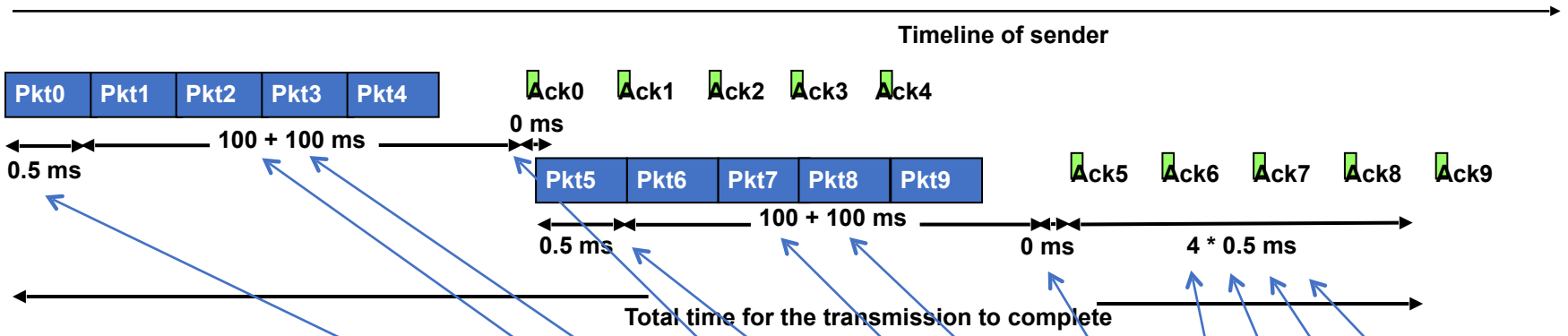
End-to-end latency for Ack0
= $S + T_w + T_f + R$
0 + 0.0 + 100 + 0
100.0 ms

Total transmission time
= 0.5 + 100 + 100 + 0 + 0.5 + 100 + 100 + 0.5 + 0.5 + 0.5 + 0.5
0.5 + 200 + 0.5 + 200 + 4 * 0.5
403 ms

Timeline example

100ms latency
10 packets
Window size = 5

Wire delay (data) = 0.5ms
Wire delay (ack) = ~0
Receiver & Sender overhead = ~0



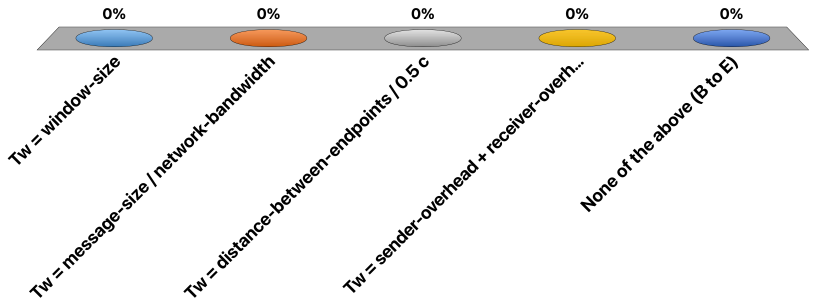
End-to-end latency for Pkt0
= $S + T_w + T_f + R$
0 + 0.5 + 100 + 0
100.5 ms

End-to-end latency for Ack0
= $S + T_w + T_f + R$
0 + 0.0 + 100 + 0
100.0 ms

Total transmission time
= 0.5 + 100 + 100 + 0 + 0.5 + 100 + 100 + 0 + 0.5 + 0.5 + 0.5 + 0.5
0.5 + 200 + 0.5 + 200 + 4 * 0.5
403 ms

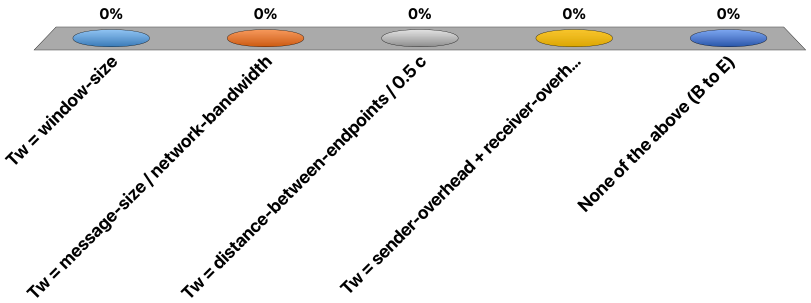
In the expression $S + T_w + T_f + R$

A. T_w = window-size
B. T_w = message-size / network-bandwidth
C. T_w = distance-between-endpoints / 0.5 *c*
D. T_w = sender-overhead + receiver-overhead
E. None of the above (B to E)



In the expression $S + T_w + T_f + R$

A. T_w = window-size
B. T_w = message-size / network-bandwidth
C. T_w = distance-between-endpoints / 0.5 *c* T_f
D. T_w = sender-overhead + receiver-overhead S+R
E. None of the above (B to E)



Acknowledgement example

- A transport protocol uses a window of 10 packets.
- For a message that consists of 101 packets, how many acknowledgement packets does the destination generate?

Acknowledgement example

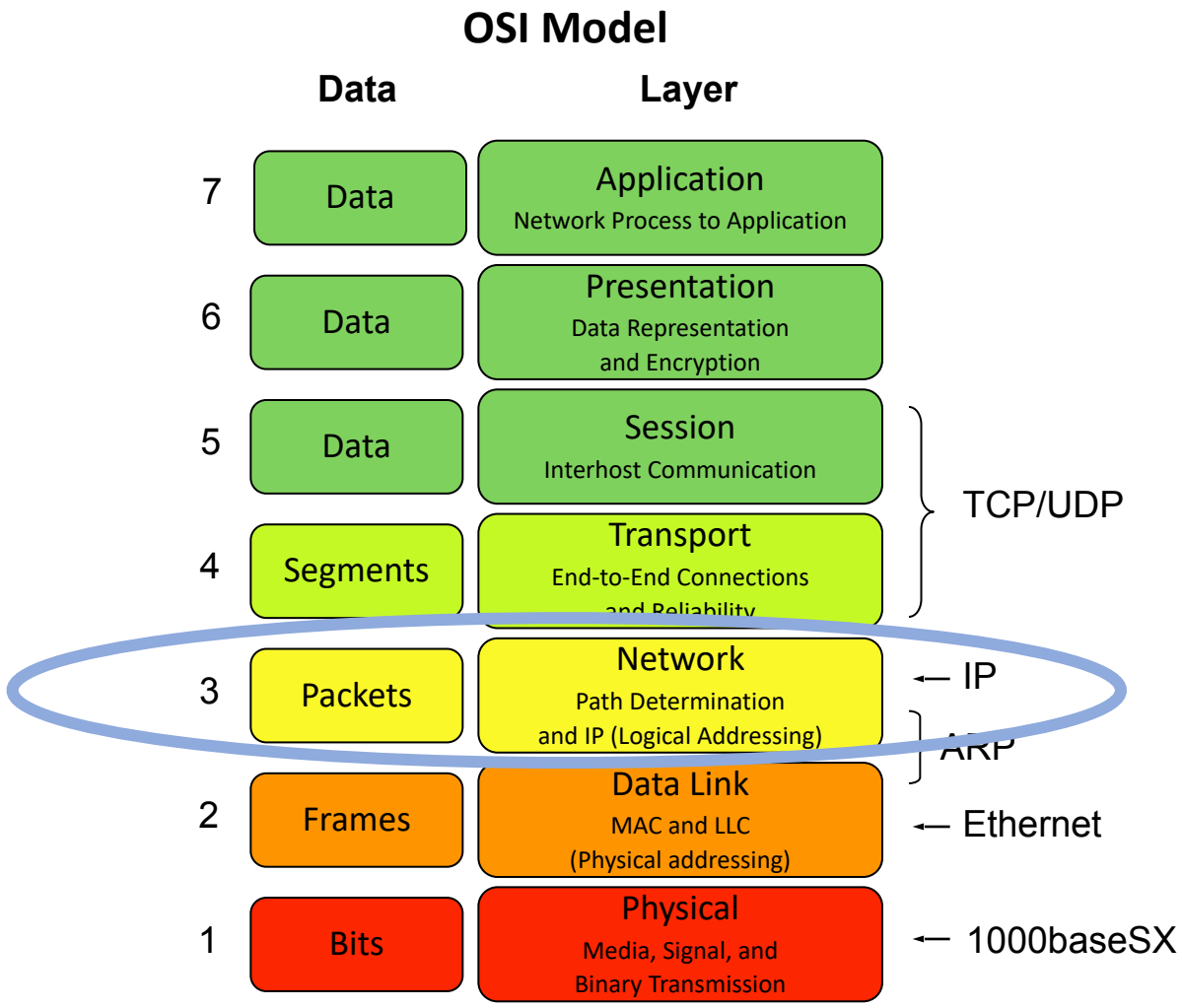
- A transport protocol uses a window of 10 packets.
- For a message that consists of 101 packets, how many acknowledgement packets does the destination generate?
- Assume no packets are lost.

Acknowledgement example

- A transport protocol uses a window of 10 packets.
- For a message that consists of 101 packets, how many acknowledgement packets does the destination generate?
- Assume no packets are lost.
- 101

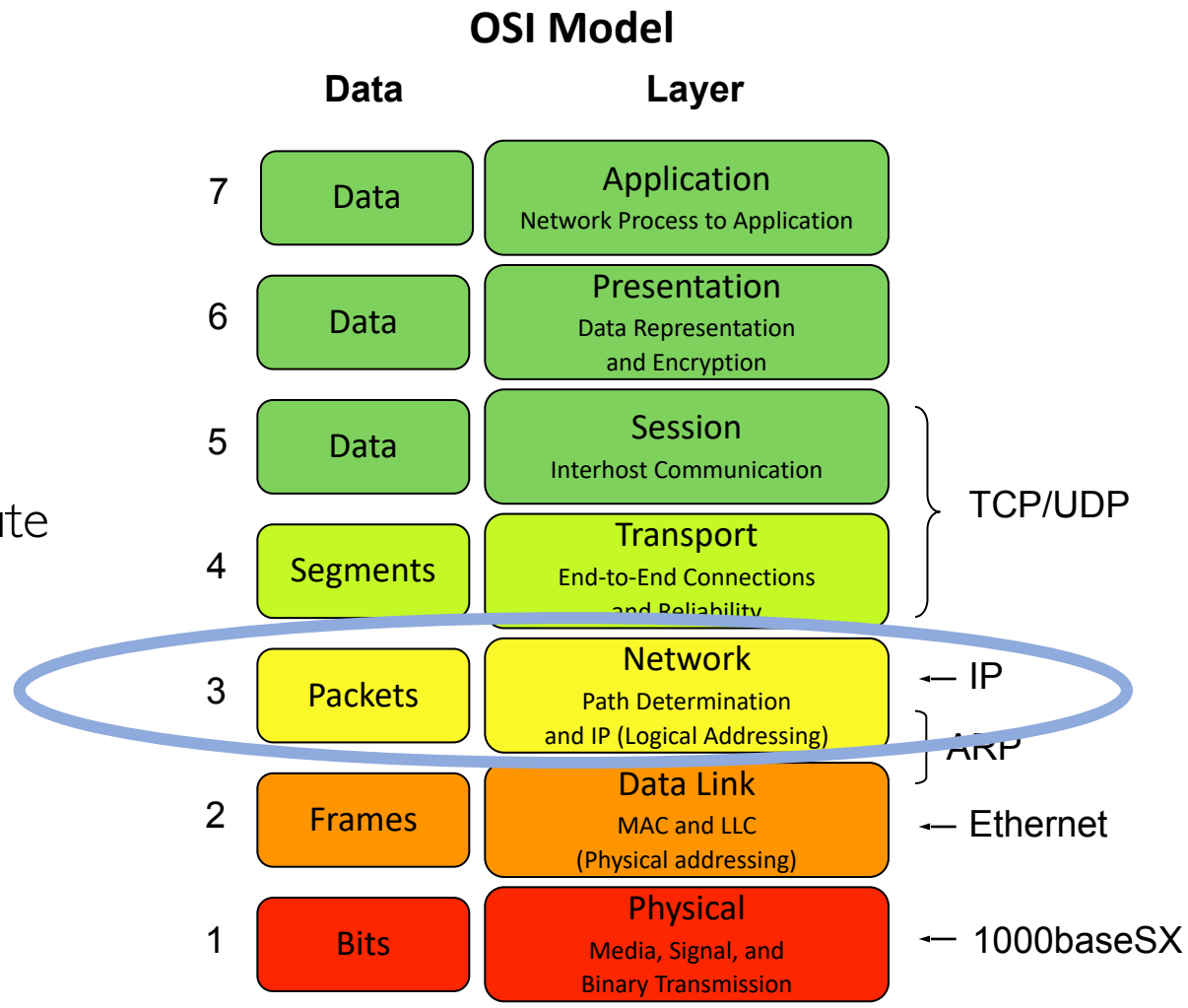
Back to the network layer

- Why are we back here?



Back to the network layer

- Why are we back here?
- Recall, IP routing happens here
- We didn't answer a question:
  How does a router know where to route (i.e. who sets up the routing table)?

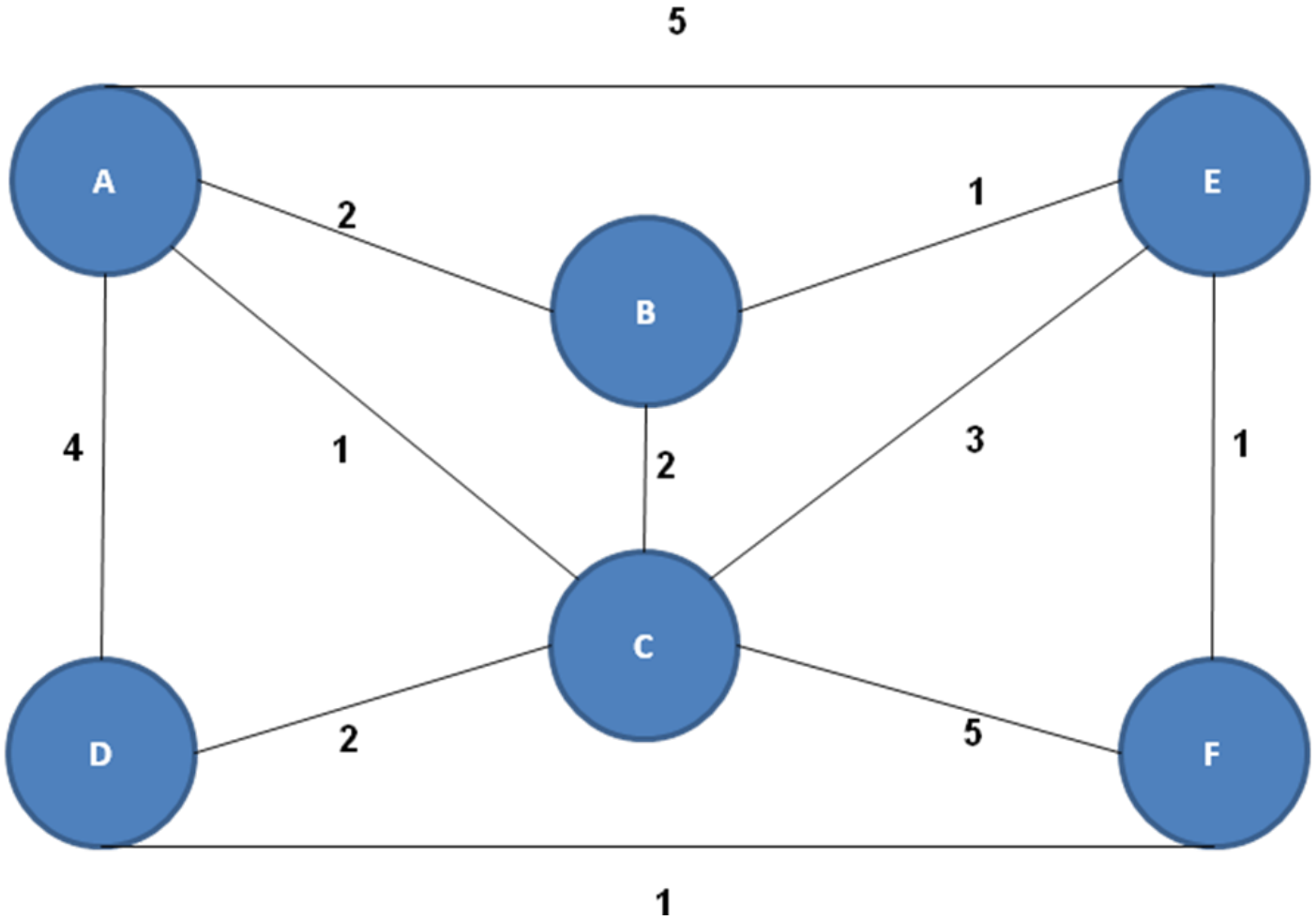


Network layer

- Routing information protocols
  - Link state protocols
  - Distance vector protocols
  - Hierarchical routing
- Addressing

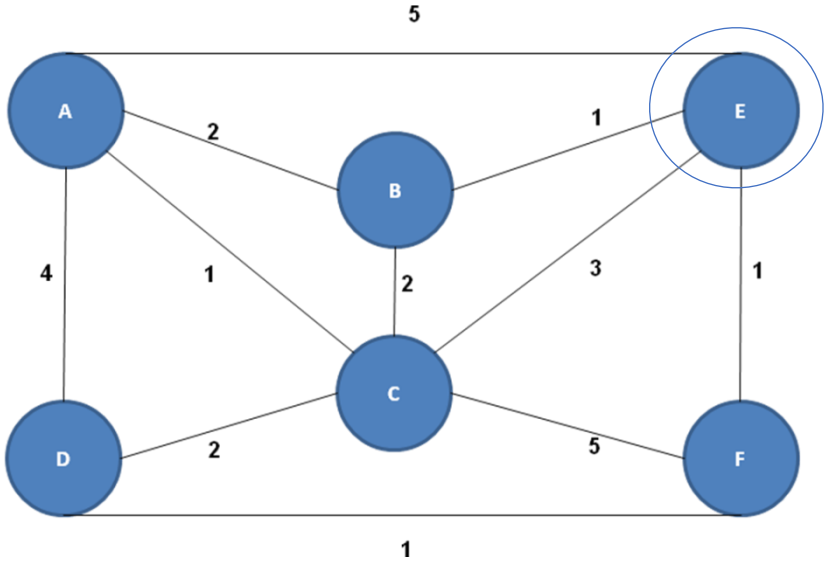
Intuition behind routing algorithms (link state, distance vector)

What's the latency to get to your neighbor?



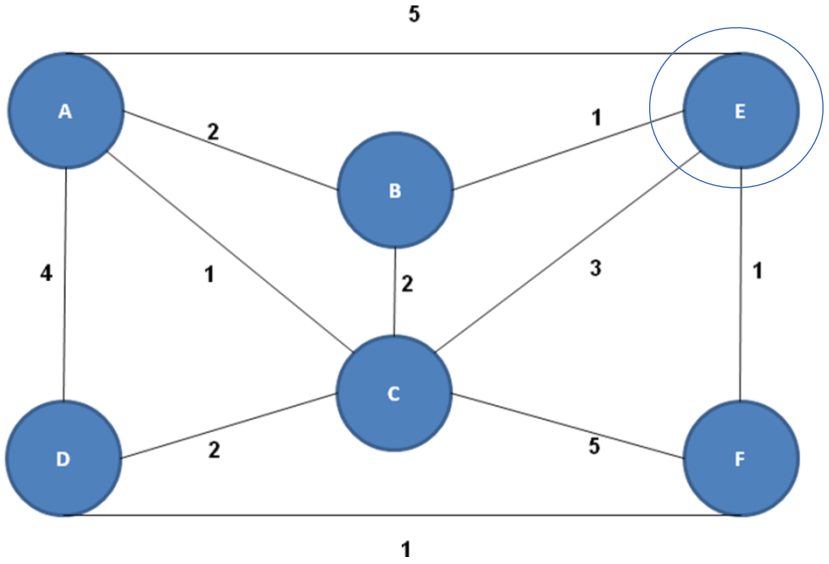
Distance vector in action

- Uses neighbor info, local algorithm (i.e. every node only sees its neighbors)
- Each node knows the cost to each of its neighbors
- Each node sends its idea of routing to each of its neighbors
- Known colloquially as "routing by rumor"
- For every other node, each node determines the lowest cost next-hop
- Algorithm runs simultaneously on every node
- How does it work?



