CSCI 466: Networks

Link Layer

Reese Pearsall
Fall 2022

All images are stolen from the internet

Announcements

NO CLASS next Friday 11/11

Might have to also cancel next Monday (11/7)*

OSI Model

Application Layer

Presentation Layer *

Session Layer *

Transport Layer

Network Layer

Data Link Layer

Physical Layer

Application Layer

Messages from Network Applications

Physical Layer

Bits being transmitted over a copper wire

**In the textbook, they condense it to a 5-layer model, but 7 layers is what is most used*

Data Link Layer

The link layer is responsible for the **actual node-to-node delivery** of data and ensure error-free transmission of information

terminology:

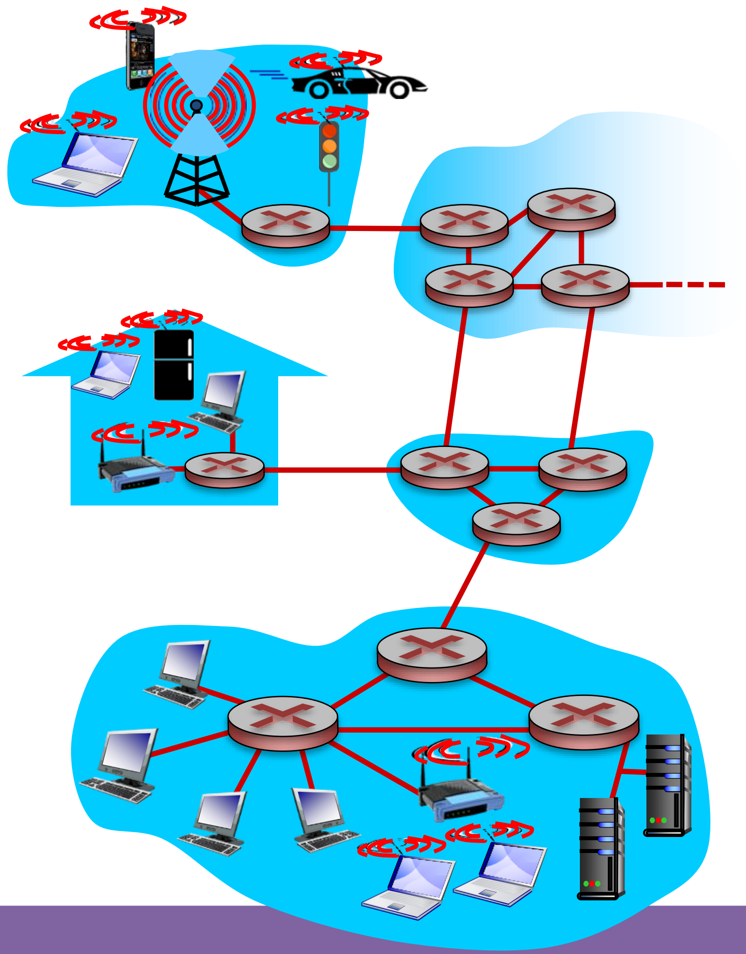
hosts and routers: **nodes**

communication channels that connect adjacent nodes along communication path: **links**

- wired links
- wireless links
- LANs

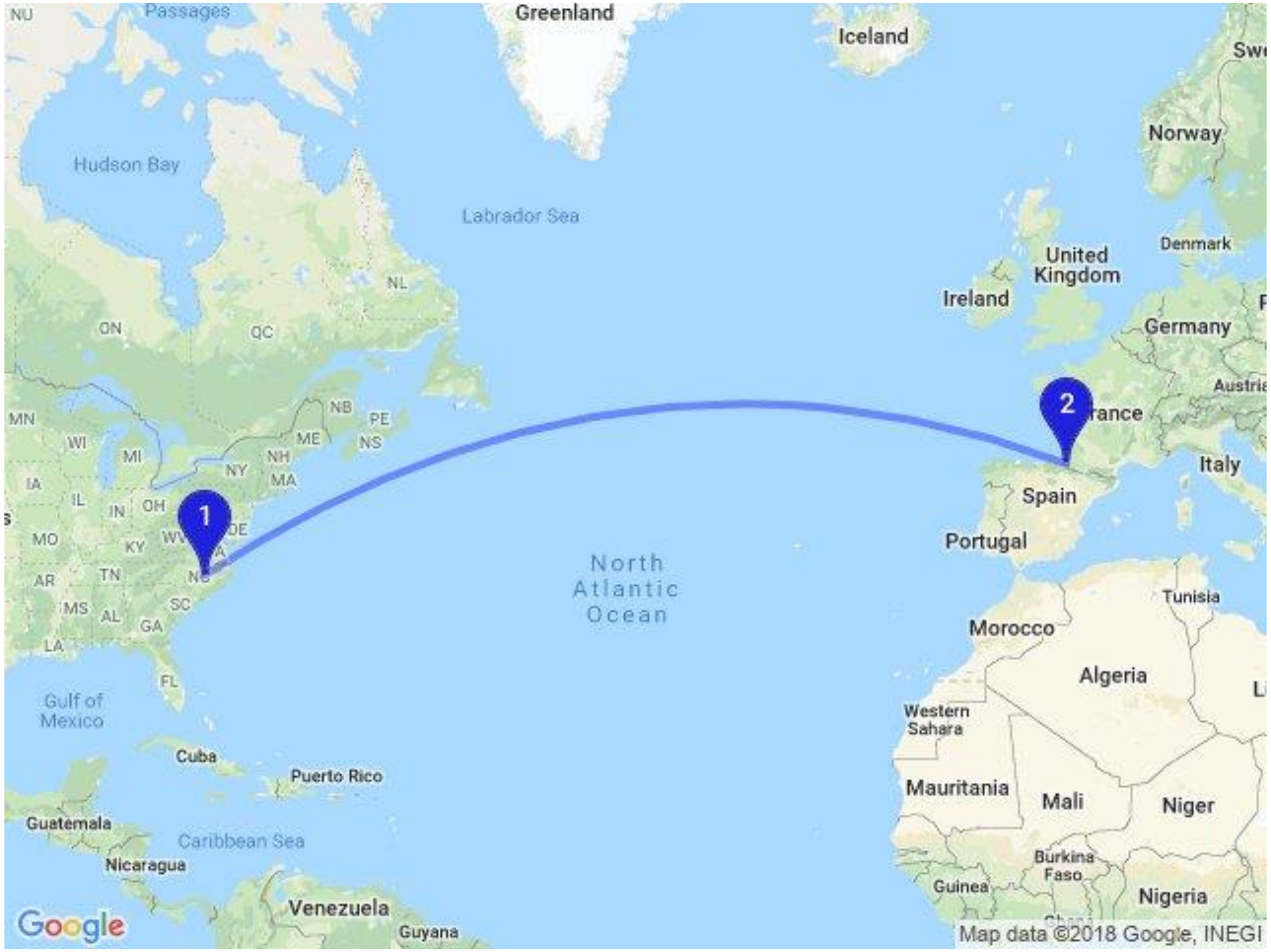
layer-2 packet: frame, encapsulates **datagram**

data-link layer has responsibility of transferring datagram from one node to *physically adjacent* node over a link



We have not addressed how we will overcome various transmission mediums!

Data Link Layer

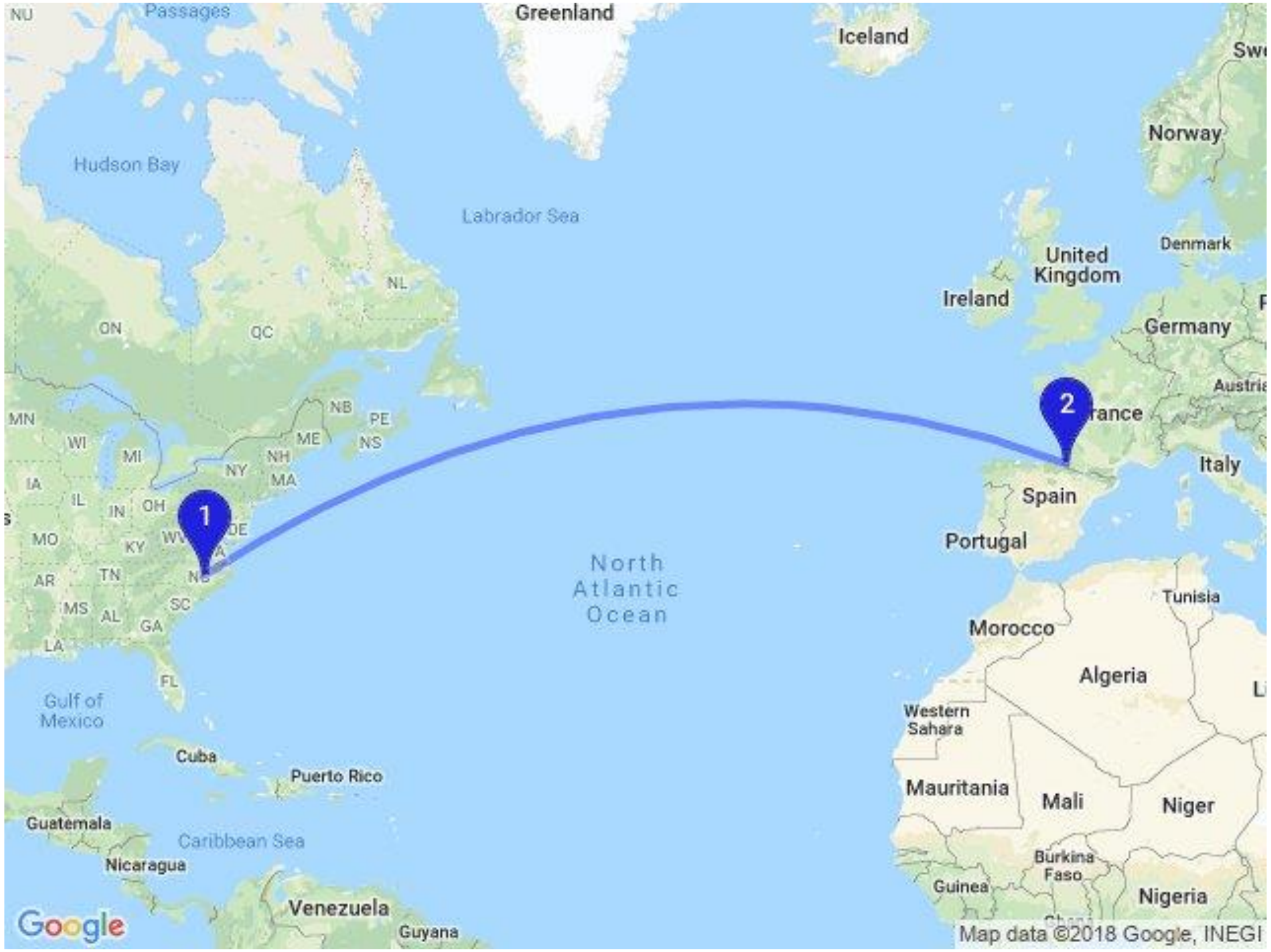


Ways to get form US to Paris?

We can visit a travel agent that will give us a travel plan

1. Take a car to the airport
2. Take a plane to France
3. Take a train from Airport to Paris
4. Take a bus from the train stop to the Eifel tower

Data Link Layer



Ways to get form US to Paris?

We can visit a travel agent that will give us a travel plan

1. Take a car to the airport
2. Take a plane to France
3. Take a train from Airport to Paris
4. Take a bus from the train stop to the Eifel tower

- Tourist = Datagram
- Transportation Segment = Link
- Airport, Bus Stop, Train Stop = Node
- Transport Mode = Link Layer Protocol
- Travel Agent = Routing Protocol (Network Layer)

Data Link Layer

Services offered by the Link Layer

- Framing
  → Encapsulate a network layer Datagram in *another* header
- Link access
  → LL dictate the rules and process of transmitting a frame over a link
- Reliable Delivery
  → For unreliable link, some reliable delivery mechanisms may need to be used
- Error Detection and Correction
  → Bits can get messed up as the are transmitted through a medium

Why do we need RDT and error detection in the link layer when it is also offered in the transport layer?

Data Link Layer

Services offered by the Link Layer

- Framing
  - → Encapsulate a network layer Datagram in *another* header
- Link access
  - → LL dictate the rules and process of transmitting a frame over a link
- Reliable Delivery
  - → For unreliable link, some reliable delivery mechanisms may need to be used
- Error Detection and Correction
  - → Bits can get messed up as the are transmitted through a medium

Why do we need RDT and error detection in the link layer when it is also offered in the transport layer?

Some packets of data don't even travel through the transport layer…

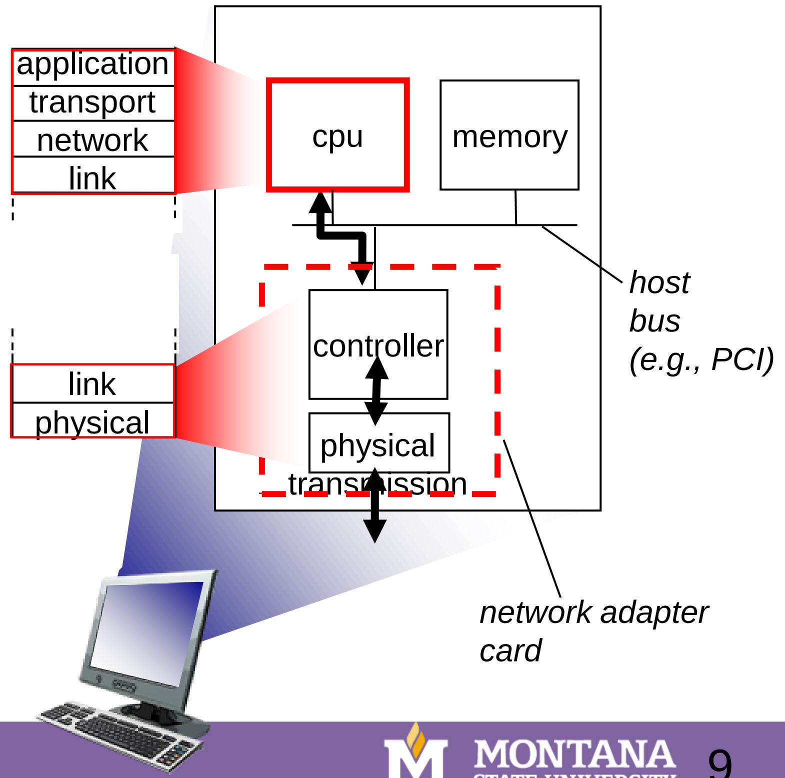
Data Link Layer

Implementation of Link Layer

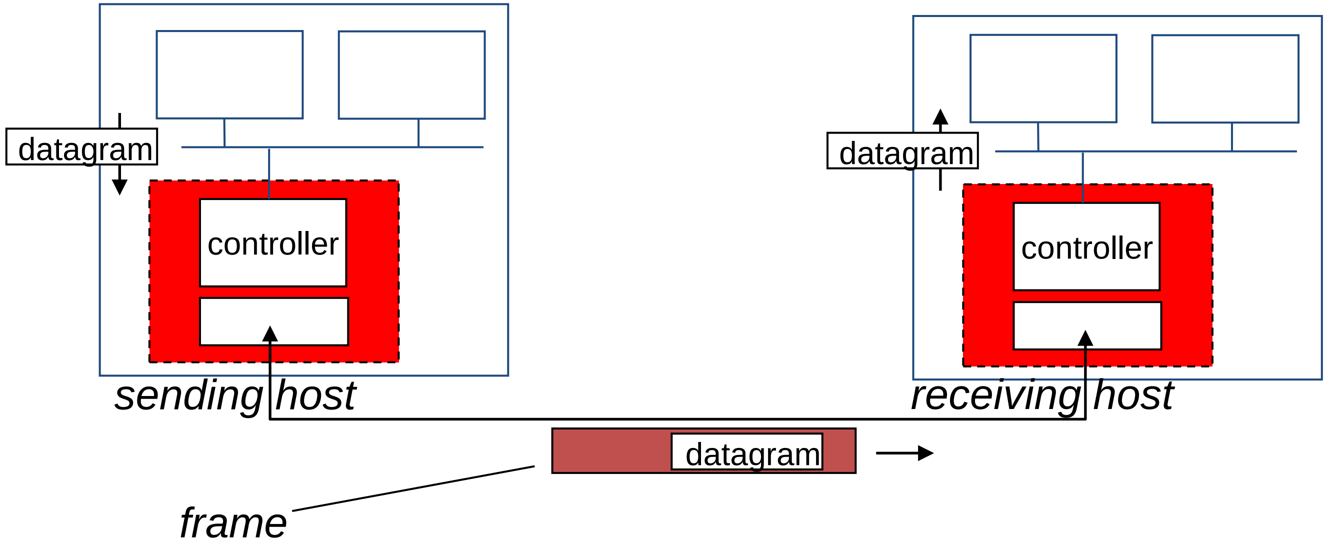
- Implemented within the hardware of your computer

NIC (Network Interface Controller)- Integrated into the motherboard and allows the machine to use LL services such as ethernet (combination of hardware, software, and some firmware)

Wireshark uses your NIC to determine which packets should be sniffed!



Data Link Layer



sending side:

encapsulates datagram in frame
adds error checking bits, rdt, flow control, etc.

receiving side

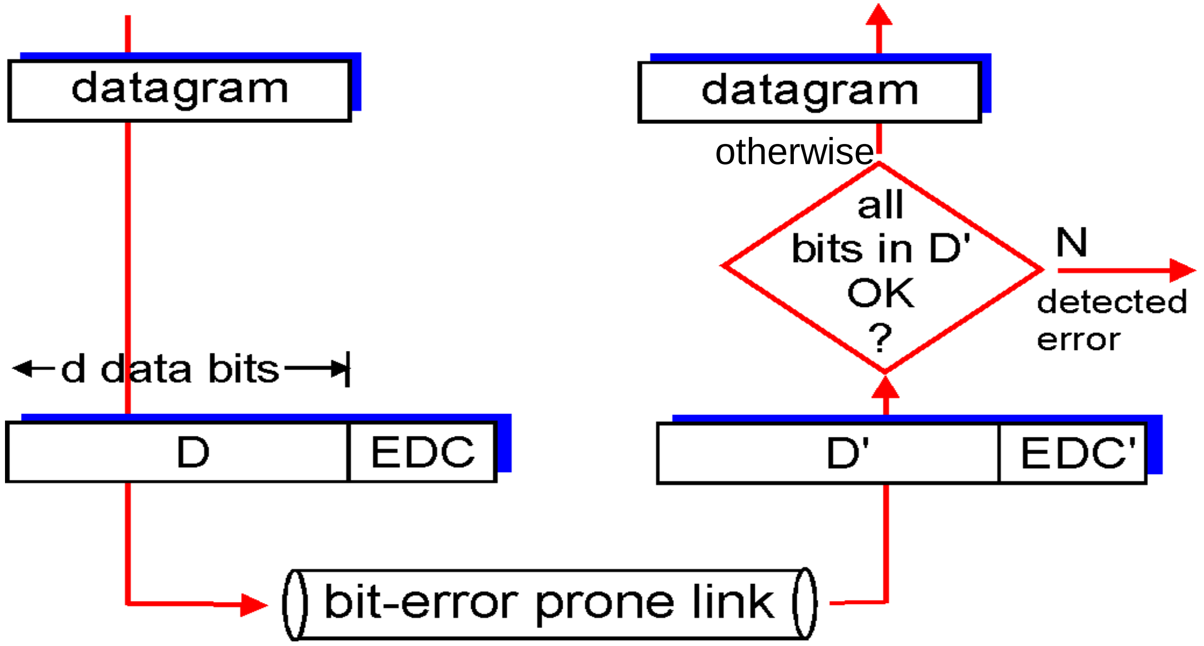
looks for errors, rdt, flow control, etc.
extracts datagram, passes to upper layer at receiving side

Data Link Layer

Bits can get messed during the physical layer and link layer

- Faulty wires
- NIC issues
- Unreliable mediums

The Data Link Layer implements services for **detecting** and **correcting errors**

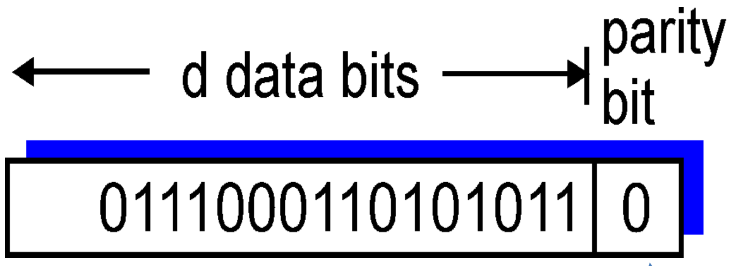


EDC= Error Detection and Correction bits
D = Data protected by error checking, may include header fields

- Error detection not 100% reliable!
- protocol may miss some errors, but rarely
- larger EDC field yields better detection and correction

Single bit parity:

Detect single bit errors



Data

Is the number of 1s odd (0) or even (1) ? (Including the parity bit)

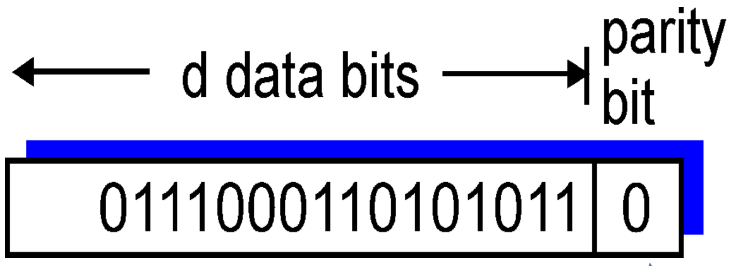
0111000**1**001010110

The receiver counts eight 1s, but the parity bit tells us it should be an odd number of 1s →
ERROR DETECTED

Data Link Layer

Single bit parity:

Detect single bit errors



Data

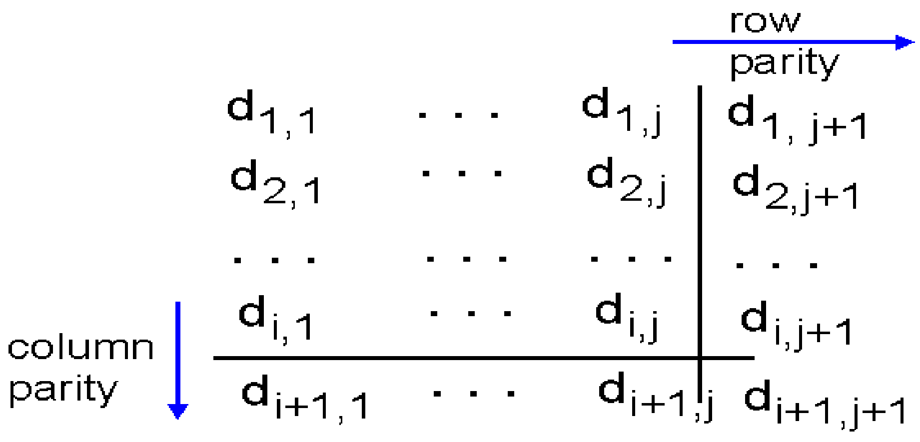
Is the number of 1s odd (0) or even (1) ? (Including the parity bit)

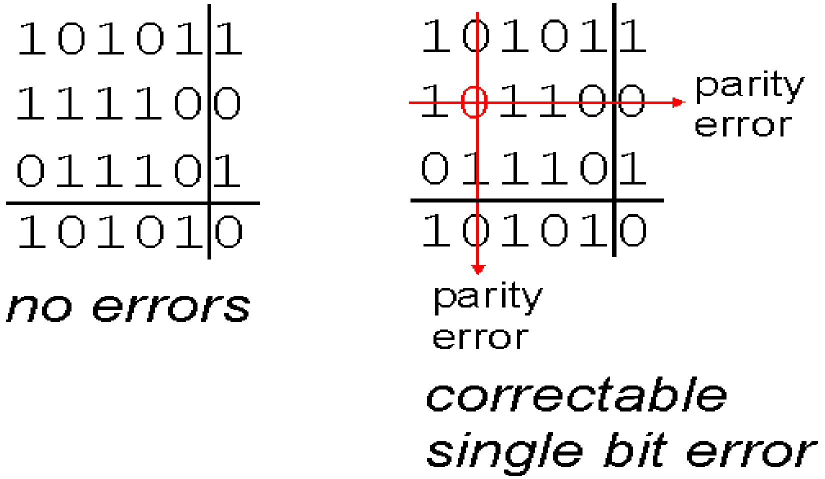
0111000100101011 0

The receiver counts eight 1s, but the parity bit tells us it should be an odd number of 1s →
ERROR DETECTED

Two-dimensional bit parity:

Detect and **correct** single bit errors





Data Link Layer

Checksum

(Sender)

$$
\begin{array}{r}
0110011001100000 \\
+\ 0101010101010101 \\
+\ 1000111100001100 \\
\hline
\mathbf{0100101011000010}
\end{array}
$$

Binary sum of words

(one's complement)

1011010100111101 Checksum!

(Receiver)

$$
\begin{array}{r}
0110011001100000 \\
+\ 0101010101010101 \\
+\ 1000111100001100 \\
+\ 0100101011000010 \\
\hline
\end{array}
$$

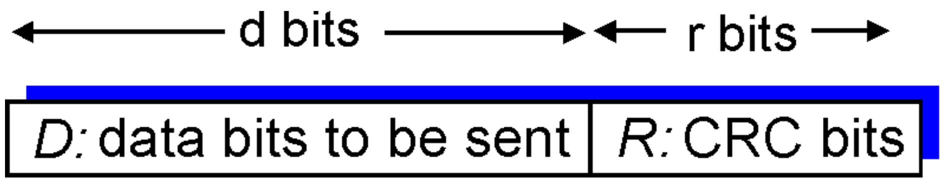
(Binary Sum → One's Complement)

= 1111111111111111

All 1s = No error!

Data Link Layer

- more powerful error-detection coding
- view data bits, D, as a binary number
- choose r+1 bit pattern (generator), G
- goal: choose r CRC bits, R, such that
  - <D,R> exactly divisible by G (modulo 2)
  - receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
  - can detect all burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi, ATM)



bit pattern

$D * 2^r$ XOR R

mathematical formula

Sender/Receiver has D and G. Need to compute R

Data Link Layer

- more powerful error-detection coding
- view data bits, D, as a binary number
- choose r+1 bit pattern (generator), G
- goal: choose r CRC bits, R, such that
  - <D,R> exactly divisible by G (modulo 2)
  - receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
  - can detect all burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi, ATM)

← d bits → ← r bits →

D: data bits to be sent	*R:* CRC bits

bit pattern

$$D * 2^r \quad \text{XOR} \quad R$$

mathematical formula

Sender/Receiver has D and G. Need to compute R

Data Link Layer

← d bits → ← r bits →

D: data bits to be sent	*R:* CRC bits

bit pattern

$D * 2^r$ XOR R *mathematical formula*

(Do some algebra to find R)

$$R = remainder[\frac{D \cdot 2^r}{G}]$$

Sender sends D + R bits.

Receiver divides D + R bits by G. Result should always be Zero if no errors occured

```
   G
                1 0 1 0 1 1
1 0 0 1 ) 1 0 1 1 1 0 0 0 0
          1 0 0 1               D
            1 0 1
            0 0 0
            1 0 1 0
            1 0 0 1
              1 1 0
              0 0 0
              1 1 0 0
              1 0 0 1
                1 0 1 0
                1 0 0 1
                  0 1 1
                    R
```

Data Link Layer

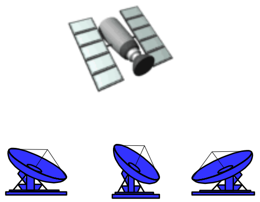
Access links

- Point to Point – Single sender, Single Receiver at each end of link

- Broadcast – shared medium

shared wire (e.g., cabled Ethernet)

shared RF (e.g., 802.11 WiFi)



shared RF (satellite)

humans at a cocktail party (shared air, acoustical)

MAC (Media Access Control) Addresses

32-bit IP address:

network-layer address for interface

used for layer 3 (network layer) forwarding

MAC (or LAN or physical or Ethernet) address:

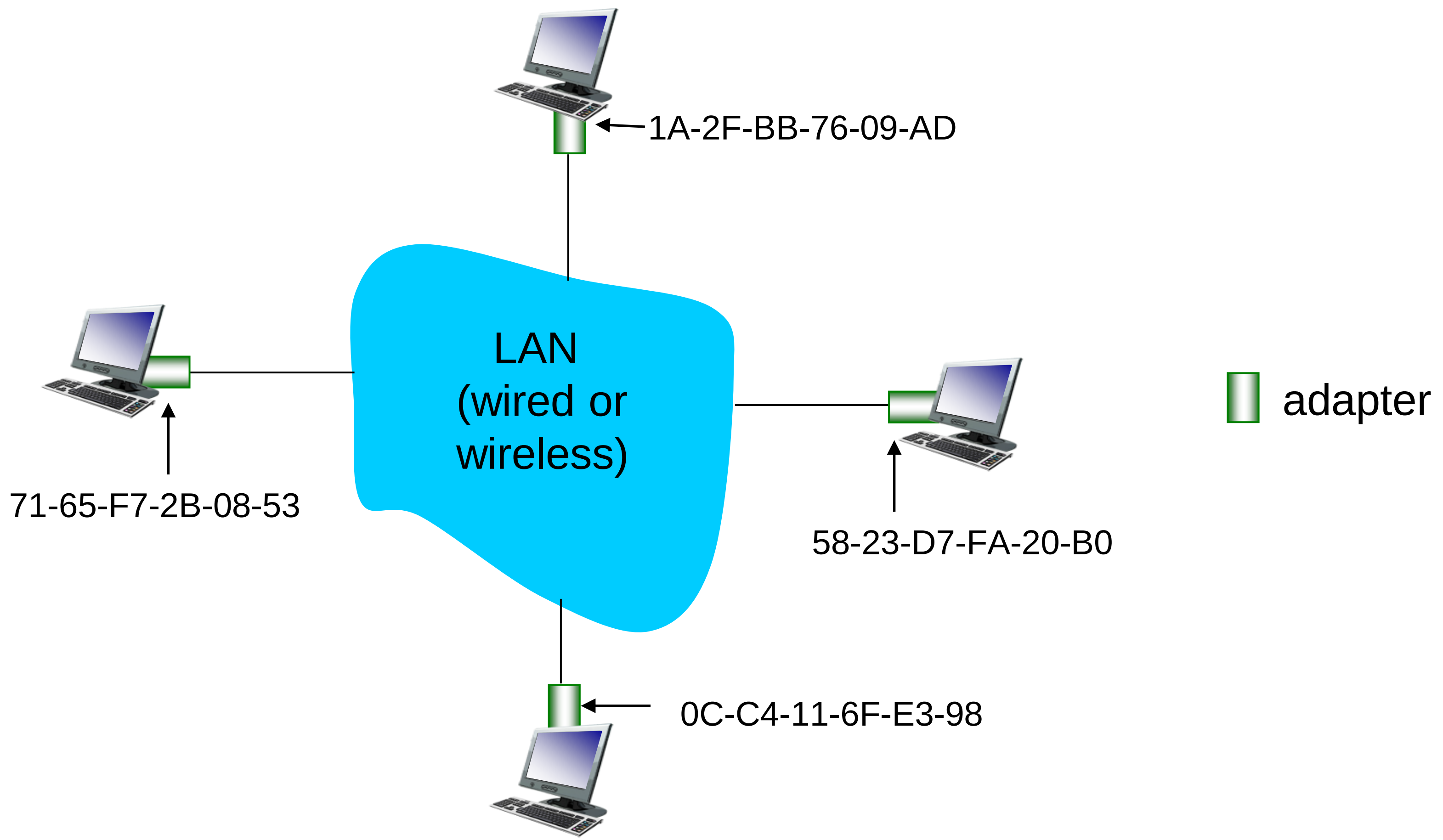
- function: *used 'locally" to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense)*
- 48 bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
- e.g.: 1A-2F-BB-76-09-AD

hexadecimal (base 16) notation
(each "numeral" represents 4 bits)

How do we know two NICs wont have the same MAC address?

MAC Address is your SSN, IP address is your Postal code ☺

Data Link Layer



Announcements

NO CLASS next Friday 11/11

Monday November 4^{th} = WORK DAY (No lecture)

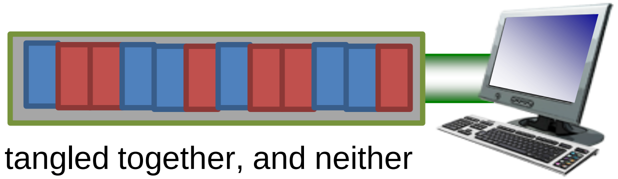
Input files + sample output for PA3 are on the website

Multiple Access Links

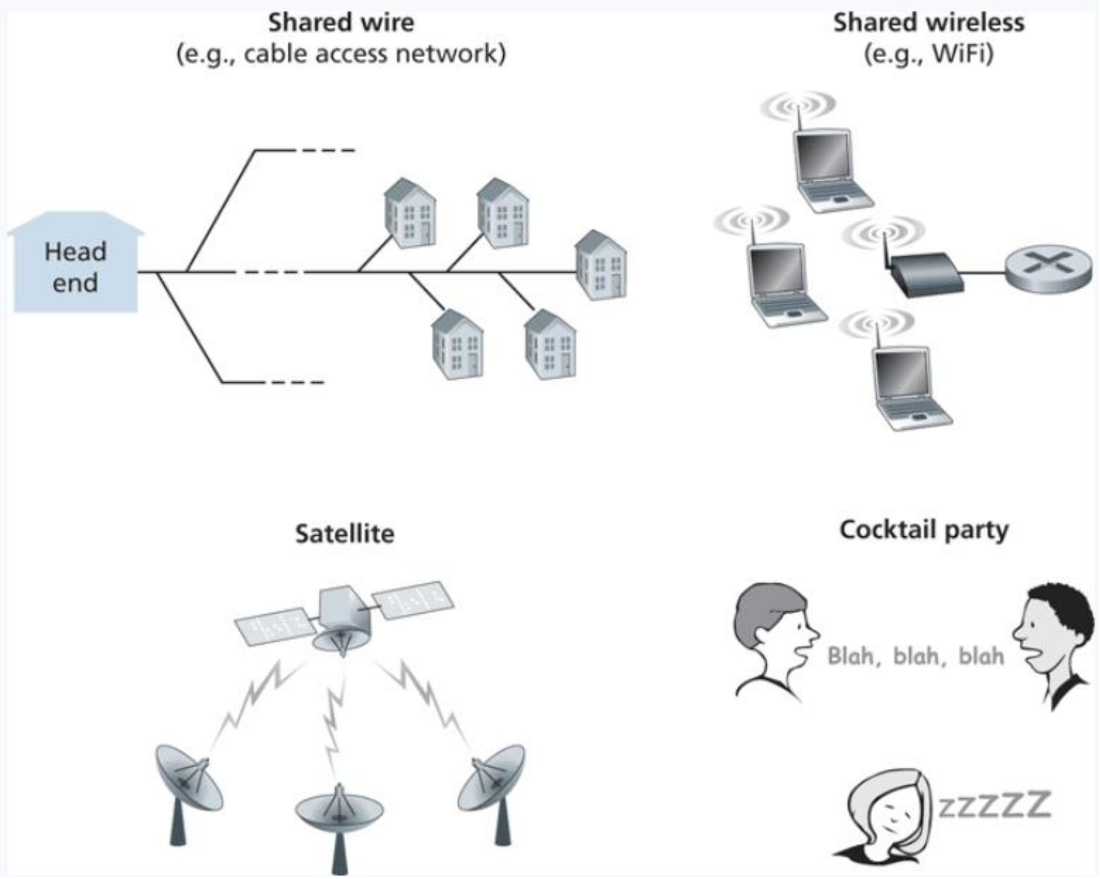
Shared medium = possibility for receivers to get two frame at the same time, AKA a **collision**

Frame X

Frame y

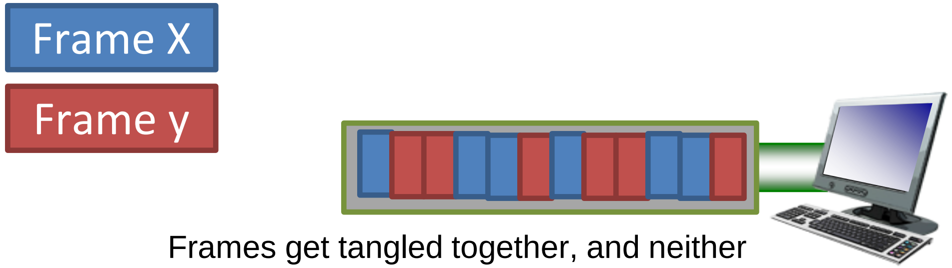


Frames get tangled together, and neither frame can be read clearly by receiver



Multiple Access Links

Shared medium = possibility for receivers to get two frame at the same time, AKA a **collision**



Frames get tangled together, and neither frame can be read clearly by receiver

"Give everyone a chance to speak."

"Don't speak until you are spoken to."

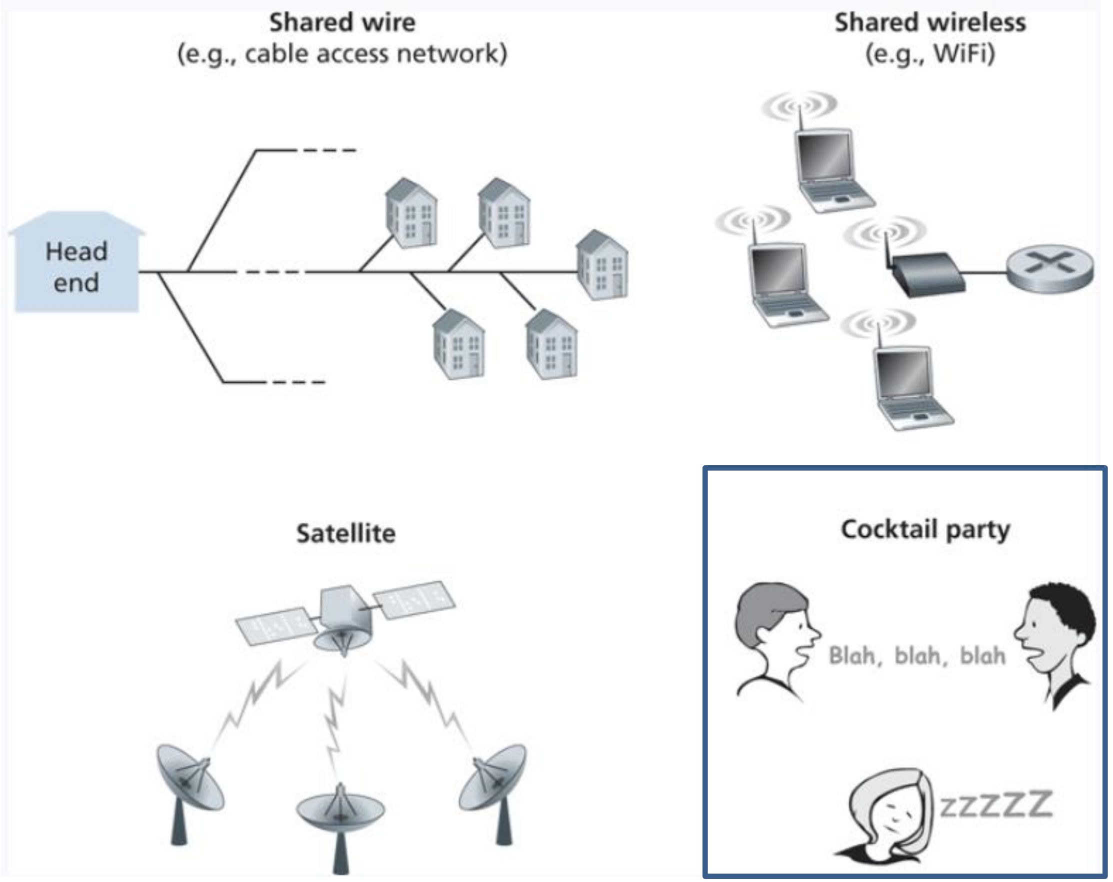
"Don't monopolize the conversation."

"Raise your hand if you have a question."

"Don't interrupt when someone is speaking."

"Don't fall asleep when someone is talking."

In English, we have some rules to prevent collisions from happening



In the link layer, we will discuss 3 multiple access protocols: **Channel Partitioning**, **Random Access**, and **Taking Turns**

Multiple Access Links

Shared medium = possibility for receivers to get two frame at the same time, AKA

Frame X

Frame y

Fran
fram

Ideally

1. When one node wants to transmit, it can send at rate R:
2. When M nodes want to transmit, each can send at average rate R/M
3. Fully decentralized:
   - no special node to coordinate transmissions
   - no synchronization of clocks, slots
4. simple

"Give everyor

"Don't speak

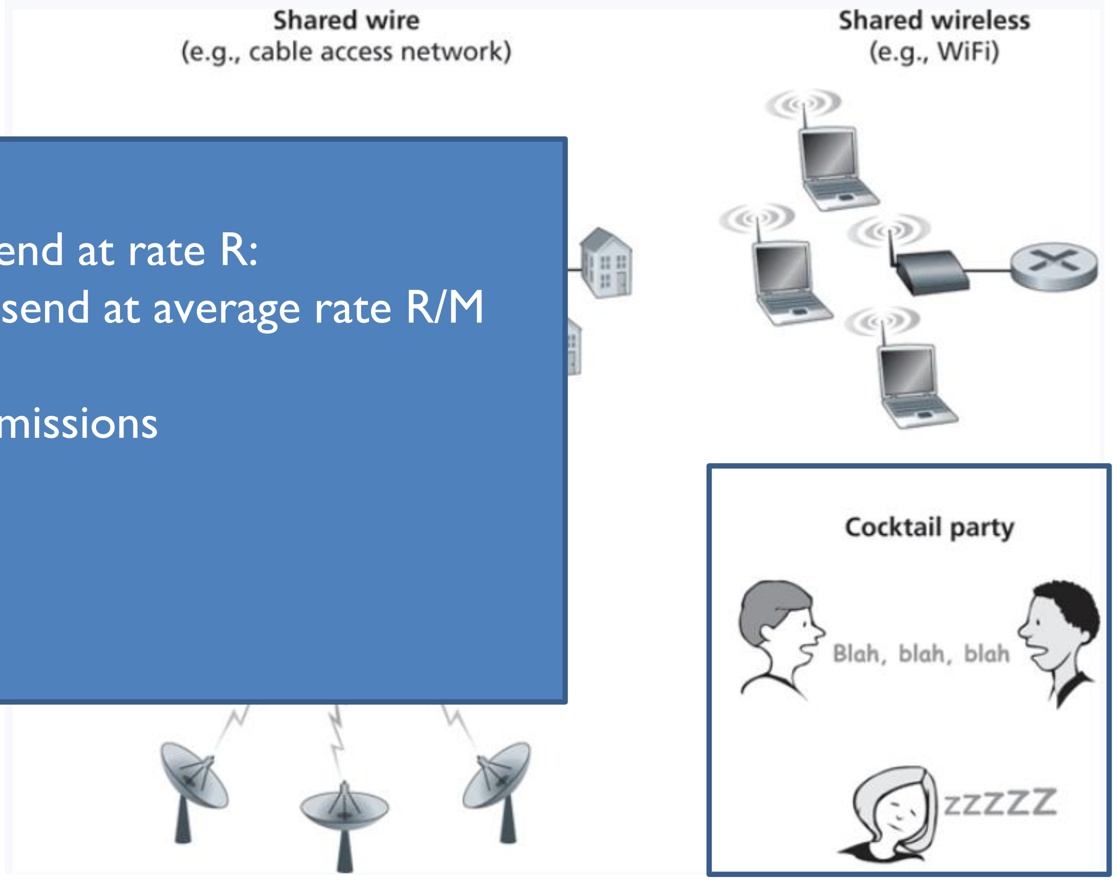
"Don't monopolize the conversation."

"Raise your hand if you have a question."

"Don't interrupt when someone is speaking."

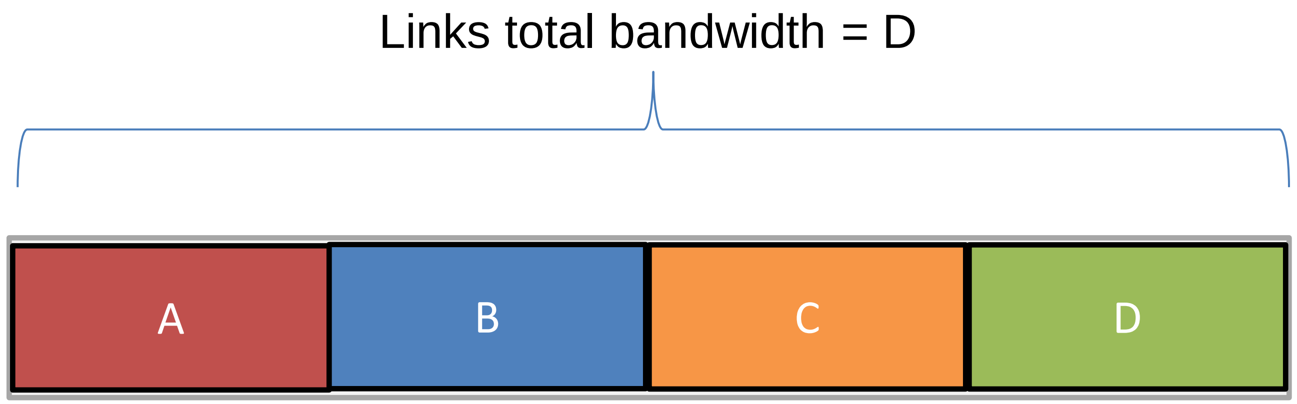
"Don't fall asleep when someone is talking."

have some rules to prevent collisions from happening

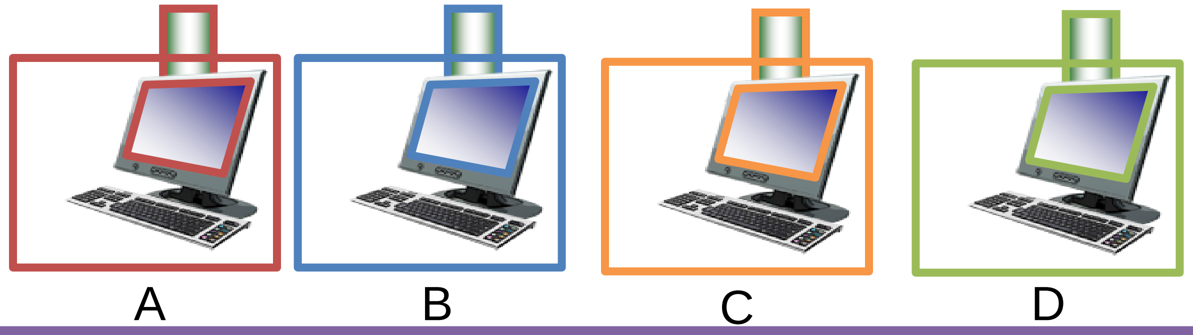


In the link layer, we will discuss 3 multiple access protocols: **Channel Partitioning**, **Random Access**, and **Taking Turns**

Channel Partitioning

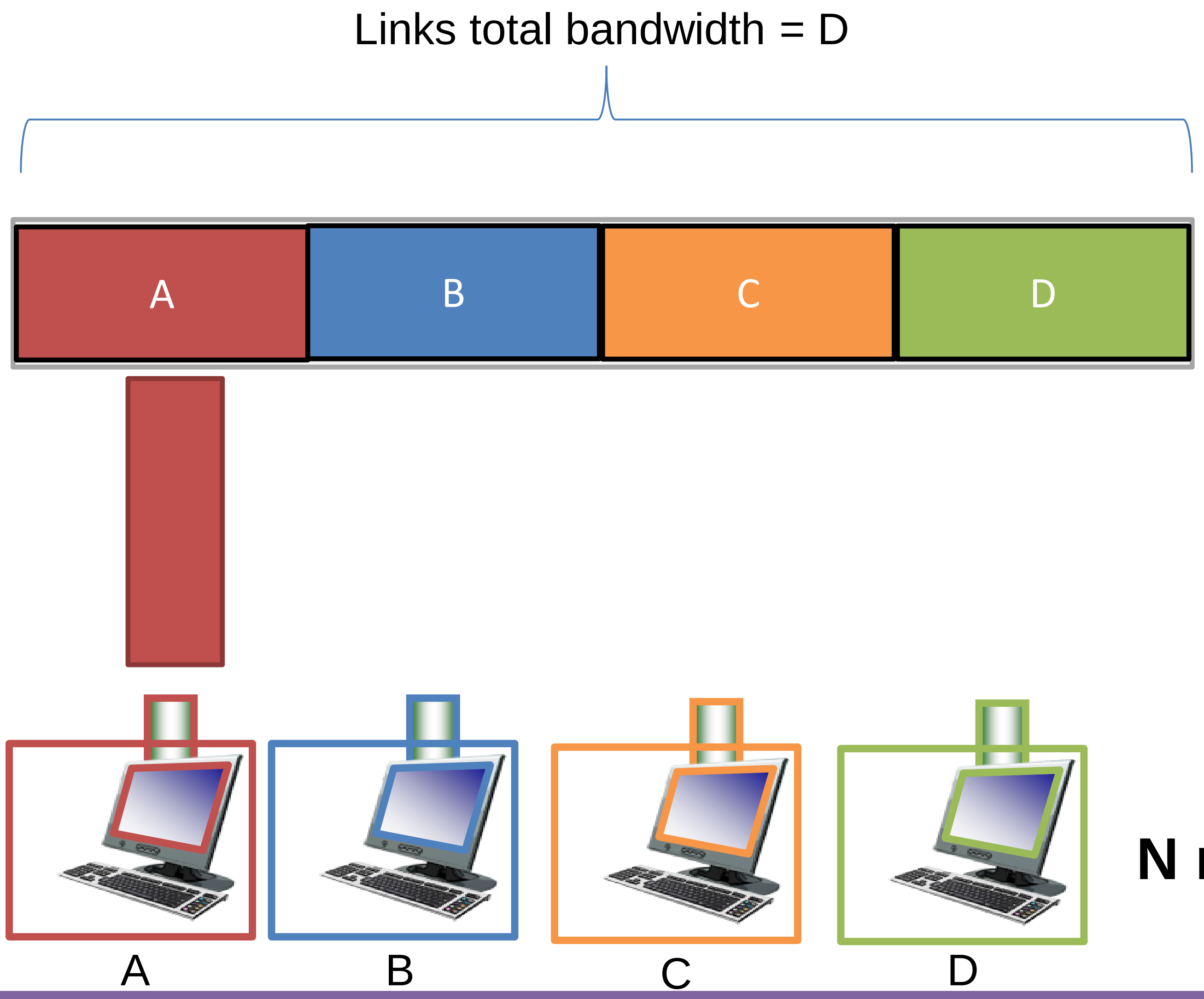


- Divide channel into **N** slots
- Each node gets (on average) D/N bandwidth
- Get to transmit data for a fixed amount of time, and then next node gets to transmit



N nodes

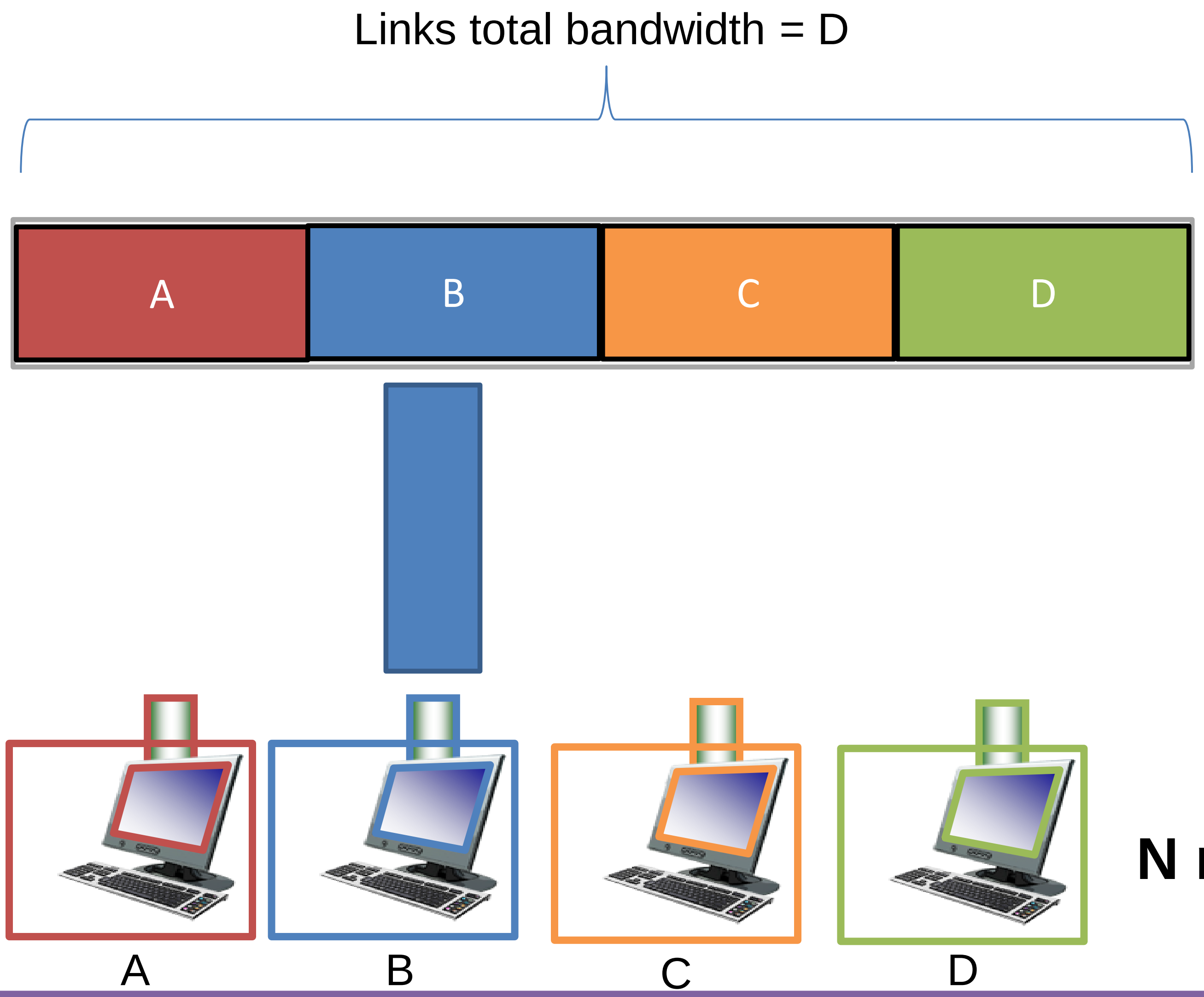
Channel Partitioning



N nodes

- Divide channel into **N** slots
- Each node gets (on average) D/N bandwidth
- Get to transmit data for a fixed amount of time, and then next node gets to transmit

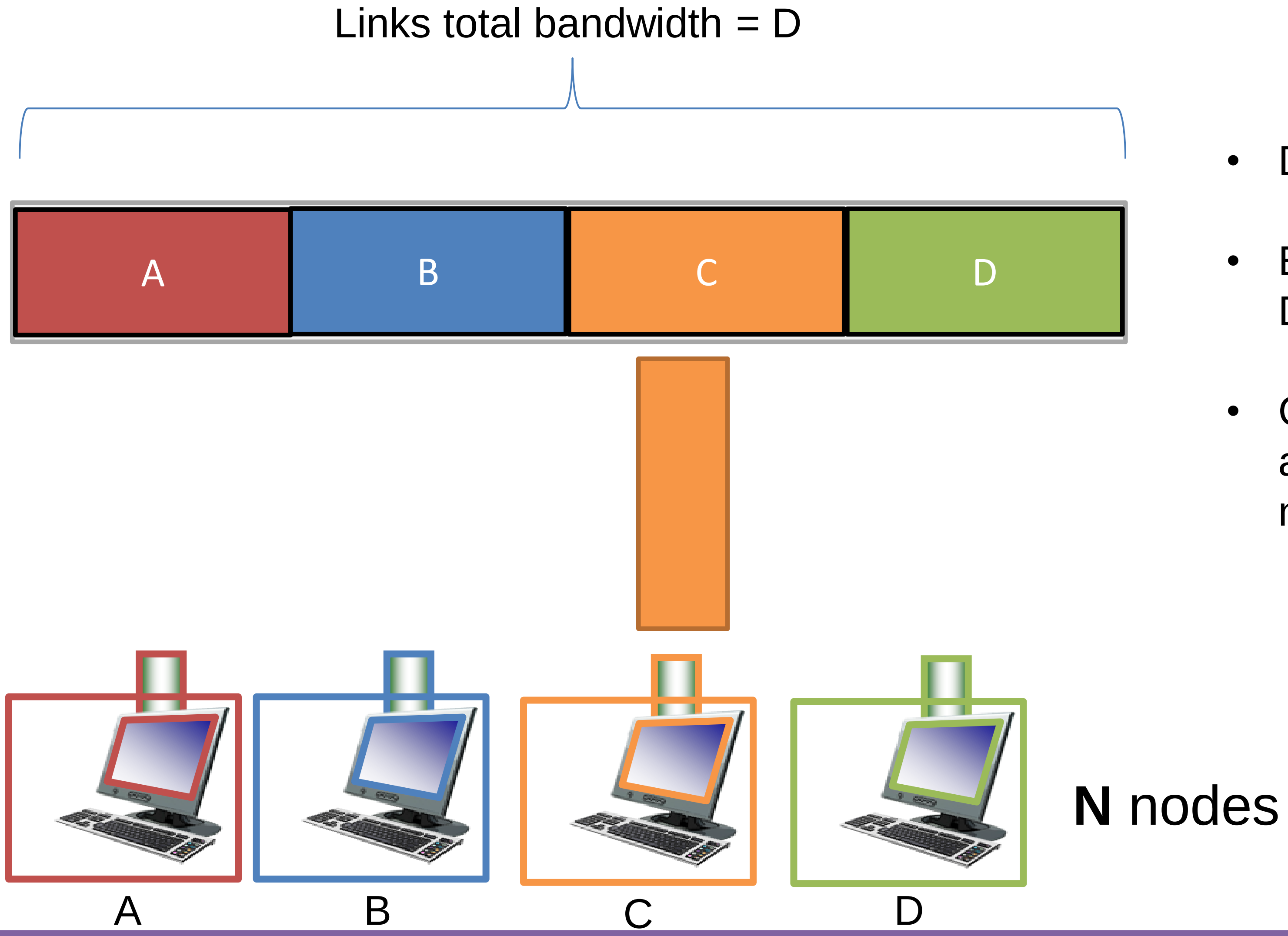
Channel Partitioning



N nodes

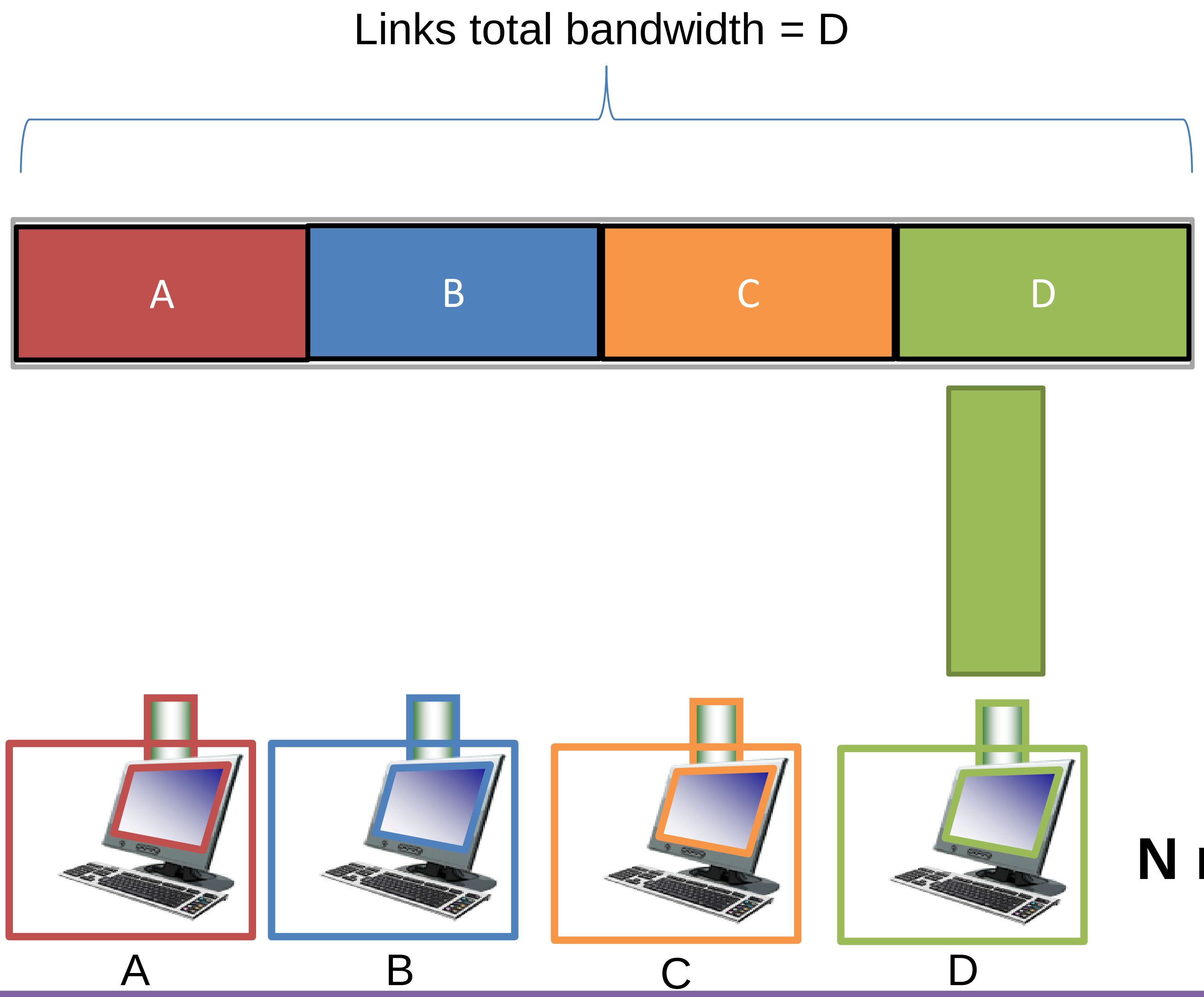
- Divide channel into **N** slots
- Each node gets (on average) D/N bandwidth
- Get to transmit data for a fixed amount of time, and then next node gets to transmit