Distance vector in action

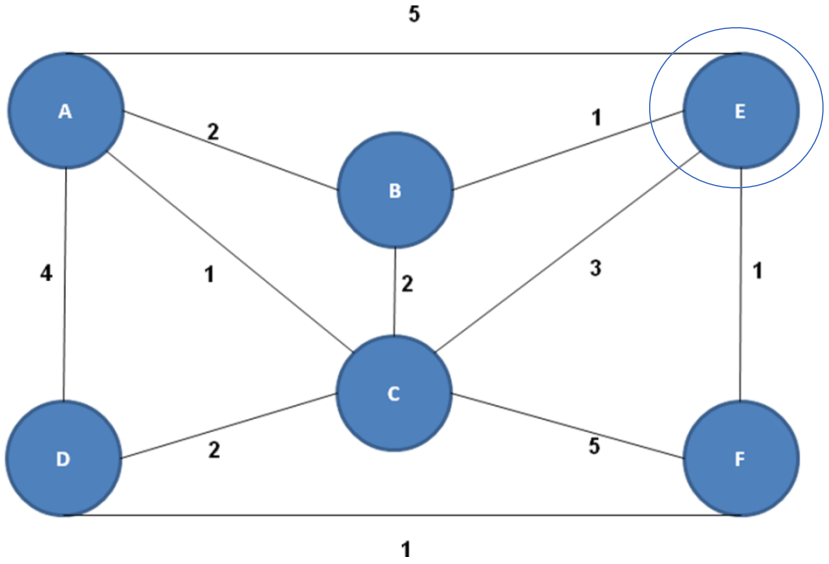
- Uses neighbor info, local algorithm (i.e. every node only sees its neighbors)
- Each node knows the cost to each of its neighbors
- Each node sends its idea of routing to each of its neighbors
- Known colloquially as "routing by rumor"
- For every other node, each node determines the lowest cost next-hop
- Algorithm runs simultaneously on every node
- How does it work?
- Node E knows its costs to its neighbors:



Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	∞/	1/F

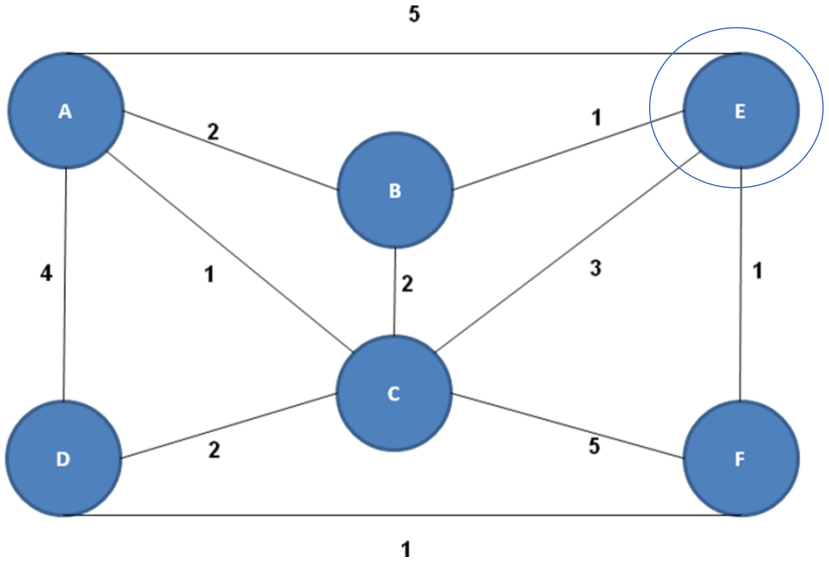
Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	∞/	1/F



Distance vector in action

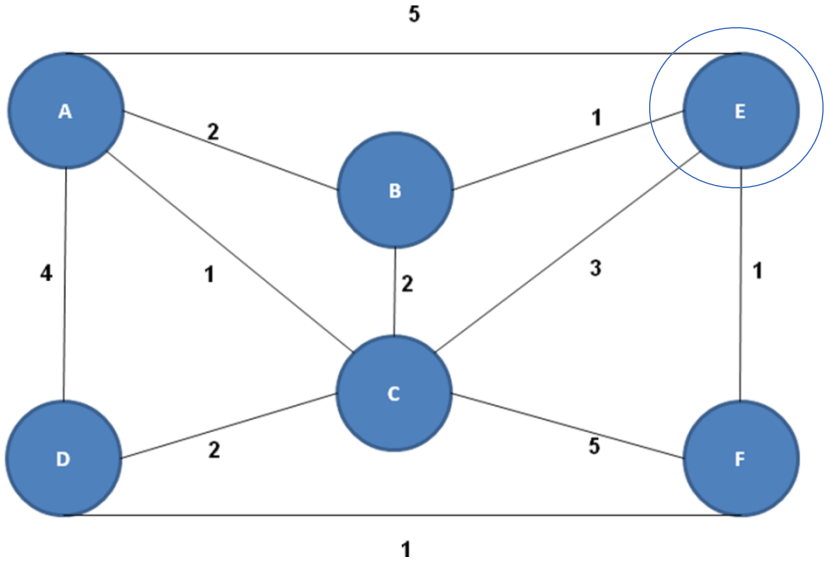
Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	∞/	1/F



- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E

Distance vector in action

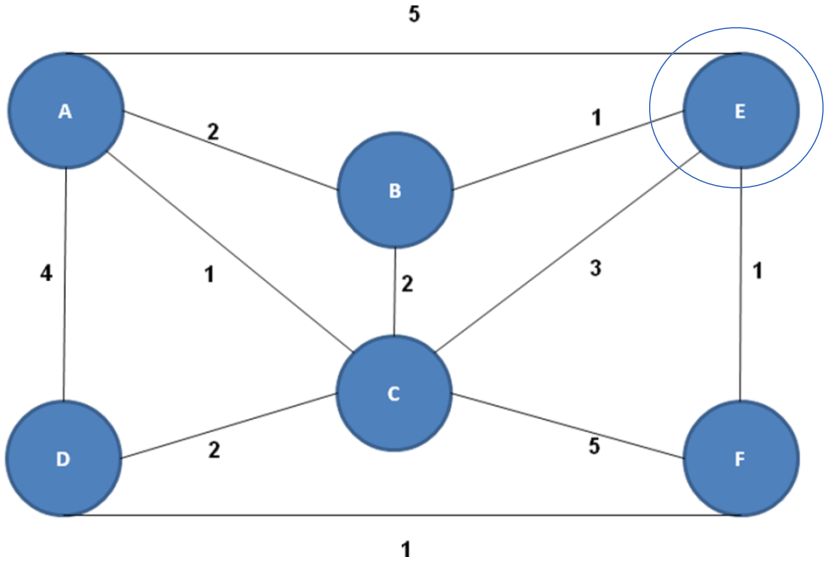
Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	∞/	1/F



- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.

Distance vector in action

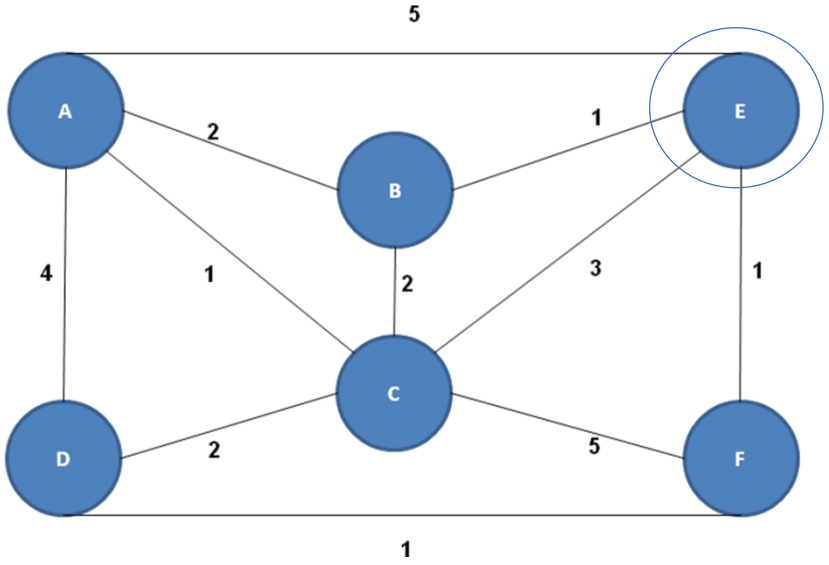
Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	∞/	1/F



- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	∞/	1/F

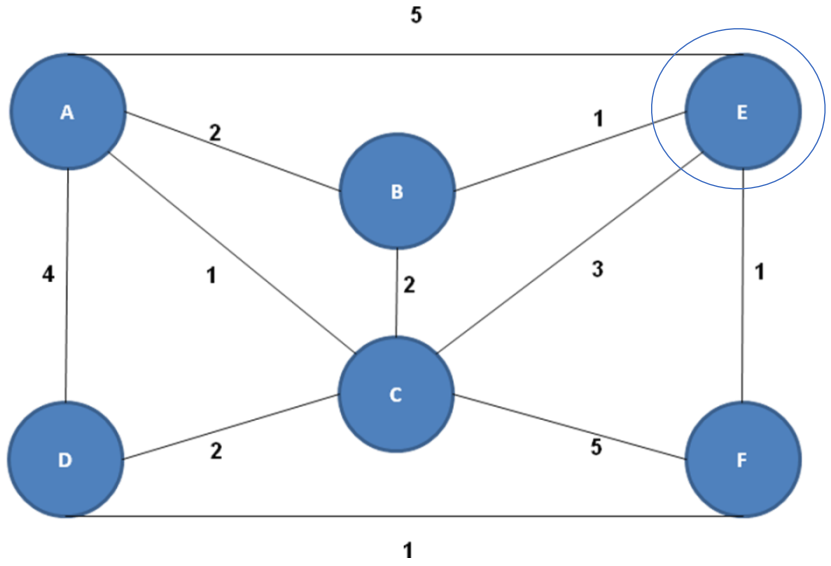


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing node A's message
  - 5+2/B, 5+1/C, 5+4/D, 5+5/F
  - Only 9/D is less
  - We'll set the next hop to A

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	∞/	1/F

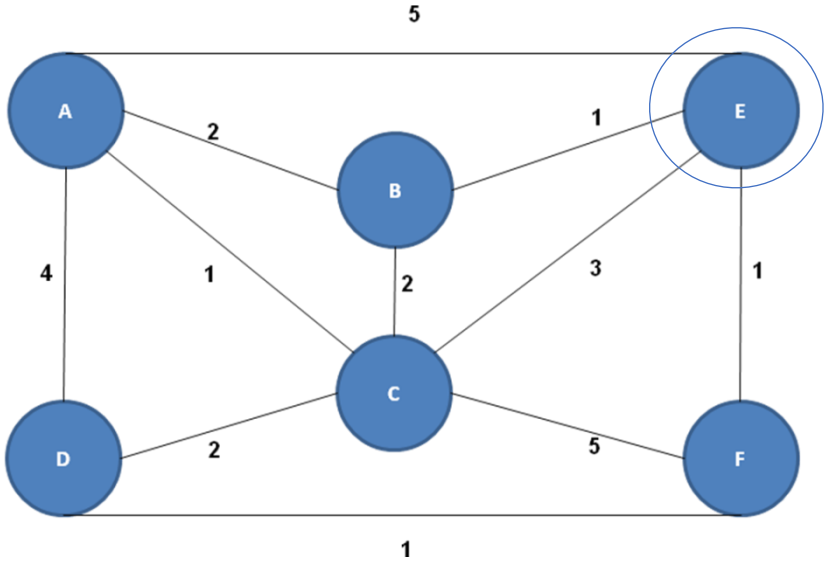


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing node A's message
  - 5+2/B, 5+1/C, 5+4/D, 5+5/F
  - Only 9/D is less
  - We'll set the next hop to A

Distance vector in action

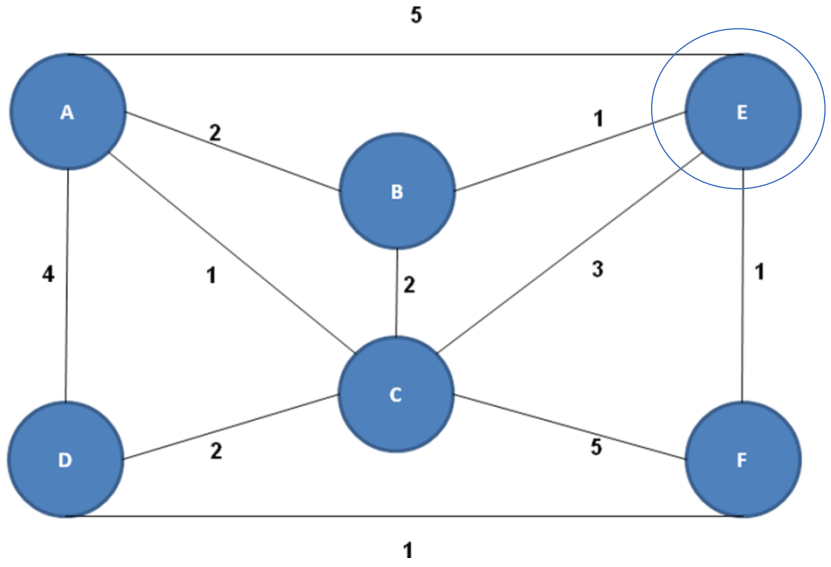
Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	9/A	1/F



- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	9/A	1/F

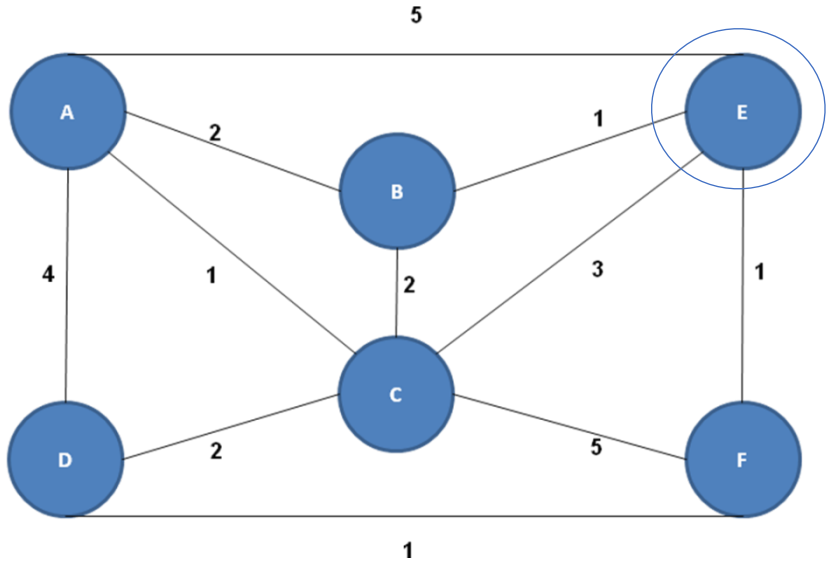


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing Node B's message

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	9/A	1/F

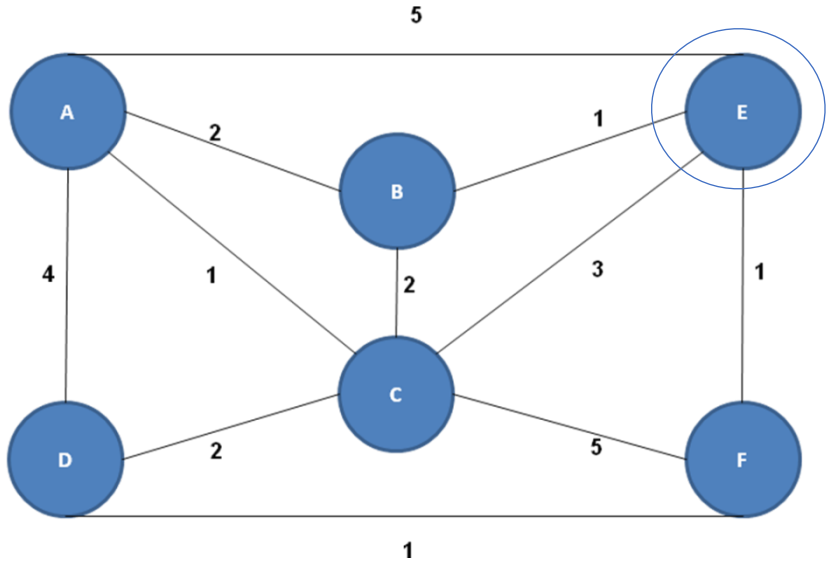


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing Node B's message
  - 1+2/C, 1+2/A, 1+5/F, 1+4/D

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	9/A	1/F

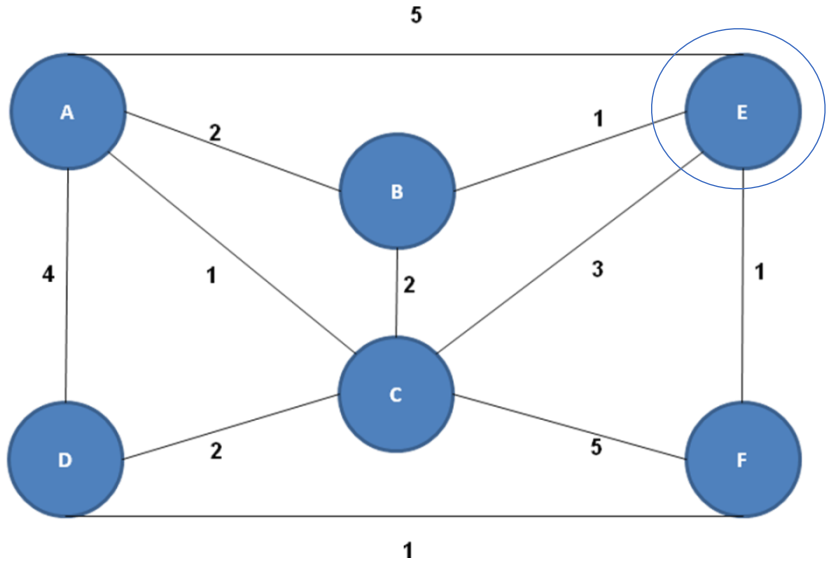


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing Node B's message
  - 1+2/C, 1+2/A, 1+5/F, 1+4/D
  - 3/A and 5/D are lower cost

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	9/A	1/F

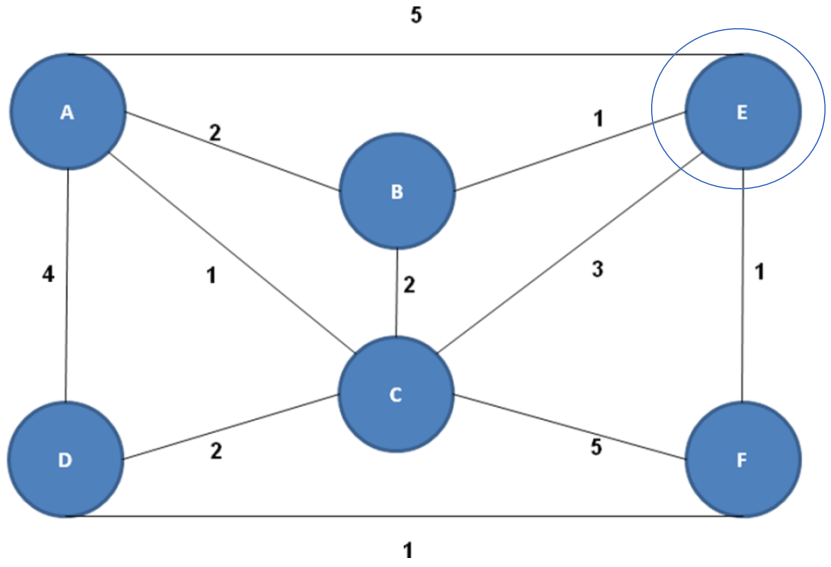


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing Node B's message
  - 1+2/C, 1+2/A, 1+5/F, 1+4/D
  - 3/A and 5/D are lower cost
  - We set their next hop to B

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	9/A	1/F

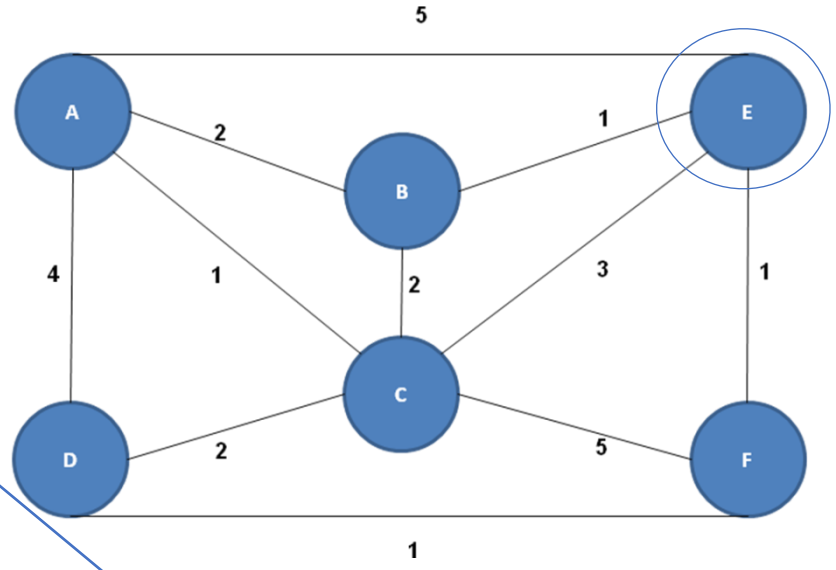


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing Node B's message
  - 1+2/C, 1+2/A, 1+5/F, 1+4/D
  - 3/A and 5/D are lower cost
  - We set their next hop to B

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	5/A	1/B	3/C	9/A	1/F

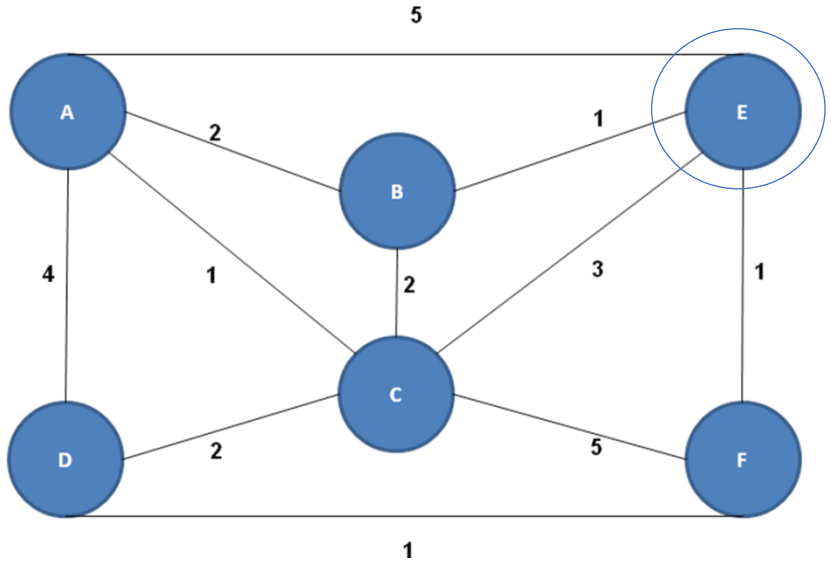


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing Node B's message
  - 1+2/C, 1+2/A, 1+5/F, 1+4/D
  - 3/A and 5/D are lower cost
  - We set their next hop to B