Channel Partitioning



- Divide channel into **N** slots
- Each node gets (on average) D/N bandwidth
- Get to transmit data for a fixed amount of time, and then next node gets to transmit

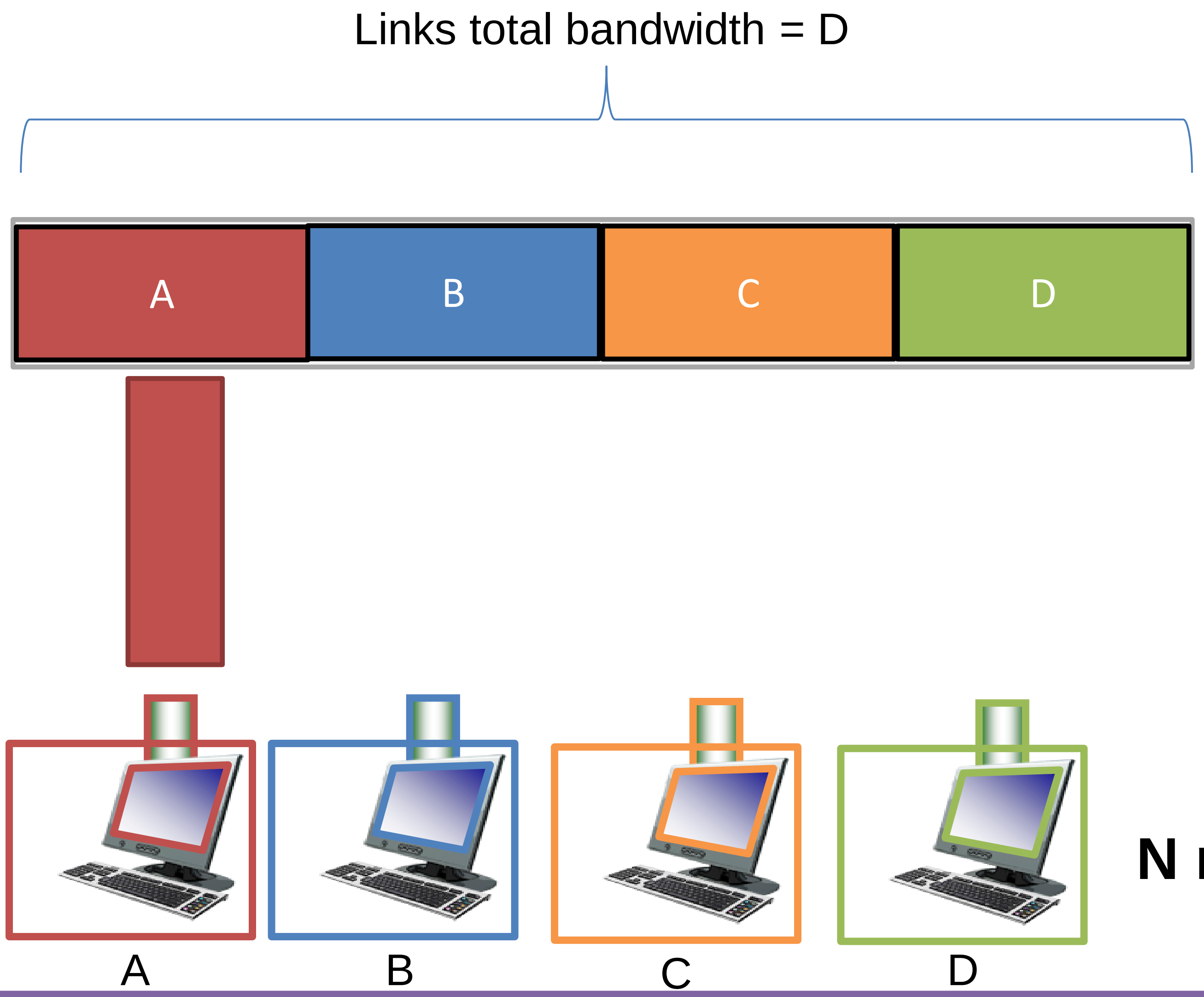
Channel Partitioning



- Divide channel into **N** slots
- Each node gets (on average) D/N bandwidth
- Get to transmit data for a fixed amount of time, and then next node gets to transmit

N nodes

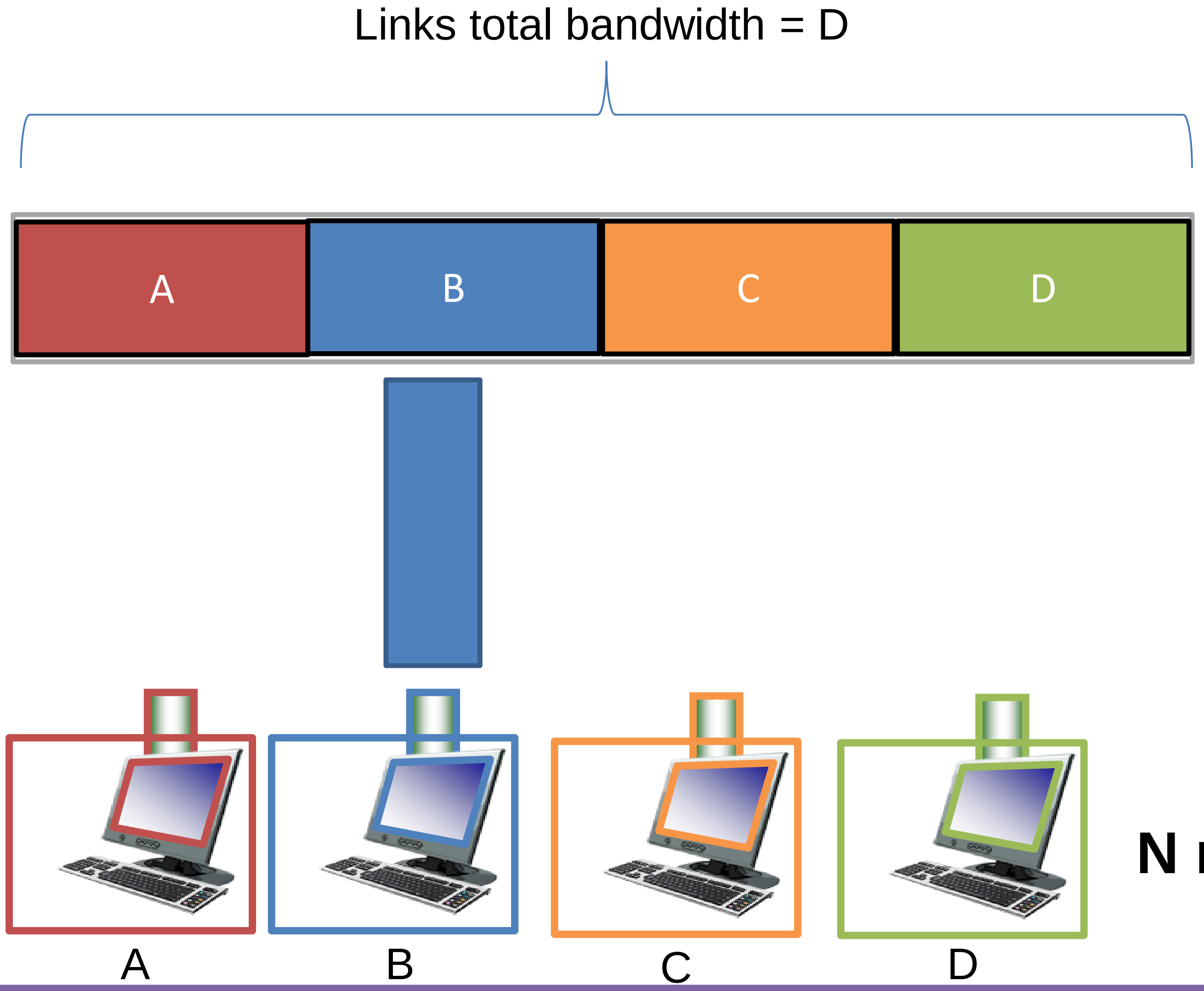
Channel Partitioning



N nodes

- Divide channel into **N** slots
- Each node gets (on average) D/N bandwidth
- Get to transmit data for a fixed amount of time, and then next node gets to transmit

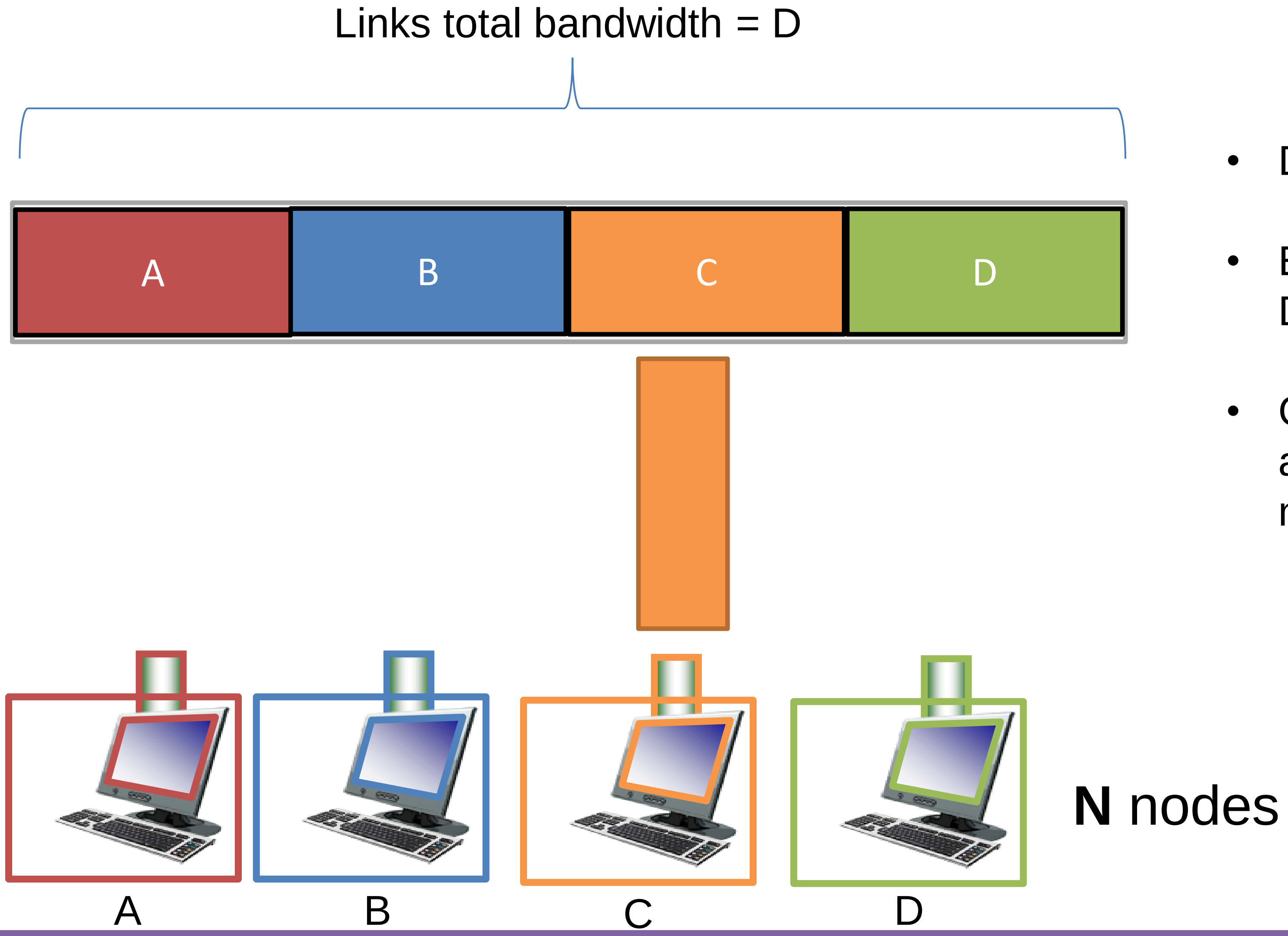
Channel Partitioning



N nodes

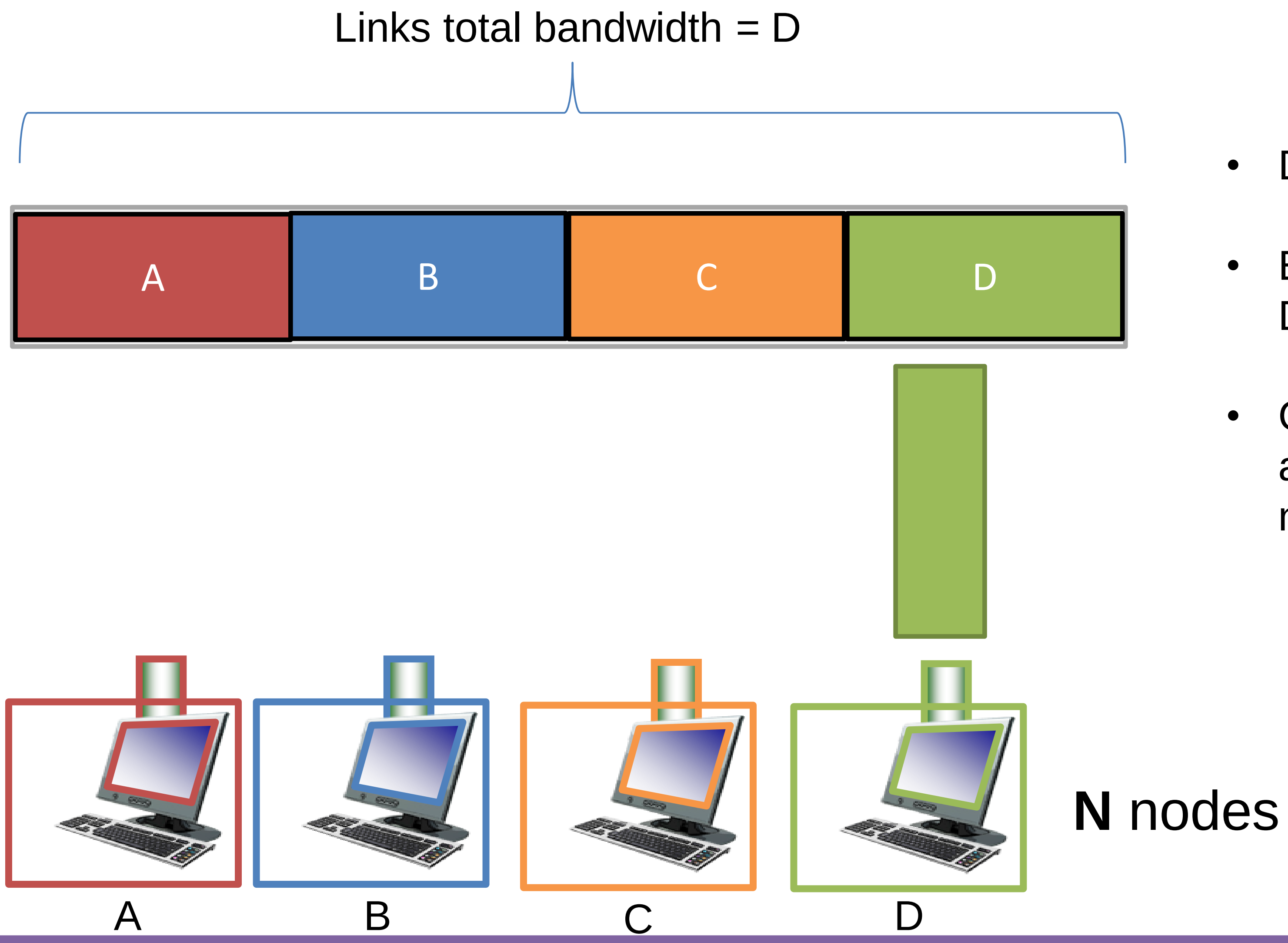
- Divide channel into **N** slots
- Each node gets (on average) D/N bandwidth
- Get to transmit data for a fixed amount of time, and then next node gets to transmit

Channel Partitioning



- Divide channel into **N** slots
- Each node gets (on average) D/N bandwidth
- Get to transmit data for a fixed amount of time, and then next node gets to transmit

Channel Partitioning



- Divide channel into **N** slots
- Each node gets (on average) D/N bandwidth
- Get to transmit data for a fixed amount of time, and then next node gets to transmit

Random Access

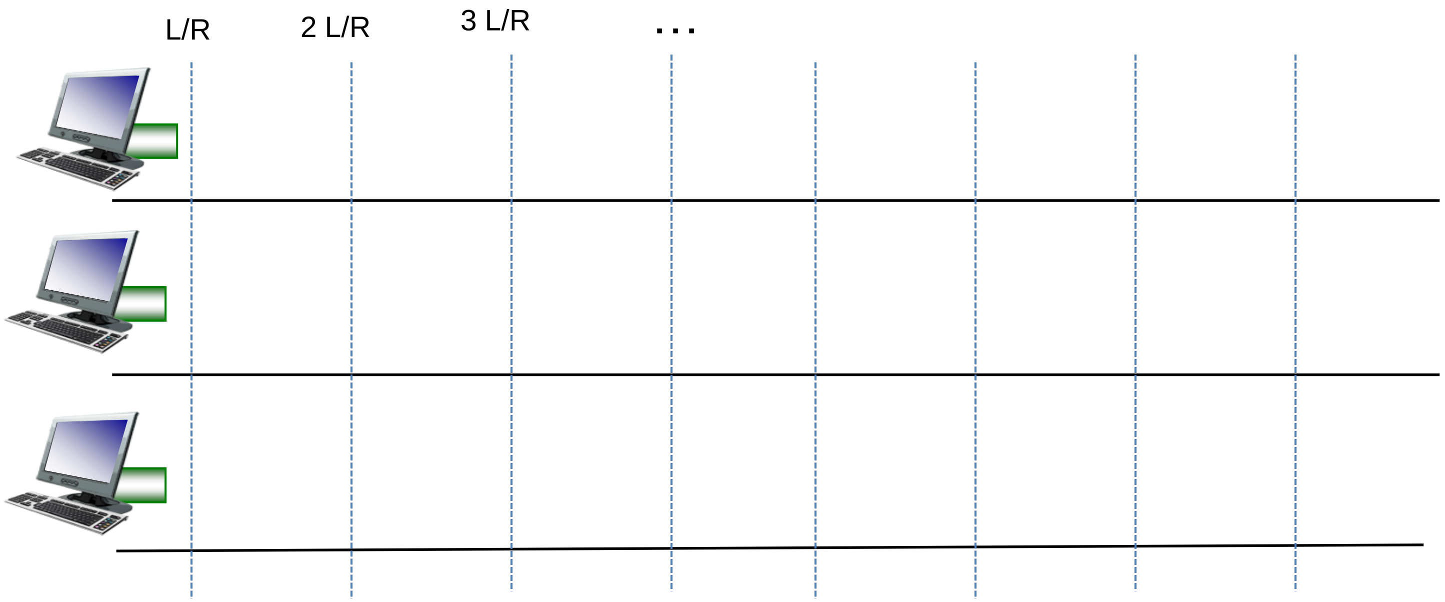
Collisions will occur, but we will try to *recover* from them

Slotted ALOHA: Divide up time into discrete L/R "slots"

If collisions occur, the colliding nodes will flip a coin to see who should retransmit

L = size of frame
R = Bandwidth

L/R = Time needed to transmit one frame



Can only transmit frames at beginning of slots. If collision occurs, the nodes can detect collision before the slot ends

Random Access

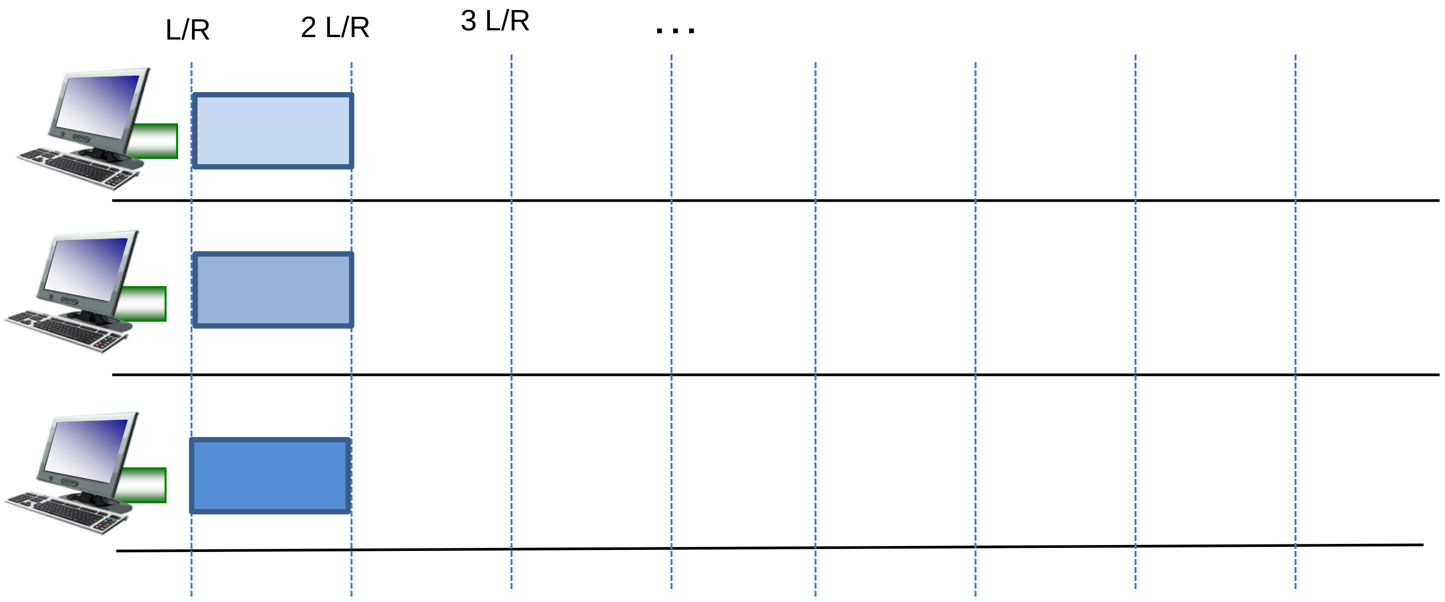
Collisions will occur, but we will try to *recover* from them

Slotted ALOHA: Divide up time into discrete L/R "slots"

If collisions occur, the colliding nodes will flip a coin to see who should retransmit

L = size of frame
R = Bandwidth

L/R = Time needed to transmit one frame



Collision

Random Access

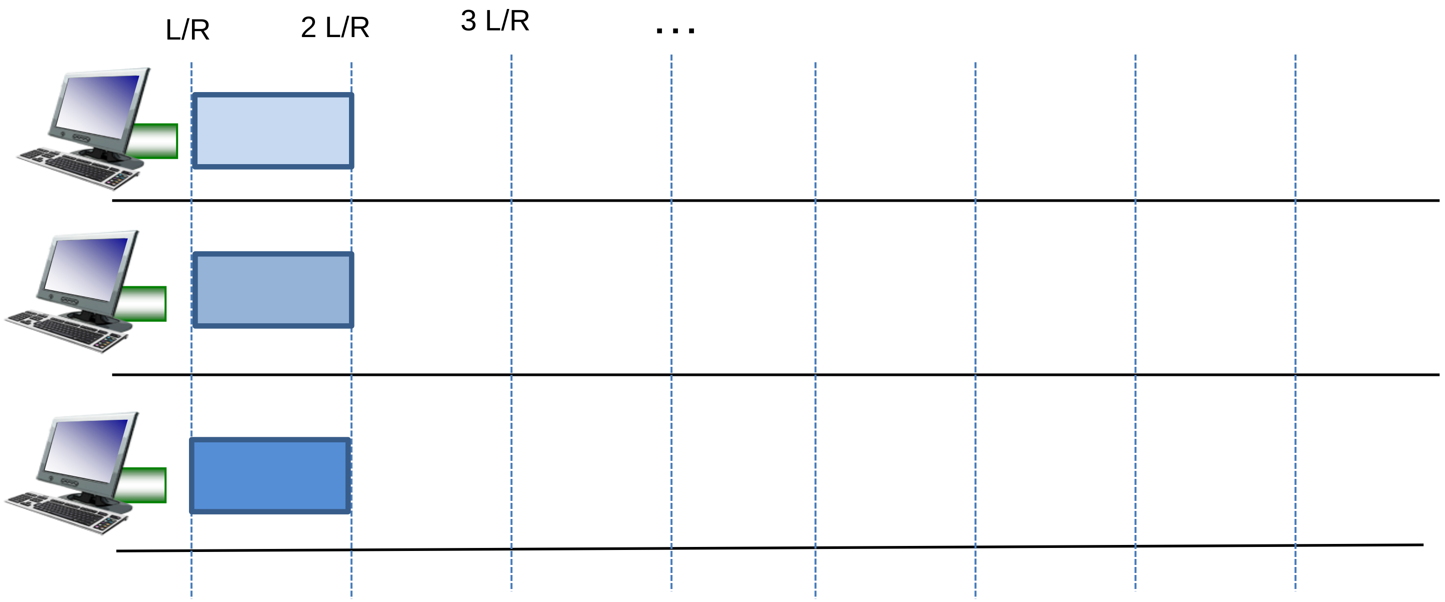
Collisions will occur, but we will try to *recover* from them

Slotted ALOHA: Divide up time into discrete L/R "slots"

If collisions occur, the colliding nodes will flip a coin to see who should retransmit