Distance vector in action

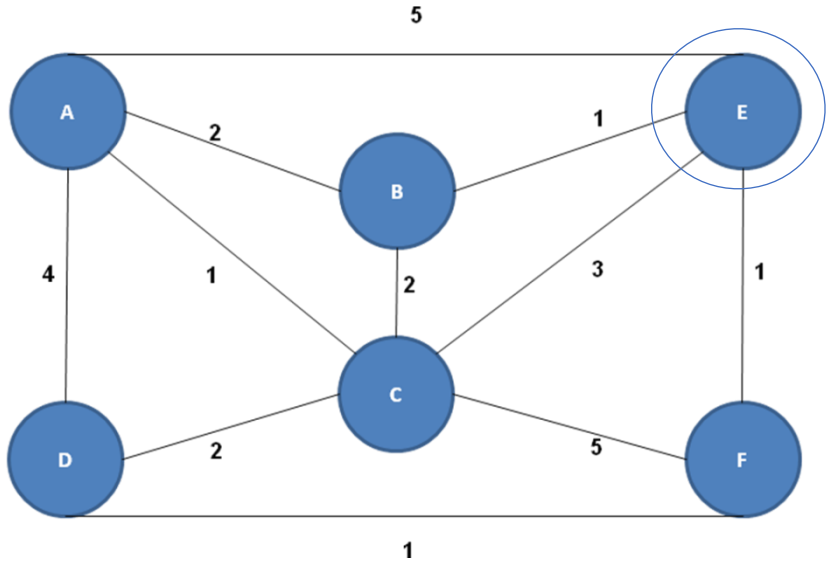
Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	5/B	1/F



- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	5/B	1/F

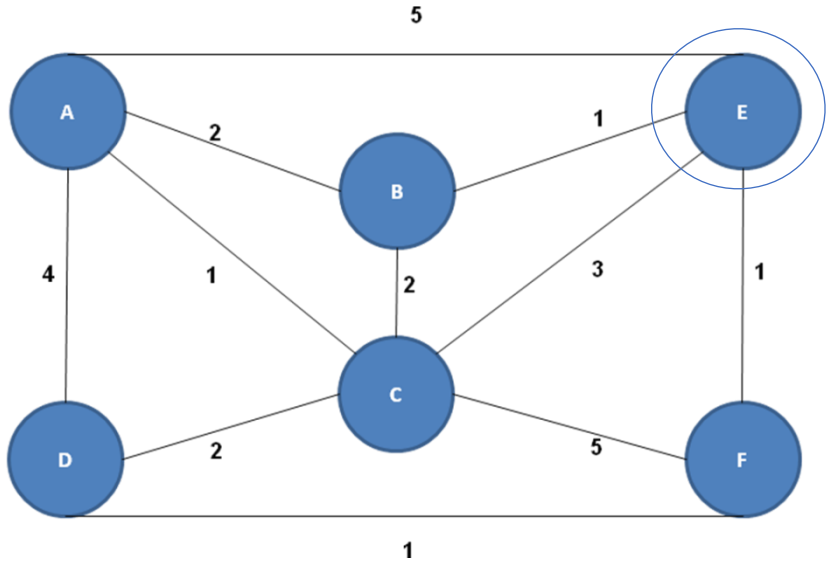


- Reviewing Node C's message

- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	5/B	1/F

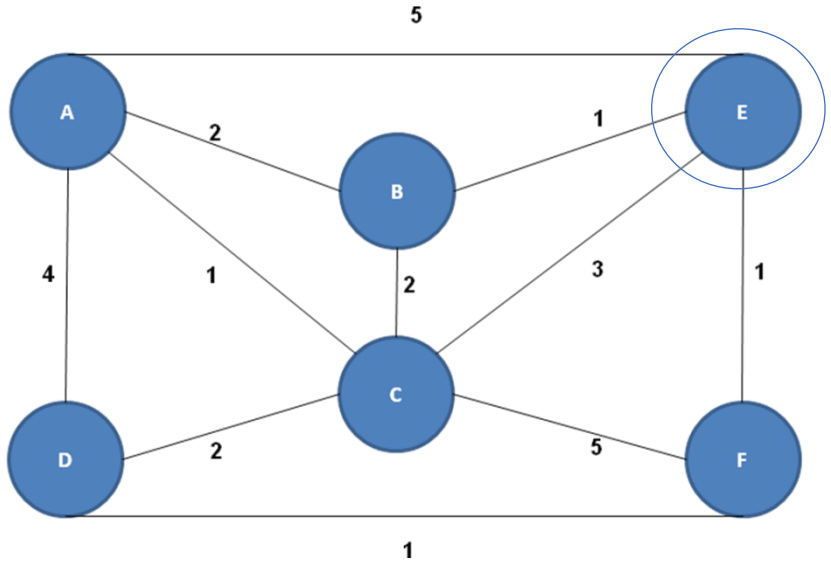


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing Node C's message
  - 3+1/A, 3+2/D, 3+5/F, 3+2/B

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	5/B	1/F

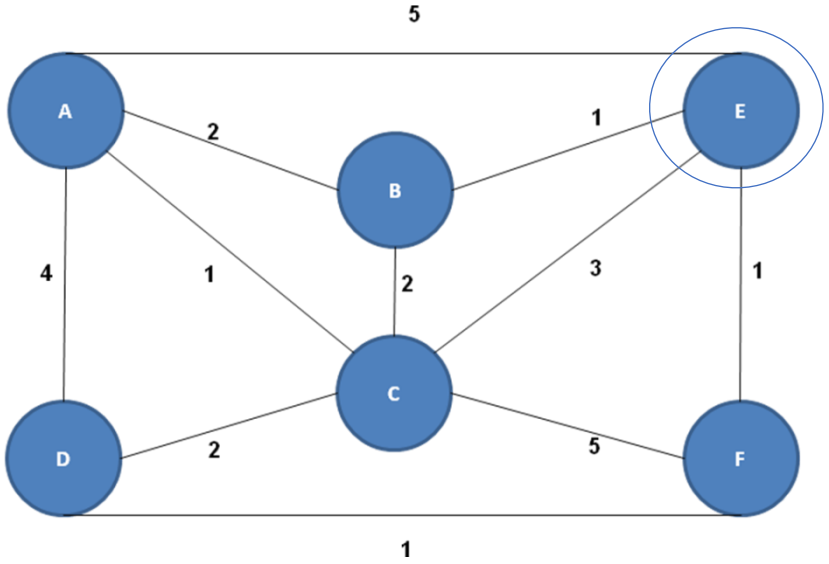


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing Node C's message
  - 3+1/A, 3+2/D, 3+5/F, 3+2/B
  - None are lower cost

Distance vector in action

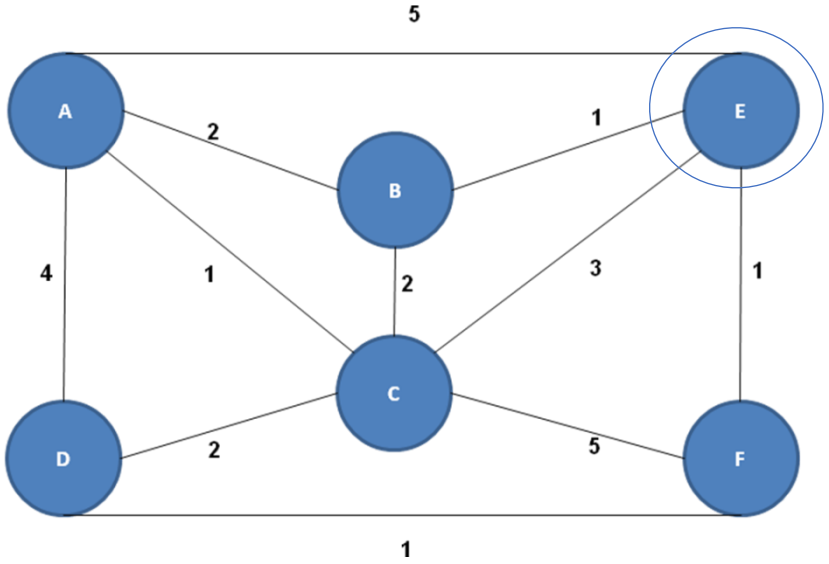
Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	5/B	1/F



- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	5/B	1/F

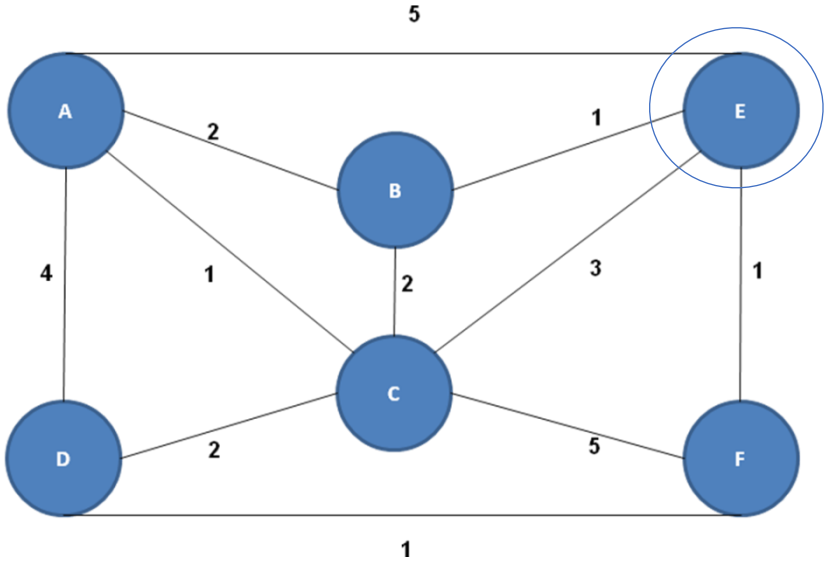


- Reviewing Node F's message

- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	5/B	1/F

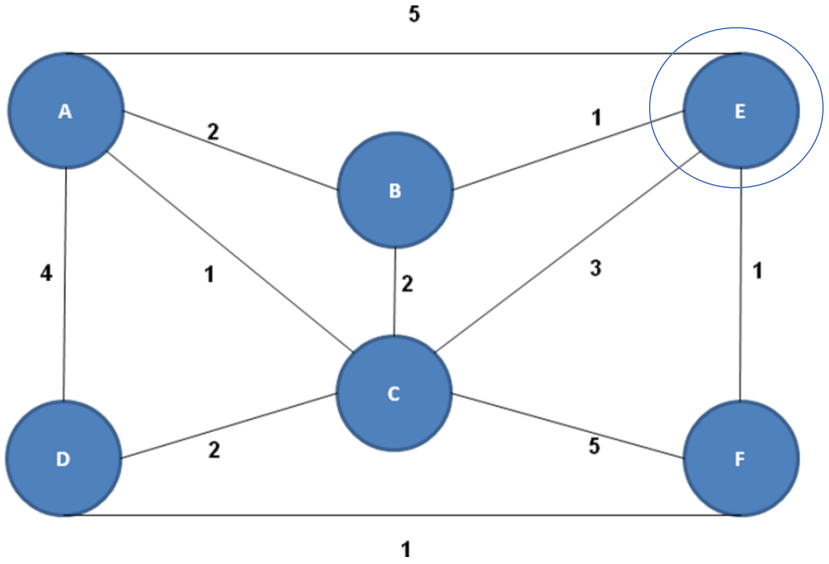


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing Node F's message
  - 1+5C, 1+1/D, 1+2/B, 1+4/A

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	5/B	1/F

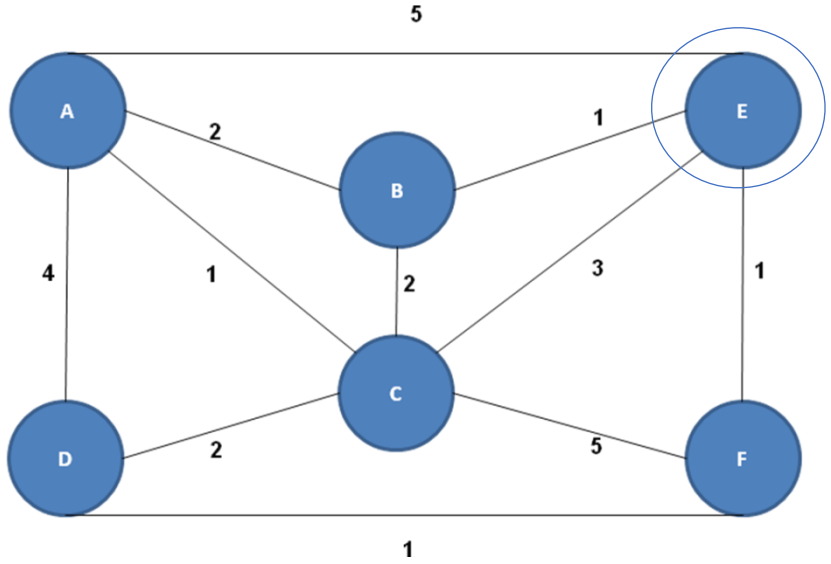


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing Node F's message
  - 1+5C, 1+1/D, 1+2/B, 1+4/A
  - 2/D is lower cost

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	5/B	1/F

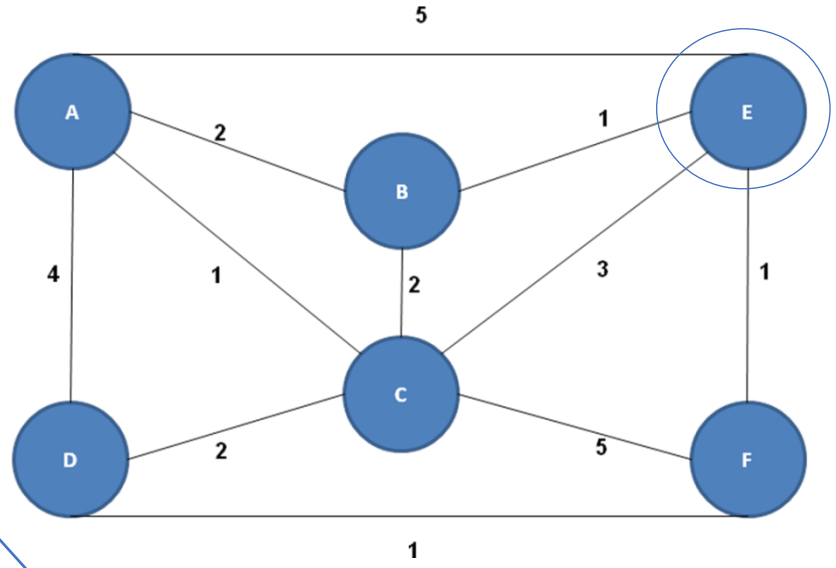


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing Node F's message
  - 1+5C, 1+1/D, 1+2/B, 1+4/A
  - 2/D is lower cost
  - We'll set the next hop to F

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	5/B	1/F

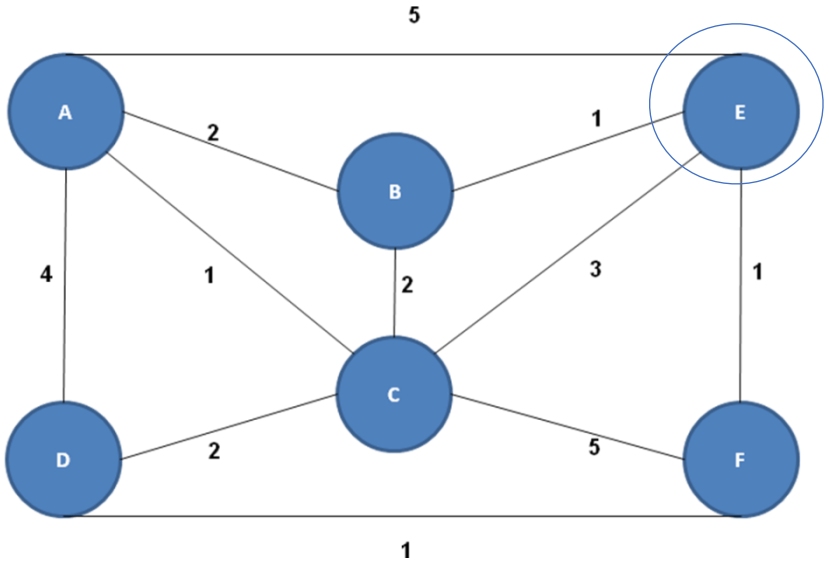


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Reviewing Node F's message
  - 1+5C, 1+1/D, 1+2/B, 1+4/A
  - 2/D is lower cost
  - We'll set the next hop to F

Distance vector in action

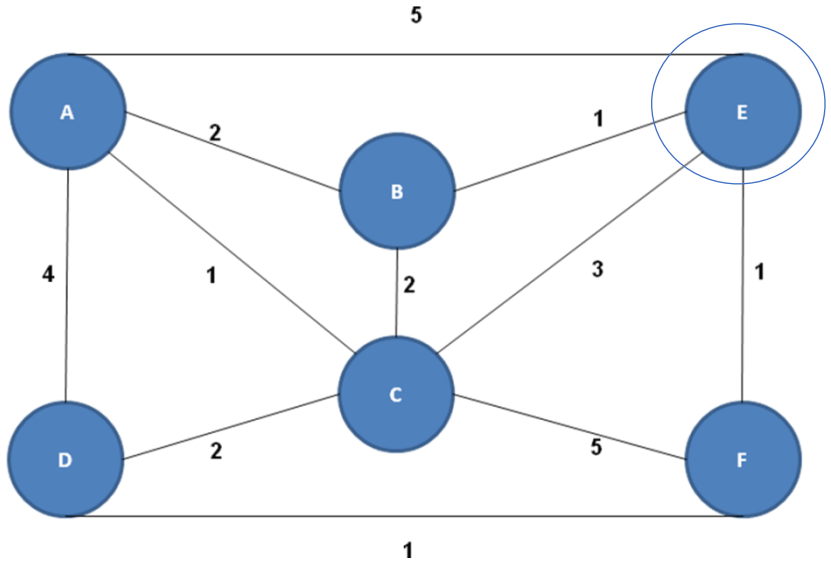
Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	2/F	1/F



- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	2/F	1/F

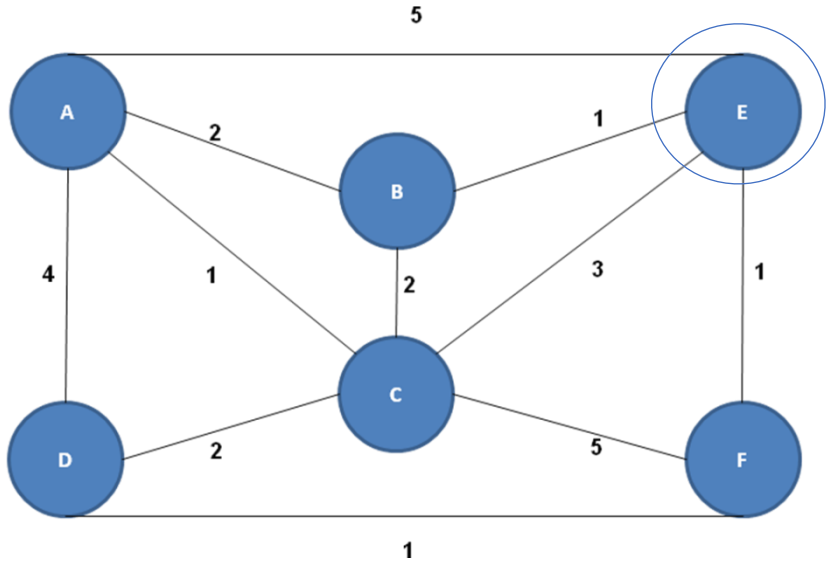


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Do you agree that E now knows the next hop for the lowest cost routes to the other nodes?