L = size of frame
R = Bandwidth

L/R = Time needed to transmit one frame



Collision

Do some probability ***p*** and retransmit if needed

Random Access

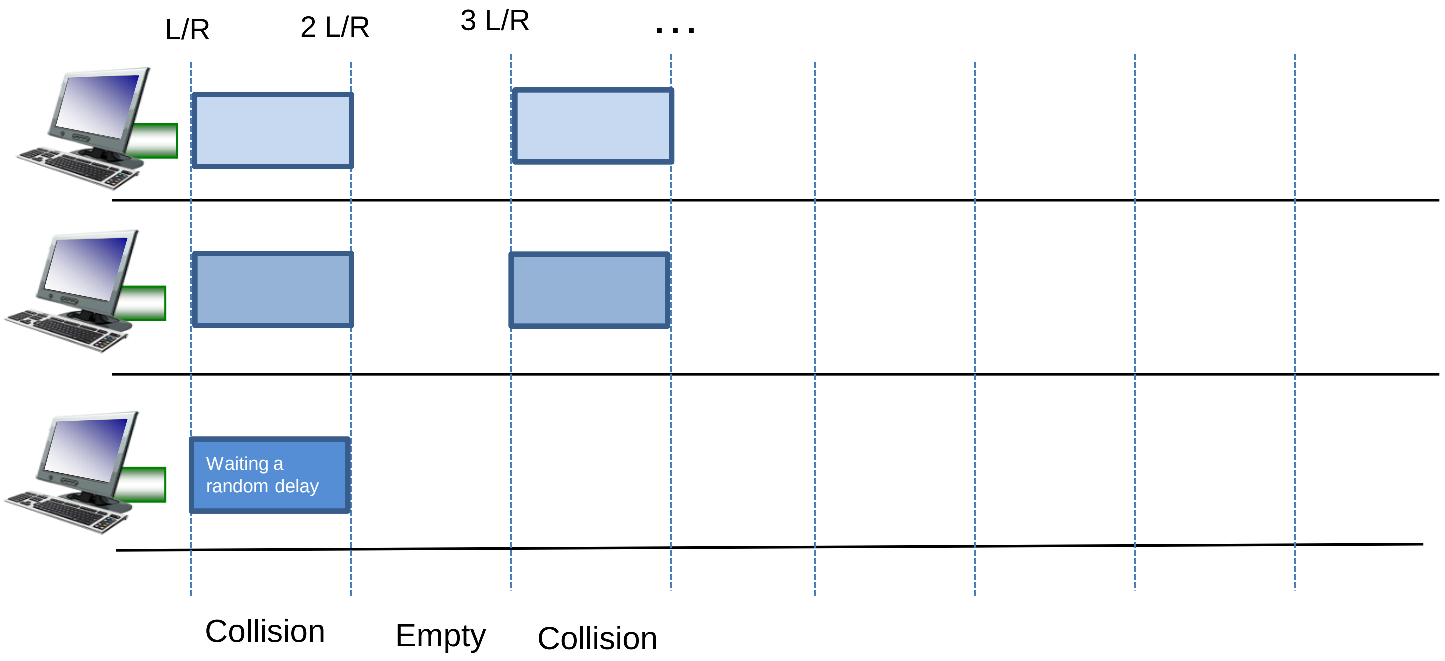
Collisions will occur, but we will try to *recover* from them

Slotted ALOHA: Divide up time into discrete L/R “slots”

If collisions occur, the colliding nodes will flip a coin to see who should retransmit

L = size of frame
R = Bandwidth

L/R = Time needed to transmit one frame



Random Access

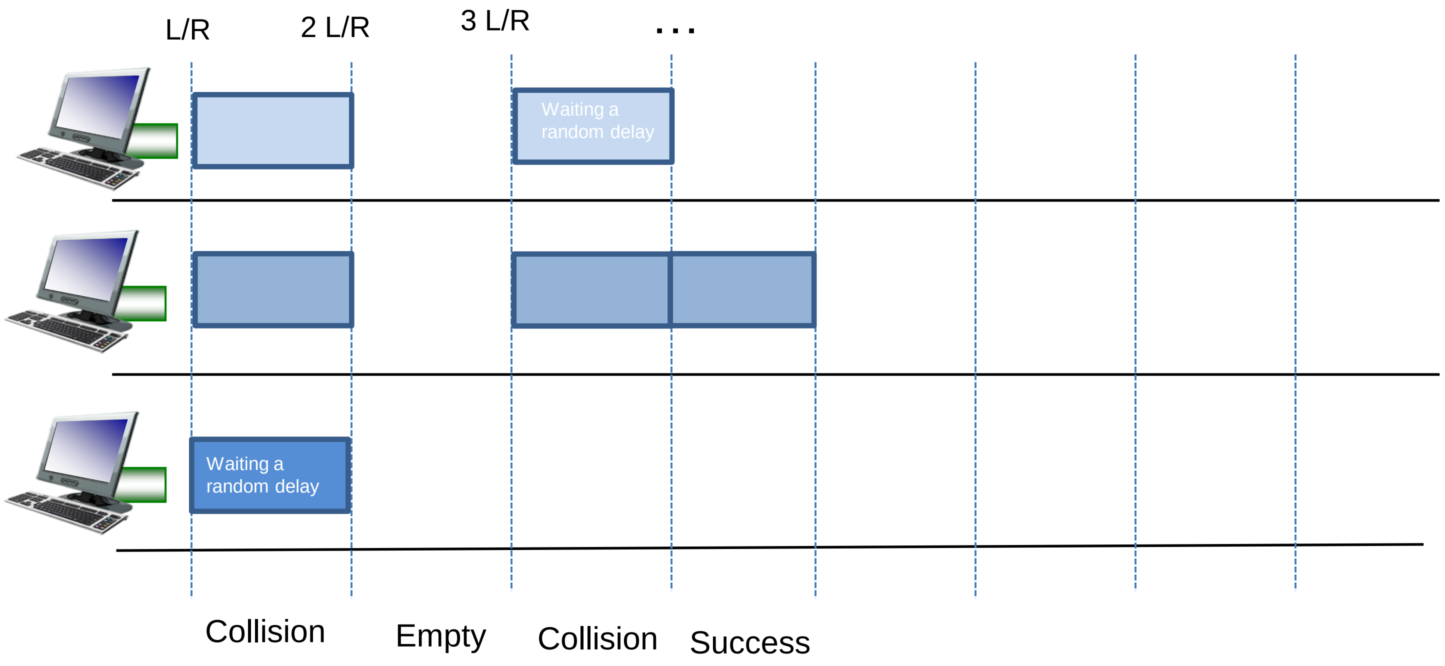
Collisions will occur, but we will try to *recover* from them

Slotted ALOHA: Divide up time into discrete L/R "slots"

If collisions occur, the colliding nodes will flip a coin to see who should retransmit

L = size of frame
R = Bandwidth

L/R = Time needed to transmit one frame



Random Access

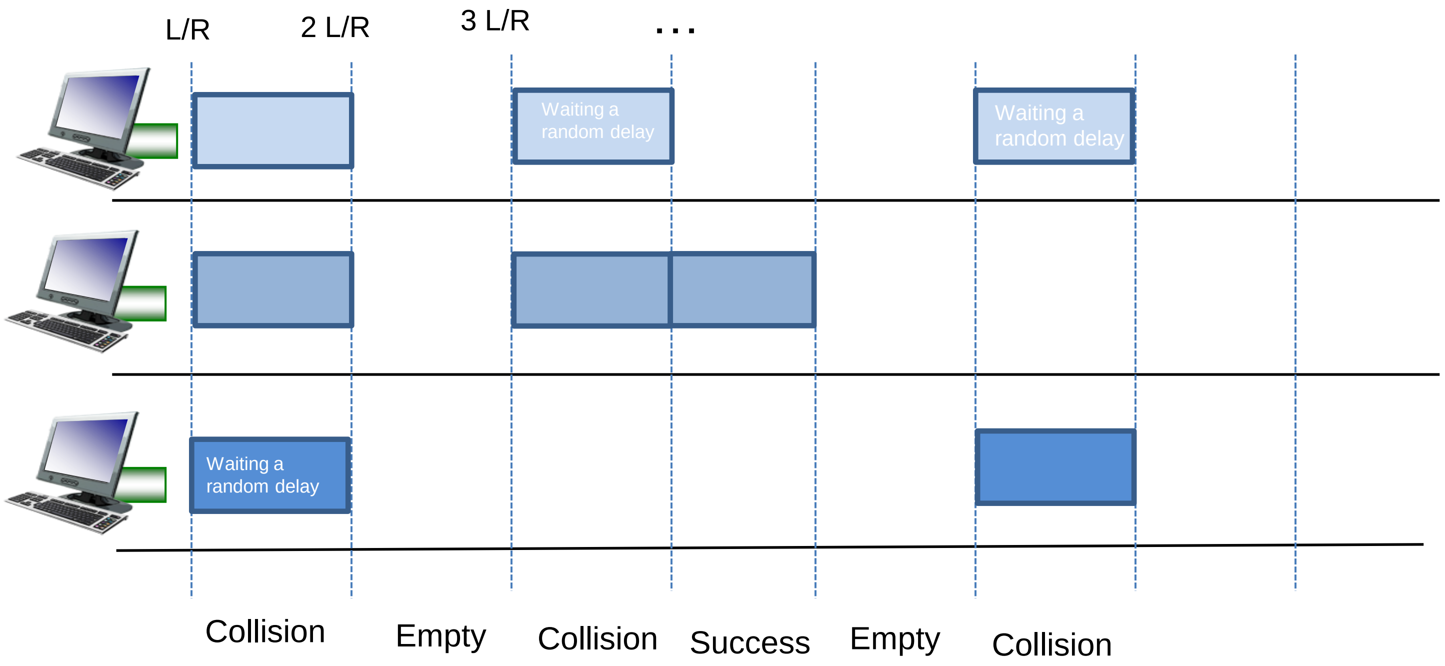
Collisions will occur, but we will try to *recover* from them

Slotted ALOHA: Divide up time into discrete L/R “slots”

If collisions occur, the colliding nodes will flip a coin to see who should retransmit

L = size of frame
R = Bandwidth

L/R = Time needed to transmit one frame



Random Access

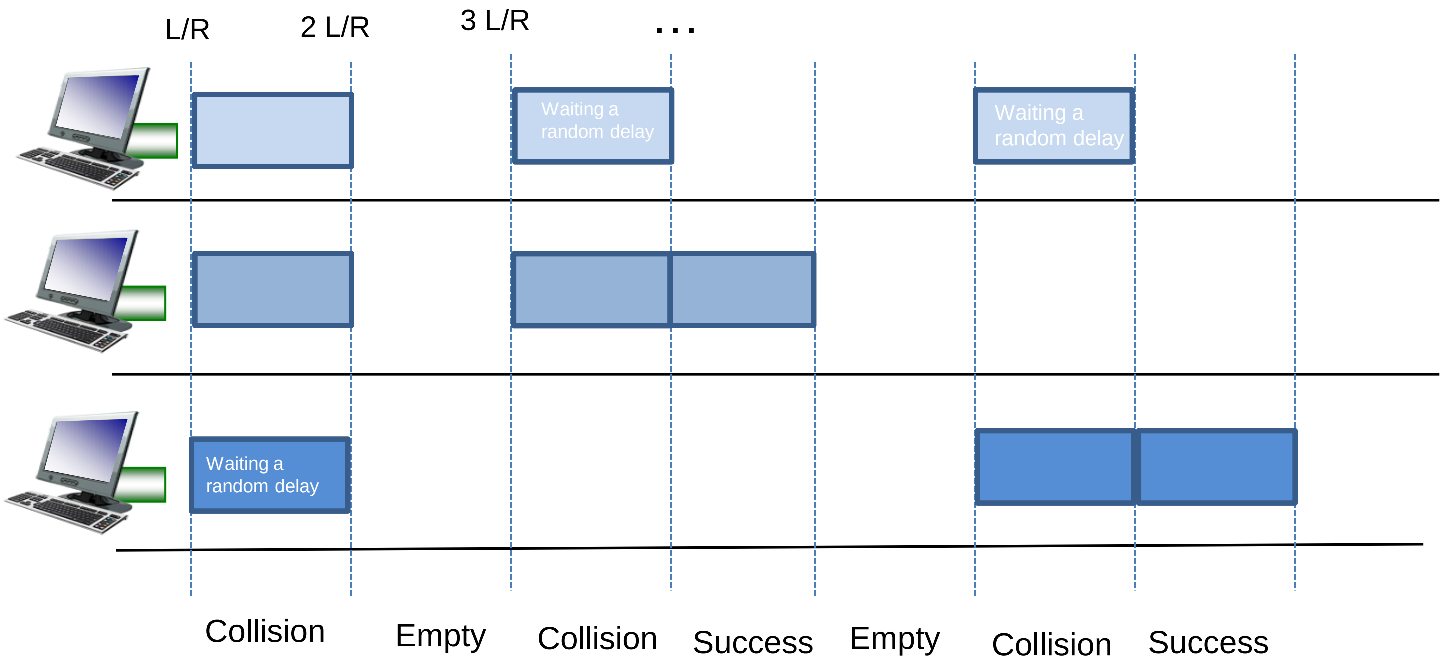
Collisions will occur, but we will try to *recover* from them

Slotted ALOHA: Divide up time into discrete L/R "slots"

If collisions occur, the colliding nodes will flip a coin to see who should retransmit

L = size of frame
R = Bandwidth

L/R = Time needed to transmit one frame



Random Access

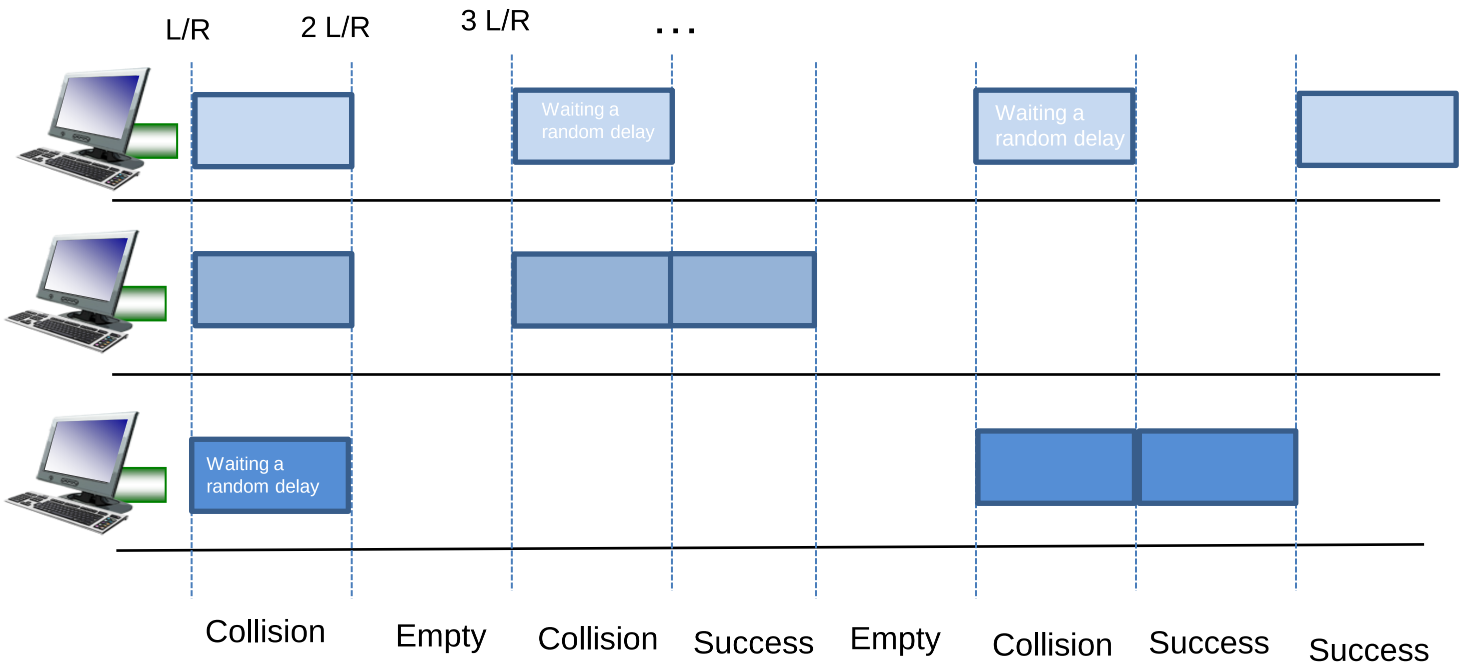
Collisions will occur, but we will try to *recover* from them

Slotted ALOHA: Divide up time into discrete L/R “slots”

If collisions occur, the colliding nodes will flip a coin to see who should retransmit

L = size of frame
R = Bandwidth

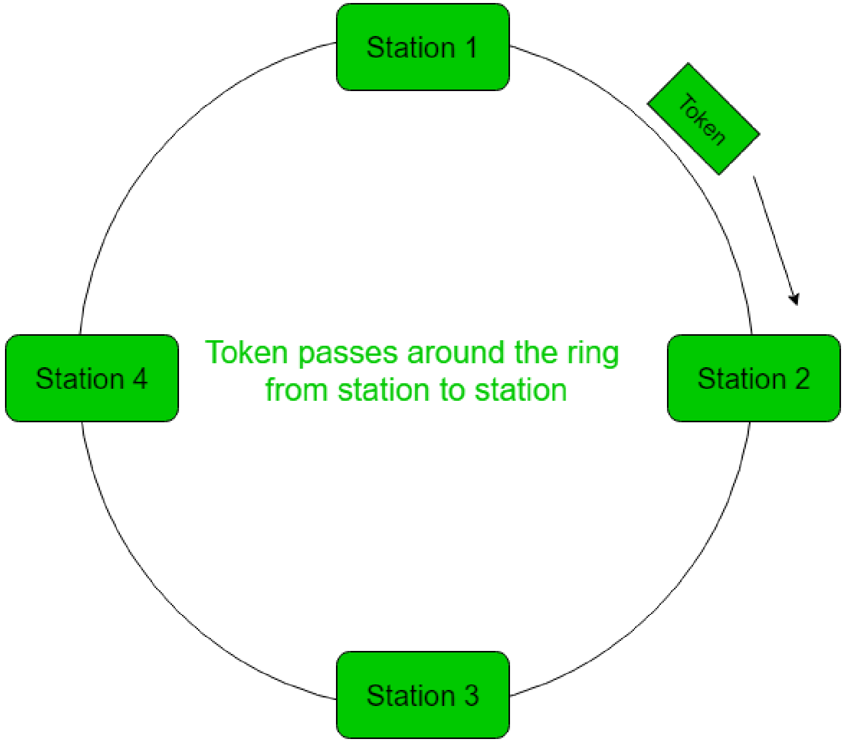
L/R = Time needed to transmit one frame



Taking Turns

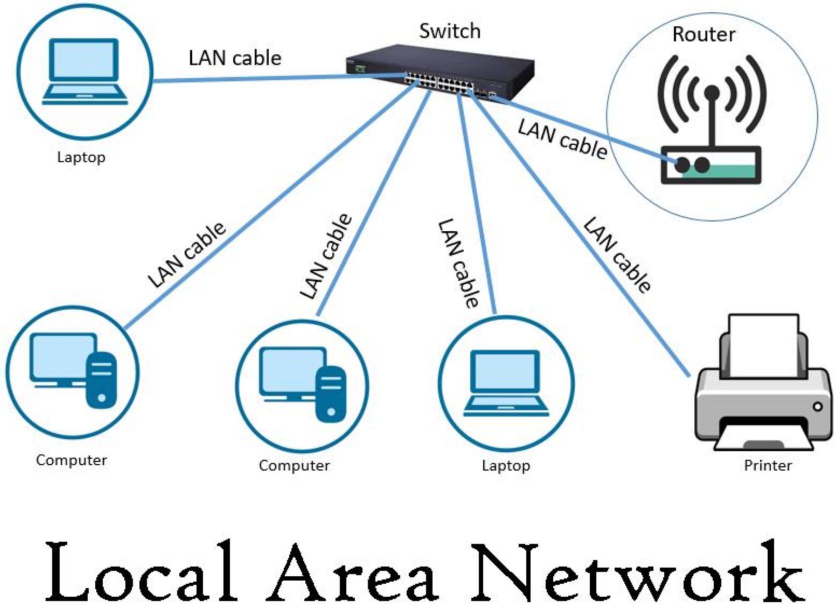
Token Passing
Nodes are connected in a circular manner, and pass a special frame (token) between each other

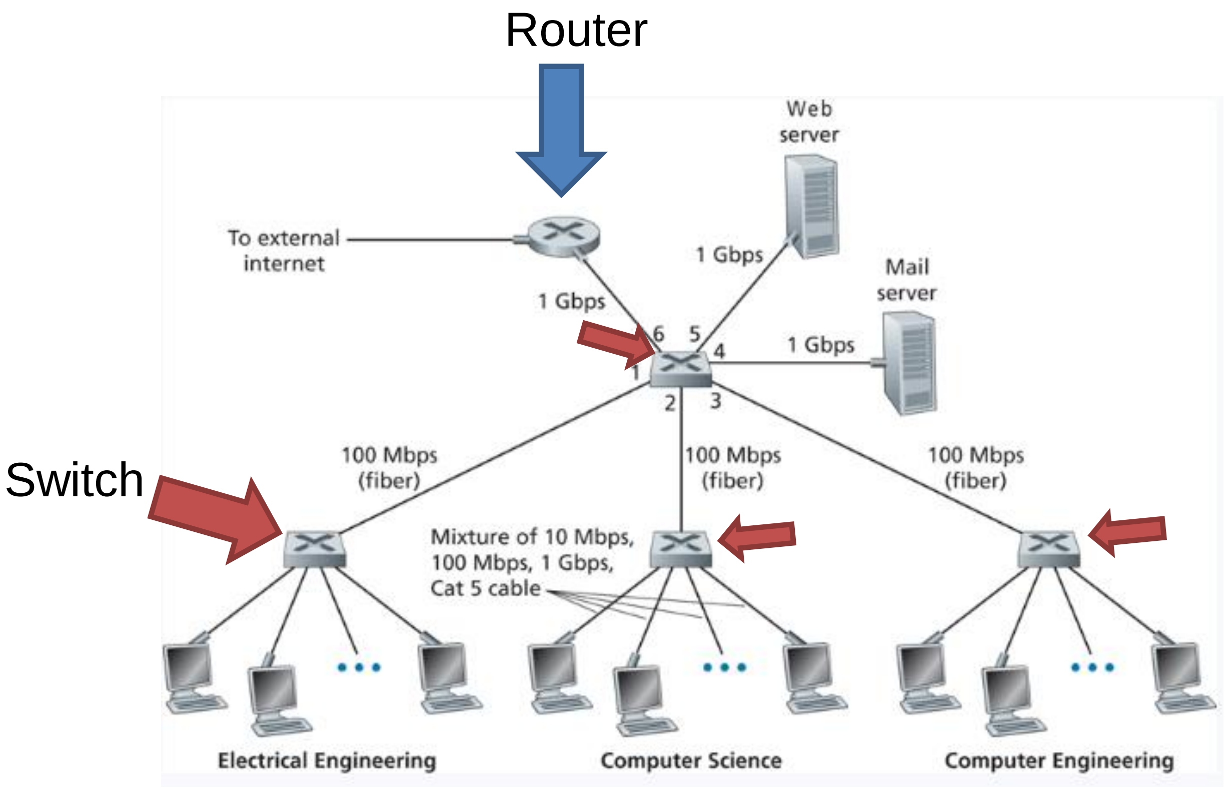
Can only transmit messages if you have the token



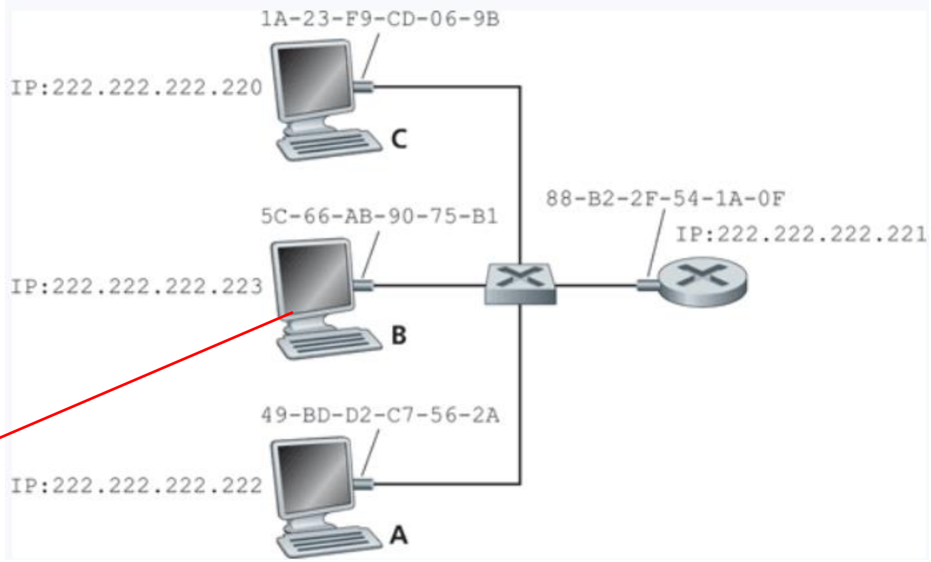
LAN

Local Area Network (LAN)- A collection of devices in one physical location, typically that share a centralized internet connection





ARP



Protocol for mapping **IP Addresses** to **MAC addresses**

Used *only* for hosts and router interfaces **on the same subnet**

First the machine checks its **ARP table**

IP Address	MAC Address	TTL
222.222.222.221	88-B2-2F-54-1A-0F	13:45:00
222.222.222.223	5C-66-AB-90-75-B1	13:52:00

If the entry does not exist in the table, construct and send an **ARP** packet

Broadcasts the ARP packet to all interfaces on the LAN (255.255.255.255)

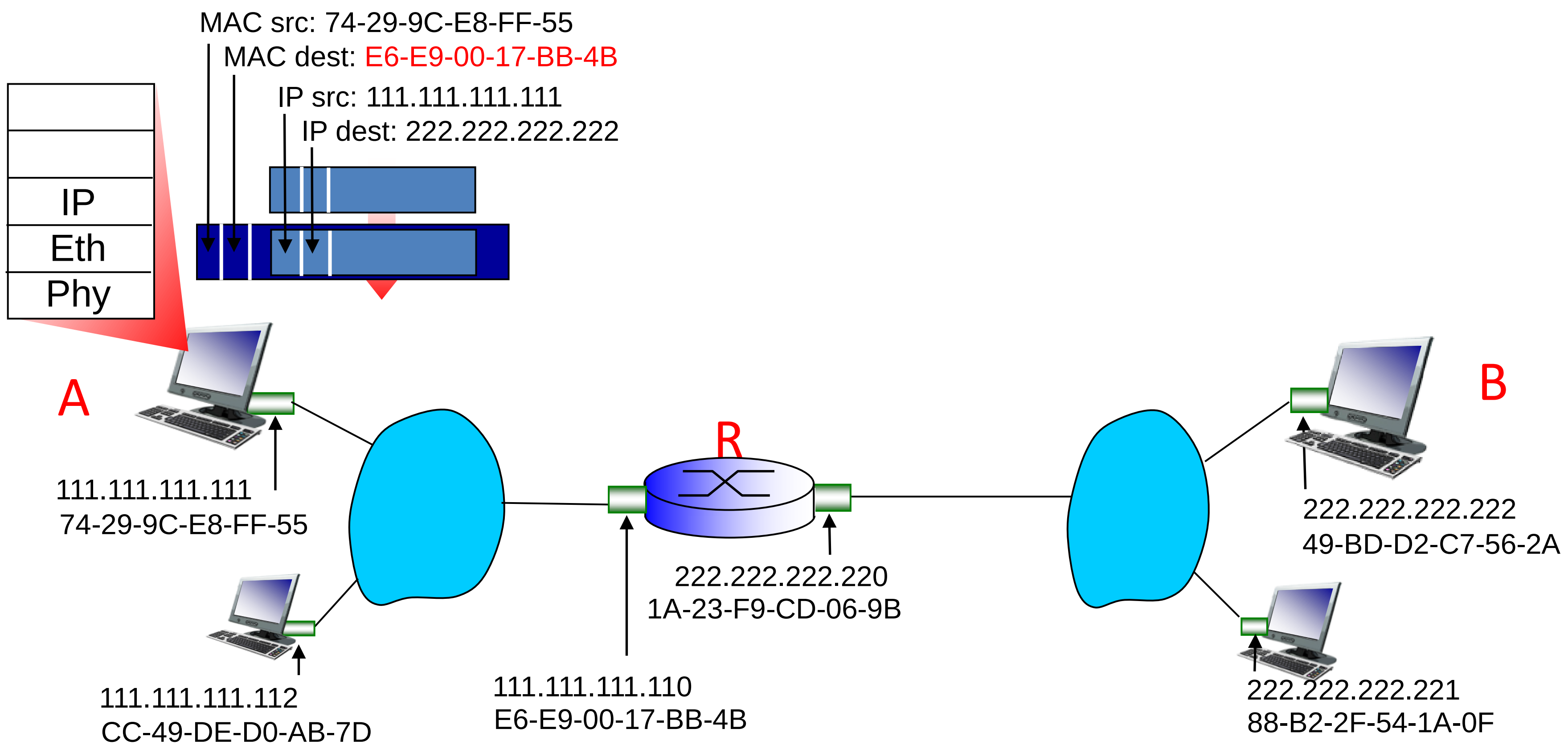
These tables are self-updated, and do not require manual entry*

ARP

- A wants to send datagram to B
  - B's MAC address not in A's ARP table.
- A broadcasts ARP query packet, containing B's IP address
  - destination MAC address = FF-FF-FF-FF-FF-FF
  - all nodes on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
  - frame sent to A's MAC address (unicast)
- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
  - soft state: information that times out (goes away) unless refreshed
- ARP is "plug-and-play":
  - nodes create their ARP tables *without intervention from net administrator*