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	2/F	1/F

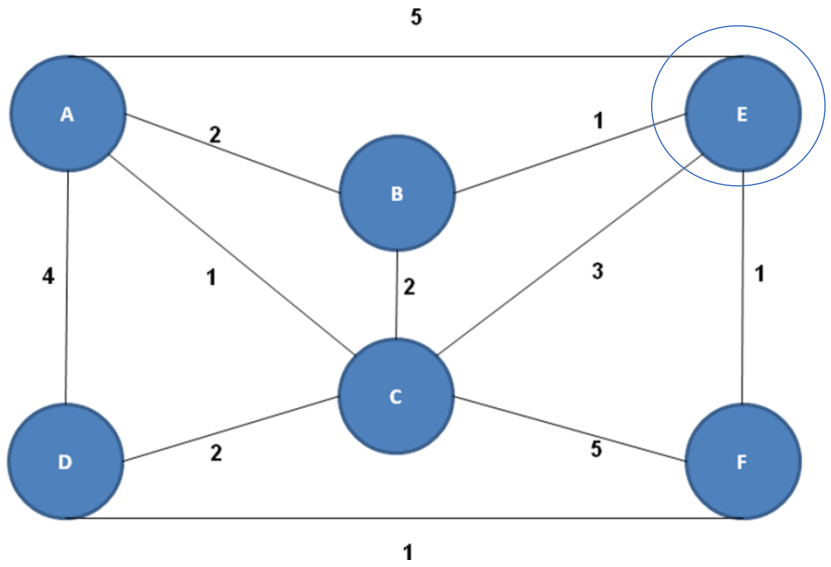


- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Do you agree that E now knows the next hop for the lowest cost routes to the other nodes?
- This of course presumed the other nodes knew their lowest cost routes

Distance vector in action

Routes from E	A	B	C	D	F
Cost/next hop	3/B	1/B	3/C	2/F	1/F



- Node E listens to messages from its neighbors:
  - A says its routes are 0/A, 5/E, 2/B, 1/C, 4/D, 5/F
  - B says its routes are 0/B, 1/E, 2/C, 2/A, 5/F, 4/D
  - C says its routes are 0/C, 1/A, 2/D, 5/F, 3/E, 2/B
  - F says its routes are 0/F, 5/C, 1/D, 1/E, 2/B, 4/A
  - D isn't heard because it isn't connected to E
- Node E looks at each route from each neighbor N; it adds its cost to get to N to N's cost to get to each other node.
- For each node, if the sum is lower cost than what E has in its route table, E changes its own route to run through the neighbor

- Do you agree that E now knows the next hop for the lowest cost routes to the other nodes?
- This of course presumed the other nodes knew their lowest cost routes
- When this runs on all nodes simultaneously, they will ***converge*** on this solution in a relatively short time (convergence depends on diameter of the network)

Distance Vector vs Link State Routing

Distance Vector vs Link State Routing

- Distance Vector
  - Each node knows the cost to reach each of its neighbors
  - Each node sends its routing table to only its neighbors
  - Each node revises its own routing table every time it receives a routing table from a neighbor
  - Slower convergence but lower traffic

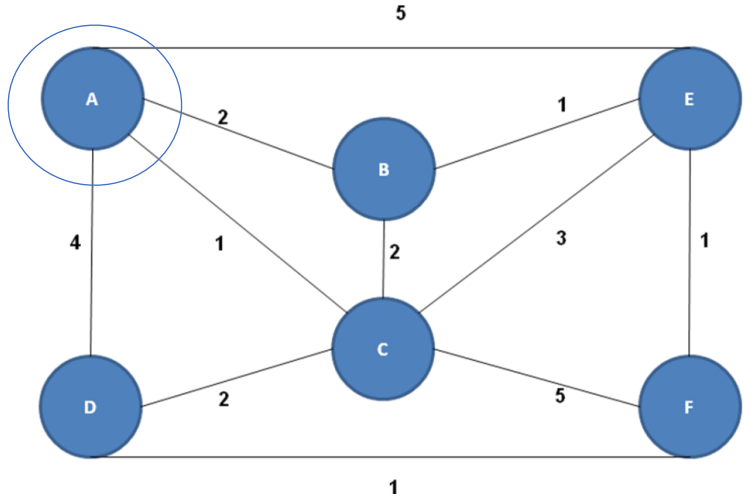
Distance Vector vs Link State Routing

- Distance Vector
  - Each node knows the cost to reach each of its neighbors
  - Each node sends its routing table to only its neighbors
  - Each node revises its own routing table every time it receives a routing table from a neighbor
  - Slower convergence but lower traffic

- Link State
  - Each node advertises its neighbor links and costs to every other node in the network
  - Each node collects the entire topology of the network from these advertisements
  - Using a "shortest path" algorithm, each node calculates its own routing table
  - Faster convergence but more traffic

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations (for n-node network)
- One least-cost route to a destination
- Runs simultaneously on each node



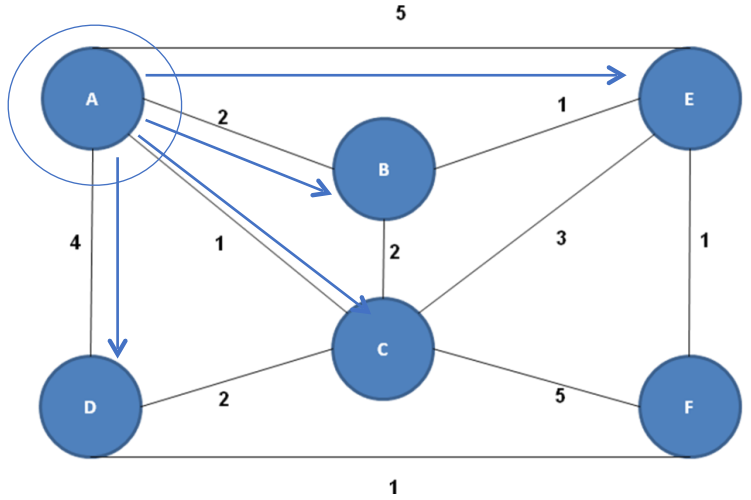
Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1						
2						
3						
4						
5						

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations (for n-node network)
- One least-cost route to a destination
- Runs simultaneously on each node



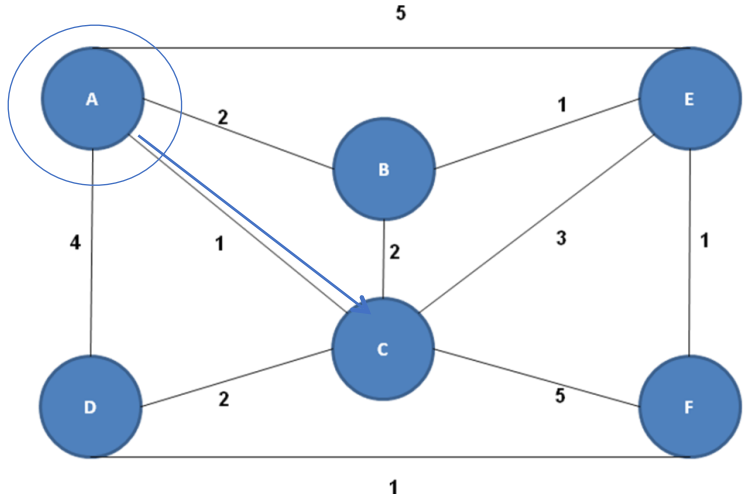
Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1						
2						
3						
4						
5						

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations (for n-node network)
- One least-cost route to a destination
- Runs simultaneously on each node



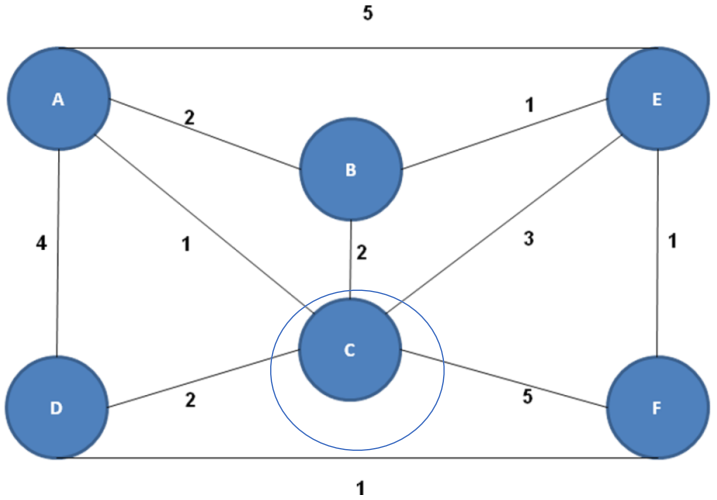
Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1						
2						
3						
4						
5						

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations
- One least-cost route to a destination
- Runs simultaneously on each node



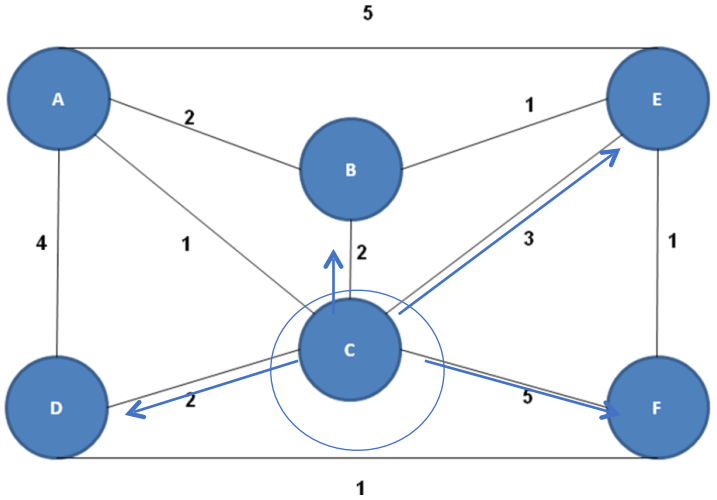
Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1	A,C	2/AB	1/AC	3/ACD	4/ACE	6/ACF
2						
3						
4						
5						

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations
- One least-cost route to a destination
- Runs simultaneously on each node



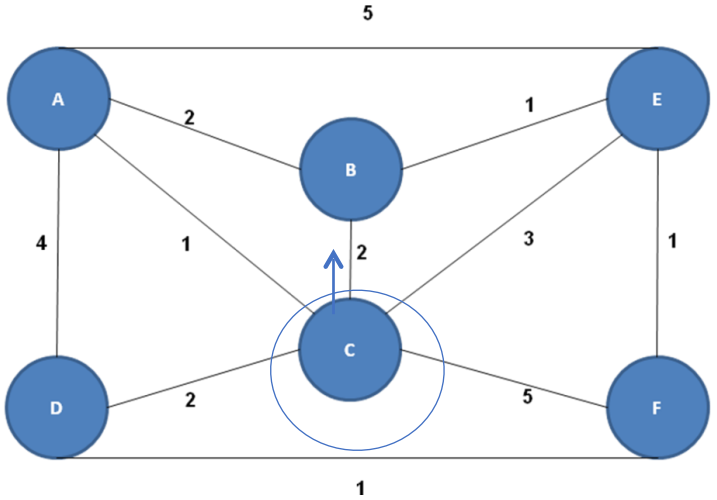
Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1	A,C	2/AB	1/AC	3/ACD	4/ACE	6/ACF
2						
3						
4						
5						

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations
- One least-cost route to a destination
- Runs simultaneously on each node



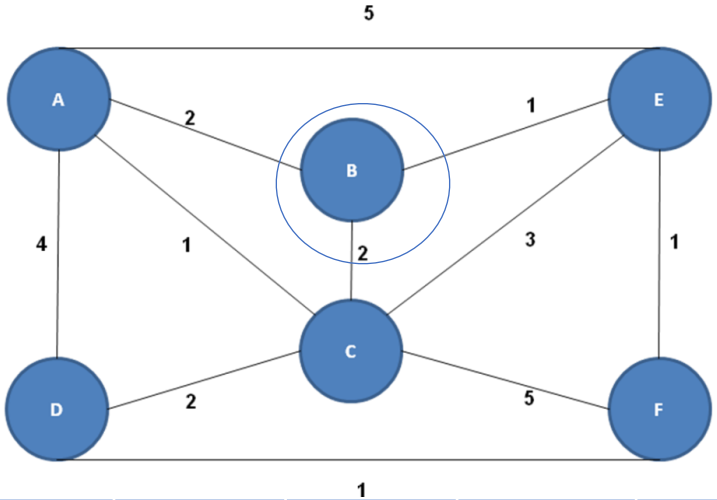
Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1	A,C	2/AB	1/AC	3/ACD	4/ACE	6/ACF
2						
3						
4						
5						

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations
- One least-cost route to a destination
- Runs simultaneously on each node



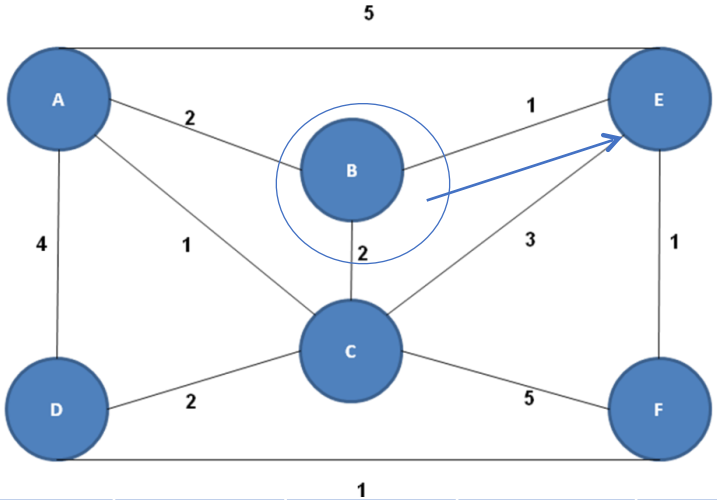
Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1	A,C	2/AB	1/AC	3/ACD	4/ACE	6/ACF
2	A,C,B	2/AB	-	3/ACD	3/ABE	6/ACF
3						
4						
5						

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations
- One least-cost route to a destination
- Runs simultaneously on each node



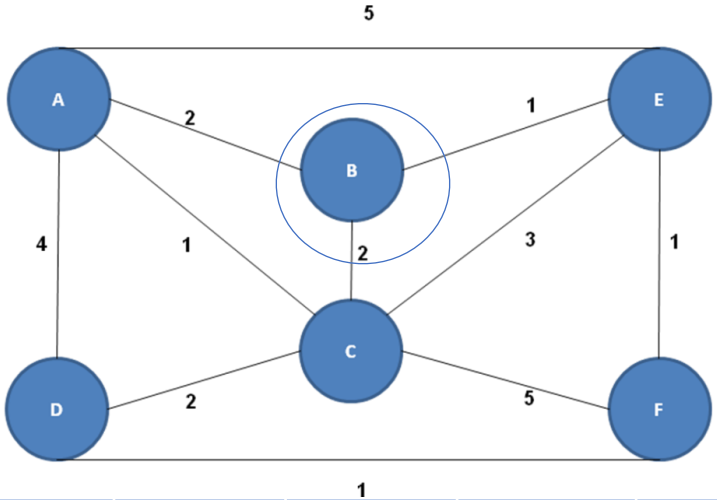
Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1	A,C	2/AB	1/AC	3/ACD	4/ACE	6/ACF
2	A,C,B	2/AB	-	3/ACD	3/ABE	6/ACF
3						
4						
5						

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations
- One least-cost route to a destination
- Runs simultaneously on each node



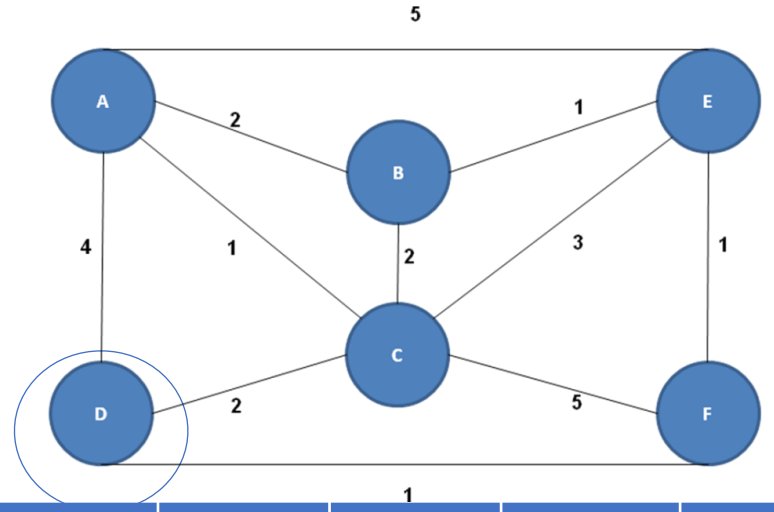
Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1	A,C	2/AB	1/AC	3/ACD	4/ACE	6/ACF
2	A,C,B	2/AB	-	3/ACD	3/ABE	6/ACF
3						
4						
5						

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations
- One least-cost route to a destination
- Runs simultaneously on each node



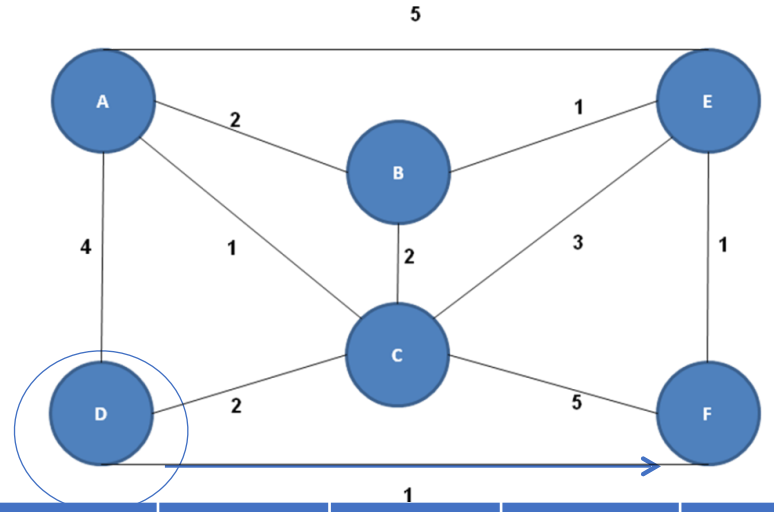
Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1	A,C	2/AB	1/AC	3/ACD	4/ACE	6/ACF
2	A,C,B	2/AB	-	3/ACD	3/ABE	6/ACF
3	A,C,B,D	-	-	3/ACD	3/ABE	4/ACDF
4						
5						

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations
- One least-cost route to a destination
- Runs simultaneously on each node



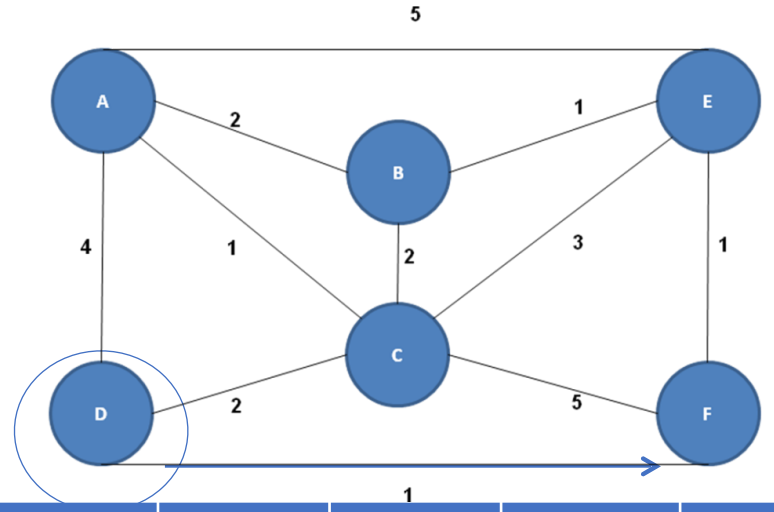
Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1	A,C	2/AB	1/AC	3/ACD	4/ACE	6/ACF
2	A,C,B	2/AB	-	3/ACD	3/ABE	6/ACF
3	A,C,B,D	-	-	3/ACD	3/ABE	4/ACDF
4						
5						

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations
- One least-cost route to a destination
- Runs simultaneously on each node



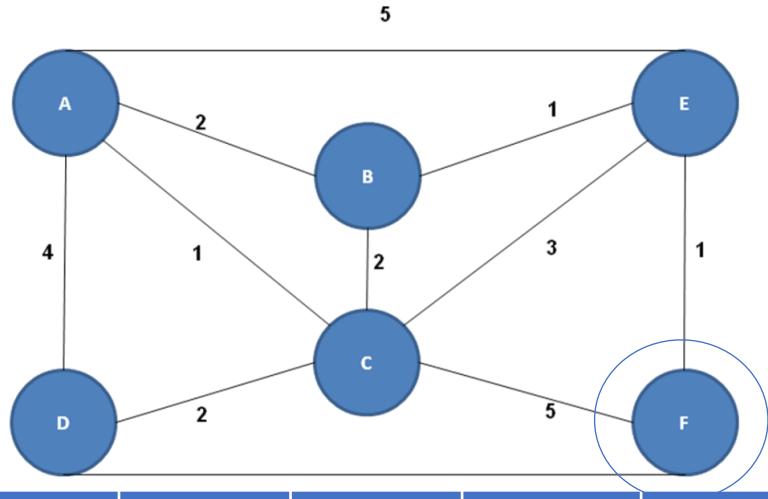
Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1	A,C	2/AB	1/AC	3/ACD	4/ACE	6/ACF
2	A,C,B	2/AB	-	3/ACD	3/ABE	6/ACF
3	A,C,B,D	-	-	3/ACD	3/ABE	4/ACDF
4						
5						

Link state in action

- Dijkstra's shortest path algorithm
- Uses global info, local algorithm (i.e. every node has this graph)
- n-1 iterations
- One least-cost route to a destination
- Runs simultaneously on each node



Calculating on Node A

(each node does its own calculation)

Iteration Count	Nodes to which least-cost routes known	B Cost/ route	C Cost/ route	D cost/ route	E Cost/ route	F Cost/ route
Init	A	2/AB	1/AC	4/AD	5/AE	∞
1	A,C	2/AB	1/AC	3/ACD	4/ACE	6/ACF
2	A,C,B	2/AB	-	3/ACD	3/ABE	6/ACF
3	A,C,B,D	-	-	3/ACD	3/ABE	4/ACDF
4	A,C,B,D,E	-	-	-	3/ABE	4/ACDF
5	A,C,B,D,E,F	-	-	-	-	4/ACDF

Hierarchical routing algorithms

Hierarchical routing algorithms

- Network of networks

Hierarchical routing algorithms

- Network of networks
- Very large scale, frequent state changes

Hierarchical routing algorithms

- Network of networks
- Very large scale, frequent state changes
- Compartmentalize so routing information protocol issues don't spread and cause widespread outages

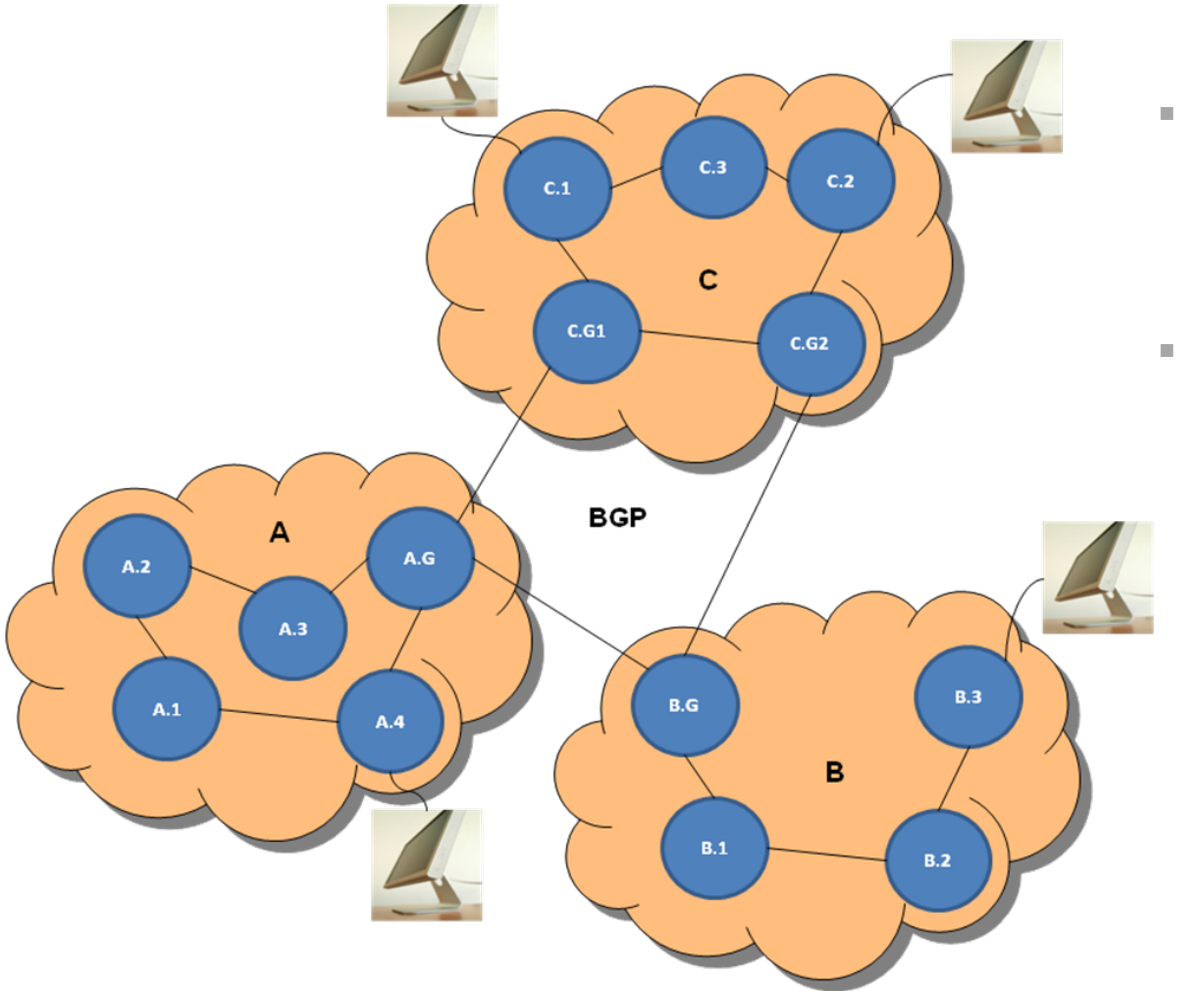
Hierarchical routing algorithms

- Network of networks
- Very large scale, frequent state changes
- Compartmentalize so routing information protocol issues don't spread and cause widespread outages
- What better example than the Internet itself!

Hierarchical routing algorithms

- Network of networks
- Very large scale, frequent state changes
- Compartmentalize so routing information protocol issues don't spread and cause widespread outages
- What better example than the Internet itself!
- Autonomous Systems (AS)
  - It is a collection of IP routing prefixes under the control of common administrative policy that presents a common, clearly defined routing policy to the internet.
  - Gateway nodes connect to other AS gateway nodes for inter-AS routing

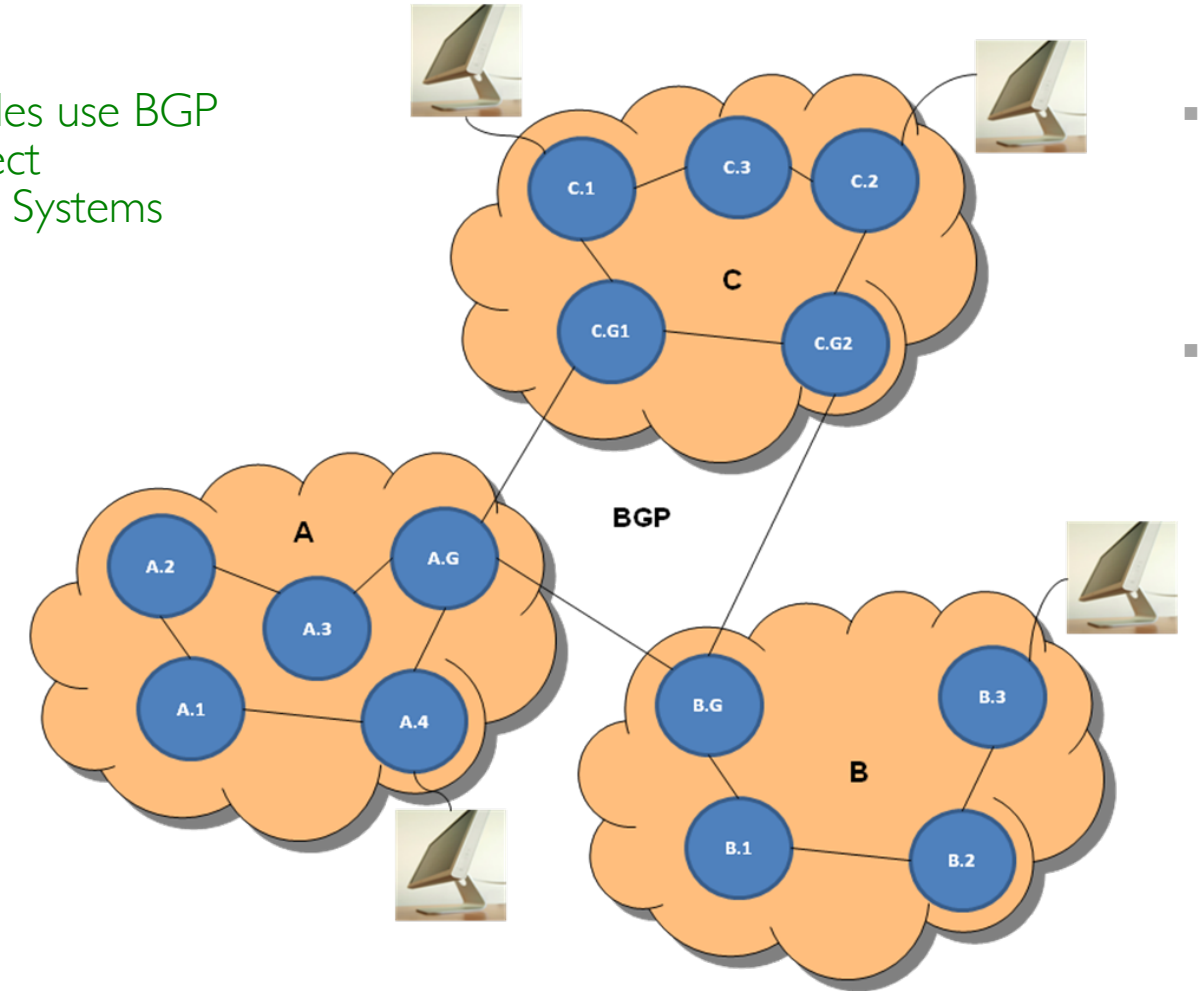
Hierarchical routing algorithms



- ASs are assigned numbers by IANA/ ARIN
- Nodes within an AS use Link State or Distance Vector (or other) routing protocol

Hierarchical routing algorithms

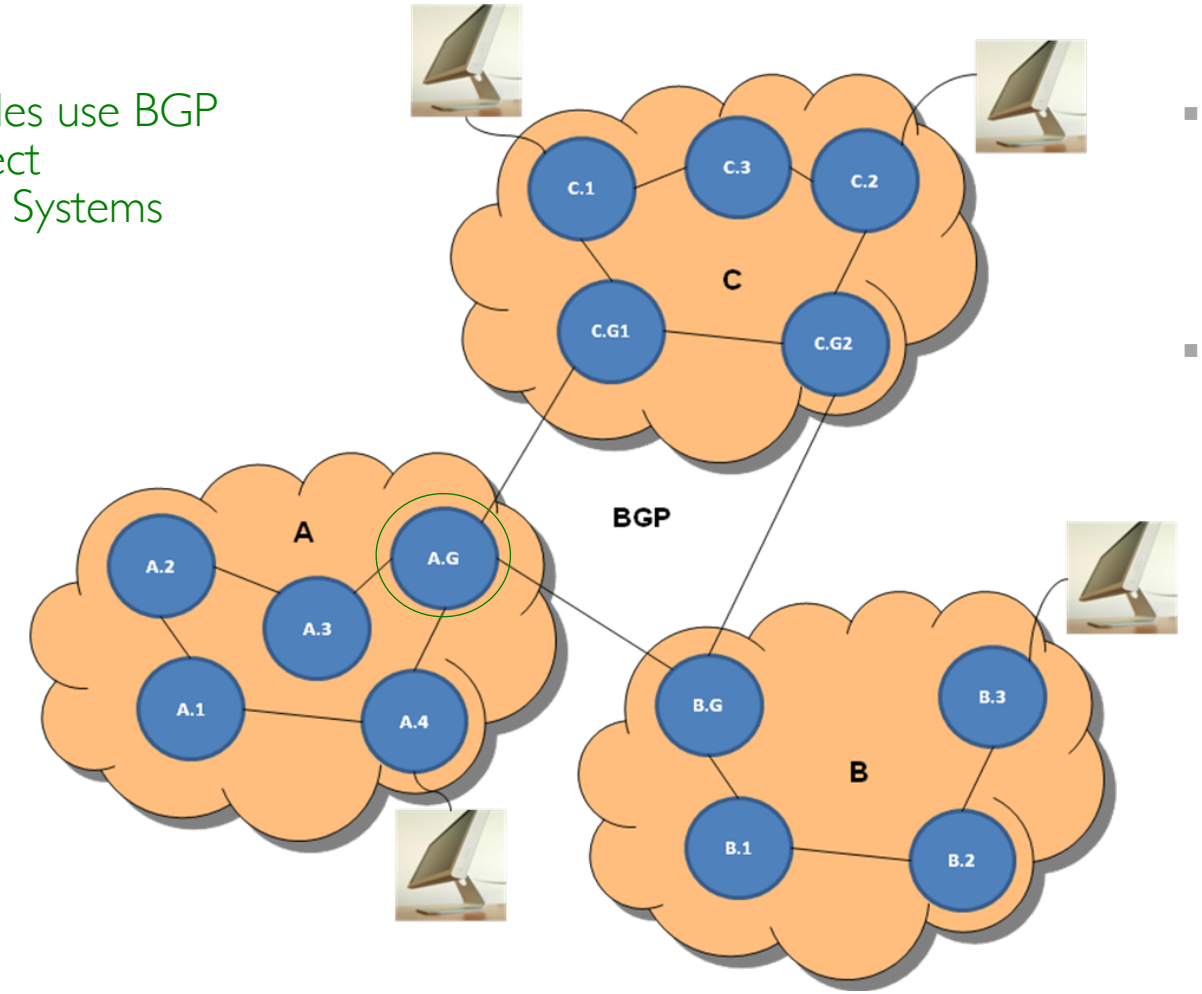
- Gateway nodes use BGP to interconnect Autonomous Systems



- ASs are assigned numbers by IANA/ ARIN
- Nodes within an AS use Link State or Distance Vector (or other) routing protocol

Hierarchical routing algorithms

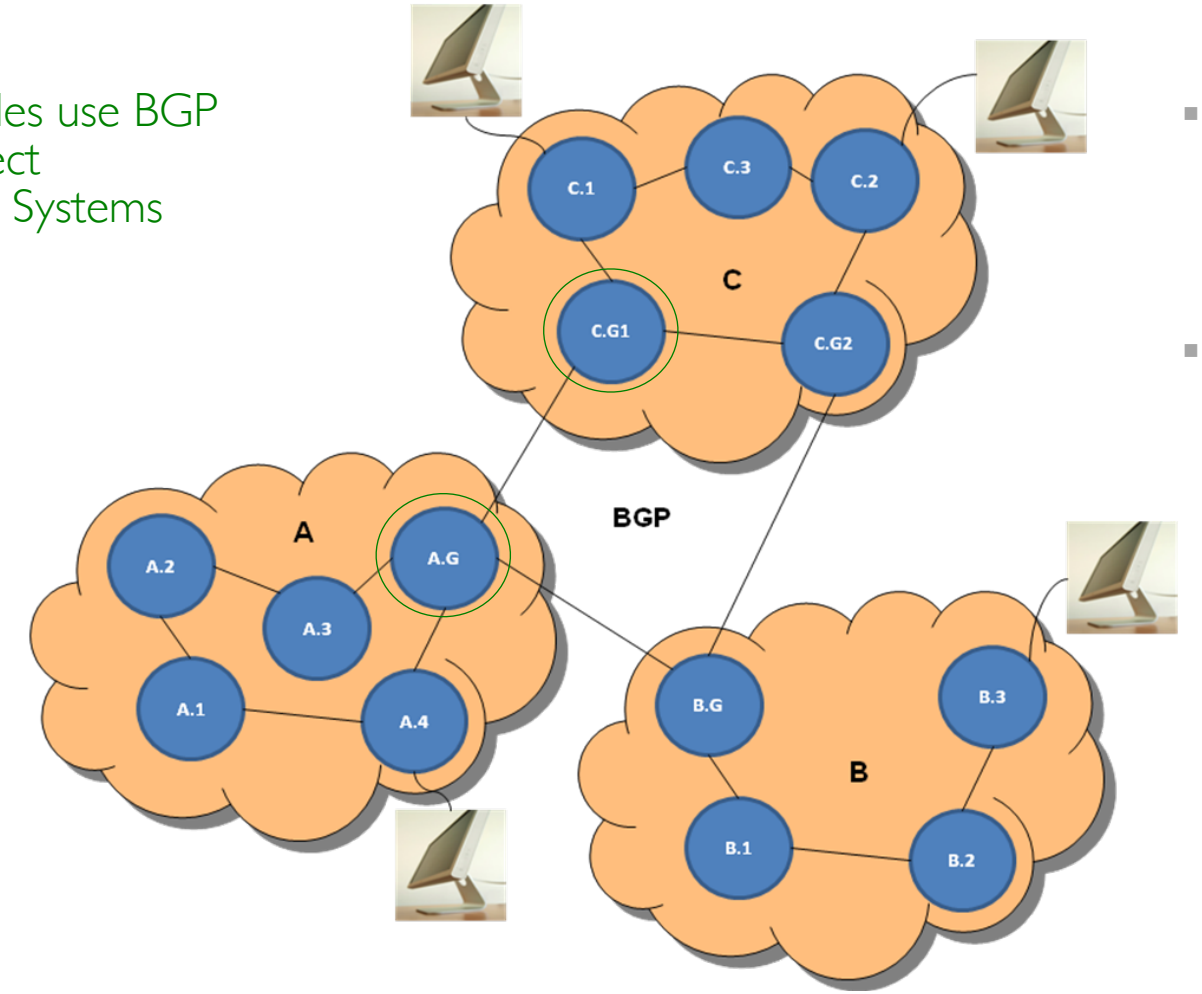
- Gateway nodes use BGP to interconnect Autonomous Systems



- ASs are assigned numbers by IANA/ ARIN
- Nodes within an AS use Link State or Distance Vector (or other) routing protocol

Hierarchical routing algorithms

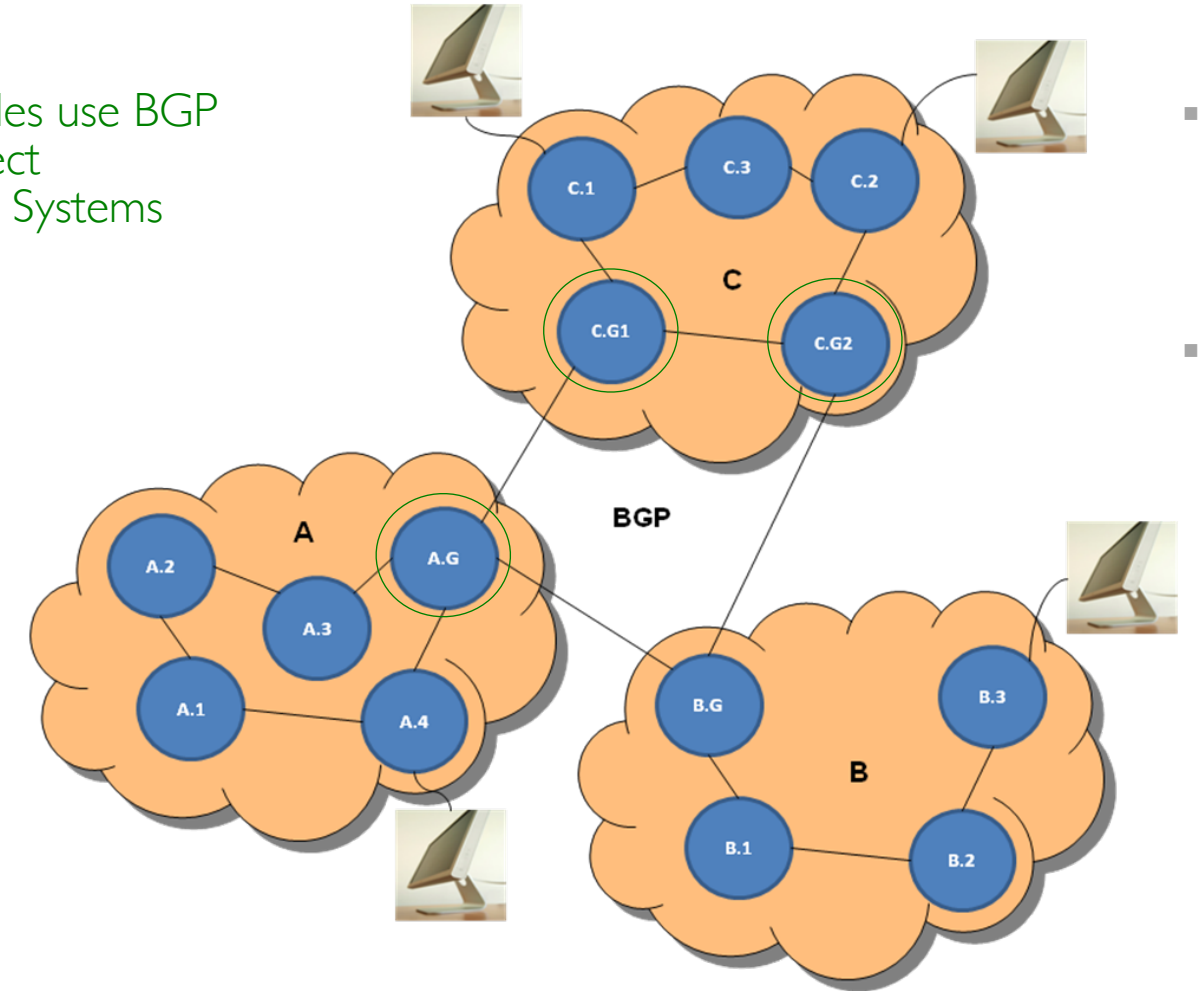
- Gateway nodes use BGP to interconnect Autonomous Systems



- ASs are assigned numbers by IANA/ ARIN
- Nodes within an AS use Link State or Distance Vector (or other) routing protocol