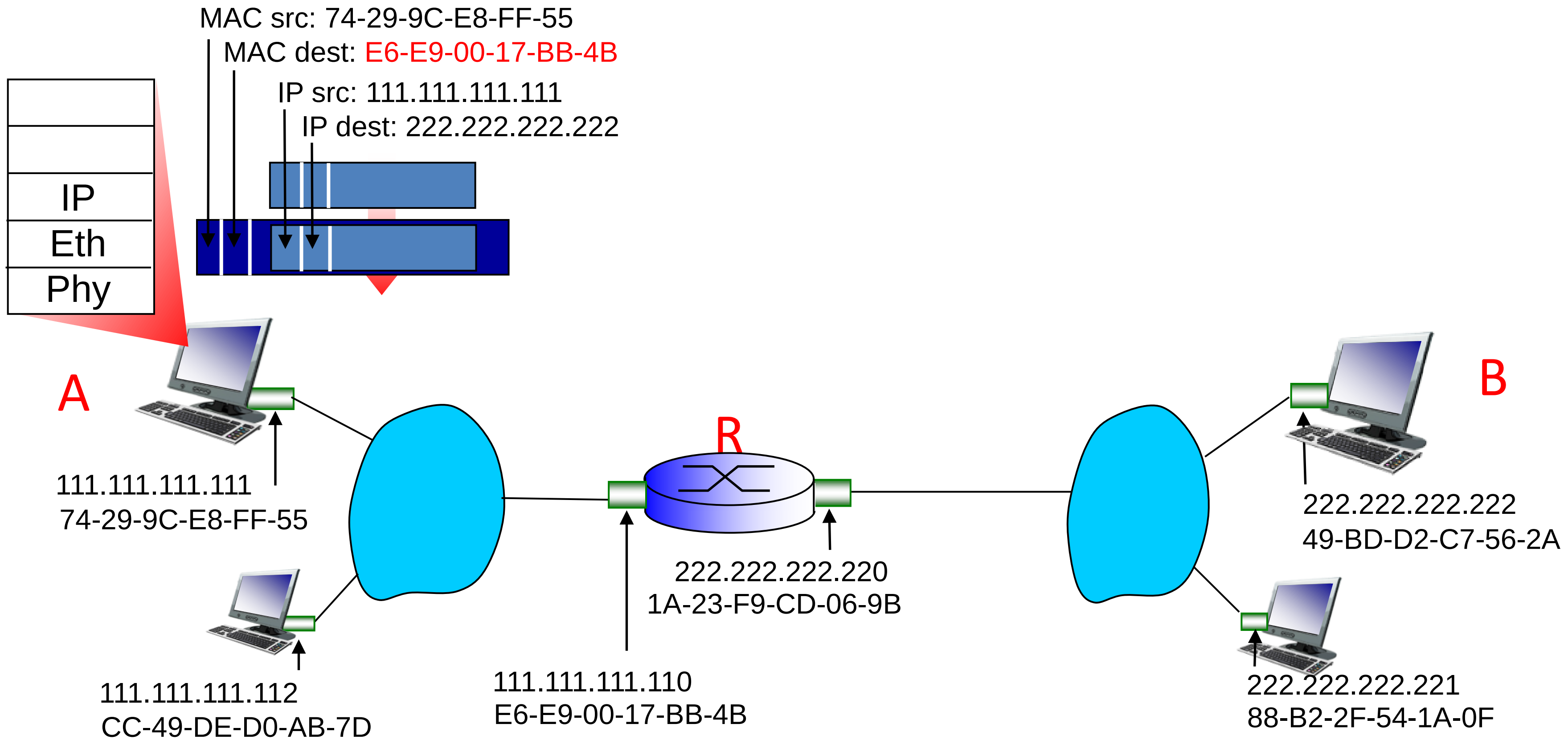
Putting it Together: Sending from LAN to LAN

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as destination address, frame contains A-to-B IP datagram



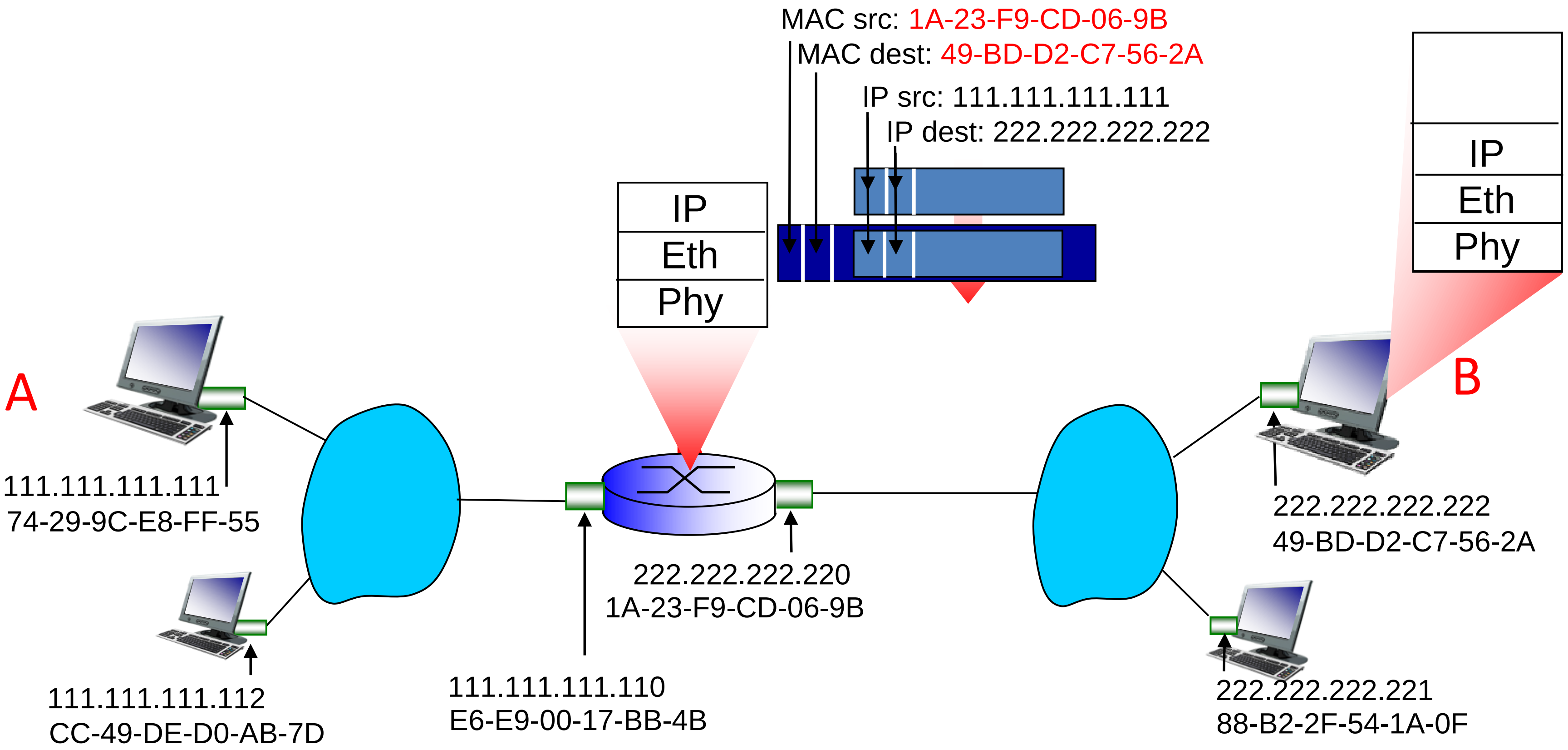
Putting it Together: Sending from LAN to LAN

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as destination address, frame contains A-to-B IP datagram



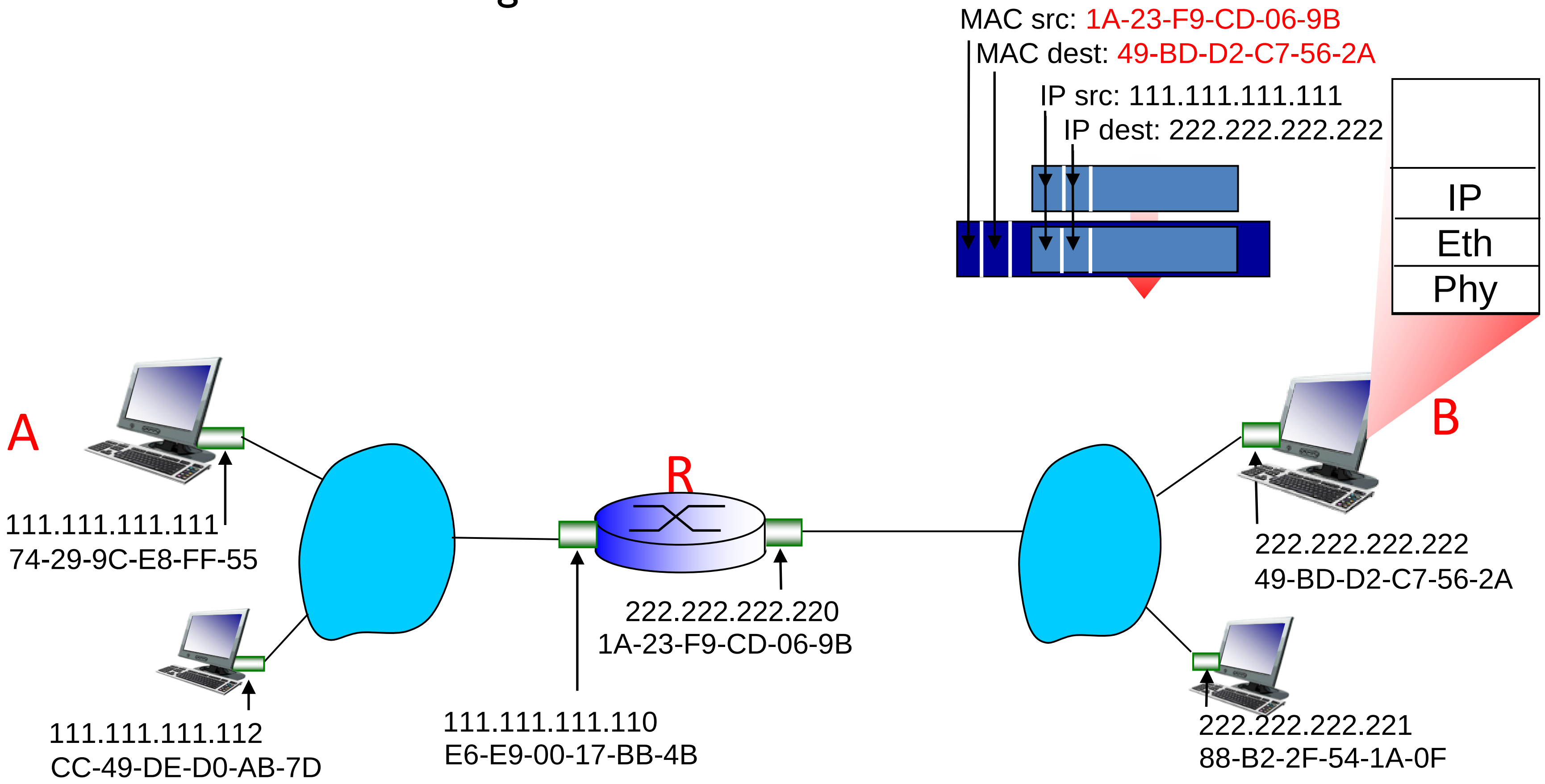
Putting it Together: Sending from LAN to LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as destination address, frame contains A-to-B IP datagram



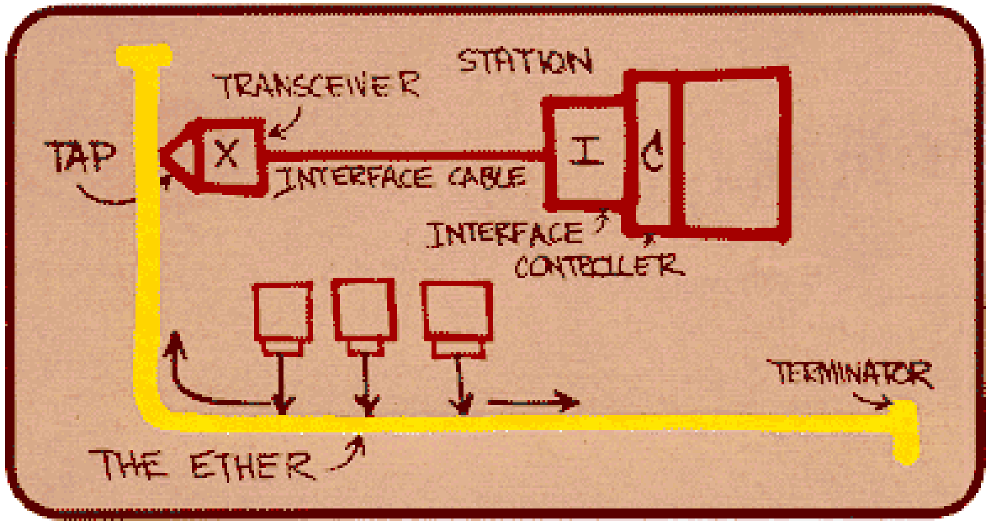
Putting it Together: Sending from LAN to LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



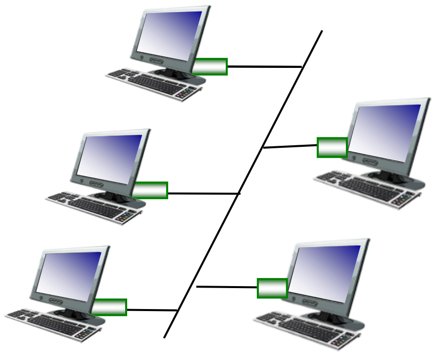
Ethernet

- "dominant" wired LAN technology:
- single chip, multiple speeds (e.g., Broadcom BCM5761)
- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps – 10 Gbps



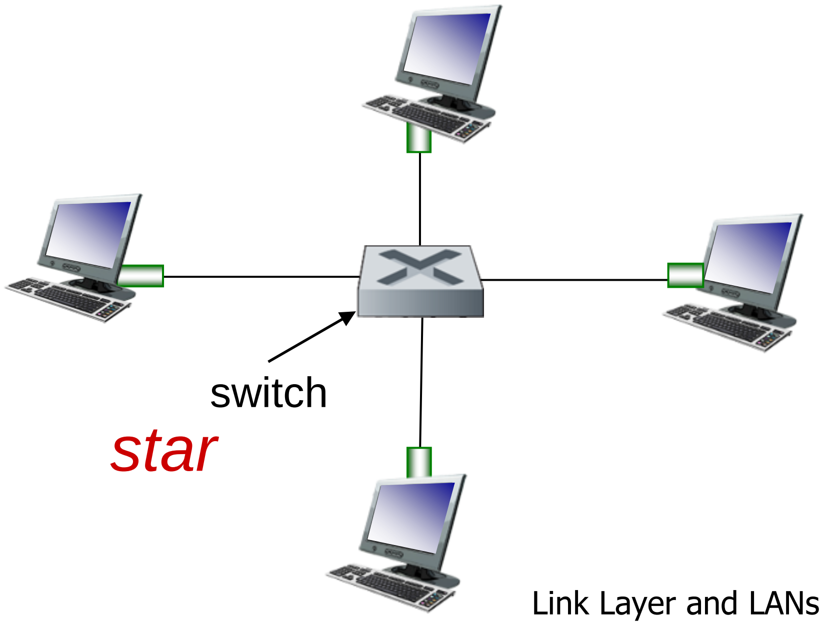
Metcalfe's Ethernet sketch

Ethernet Topology



bus: coaxial cable

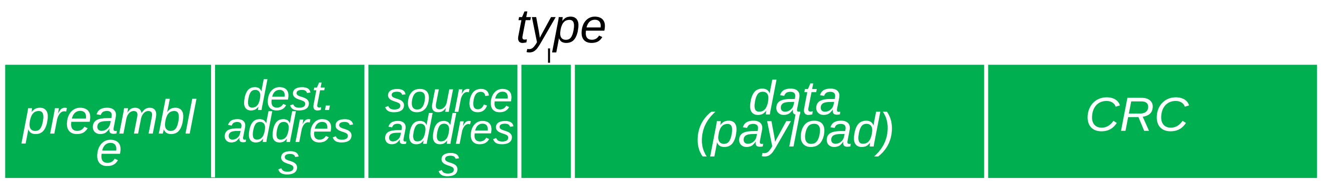
(outdated)



star

Ethernet frame structure

sending adapter encapsulates IP datagram (or other network layer protocol packet) in Ethernet frame



preamble:

7 bytes with pattern 10101010 followed by one byte with pattern 10101011

used to synchronize receiver, sender clock rates

Ethernet frame structure (more)

addresses: 6 byte source, destination MAC addresses

- if adapter receives frame with matching destination address, or with broadcast address (e.g. ARP packet), it passes data in frame to network layer protocol
- otherwise, adapter discards frame

type: indicates higher layer protocol (mostly IP but others possible, e.g., Novell IPX, AppleTalk)

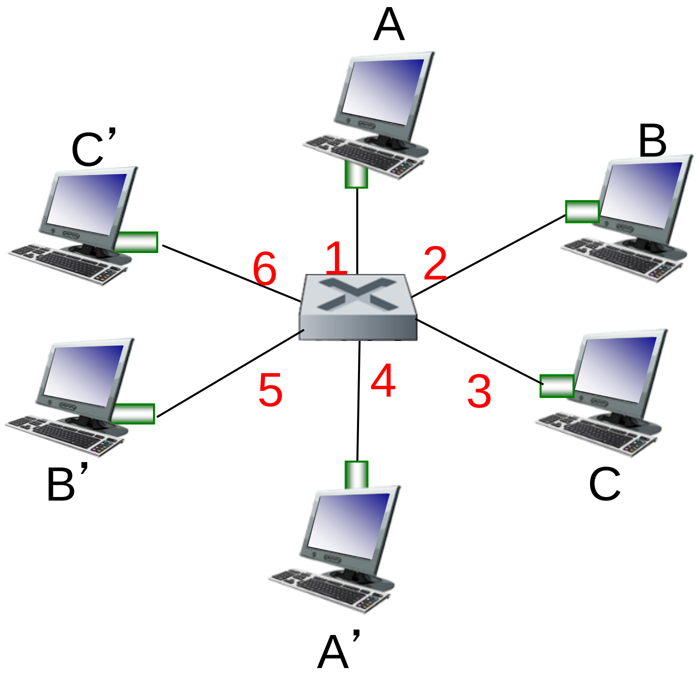
CRC: cyclic redundancy check at receiver

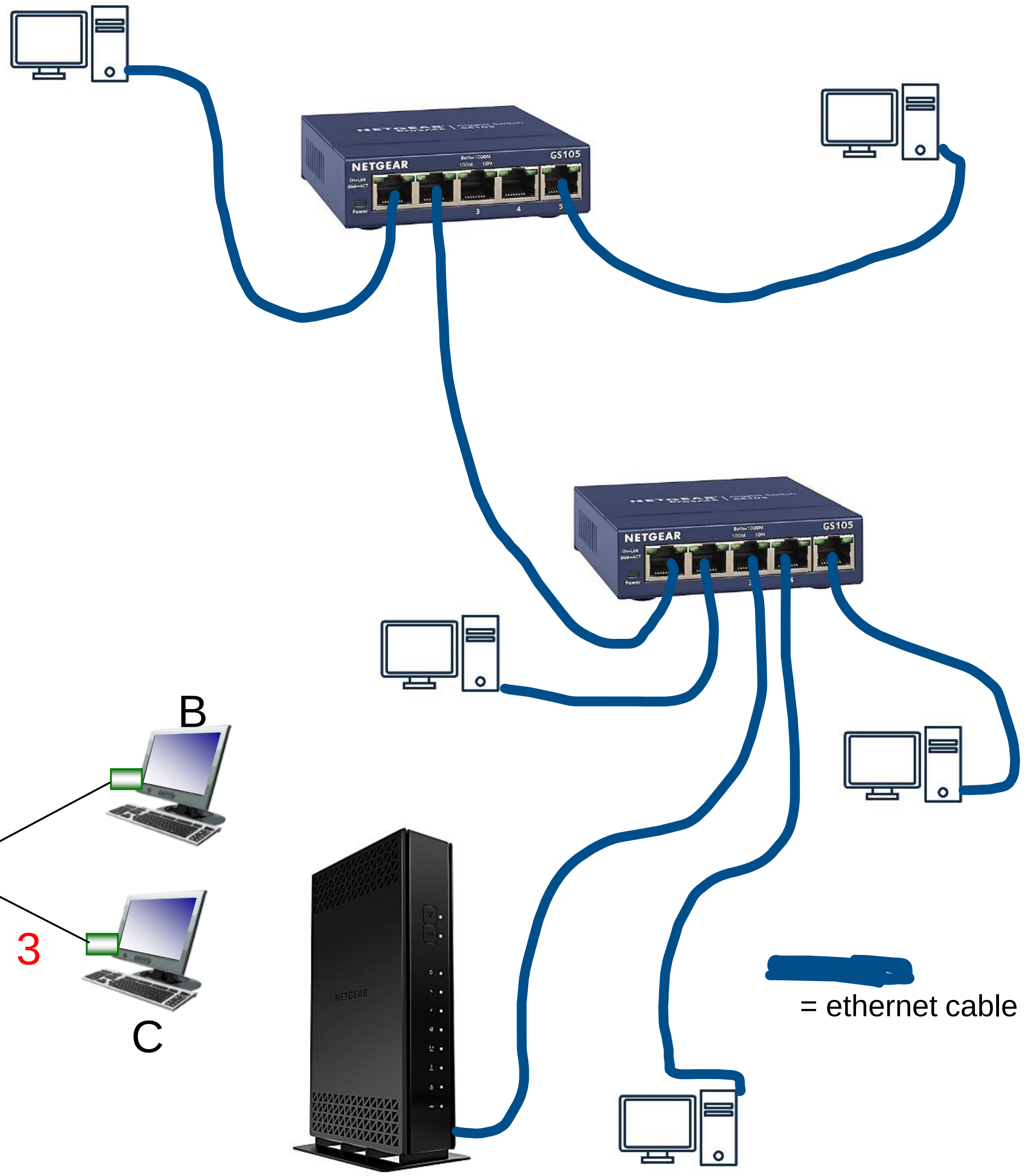
- error detected: frame is dropped

Ethernet switch

- Switches will store and forward ethernet frames
- Hosts have *dedicated*, direct connection to switch
- Ethernet protocol used on each incoming link, **but no collisions between links**
- Switching: A-A’ and B-B’ can transmit simultaneously, without collisions

- Transparent: Hosts are not aware they are connected to a switch
- Plug and play; self-learning





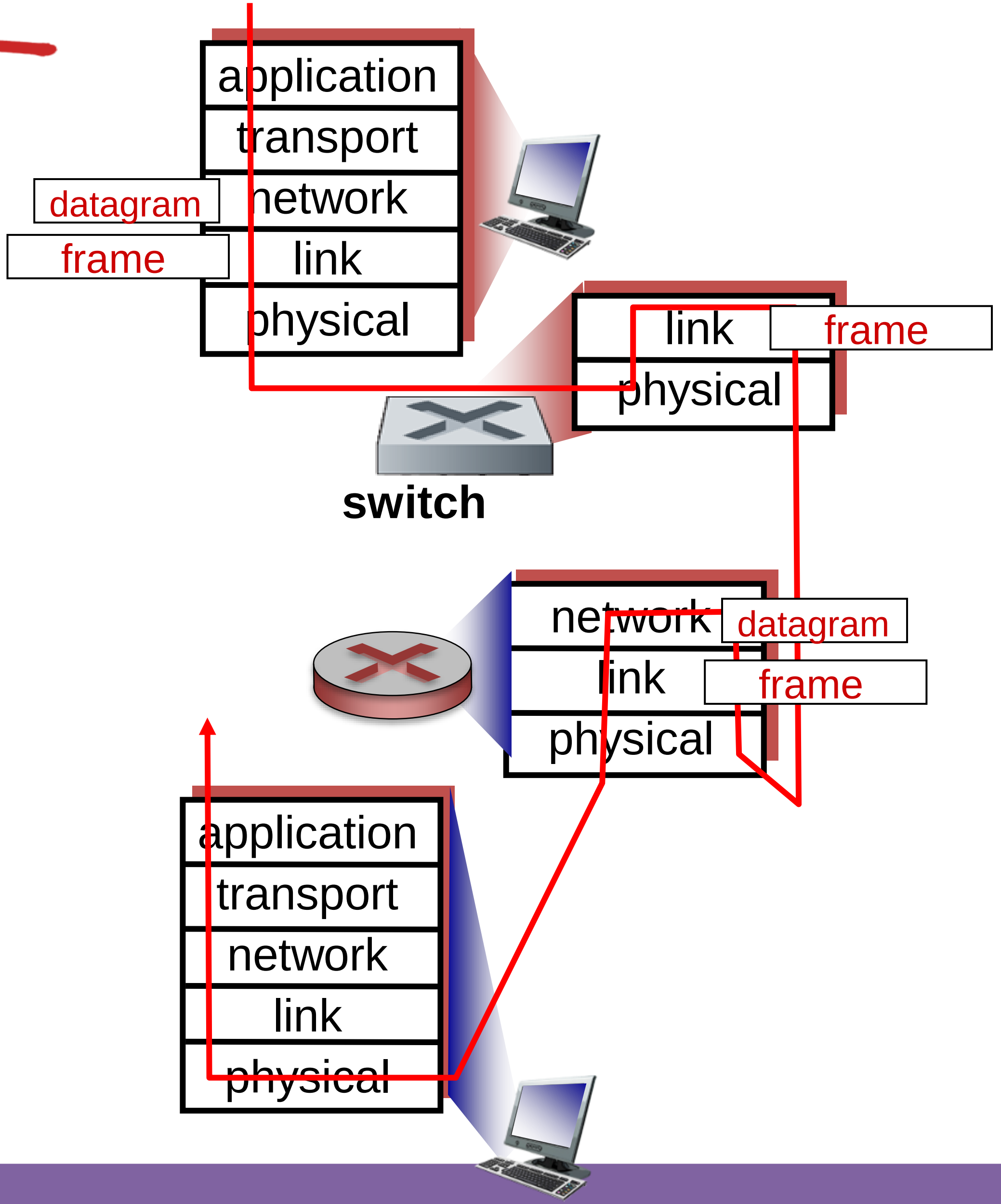
Switches vs. routers

both are store-and-forward:

- *routers:* network-layer devices (examine network-layer headers)
- *switches*: link-layer devices (examine link-layer headers)

both have forwarding tables:

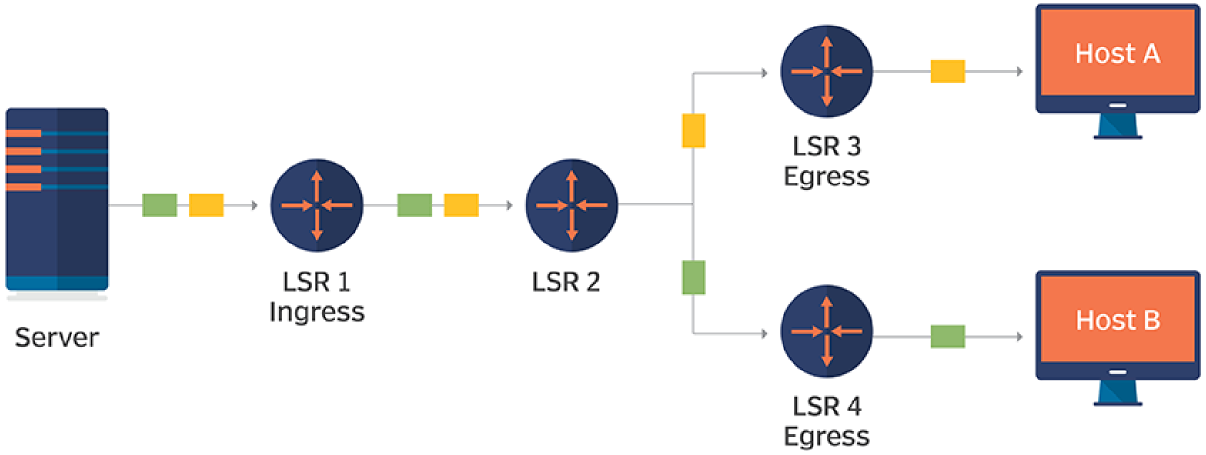
- *routers:* compute tables using routing algorithms, IP addresses
- *switches:* learn forwarding table using flooding, learning, MAC addresses



Multiprotocol Label Switching

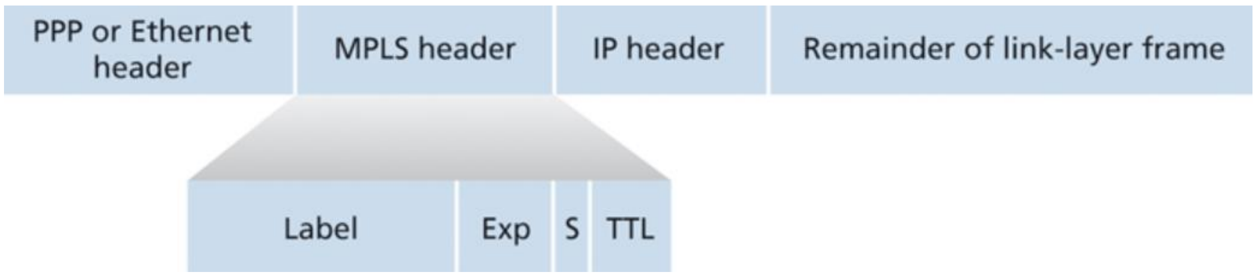
Basic MPLS network

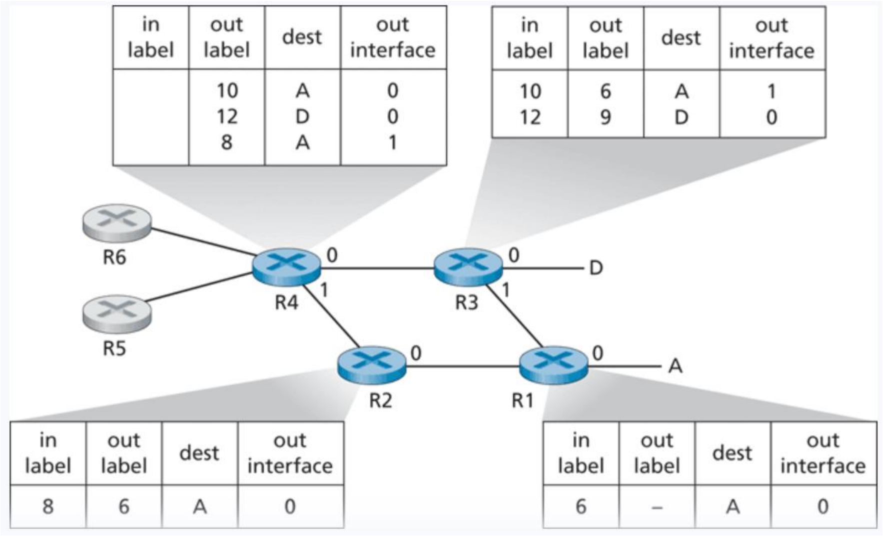
An MPLS network uses path labels instead of network addresses to direct traffic. These labels include information about which label switched path should be used to make sure a packet gets to where it's supposed to go.