Hierarchical routing algorithms

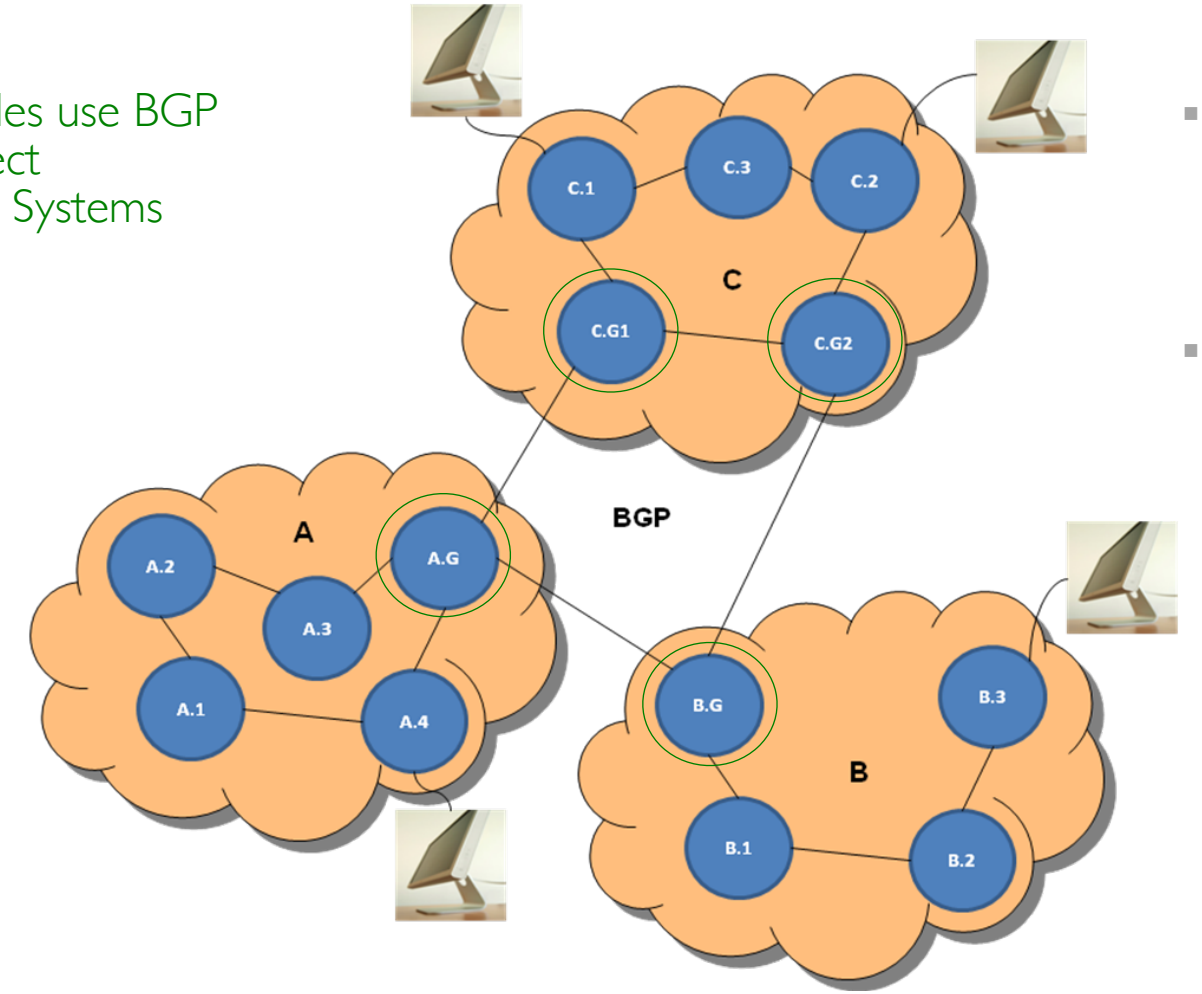
- Gateway nodes use BGP to interconnect Autonomous Systems



- ASs are assigned numbers by IANA/ ARIN
- Nodes within an AS use Link State or Distance Vector (or other) routing protocol

Hierarchical routing algorithms

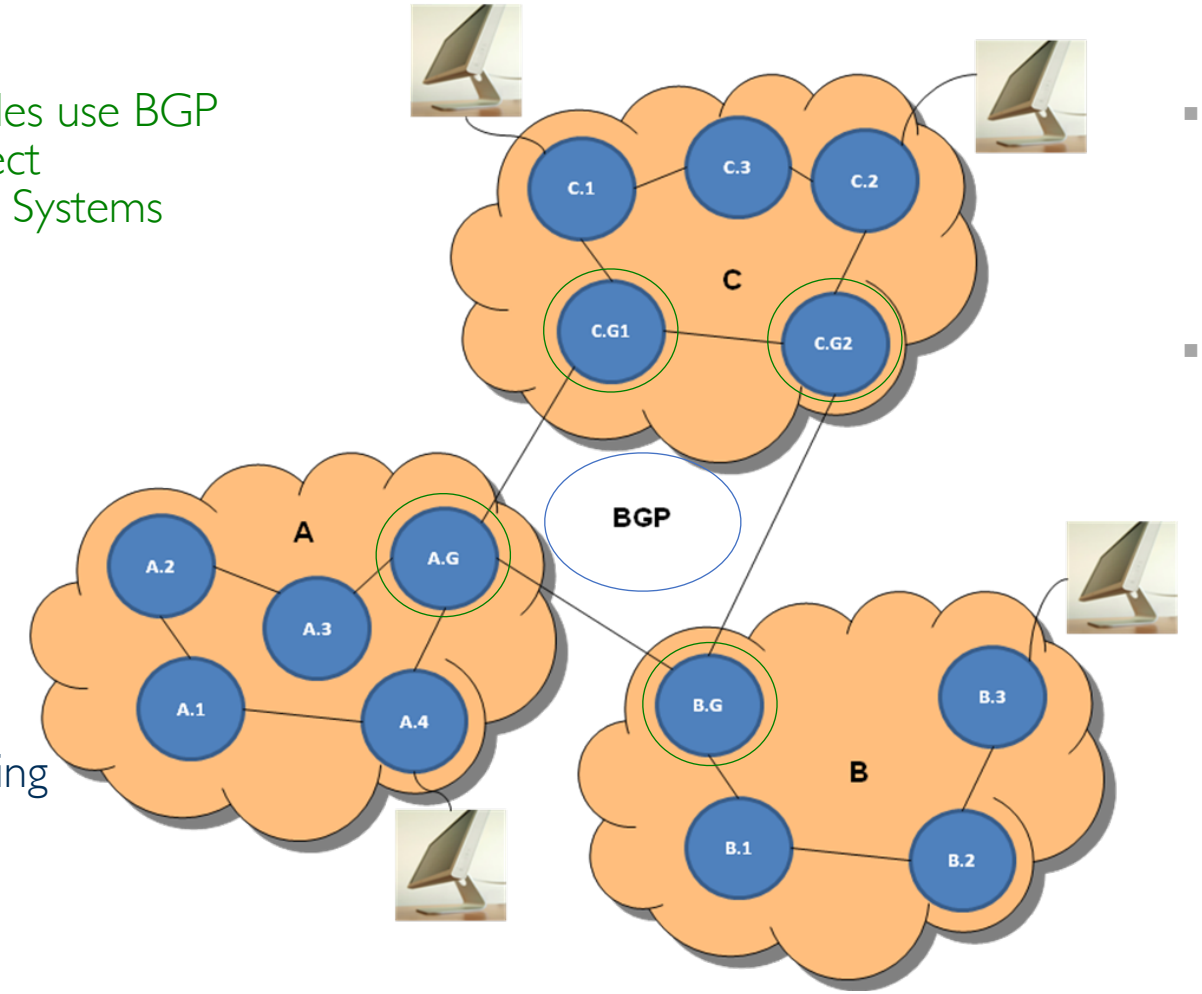
- Gateway nodes use BGP to interconnect Autonomous Systems



- ASs are assigned numbers by IANA/ ARIN
- Nodes within an AS use Link State or Distance Vector (or other) routing protocol

Hierarchical routing algorithms

- Gateway nodes use BGP to interconnect Autonomous Systems
- We use BGP as the internet inter-AS routing protocol

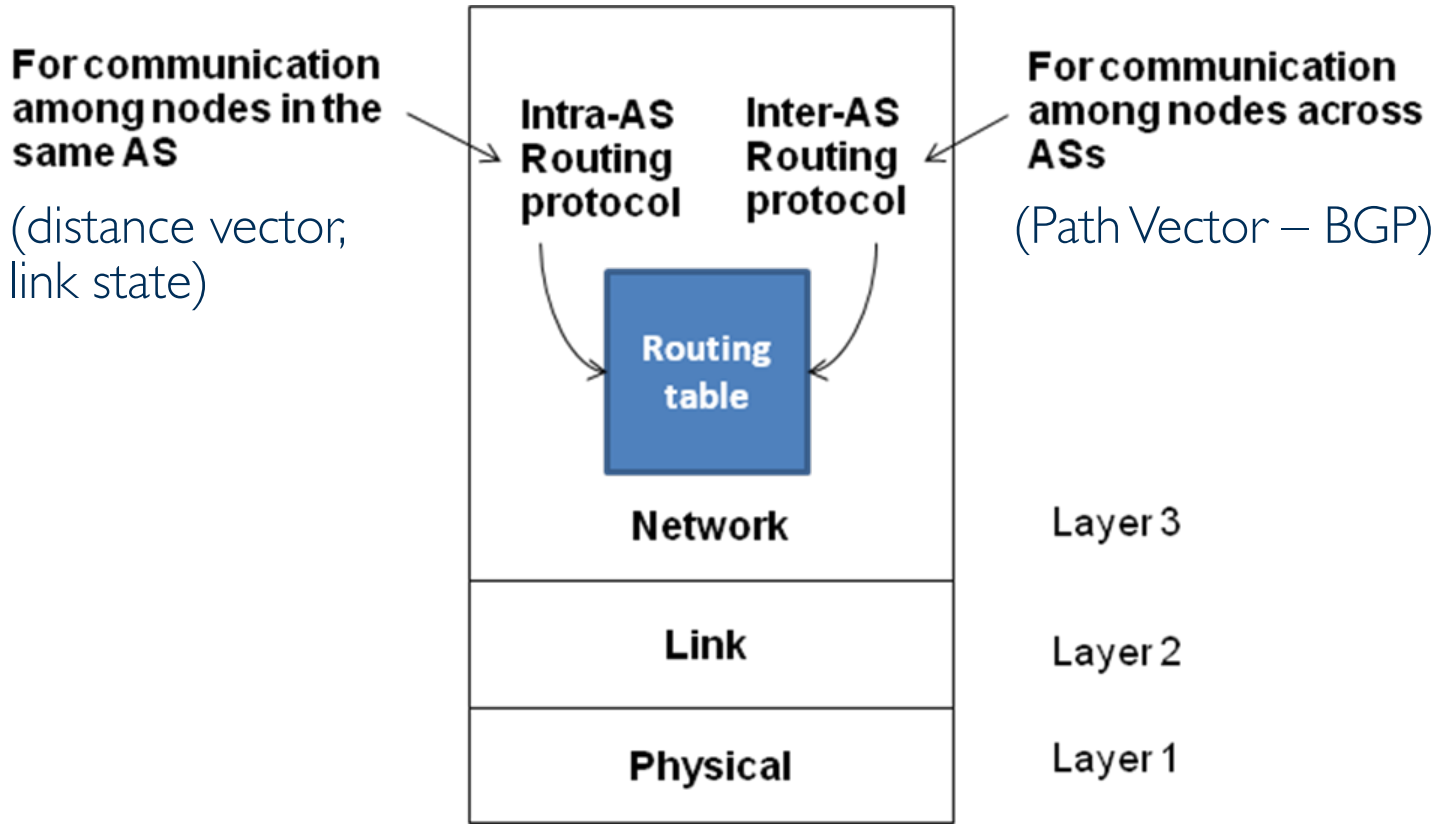


- ASs are assigned numbers by IANA/ ARIN
- Nodes within an AS use Link State or Distance Vector (or other) routing protocol

Border Gateway Protocol (BGP)

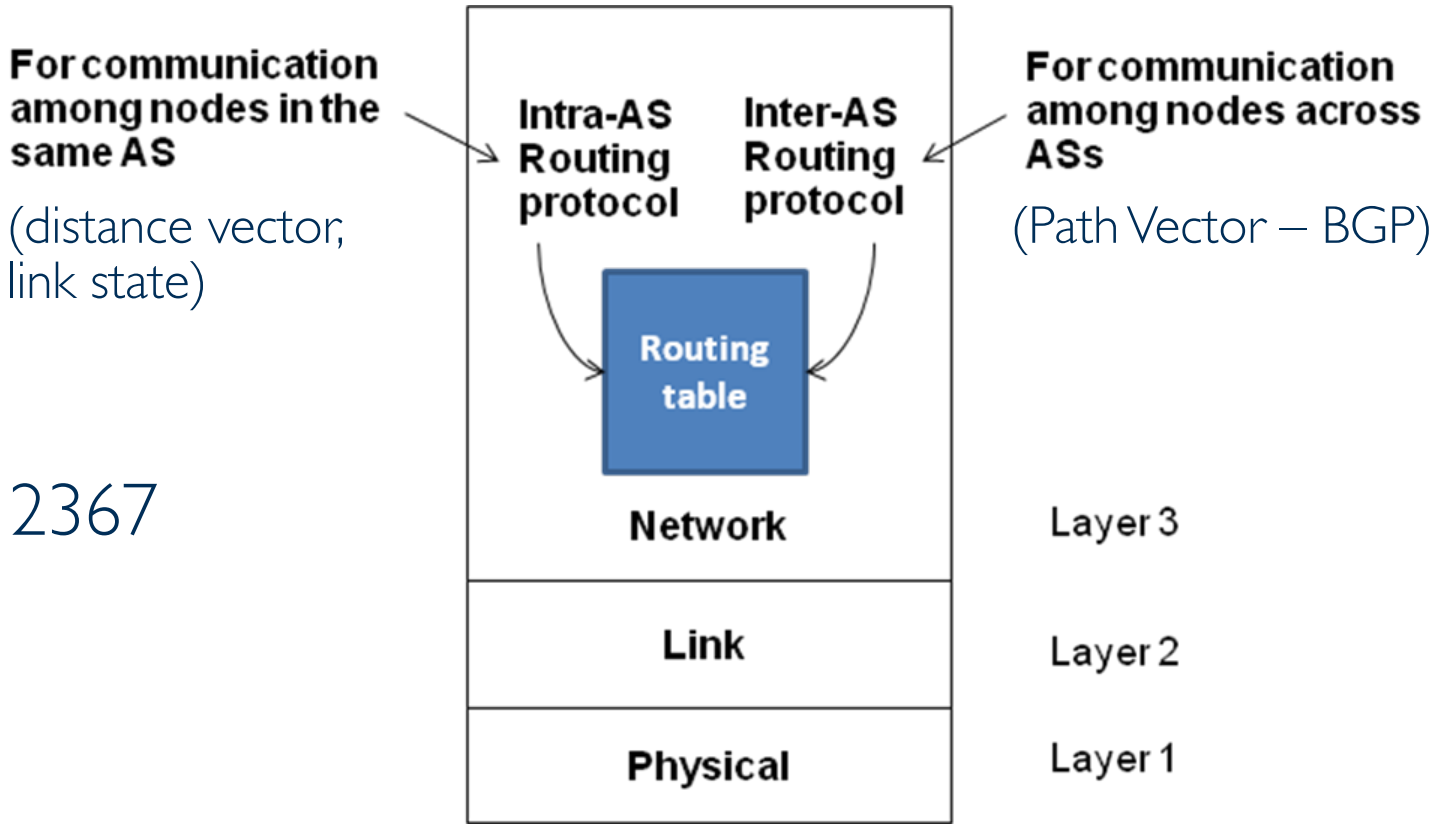
- BGP is a path-vector protocol. (We're not going into detail about that.)
- The gateway routers send path-vector messages to advertise the reachability of networks.
- Each router that receives a path vector message must verify the advertised path according to its policy.
- If the message is compliant, the router modifies its routing table and the message before sending the message to the next neighbor.
  - It modifies the routing table to maintain the autonomous systems that are traversed in order to reach the destination system.
  - It modifies the message to add its AS number and to replace the next router entry with its identification.

Routing at GT

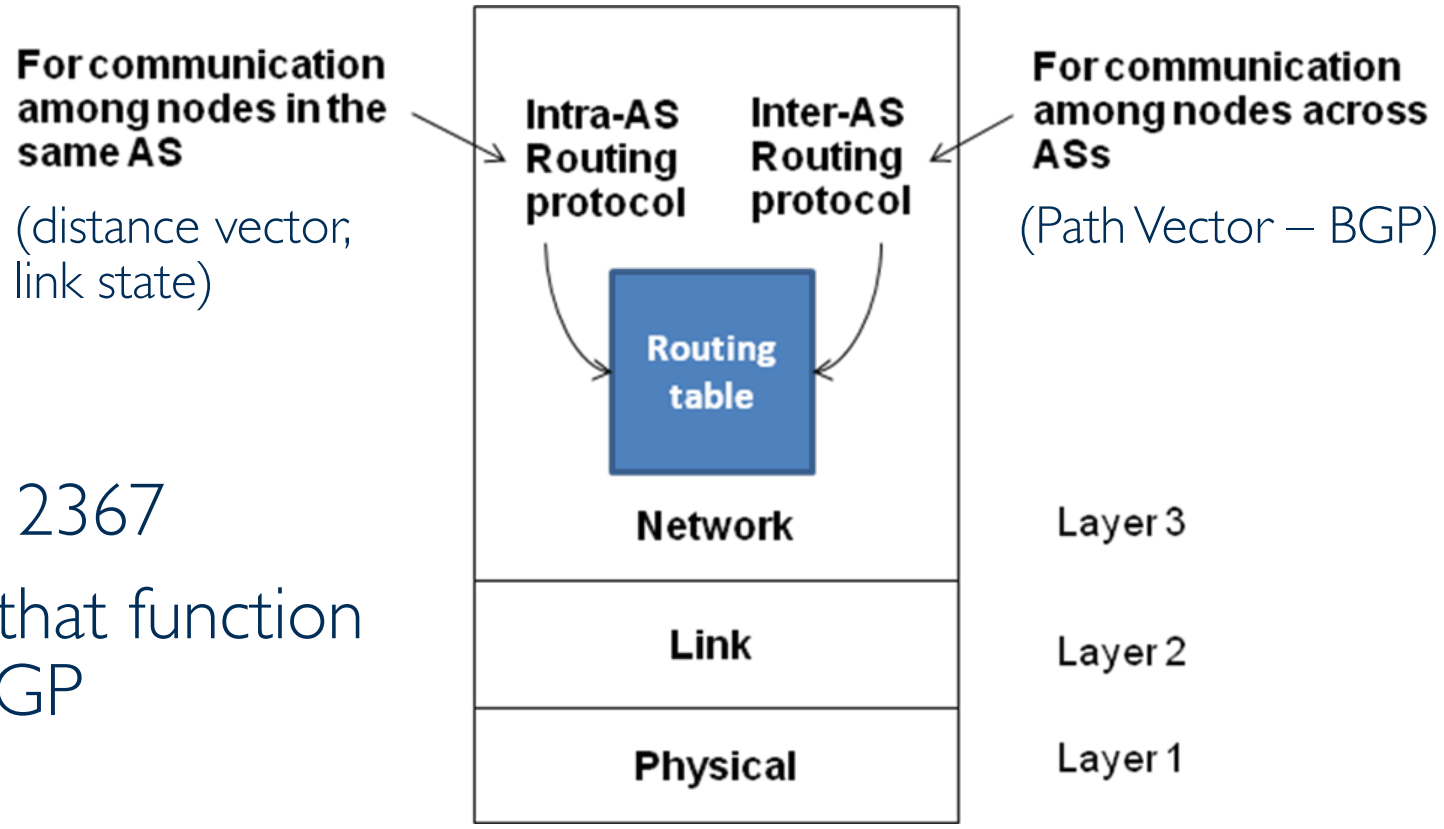


Routing at GT

- The GT campus network is AS 2367

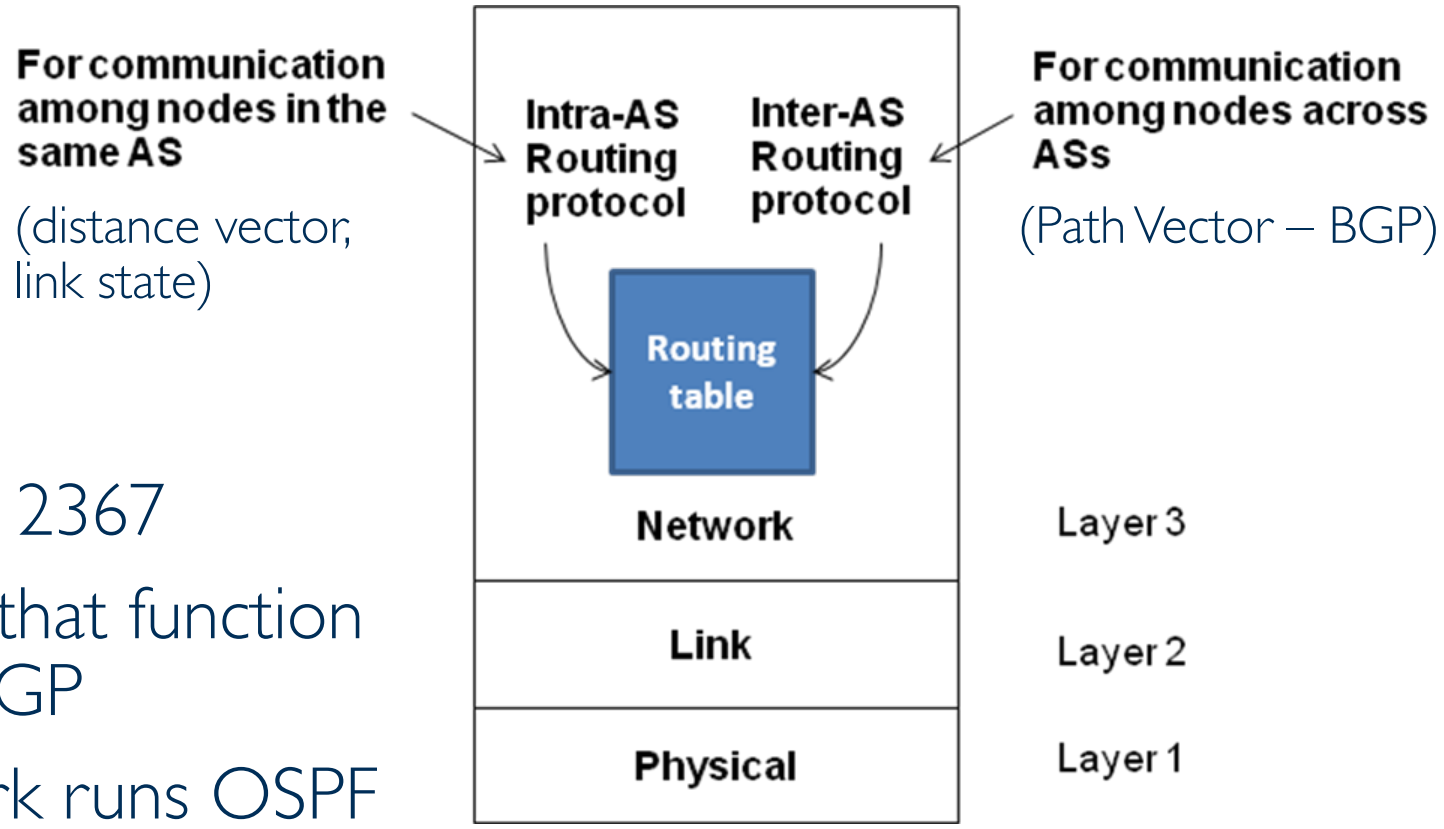


Routing at GT



- The GT campus network is AS 2367
- There are two Juniper routers that function as Gateway Routers and run BGP