Label Switch Router = LSR

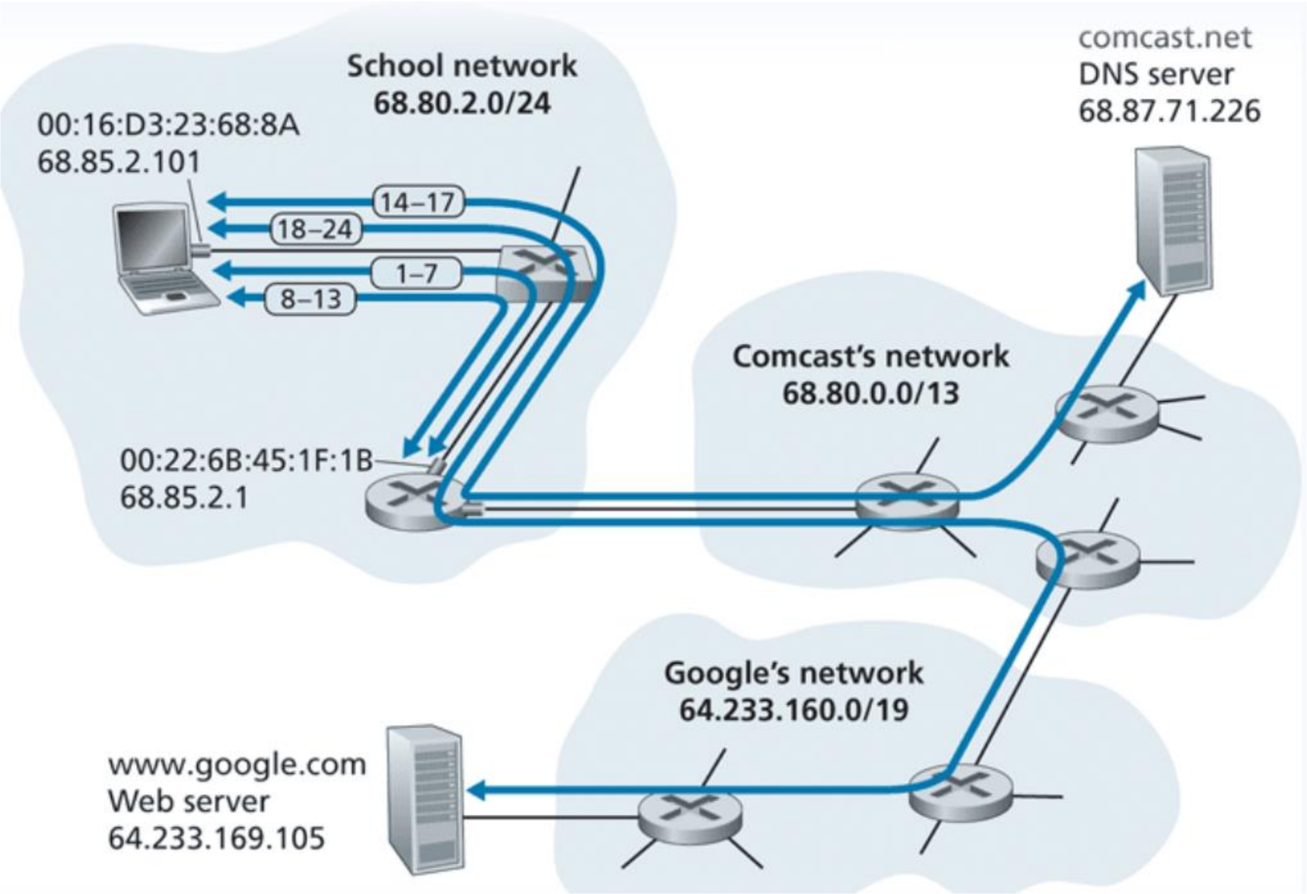
This is a layer 2.5 protocol (between network layer and link layer)



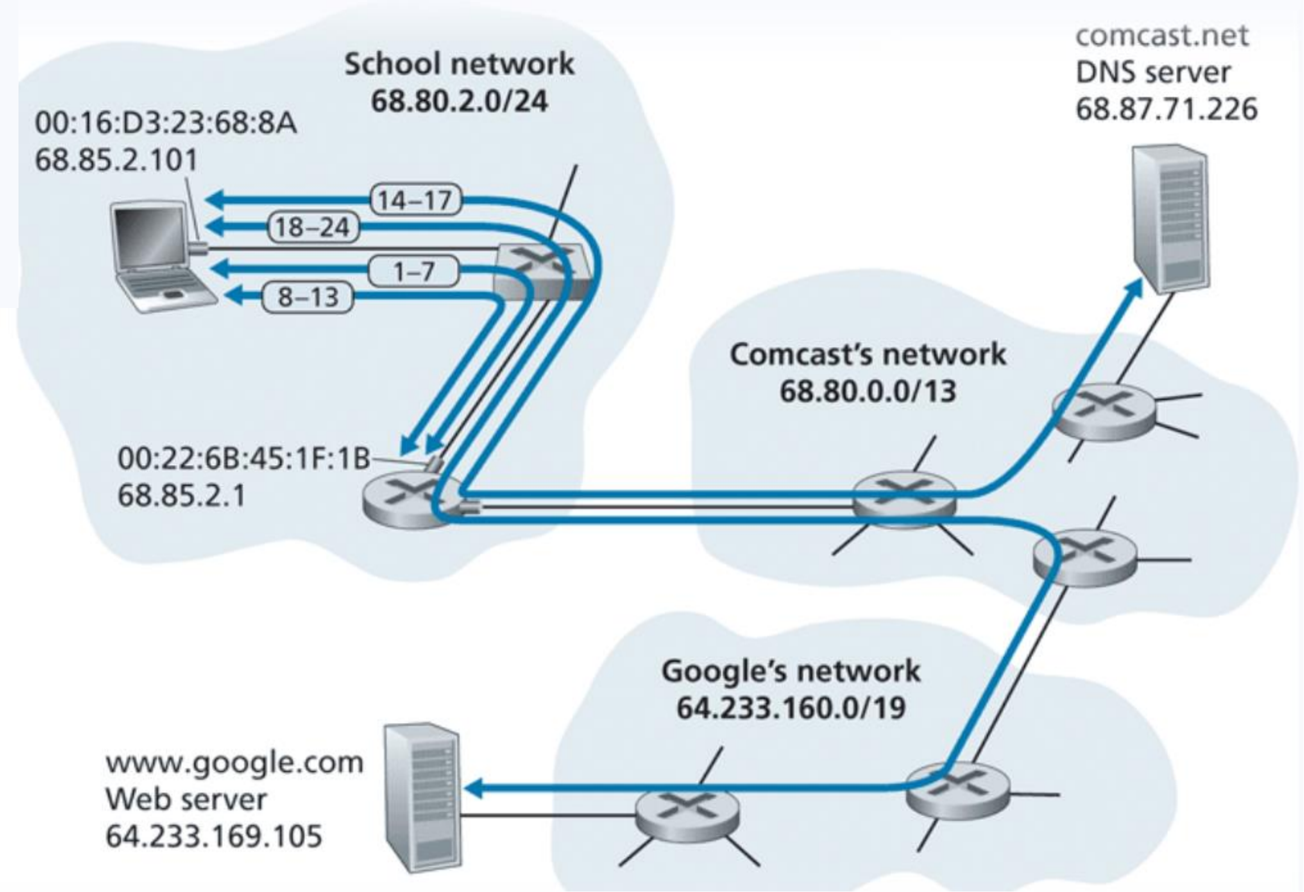


The End of the OSI Model…

We have reached the end of the OSI, now lets review…

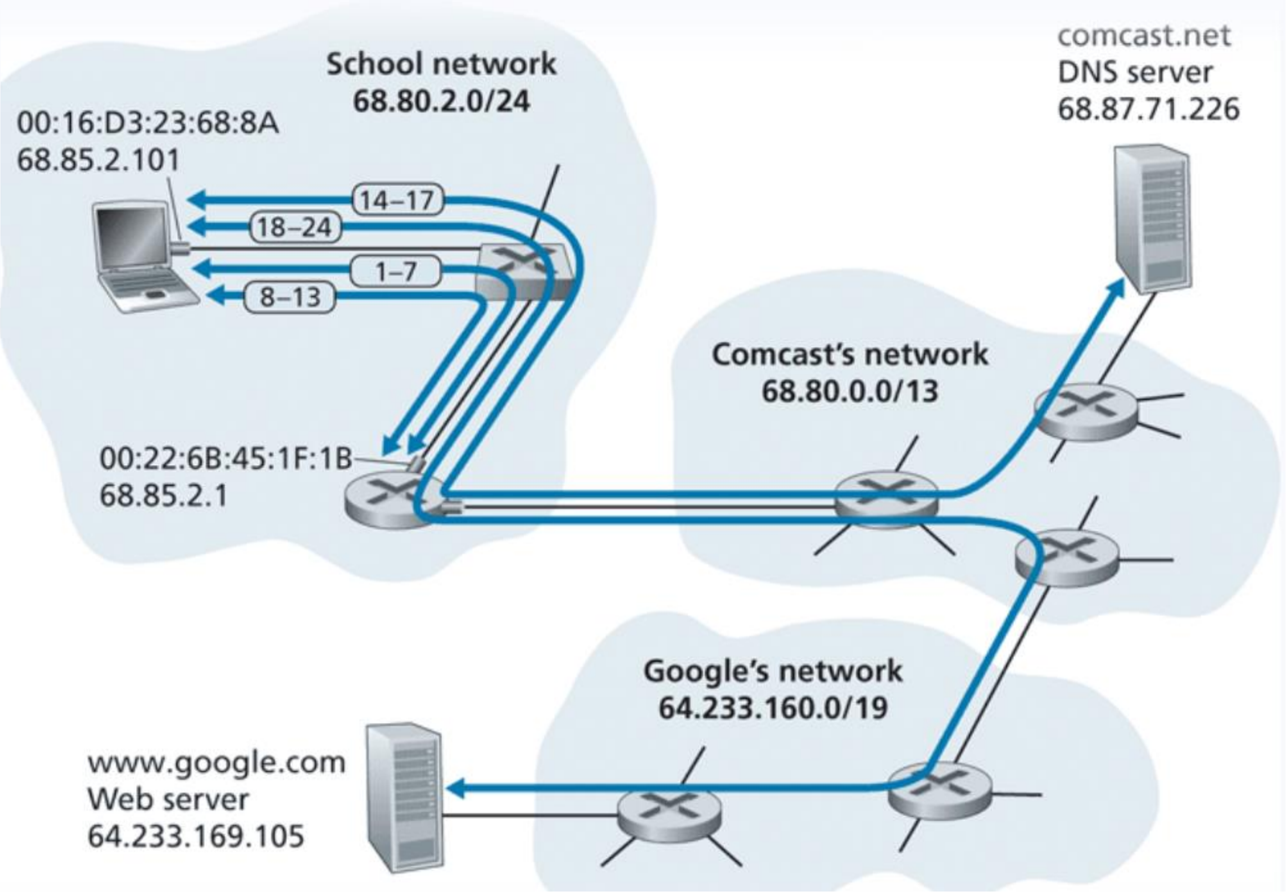

comcast.net
DNS server
68.87.71.226
School network
68.80.2.0/24
00:16:D3:23:68:8A
68.85.2.101
14–17
18–24
1–7
8–13
00:22:6B:45:1F:1B
68.85.2.1
Comcast's network
68.80.0.0/13
Google's network
64.233.160.0/19
www.google.com
Web server
64.233.169.105

1. The operating system on Bob's laptop creates a **DHCP request message** (**Section 4.3.3**) and puts this message within a **UDP segment** (**Section 3.3**) with destination port 67 (DHCP server) and source port 68 (DHCP client). The UDP segment is then placed within an **IP datagram** (**Section 4.3.1**) with a broadcast IP destination address (255.255.255.255) and a source IP address of 0.0.0.0, since Bob's laptop doesn't yet have an IP address.



1. The operating system on Bob's laptop creates a **DHCP request message** (**Section 4.3.3**) and puts this message within a **UDP segment** (**Section 3.3**) with destination port 67 (DHCP server) and source port 68 (DHCP client). The UDP segment is then placed within an **IP datagram** (**Section 4.3.1**) with a broadcast IP destination address (255.255.255.255) and a source IP address of 0.0.0.0, since Bob's laptop doesn't yet have an IP address.

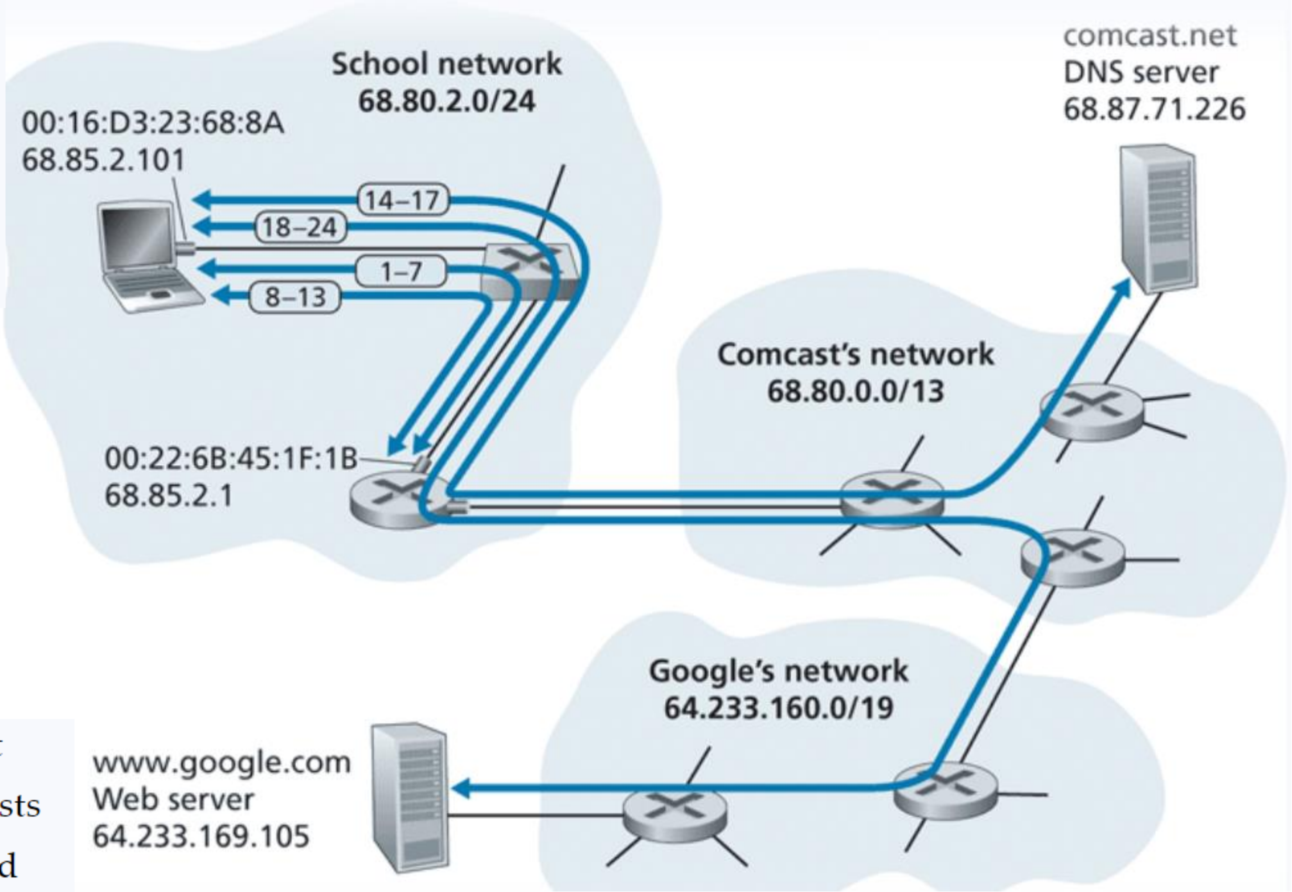
2. The IP datagram containing the DHCP request message is then placed within an **Ethernet frame** (**Section 6.4.2**). The Ethernet frame has a destination MAC addresses of FF:FF:FF:FF:FF:FF so that the frame will be broadcast to all devices connected to the switch (hopefully including a DHCP server); the frame's source MAC address is that of Bob's laptop, 00:16:D3:23:68:8A.



1. The operating system on Bob's laptop creates a **DHCP request message** (**Section 4.3.3**) and puts this message within a **UDP segment** (**Section 3.3**) with destination port 67 (DHCP server) and source port 68 (DHCP client). The UDP segment is then placed within an **IP datagram** (**Section 4.3.1**) with a broadcast IP destination address (255.255.255.255) and a source IP address of 0.0.0.0, since Bob's laptop doesn't yet have an IP address.

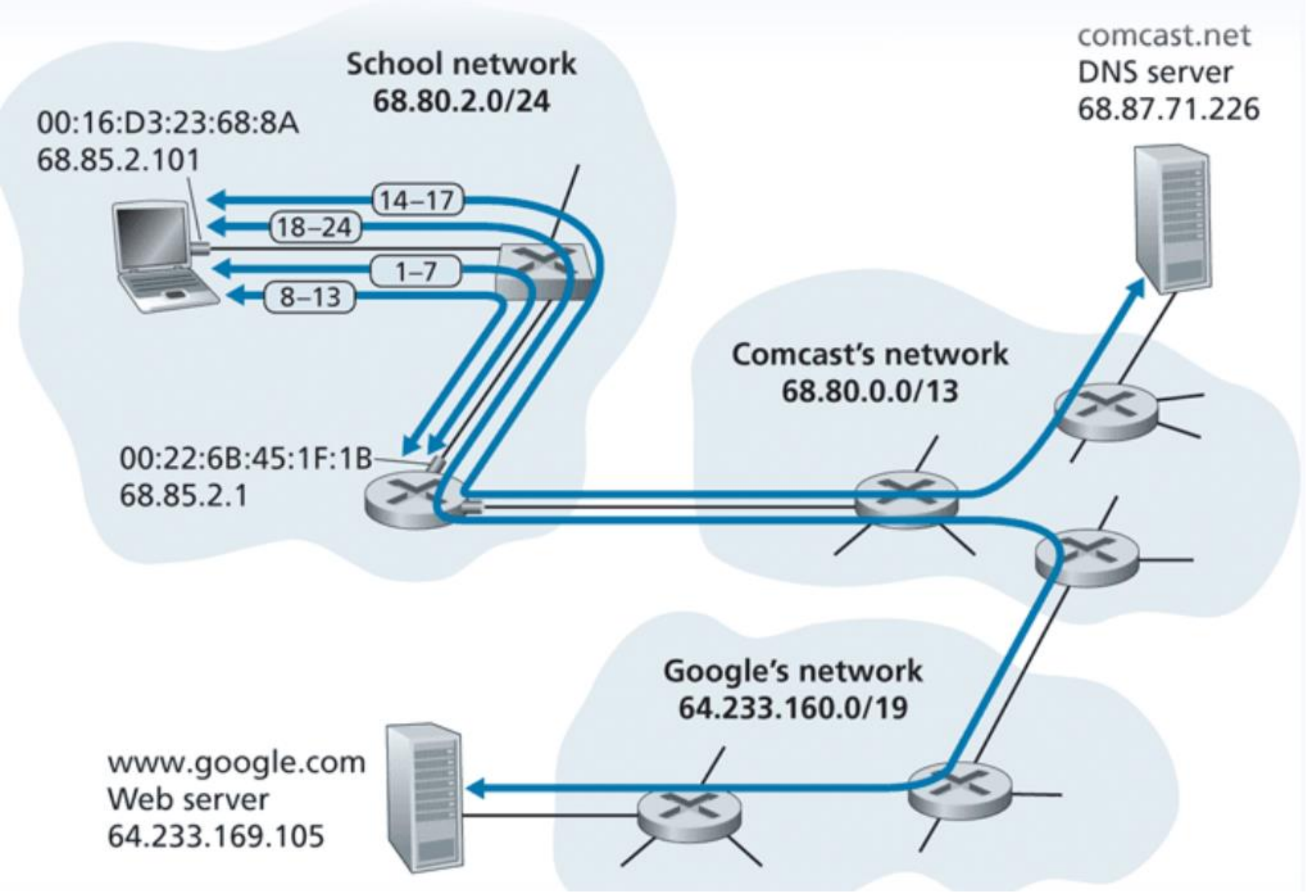
2. The IP datagram containing the DHCP request message is then placed within an **Ethernet frame** (**Section 6.4.2**). The Ethernet frame has a destination MAC addresses of FF:FF:FF:FF:FF:FF so that the frame will be broadcast to all devices connected to the switch (hopefully including a DHCP server); the frame's source MAC address is that of Bob's laptop, 00:16:D3:23:68:8A.

3. The broadcast Ethernet frame containing the DHCP request is the first frame sent by Bob's laptop to the Ethernet switch. The switch broadcasts the incoming frame on all outgoing ports, including the port connected to the router.



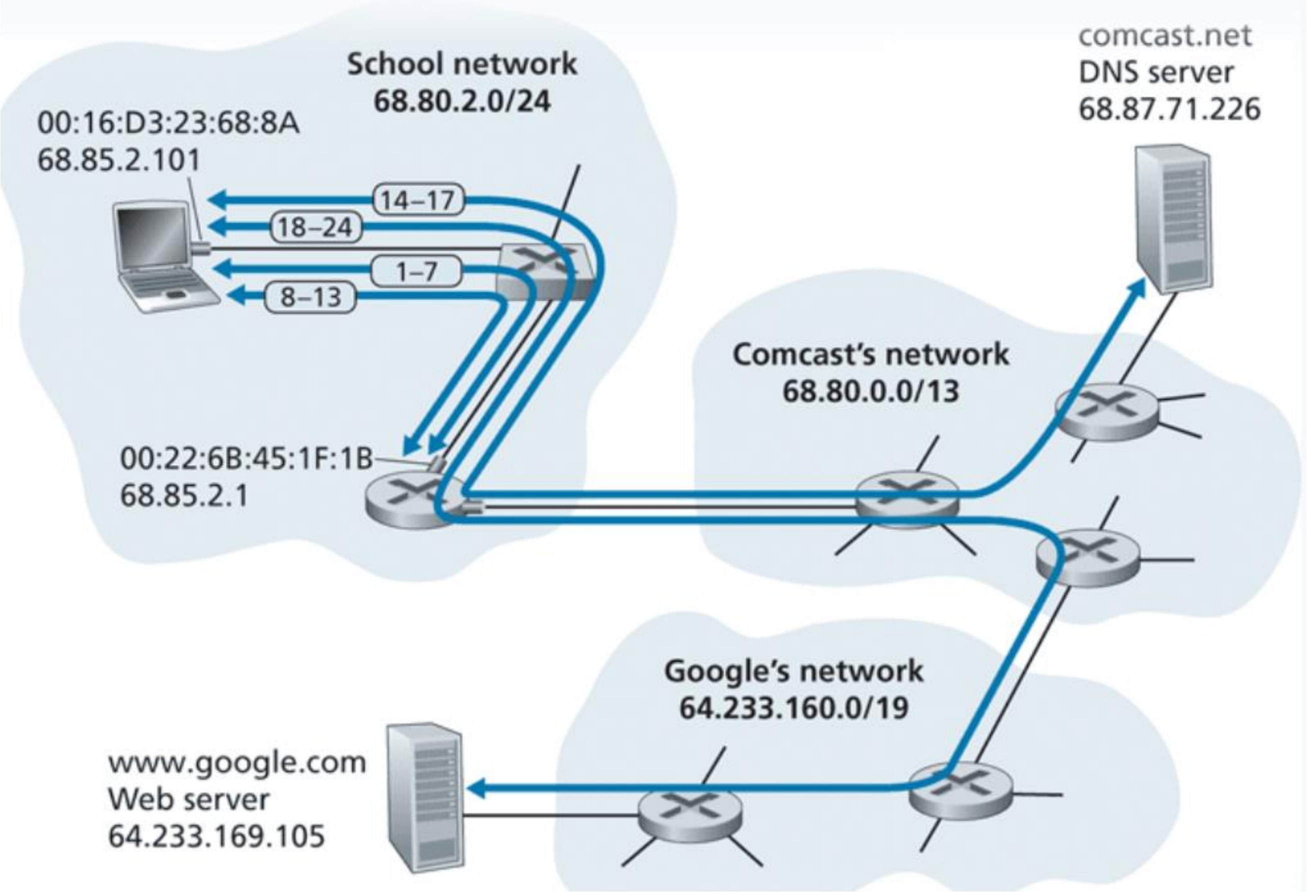
4. The router receives the broadcast Ethernet frame containing the DHCP request on its interface with MAC address 00:22:6B:45:1F:1B and the IP datagram is extracted from the Ethernet frame. The datagram's broadcast IP destination address indicates that this IP datagram should be processed by upper layer protocols at this node, so the datagram's payload (a UDP segment) is thus **demultiplexed** (**Section 3.2**) up to UDP, and the DHCP request message is extracted from the UDP segment. The DHCP server now has the DHCP request message.

5. Let's suppose that the DHCP server running within the router can allocate IP addresses in the **CIDR** (**Section 4.3.3**) block 68.85.2.0/24. In this example, all IP addresses used within the school are thus within Comcast's address block. Let's suppose the DHCP server allocates address 68.85.2.101 to Bob's laptop. The DHCP server creates a **DHCP ACK message** (**Section 4.3.3**) containing this IP address, as well as the IP address of the DNS server (68.87.71.226), the IP address for the default gateway router (68.85.2.1), and the subnet block (68.85.2.0/24) (equivalently, the "network mask"). The DHCP message is put inside a UDP segment, which is put inside an IP datagram, which is put inside an Ethernet frame. The Ethernet frame has a source MAC address of the router's interface to the home network (00:22:6B:45:1F:1B) and a destination MAC address of Bob's laptop (00:16:D3:23:68:8A).



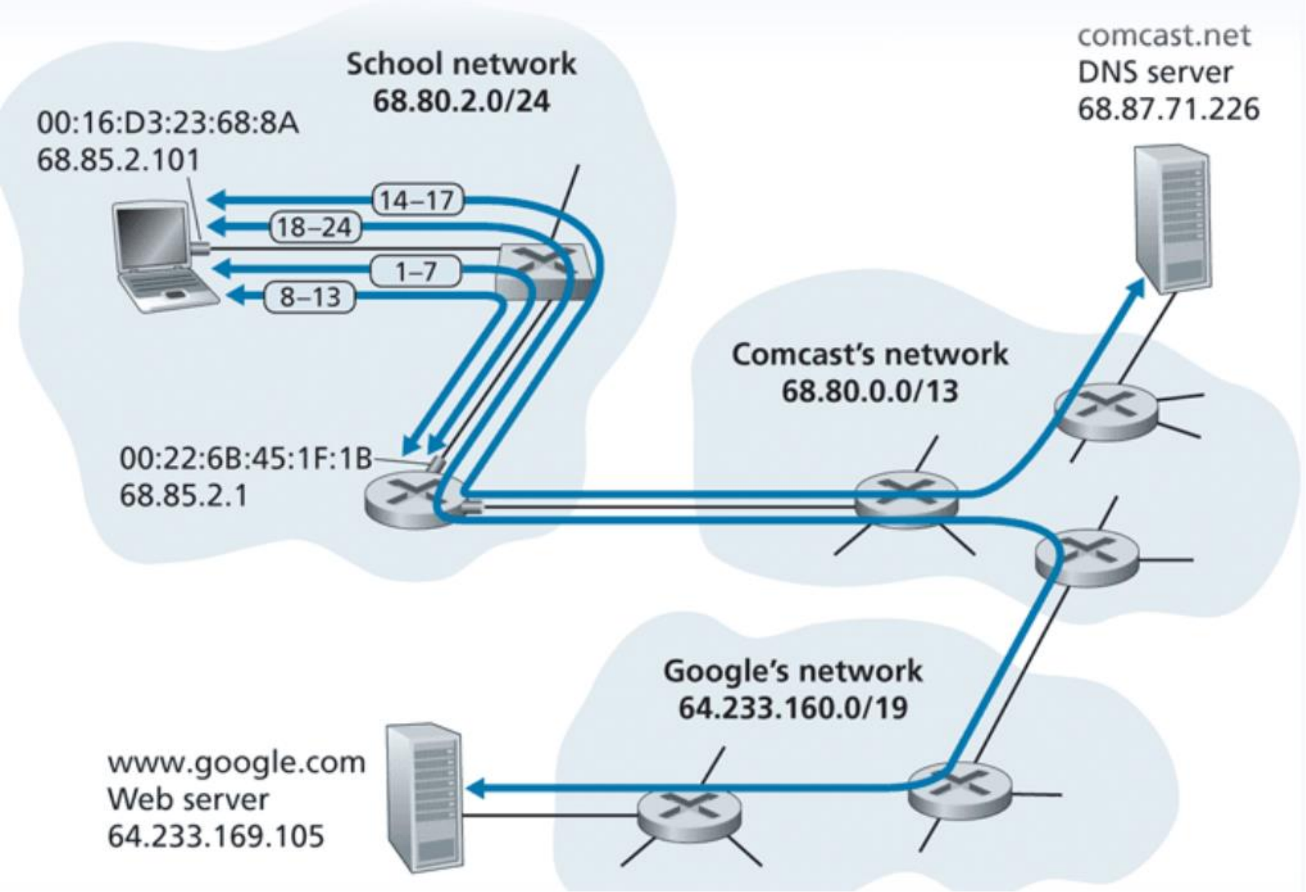
6. The Ethernet frame containing the DHCP ACK is sent (unicast) by the router to the switch. Because the switch is **self-learning** (**Section 6.4.3**) and previously received an Ethernet frame (containing the DHCP request) from Bob's laptop, the switch knows to forward a frame addressed to 00:16:D3:23:68:8A only to the output port leading to Bob's laptop.