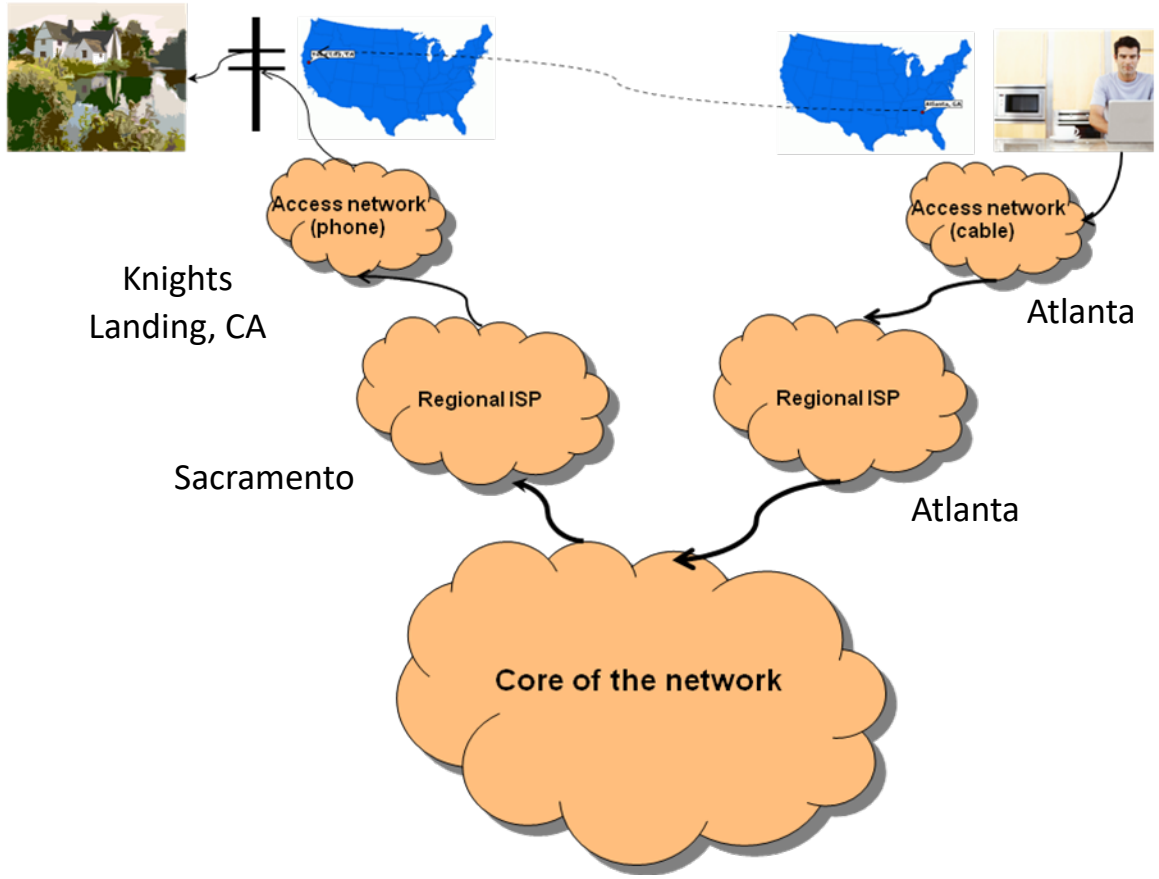
Routing at GT



- The GT campus network is AS 2367
- There are two Juniper routers that function as Gateway Routers and run BGP
- The rest of the campus network runs OSPF (a link-state routing protocol)

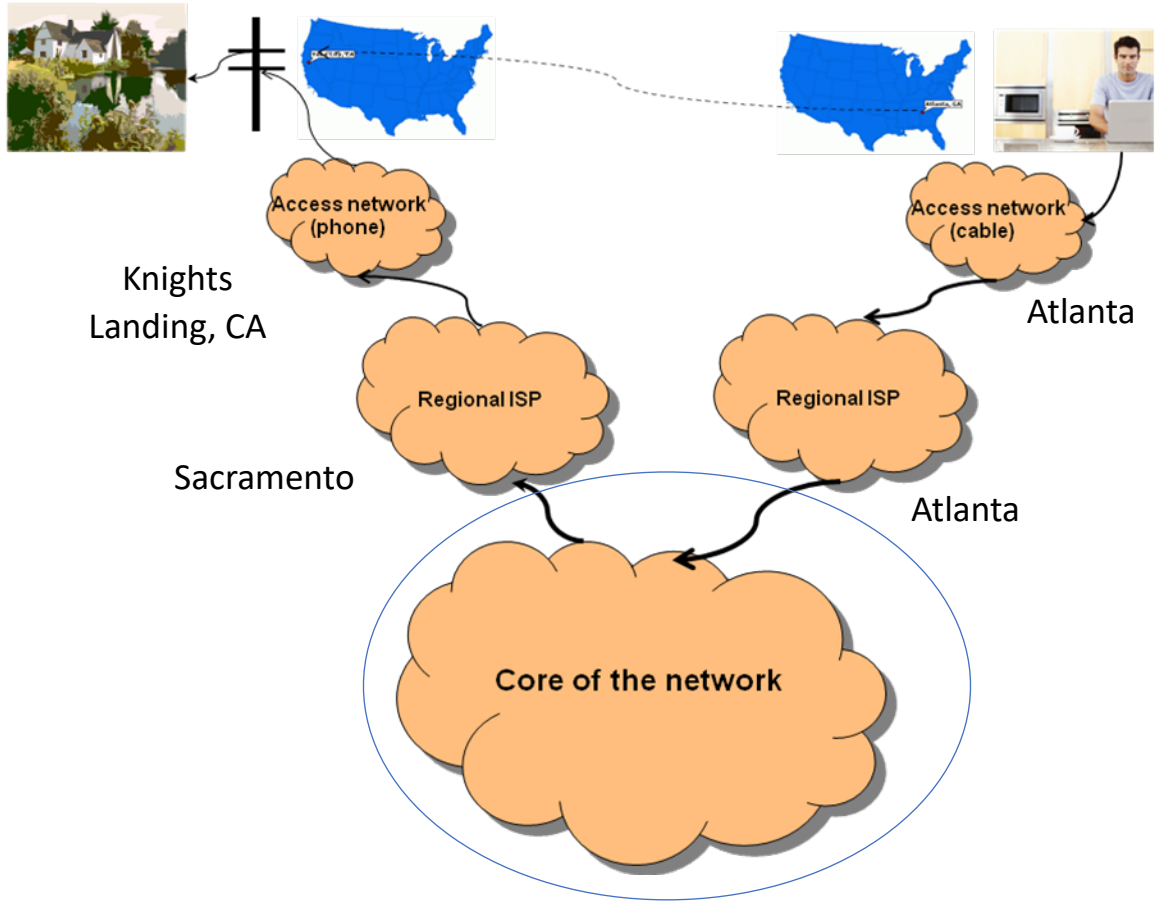
Remember this slide?

- Now consider an email from Joe to his grandmother



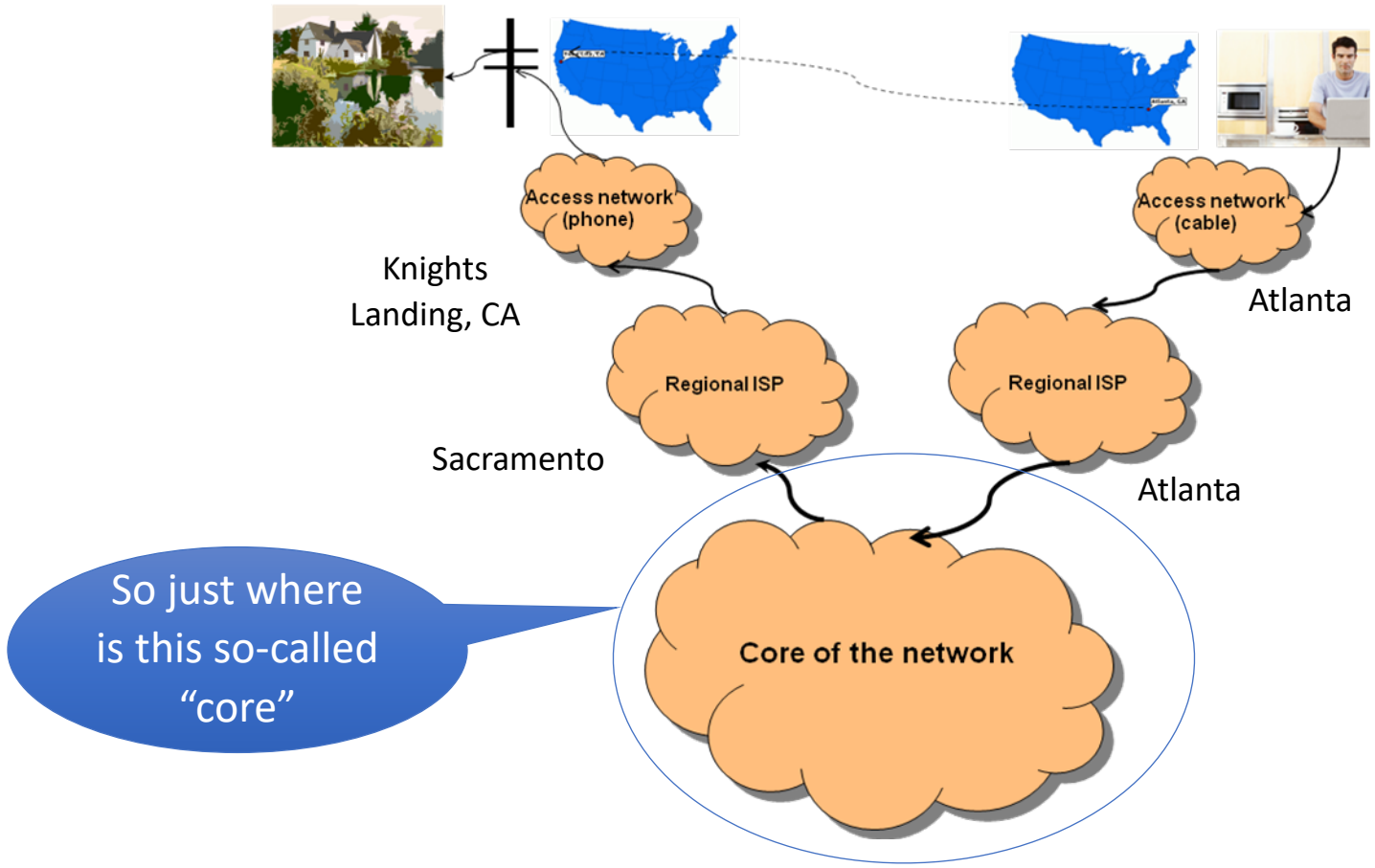
Remember this slide?

- Now consider an email from Joe to his grandmother



Remember this slide?

- Now consider an email from Joe to his grandmother



There really is no internet core …

- Instead there are dozens of Internet Exchange Points (IXPs) where groups of Internet Service Providers (ISPs) interconnect their networks

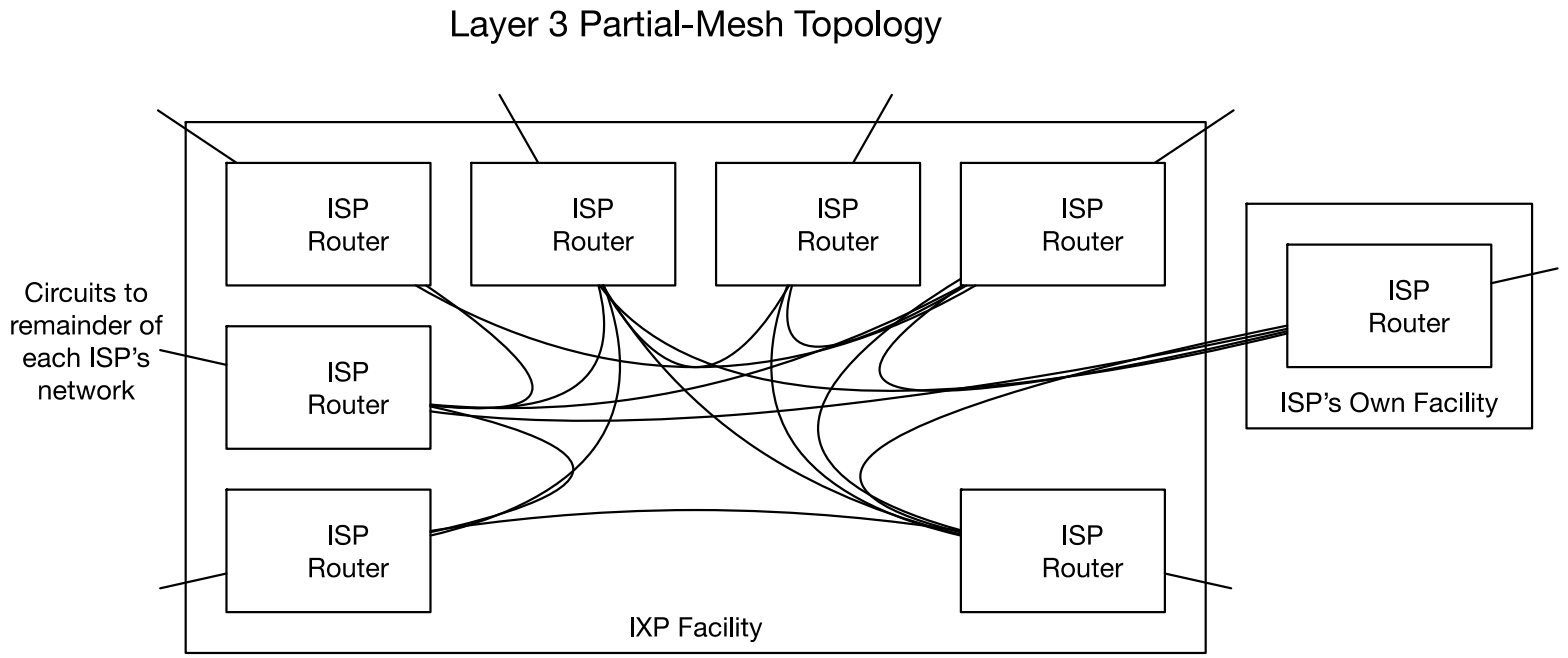
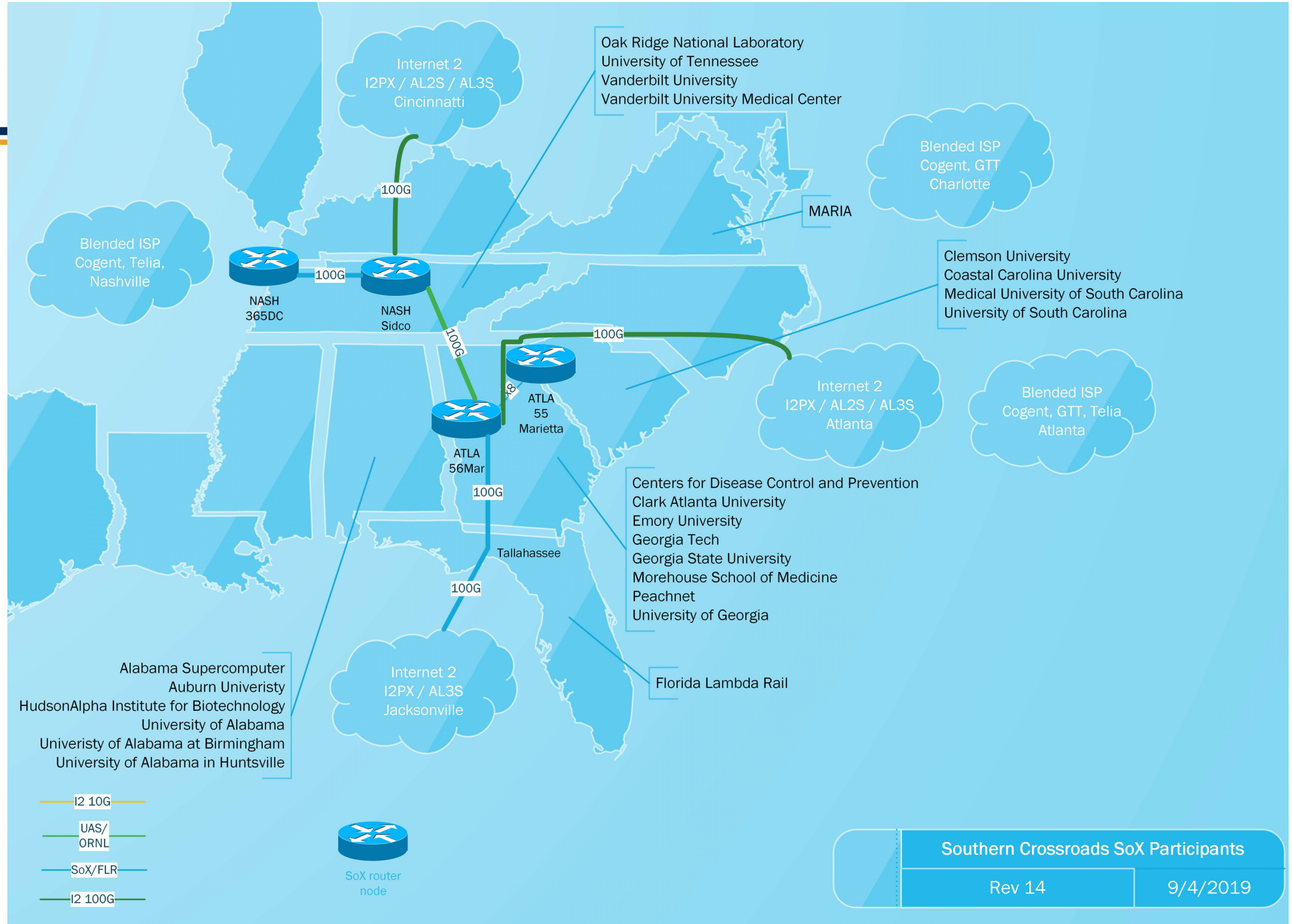


Diagram by Bill Woodcock, Packet Clearing House, rights released for use on Wikipedia

There really is no internet core …

- The ISPs peer using BGP such that all the gateway routers know multiple routes to every regional ISP
- You can begin to see that the Internet has a highly distributed "core" created by multiple IXPs
- There are at least four IXPs in Atlanta; GT participates in and operates an ISP, SoX, which provides network access for about 3 dozen southeastern universities and research centers
- The diagram on the next slide shows just the IXPs in which SoX participates. Note that traffic is allowed to traverse SoX networks to get from one ISP to another.
- The diagram doesn't show the SoX peering with other entities such as Google, AT&T, NASA, Dept of Energy, and Comcast


Internet 2
I2PX / AL2S / AL3S
Cincinnati
Oak Ridge National Laboratory
University of Tennessee
Vanderbilt University
Vanderbilt University Medical Center
Blended ISP
Cogent, GTT
Charlotte
MARIA
Blended ISP
Cogent, Telia,
Nashville
100G
NASH
365DC
NASH
Sidco
Clemson University
Coastal Carolina University
Medical University of South Carolina
University of South Carolina
100G
100G
ATLA
55
Marietta
Internet 2
I2PX / AL2S / AL3S
Atlanta
Blended ISP
Cogent, GTT, Telia
Atlanta
ATLA
56Mar
100G
Centers for Disease Control and Prevention
Clark Atlanta University
Emory University
Georgia Tech
Georgia State University
Morehouse School of Medicine
Peachnet
University of Georgia
Tallahassee
100G
Internet 2
I2PX / AL3S
Jacksonville
Florida Lambda Rail
Alabama Supercomputer
Auburn Univeristy
HudsonAlpha Institute for Biotechnology
University of Alabama
Univeristy of Alabama at Birmingham
University of Alabama in Huntsville
I2 10G
UAS/
ORNL
SoX/FLR
I2 100G
SoX router
node
Southern Crossroads SoX Participants
Rev 14
9/4/2019

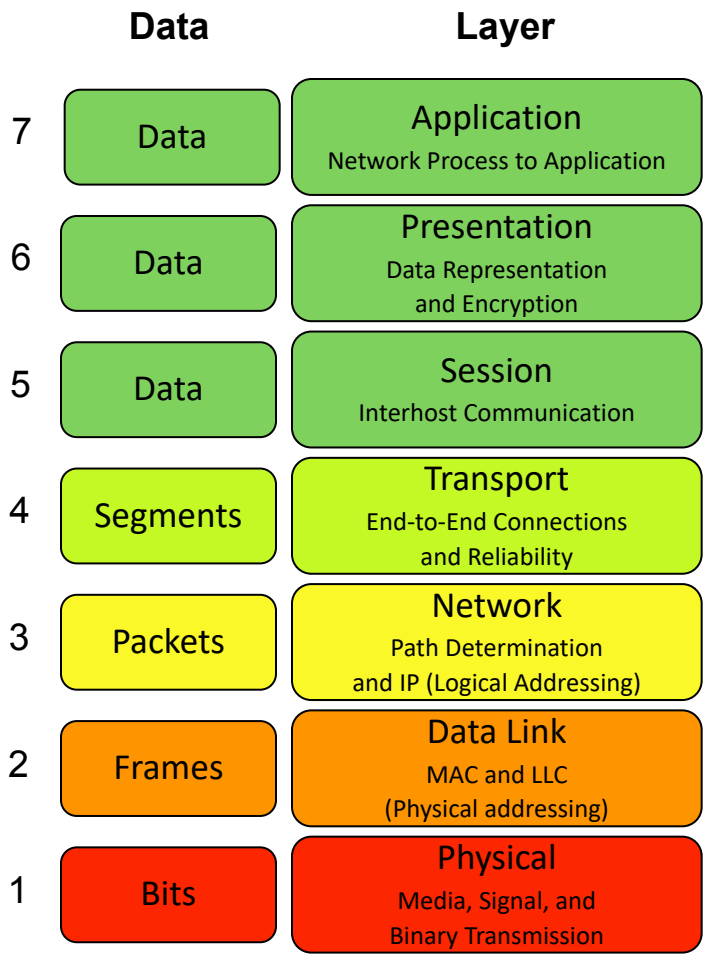
Network Layer Summary

Network Terminology	Definition/Use
Circuit switching	A network layer technology used in telephony. Reserves the network resources (link bandwidth in all the links from source to destination) for the duration of the call; no queuing or store-and-forward delays
Packet switching	A network layer technology used in wide area Internet. It supports best effort delivery of packets from source to destination without reserving any network resources en route.
Message switching	Similar to packet switching but at the granularity of the whole message (at the transport level) instead of packets.
Switch/Router	A device that supports the network layer functionality. It may simply be a computer with a number of network interfaces and adequate memory to serve as input and output buffers.
Input buffers	These are buffers associated with each input link to a switch for assembling incoming packets.
Output buffers	These are buffers associated with each outgoing link from a switch if in case the link is busy.
Routing table	This is table that gives the next hop to be used by this switch for an incoming packet based on the destination address. The initial contents of the table as well as periodic updates are a result of routing algorithms in use by the network layer.

Network Layer Summary

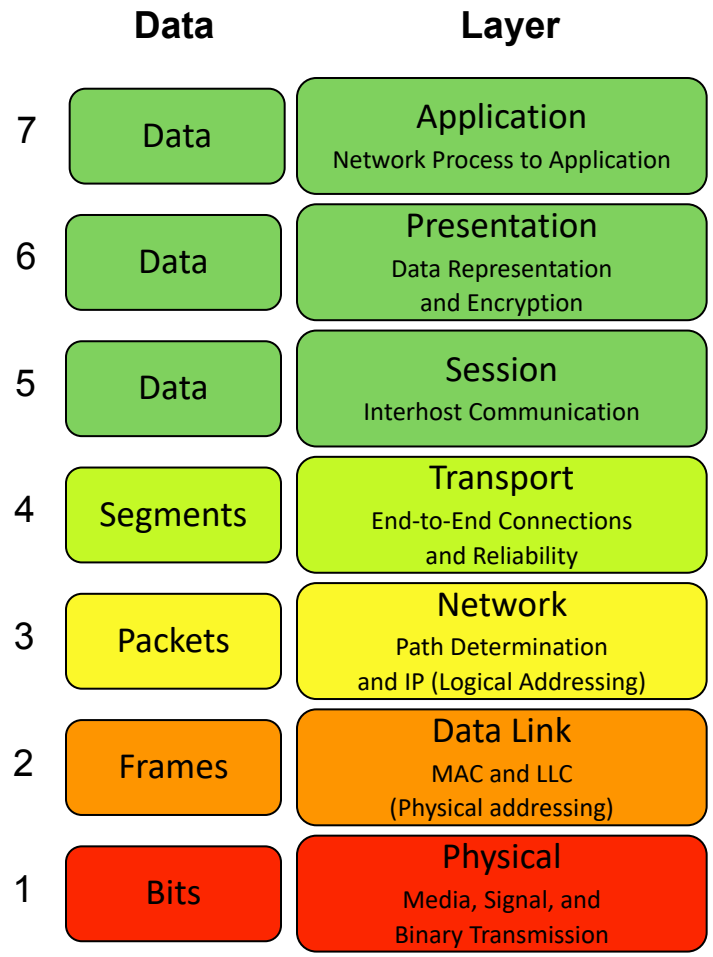
Network Terminology	Definition/Use
Delays	The delays experienced by packets in a packet-switched network
Store and forward	This delay is due to the waiting time for the packet to be fully formed in the input buffer before the switch can act on it.
Queuing	This delay accounts for the waiting time experienced by a packet on either the input or the output buffer before it is finally sent out on an outgoing link.
Packet loss	This is due to the switch having to drop a packet due to either the input or the output buffer being full and is indicative of traffic congestion on specific routes of the network.
Service Model	This is the contract between the network layer and the upper layers of the protocol stack. Both the datagram and virtual circuit models used in packet-switched networks provide **best effort delivery** of packets.
Virtual Circuit (VC)	This model sets up a virtual circuit between the source and destination so that individual packets may simply use this number instead of the destination address. This also helps to simplify the routing decision a switch has to make on an incoming packet. (ATM and X.25)
Datagram	This model does not need any call setup or tear down. Each packet is independent of the others and the switch provides a best effort service model to deliver it to the ultimate destination using information in its routing table. (IP)

Summary


Data
Layer
7
Data
Application
Network Process to Application
6
Data
Presentation
Data Representation
and Encryption
5
Data
Session
Interhost Communication
4
Segments
Transport
End-to-End Connections
and Reliability
3
Packets
Network
Path Determination
and IP (Logical Addressing)
2
Frames
Data Link
MAC and LLC
(Physical addressing)
1
Bits
Physical
Media, Signal, and
Binary Transmission

Summary

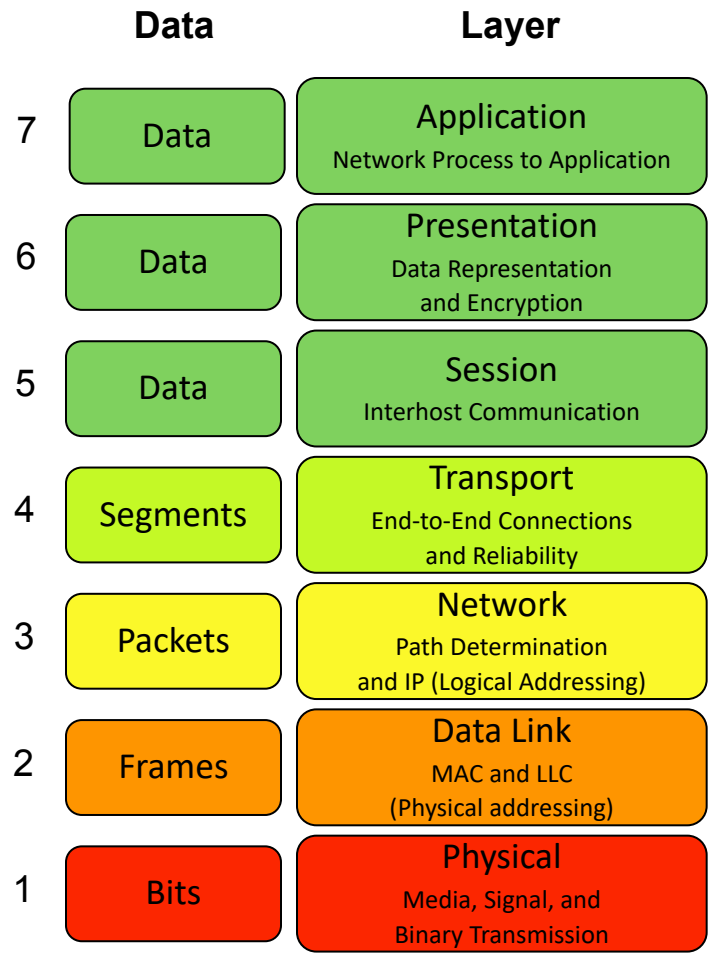
Sockets, http



Summary

Sockets, http

RPC

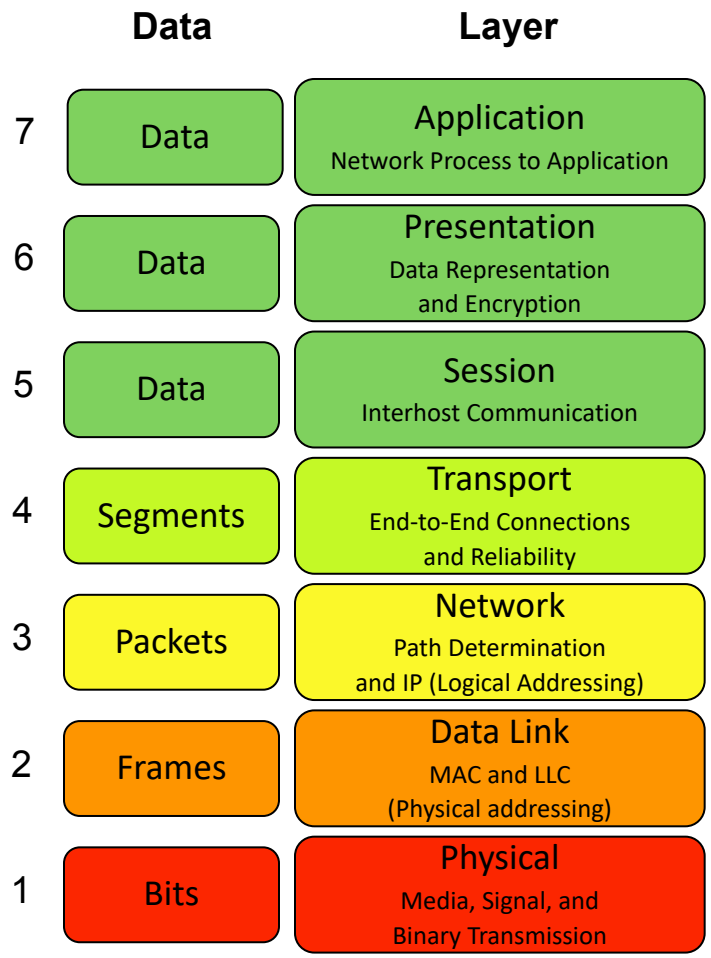


Summary

Sockets, http

RPC

TCP



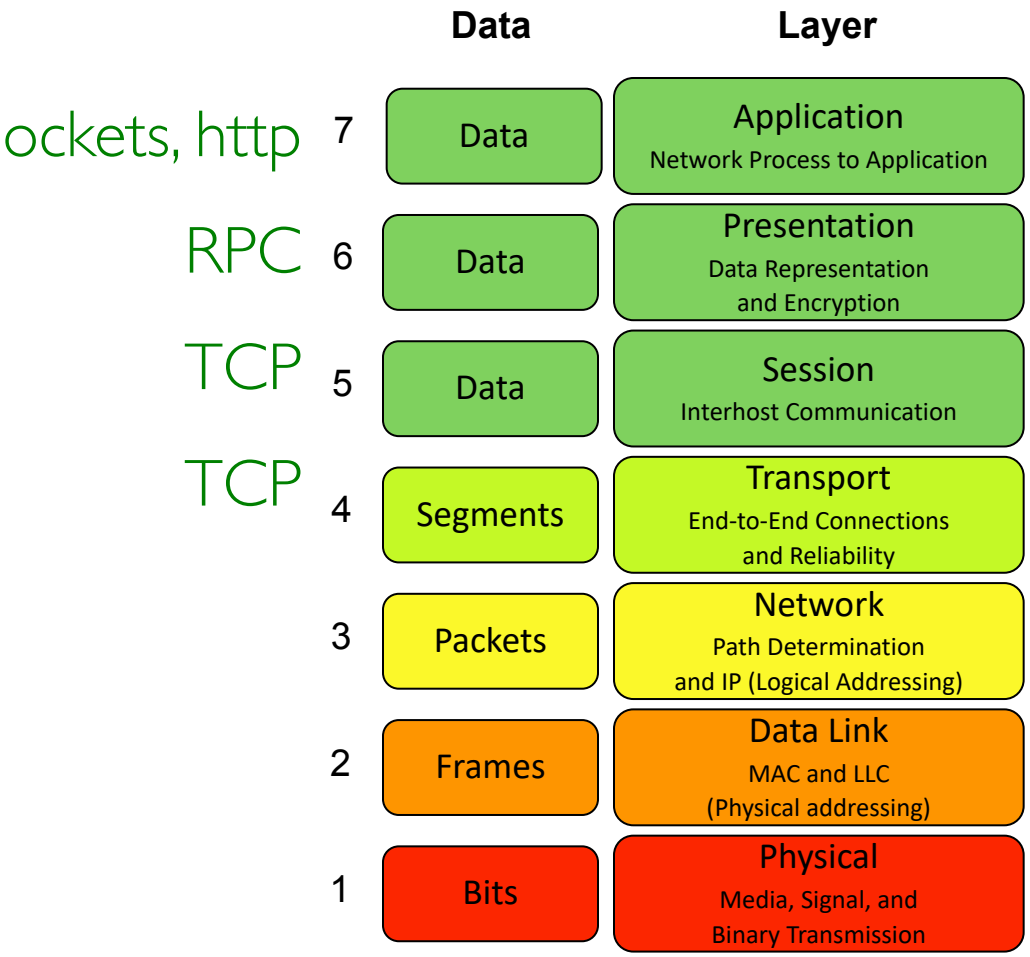
Summary

Sockets, http

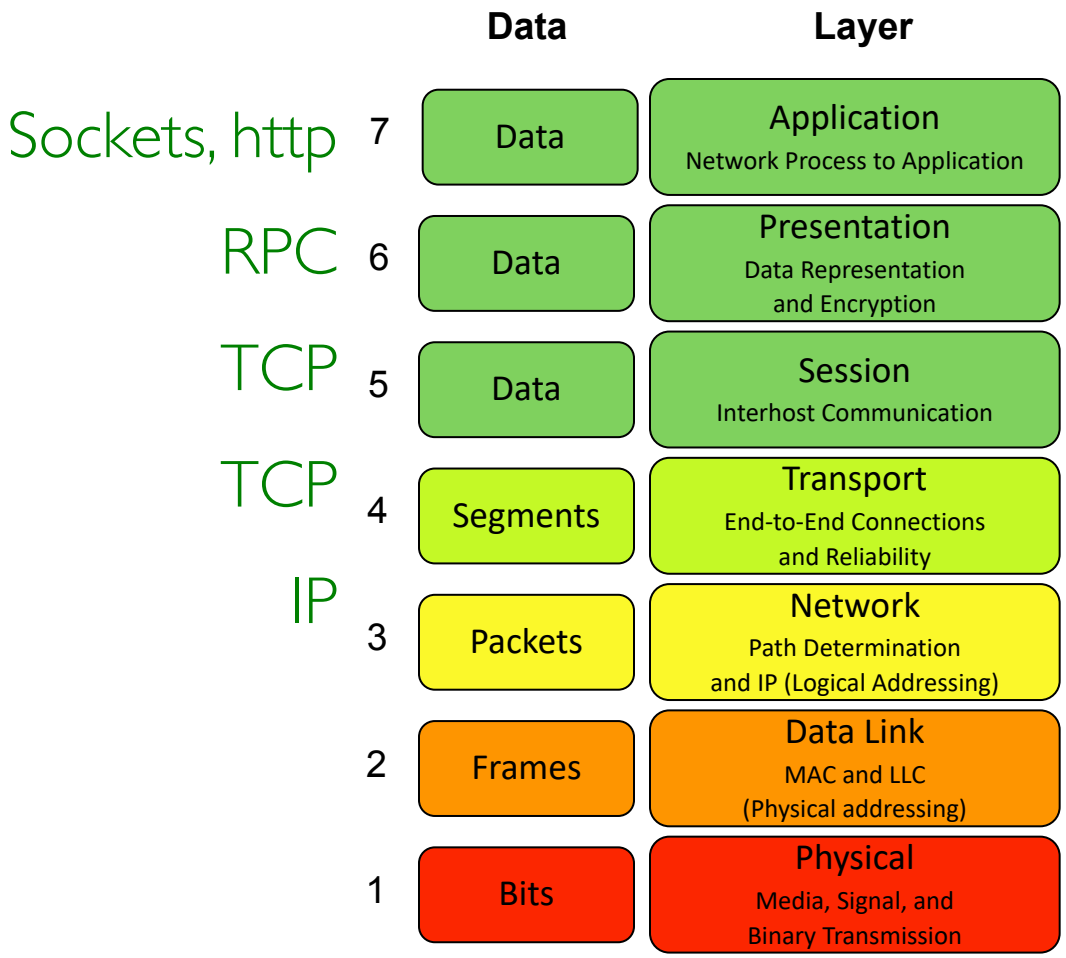
RPC

TCP

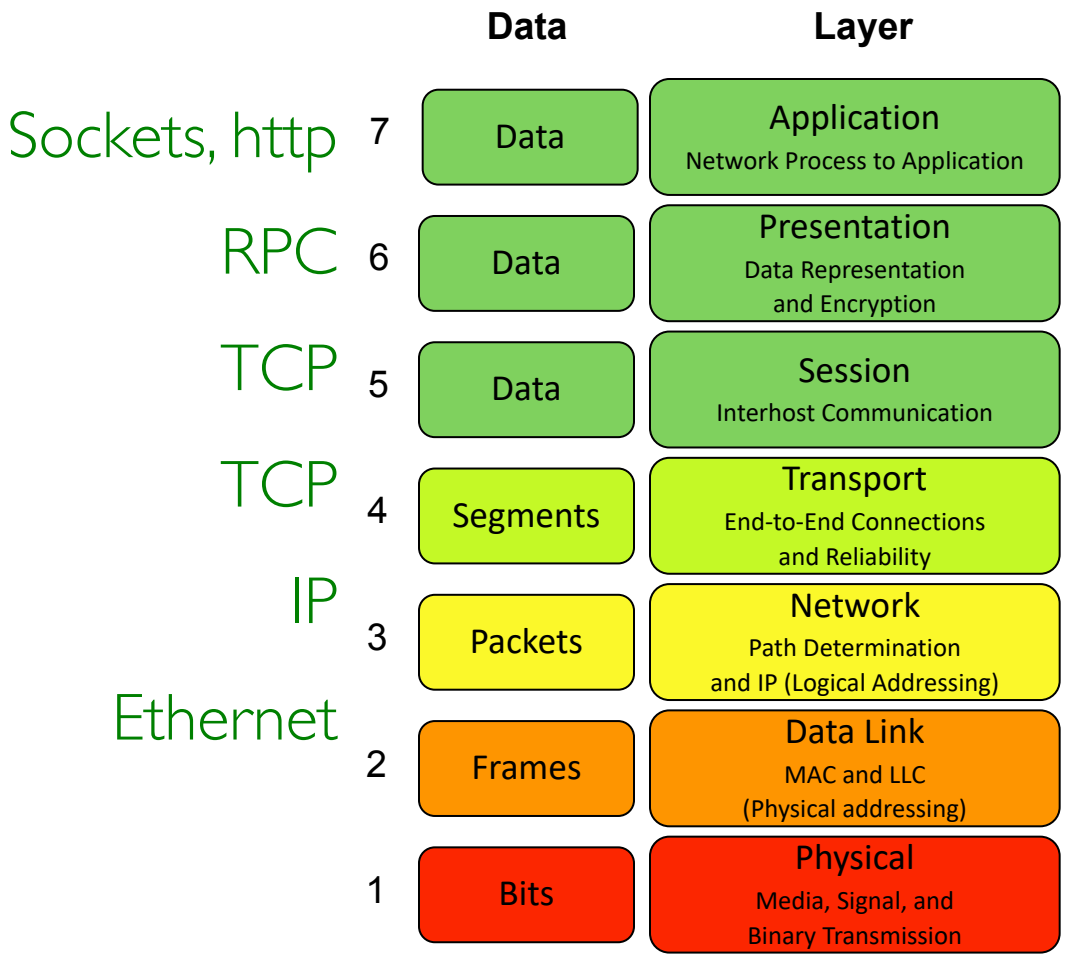
TCP



Summary


Data
Layer
Sockets, http
RPC
TCP
TCP
IP
7
6
5
4
3
2
1
Data
Data
Data
Segments
Packets
Frames
Bits
Application
Network Process to Application
Presentation
Data Representation
and Encryption
Session
Interhost Communication
Transport
End-to-End Connections
and Reliability
Network
Path Determination
and IP (Logical Addressing)
Data Link
MAC and LLC
(Physical addressing)
Physical
Media, Signal, and
Binary Transmission

Summary


Data
Layer
Sockets, http
RPC
TCP
TCP
IP
Ethernet
7
6
5
4
3
2
1
Data
Data
Data
Segments
Packets
Frames
Bits
Application
Network Process to Application
Presentation
Data Representation
and Encryption
Session
Interhost Communication
Transport
End-to-End Connections
and Reliability
Network
Path Determination
and IP (Logical Addressing)
Data Link
MAC and LLC
(Physical addressing)
Physical
Media, Signal, and
Binary Transmission

Summary

Sockets, http

RPC

TCP

TCP

IP

Ethernet

Wired, wireless hardware