7. Bob's laptop receives the Ethernet frame containing the DHCP ACK, extracts the IP datagram from the Ethernet frame, extracts the UDP segment from the IP datagram, and extracts the DHCP ACK message from the UDP segment. Bob's DHCP client then records its IP address and the IP address of its DNS server. It also installs the address of the default gateway into its **IP forwarding table** (**Section 4.1**). Bob's laptop will send all datagrams with destination address outside of its subnet 68.85.2.0/24 to the default gateway. At this point, Bob's laptop has initialized its networking components and is ready to begin processing the Web page fetch. (Note that only the last two DHCP steps of the four presented in **Chapter 4** are actually necessary.)



OK.. So that was
just DHCP

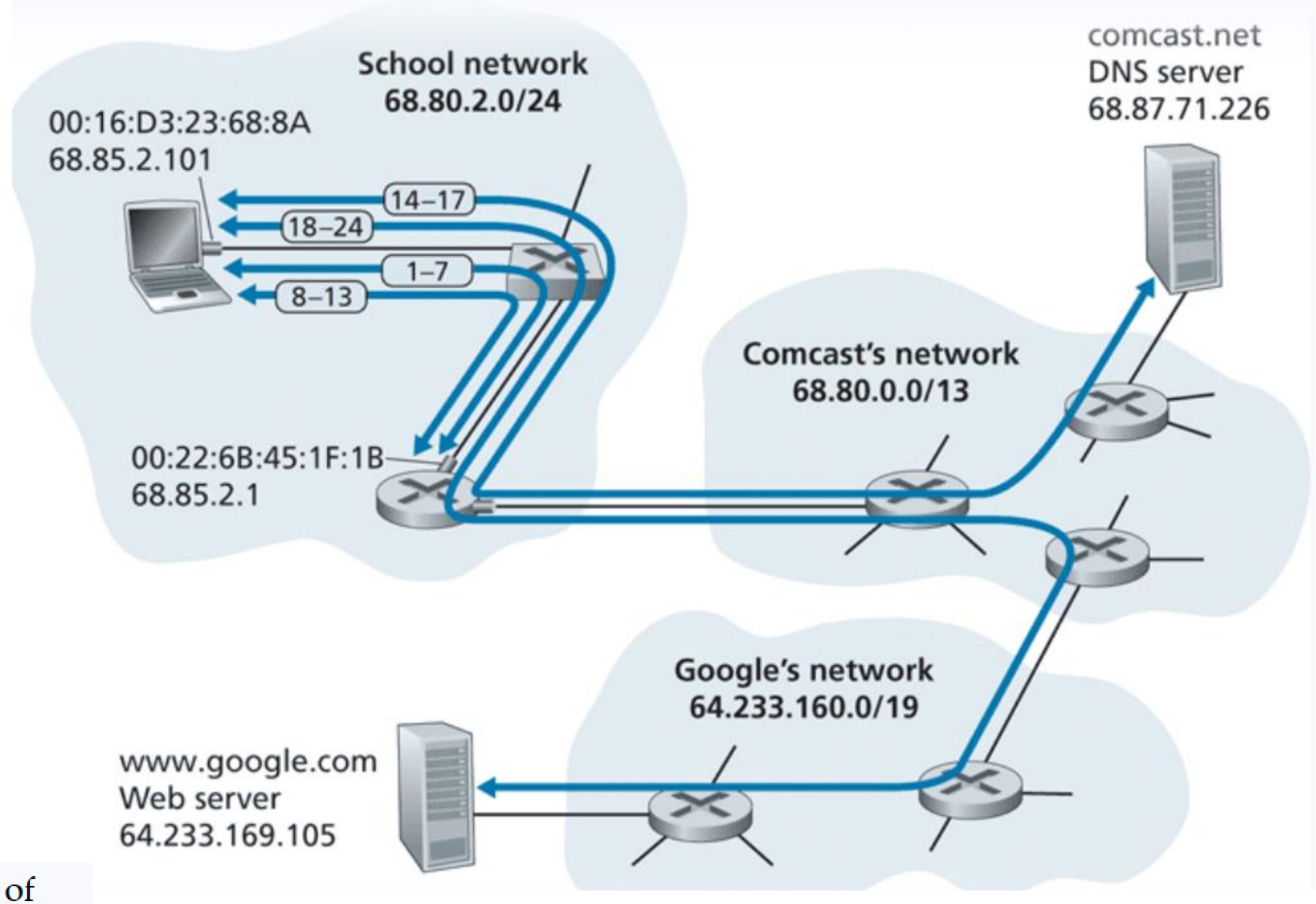
8. The operating system on Bob's laptop thus creates a **DNS query message (Section 2.5.3)**, putting the string "www.google.com" in the question section of the DNS message. This DNS message is then placed within a UDP segment with a destination port of 53 (DNS server). The UDP segment is then placed within an IP datagram with an IP destination address of 68.87.71.226 (the address of the DNS server returned in the DHCP ACK in step 5) and a source IP address of 68.85.2.101.



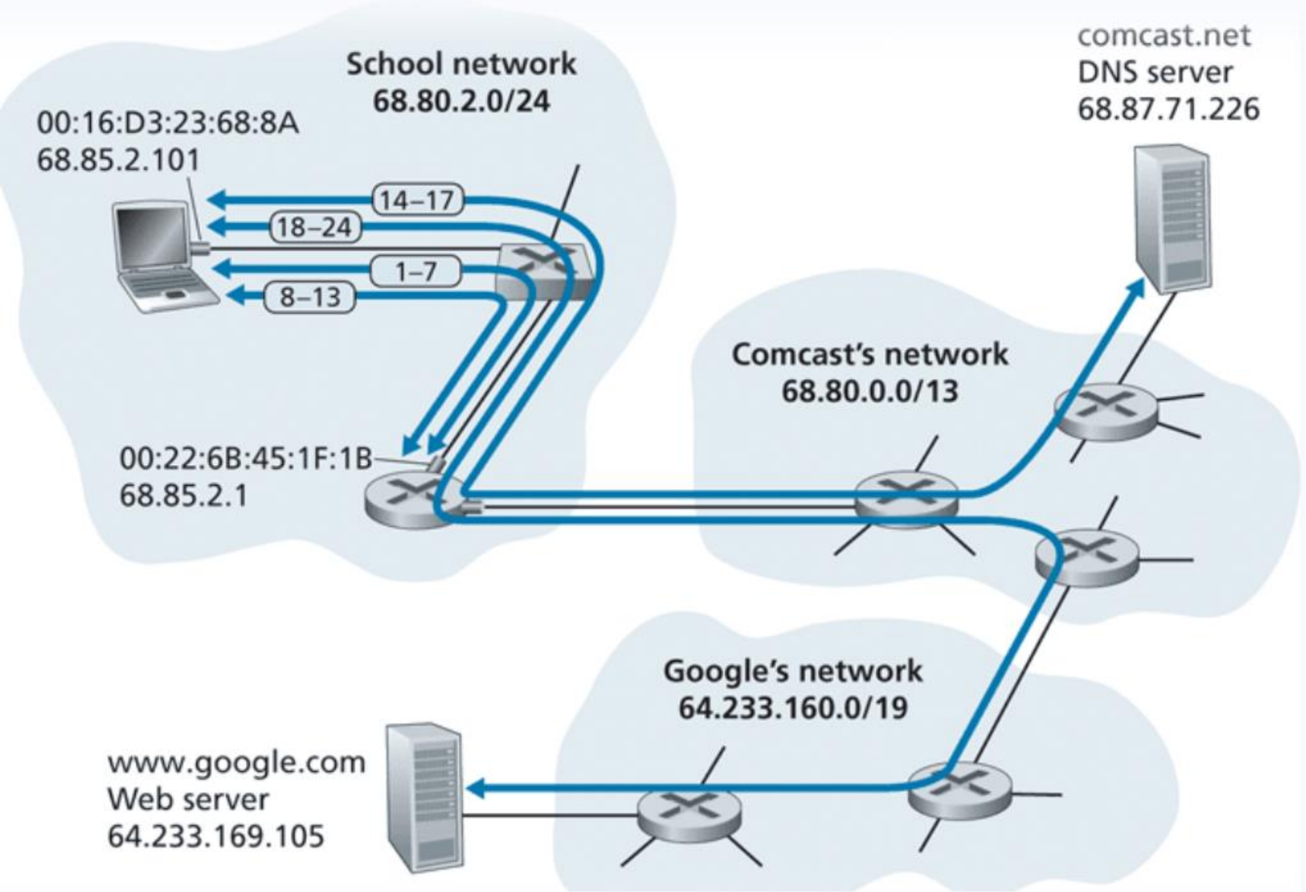
8. The operating system on Bob's laptop thus creates a **DNS query message (Section 2.5.3)**, putting the string "www.google.com" in the question section of the DNS message. This DNS message is then placed within a UDP segment with a destination port of 53 (DNS server). The UDP segment is then placed within an IP datagram with an IP destination address of 68.87.71.226 (the address of the DNS server returned in the DHCP ACK in step 5) and a source IP address of 68.85.2.101.

9. Bob's laptop then places the datagram containing the DNS query message in an Ethernet frame. This frame will be sent (addressed, at the link layer) to the gateway router in Bob's school's network. However, even though Bob's laptop knows the IP address of the school's gateway router (68.85.2.1) via the DHCP ACK message in step 5 above, it doesn't know the gateway router's MAC address. In order to obtain the MAC address of the gateway router, Bob's laptop will need to use the **ARP protocol** (**Section 6.4.1**).

10. Bob's laptop creates an **ARP query** message with a target IP address of 68.85.2.1 (the default gateway), places the ARP message within an Ethernet frame with a broadcast destination address (FF:FF:FF:FF:FF:FF) and sends the Ethernet frame to the switch, which delivers the frame to all connected devices, including the gateway router.

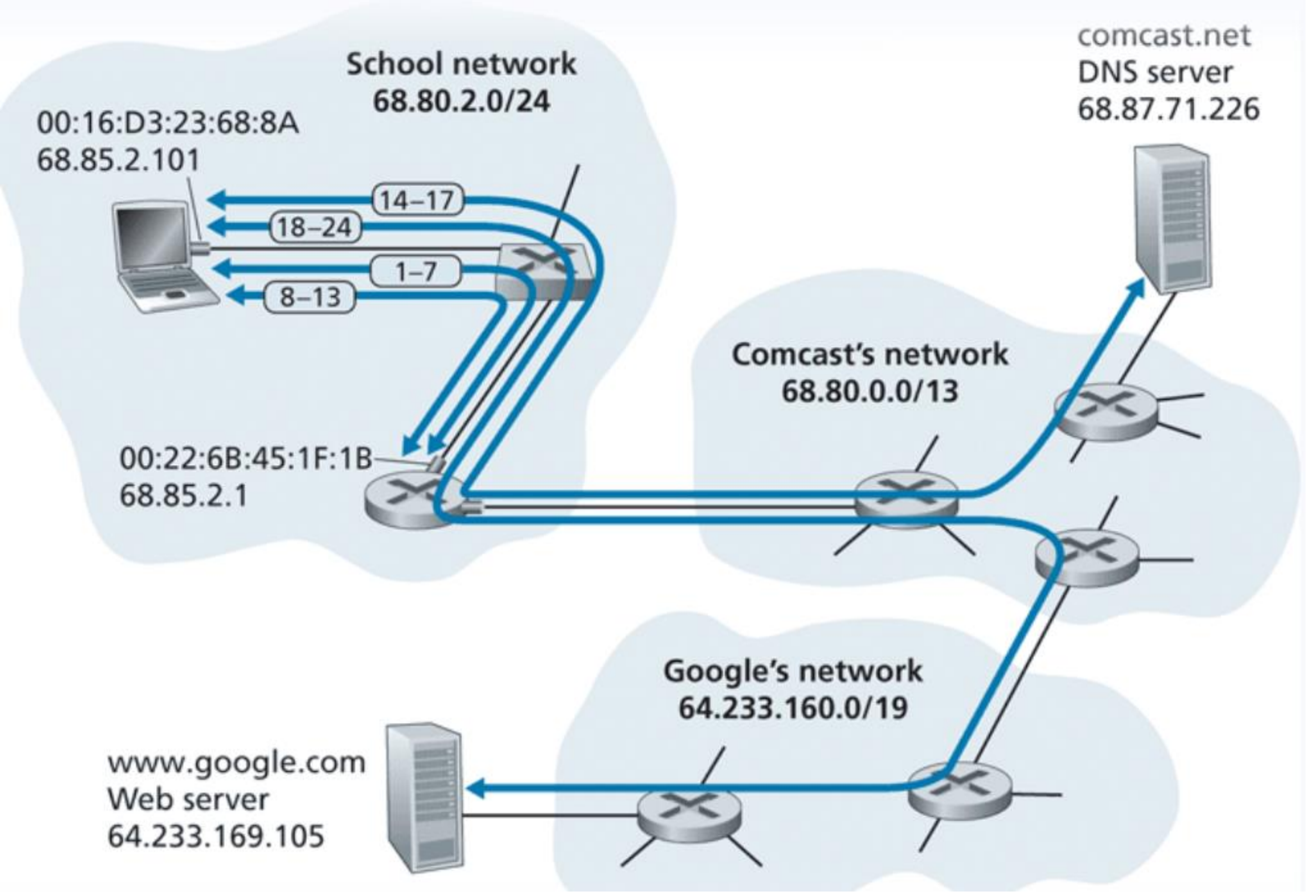


11. The gateway router receives the frame containing the ARP query message on the interface to the school network, and finds that the target IP address of 68.85.2.1 in the ARP message matches the IP address of its interface. The gateway router thus prepares an **ARP reply**, indicating that its MAC address of 00:22:6B:45:1F:1B corresponds to IP address 68.85.2.1. It places the ARP reply message in an Ethernet frame, with a destination address of 00:16:D3:23:68:8A (Bob's laptop) and sends the frame to the switch, which delivers the frame to Bob's laptop.

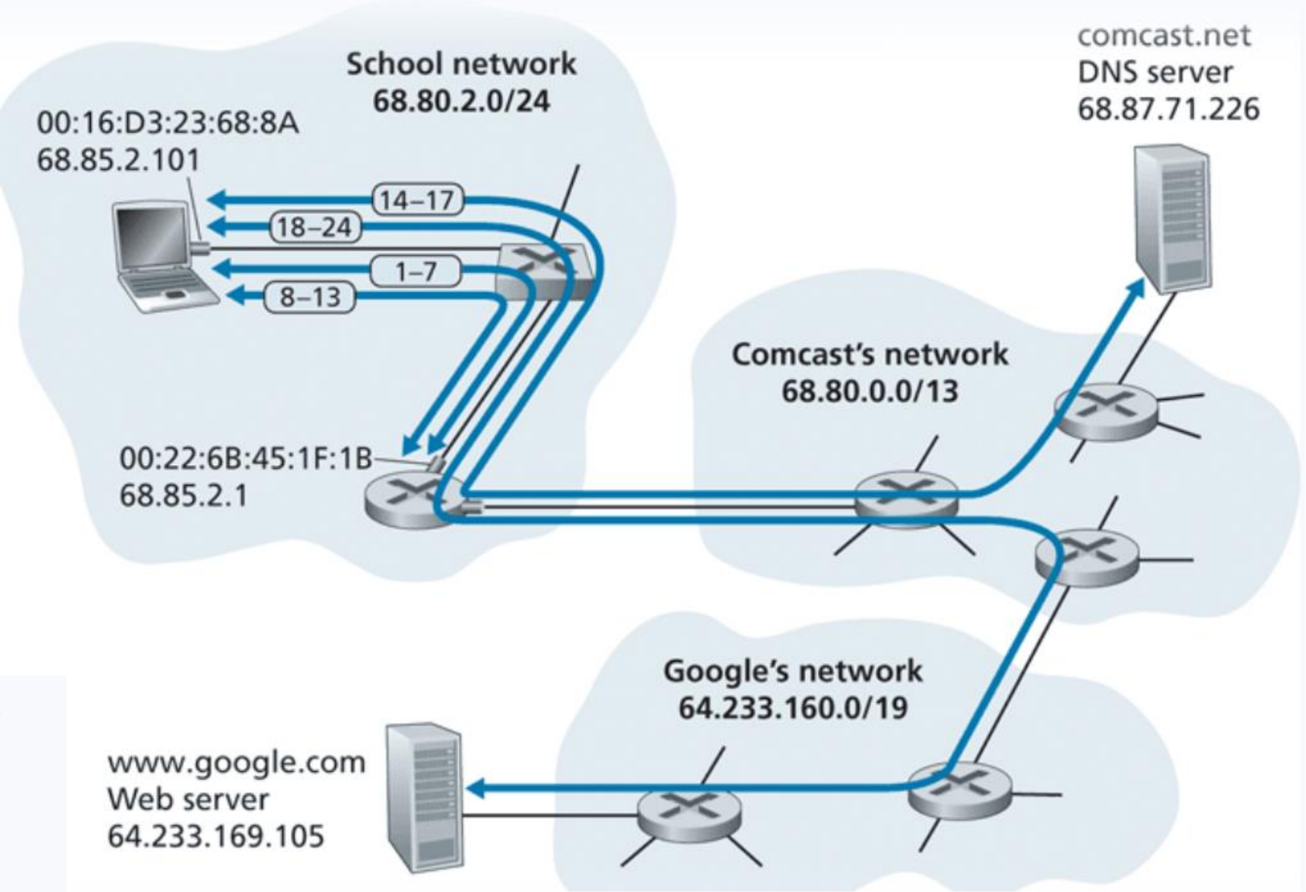


11. The gateway router receives the frame containing the ARP query message on the interface to the school network, and finds that the target IP address of 68.85.2.1 in the ARP message matches the IP address of its interface. The gateway router thus prepares an **ARP reply**, indicating that its MAC address of 00:22:6B:45:1F:1B corresponds to IP address 68.85.2.1. It places the ARP reply message in an Ethernet frame, with a destination address of 00:16:D3:23:68:8A (Bob's laptop) and sends the frame to the switch, which delivers the frame to Bob's laptop.

12. Bob's laptop receives the frame containing the ARP reply message and extracts the MAC address of the gateway router (00:22:6B:45:1F:1B) from the ARP reply message.



11. The gateway router receives the frame containing the ARP query message on the interface to the school network, and finds that the target IP address of 68.85.2.1 in the ARP message matches the IP address of its interface. The gateway router thus prepares an **ARP reply**, indicating that its MAC address of 00:22:6B:45:1F:1B corresponds to IP address 68.85.2.1. It places the ARP reply message in an Ethernet frame, with a destination address of 00:16:D3:23:68:8A (Bob's laptop) and sends the frame to the switch, which delivers the frame to Bob's laptop.

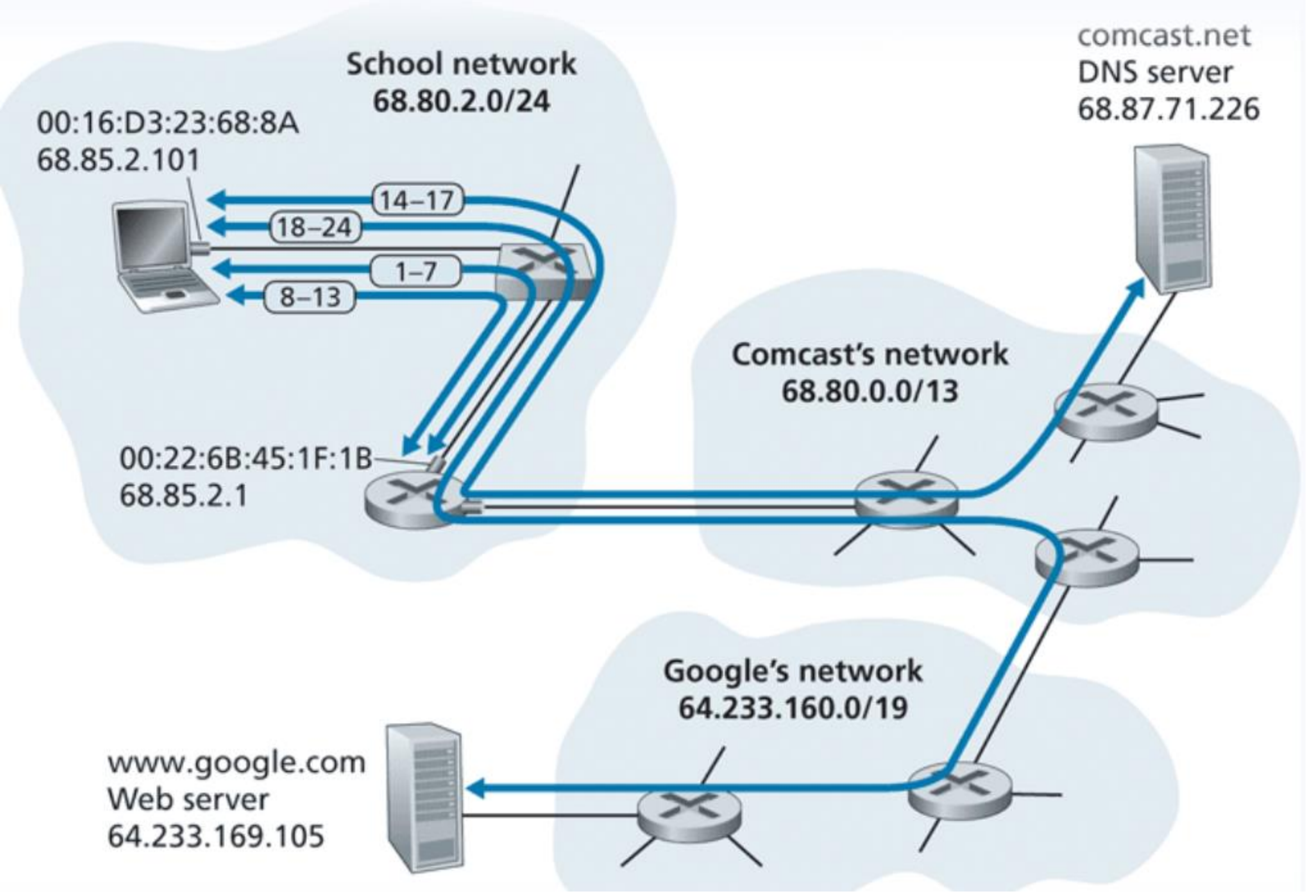


12. Bob's laptop receives the frame containing the ARP reply message and extracts the MAC address of the gateway router (00:22:6B:45:1F:1B) from the ARP reply message.

13. Bob's laptop can now (*finally!*) address the Ethernet frame containing the DNS query to the gateway router's MAC address. Note that the IP datagram in this frame has an IP destination address of 68.87.71.226 (the DNS server), while the frame has a destination address of 00:22:6B:45:1F:1B (the gateway router). Bob's laptop sends this frame to the switch, which delivers the frame to the gateway router.

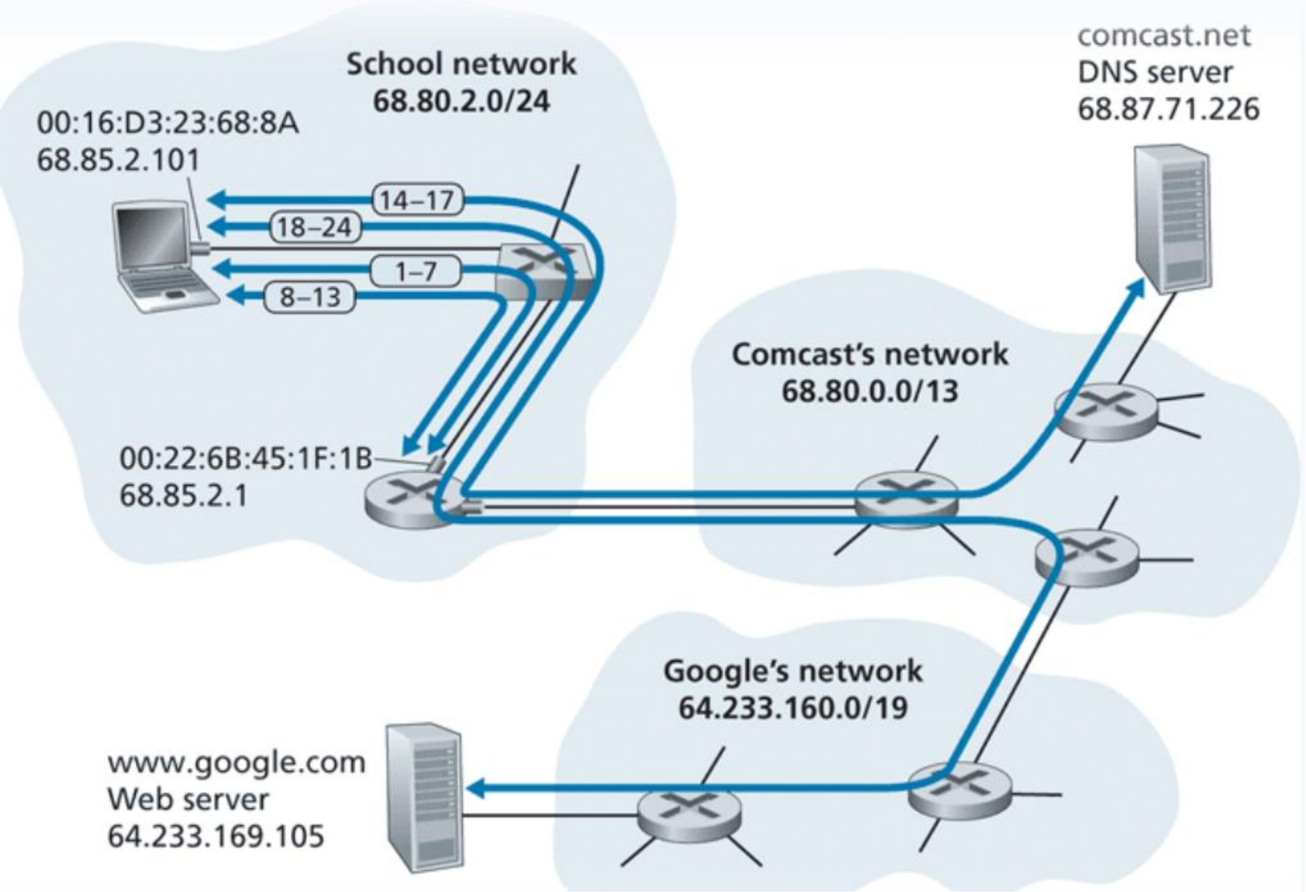
Whew… ok we can now address something to the DNS server

14. The gateway router receives the frame and extracts the IP datagram containing the DNS query. The router looks up the destination address of this datagram (68.87.71.226) and determines from its forwarding table that the datagram should be sent to the leftmost router in the Comcast network in **Figure 6.32**. The IP datagram is placed inside a link-layer frame appropriate for the link connecting the school's router to the leftmost Comcast router and the frame is sent over this link.



14. The gateway router receives the frame and extracts the IP datagram containing the DNS query. The router looks up the destination address of this datagram (68.87.71.226) and determines from its forwarding table that the datagram should be sent to the leftmost router in the Comcast network in **Figure 6.32**. The IP datagram is placed inside a link-layer frame appropriate for the link connecting the school's router to the leftmost Comcast router and the frame is sent over this link.

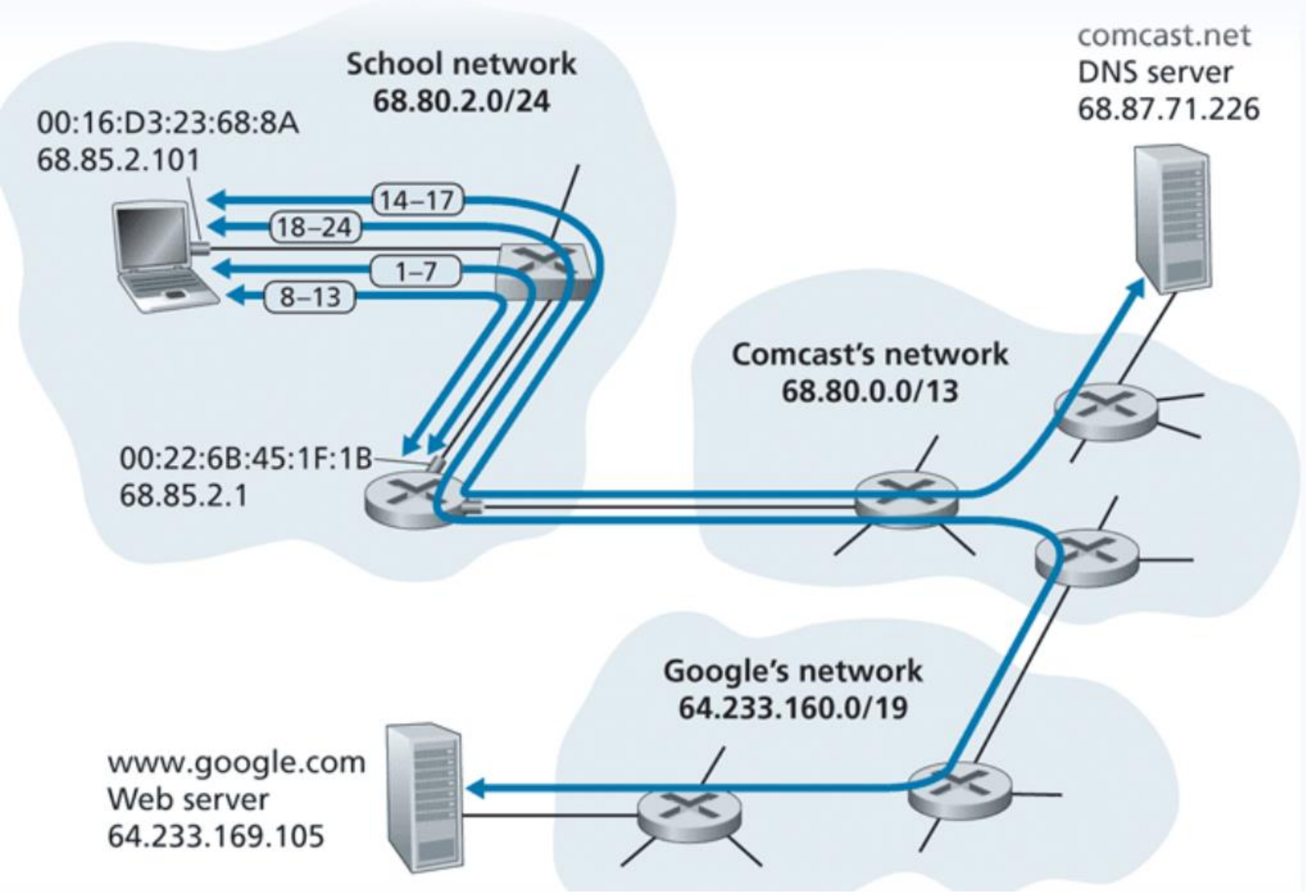
15. The leftmost router in the Comcast network receives the frame, extracts the IP datagram, examines the datagram's destination address (68.87.71.226) and determines the outgoing interface on which to forward the datagram toward the DNS server from its forwarding table, which has been filled in by Comcast's intra-domain protocol (such as **RIP**, **OSPF** or **IS-IS**, **Section 5.3**) as well as the **Internet's inter-domain protocol**, **BGP** (**Section 5.4**).



14. The gateway router receives the frame and extracts the IP datagram containing the DNS query. The router looks up the destination address of this datagram (68.87.71.226) and determines from its forwarding table that the datagram should be sent to the leftmost router in the Comcast network in **Figure 6.32**. The IP datagram is placed inside a link-layer frame appropriate for the link connecting the school's router to the leftmost Comcast router and the frame is sent over this link.

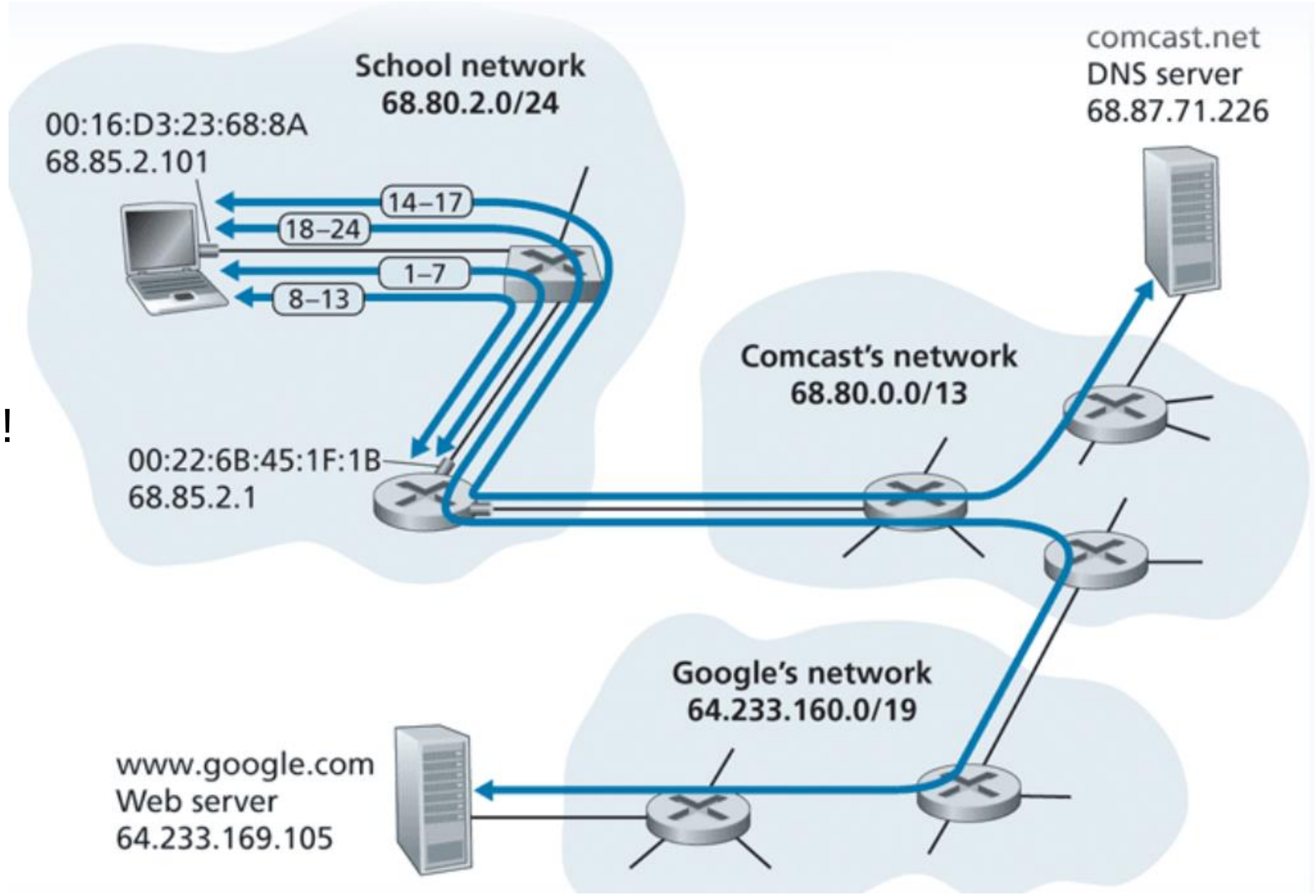
15. The leftmost router in the Comcast network receives the frame, extracts the IP datagram, examines the datagram's destination address (68.87.71.226) and determines the outgoing interface on which to forward the datagram toward the DNS server from its forwarding table, which has been filled in by Comcast's intra-domain protocol (such as **RIP**, **OSPF** or **IS-IS**, **Section 5.3**) as well as the **Internet's inter-domain protocol**, **BGP** (**Section 5.4**).

16. Eventually the IP datagram containing the DNS query arrives at the DNS server. The DNS server extracts the DNS query message, looks up the name www.google.com in its DNS database (**Section 2.5**), and finds the **DNS resource record** that contains the IP address (64.233.169.105) for www.google.com. (assuming that it is currently cached in the DNS server). Recall that this cached data originated in the **authoritative DNS server** (**Section 2.5.2**) for google.com. The DNS server forms a **DNS reply message** containing this hostname-to-IP-address mapping, and places the DNS reply message in a UDP segment, and the segment within an IP datagram addressed to Bob's laptop (68.85.2.101). This datagram will be forwarded back through the Comcast network to the school's router and from there, via the Ethernet switch to Bob's laptop.

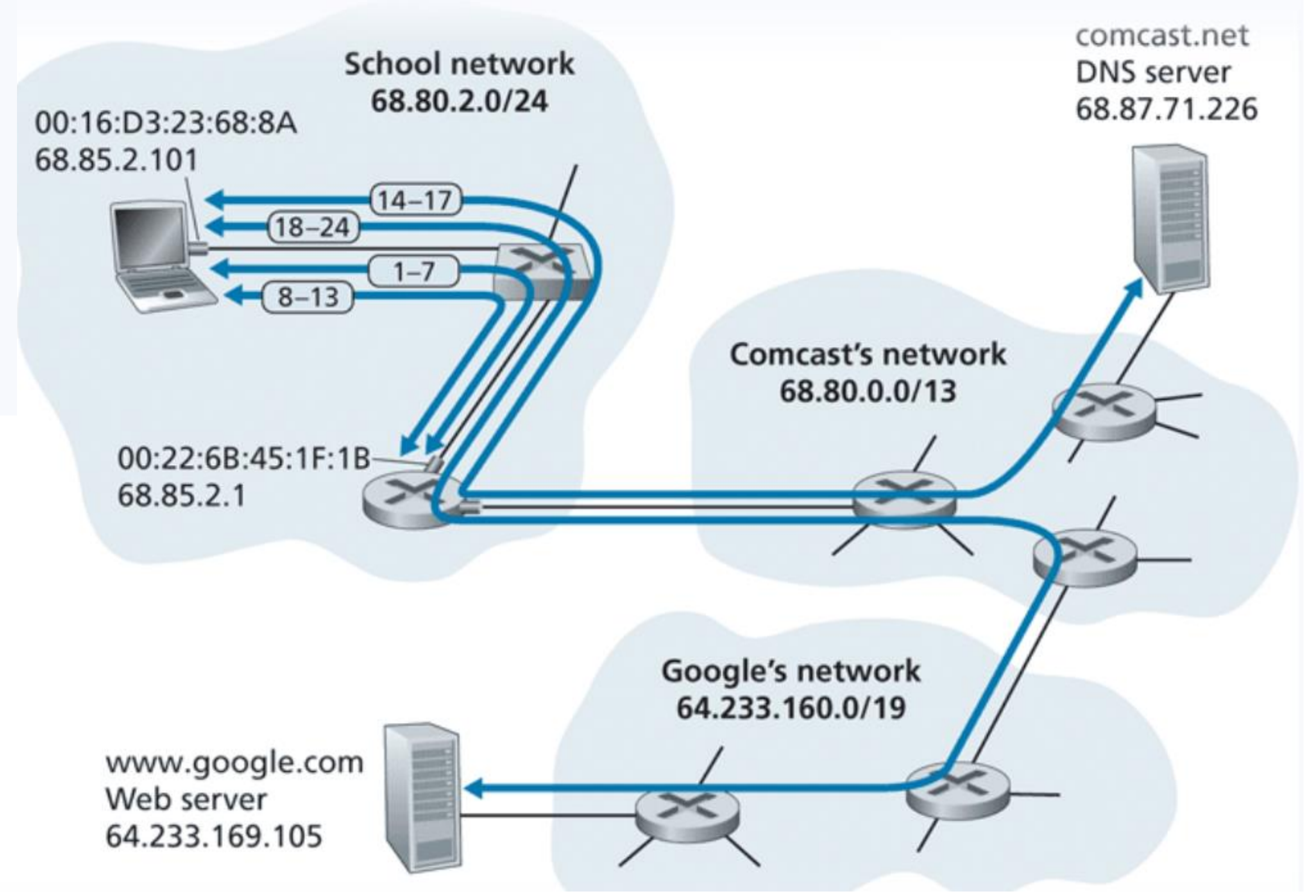


17. Bob's laptop extracts the IP address of the server www.google.com from the DNS message. *Finally*, after a *lot* of work, Bob's laptop is now ready to contact the www.google.com server!

We are finally ready to contact the google server!

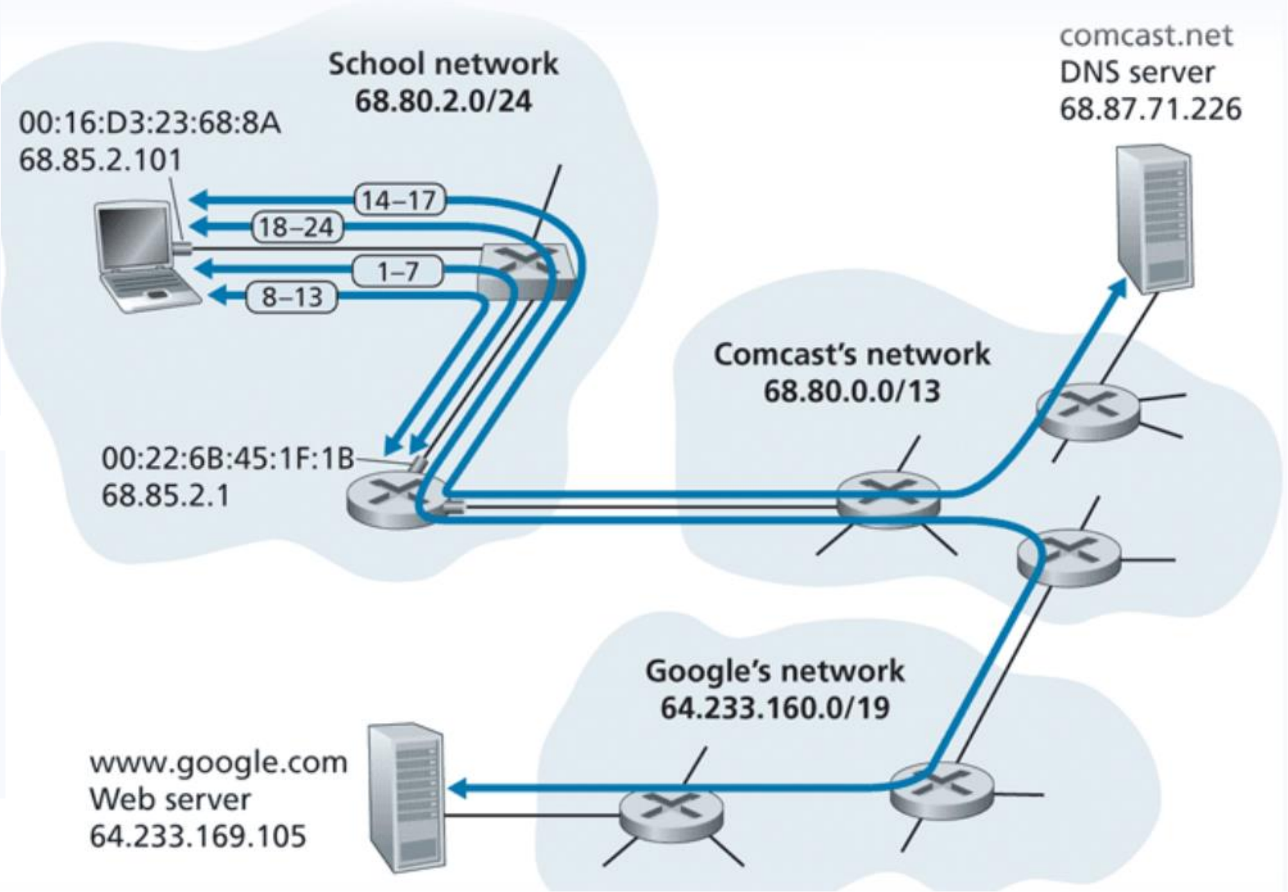


18. Now that Bob's laptop has the IP address of www.google.com, it can create the **TCP socket** (**Section 2.7**) that will be used to send the **HTTP GET** message (**Section 2.2.3**) to www.google.com. When Bob creates the TCP socket, the TCP in Bob's laptop must first perform a **three-way handshake** (**Section 3.5.6**) with the TCP in www.google.com. Bob's laptop thus first creates a **TCP SYN** segment with destination port 80 (for HTTP), places the TCP segment inside an IP datagram with a destination IP address of 64.233.169.105 (www.google.com), places the datagram inside a frame with a destination MAC address of 00:22:6B:45:1F:1B (the gateway router) and sends the frame to the switch.

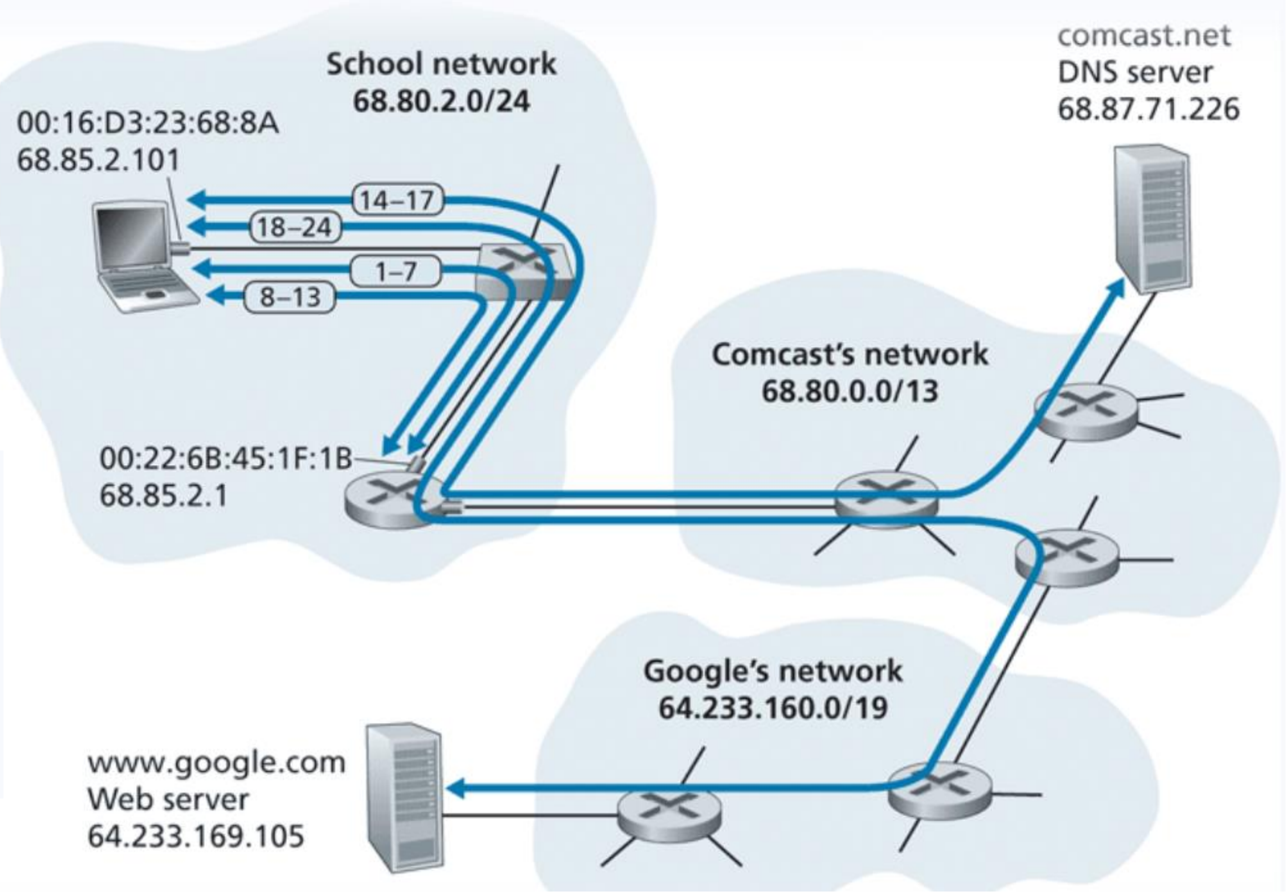


18. Now that Bob's laptop has the IP address of www.google.com, it can create the **TCP socket** (**Section 2.7**) that will be used to send the **HTTP GET** message (**Section 2.2.3**) to www.google.com. When Bob creates the TCP socket, the TCP in Bob's laptop must first perform a **three-way handshake** (**Section 3.5.6**) with the TCP in www.google.com. Bob's laptop thus first creates a **TCP SYN** segment with destination port 80 (for HTTP), places the TCP segment inside an IP datagram with a destination IP address of 64.233.169.105 (www.google.com), places the datagram inside a frame with a destination MAC address of 00:22:6B:45:1F:1B (the gateway router) and sends the frame to the switch.

19. The routers in the school network, Comcast's network, and Google's network forward the datagram containing the TCP SYN toward www.google.com, using the forwarding table in each router, as in steps 14–16 above. Recall that the router forwarding table entries governing forwarding of packets over the inter-domain link between the Comcast and Google networks are determined by the **BGP** protocol (**Chapter 5**).



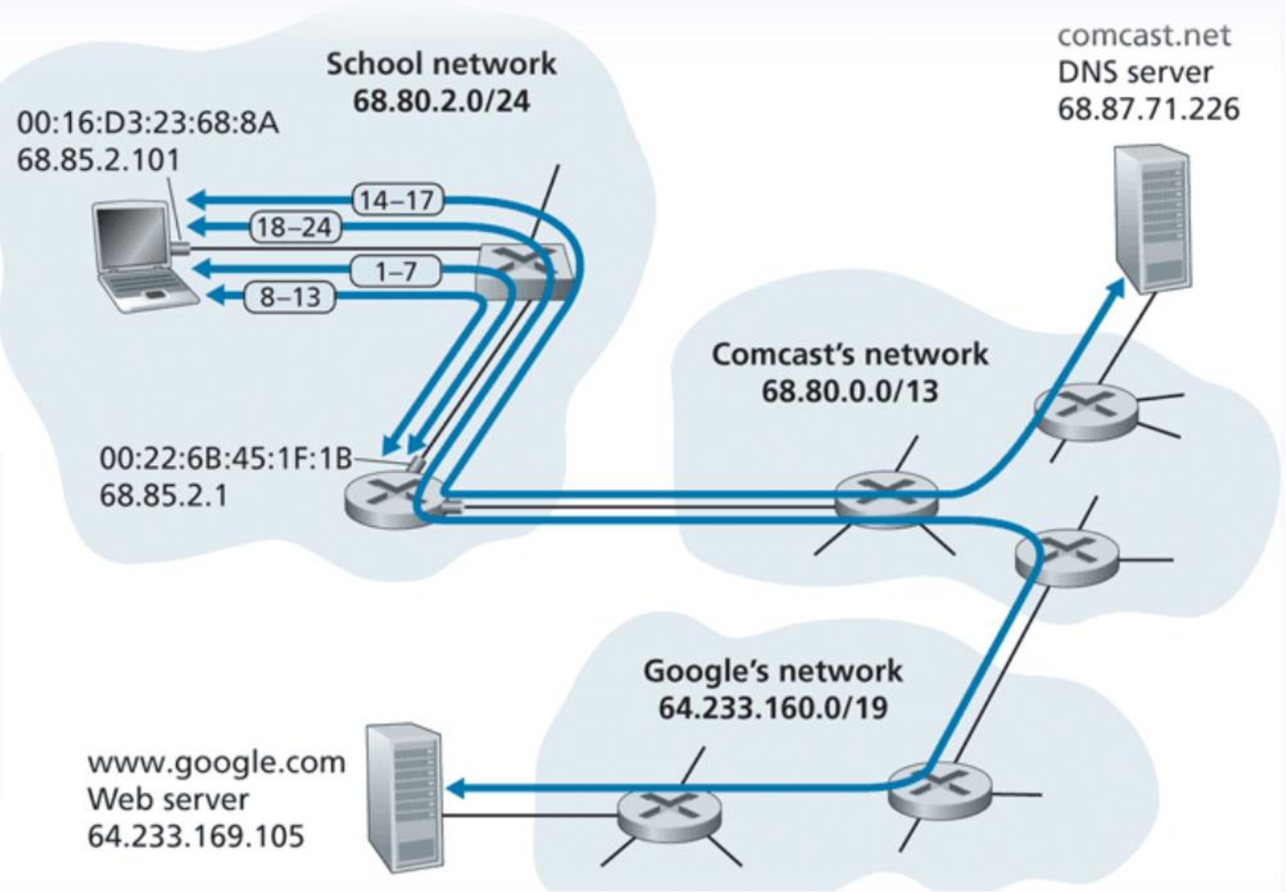
18. Now that Bob's laptop has the IP address of www.google.com, it can create the **TCP socket** (**Section 2.7**) that will be used to send the **HTTP GET** message (**Section 2.2.3**) to www.google.com. When Bob creates the TCP socket, the TCP in Bob's laptop must first perform a **three-way handshake** (**Section 3.5.6**) with the TCP in www.google.com. Bob's laptop thus first creates a **TCP SYN** segment with destination port 80 (for HTTP), places the TCP segment inside an IP datagram with a destination IP address of 64.233.169.105 (www.google.com), places the datagram inside a frame with a destination MAC address of 00:22:6B:45:1F:1B (the gateway router) and sends the frame to the switch.



19. The routers in the school network, Comcast's network, and Google's network forward the datagram containing the TCP SYN toward www.google.com, using the forwarding table in each router, as in steps 14–16 above. Recall that the router forwarding table entries governing forwarding of packets over the inter-domain link between the Comcast and Google networks are determined by the **BGP** protocol (**Chapter 5**).

20. Eventually, the datagram containing the TCP SYN arrives at www.google.com. The TCP SYN message is extracted from the datagram and demultiplexed to the welcome socket associated with port 80. A connection socket (**Section 2.7**) is created for the TCP connection between the Google HTTP server and Bob's laptop. A TCP SYNACK (**Section 3.5.6**) segment is generated, placed inside a datagram addressed to Bob's laptop, and finally placed inside a link-layer frame appropriate for the link connecting www.google.com to its first-hop router.

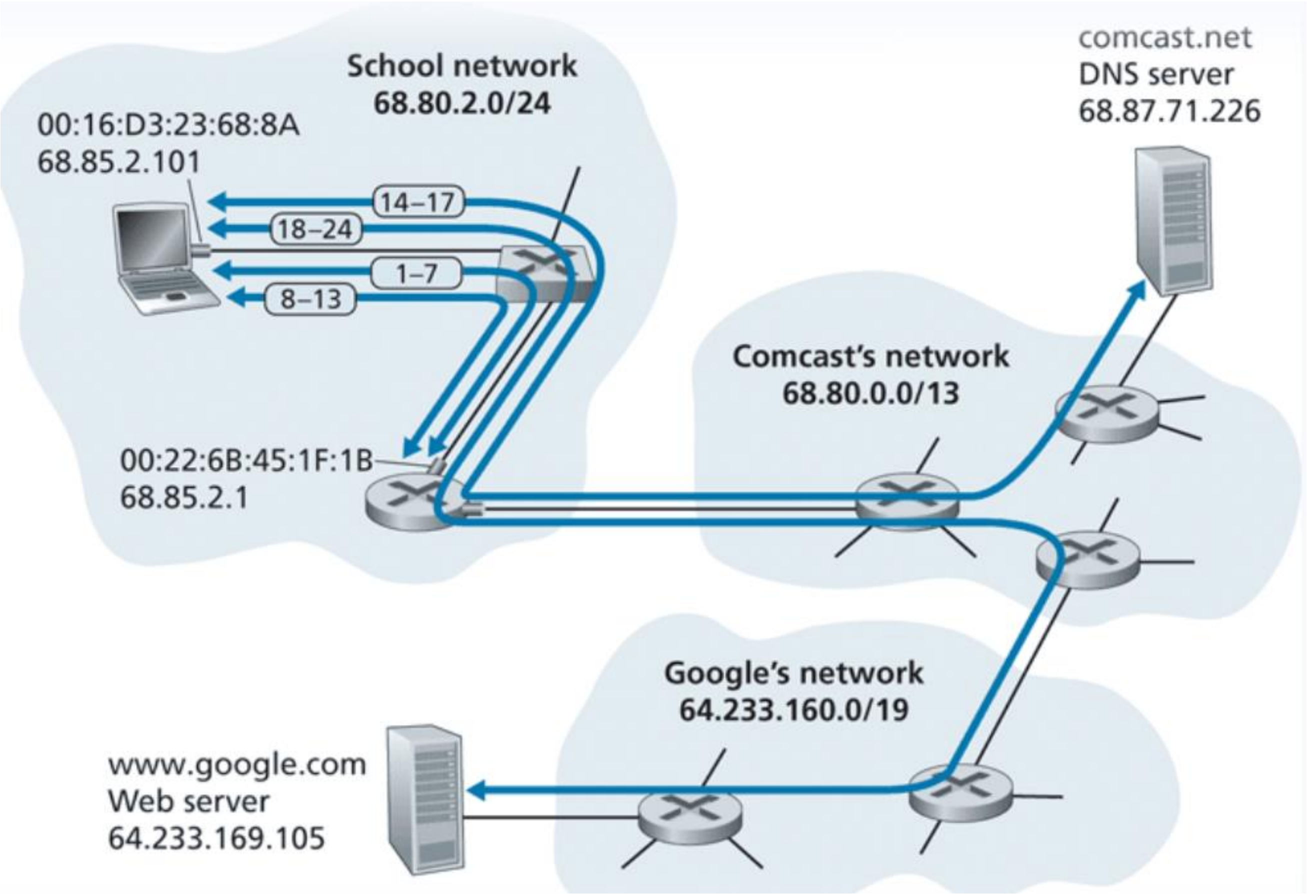
18. Now that Bob's laptop has the IP address of www.google.com, it can create the **TCP socket** (**Section 2.7**) that will be used to send the **HTTP GET** message (**Section 2.2.3**) to www.google.com. When Bob creates the TCP socket, the TCP in Bob's laptop must first perform a **three-way handshake** (**Section 3.5.6**) with the TCP in www.google.com. Bob's laptop thus first creates a **TCP SYN** segment with destination port 80 (for HTTP), places the TCP segment inside an IP datagram with a destination IP address of 64.233.169.105 (www.google.com), places the datagram inside a frame with a destination MAC address of 00:22:6B:45:1F:1B (the gateway router) and sends the frame to the switch.



19. The routers in the school network, Comcast's network, and Google's network forward the datagram containing the TCP SYN toward www.google.com, using the forwarding table in each router, as in steps 14–16 above. Recall that the router forwarding table entries governing forwarding of packets over the inter-domain link between the Comcast and Google networks are determined by the **BGP** protocol (**Chapter 5**).

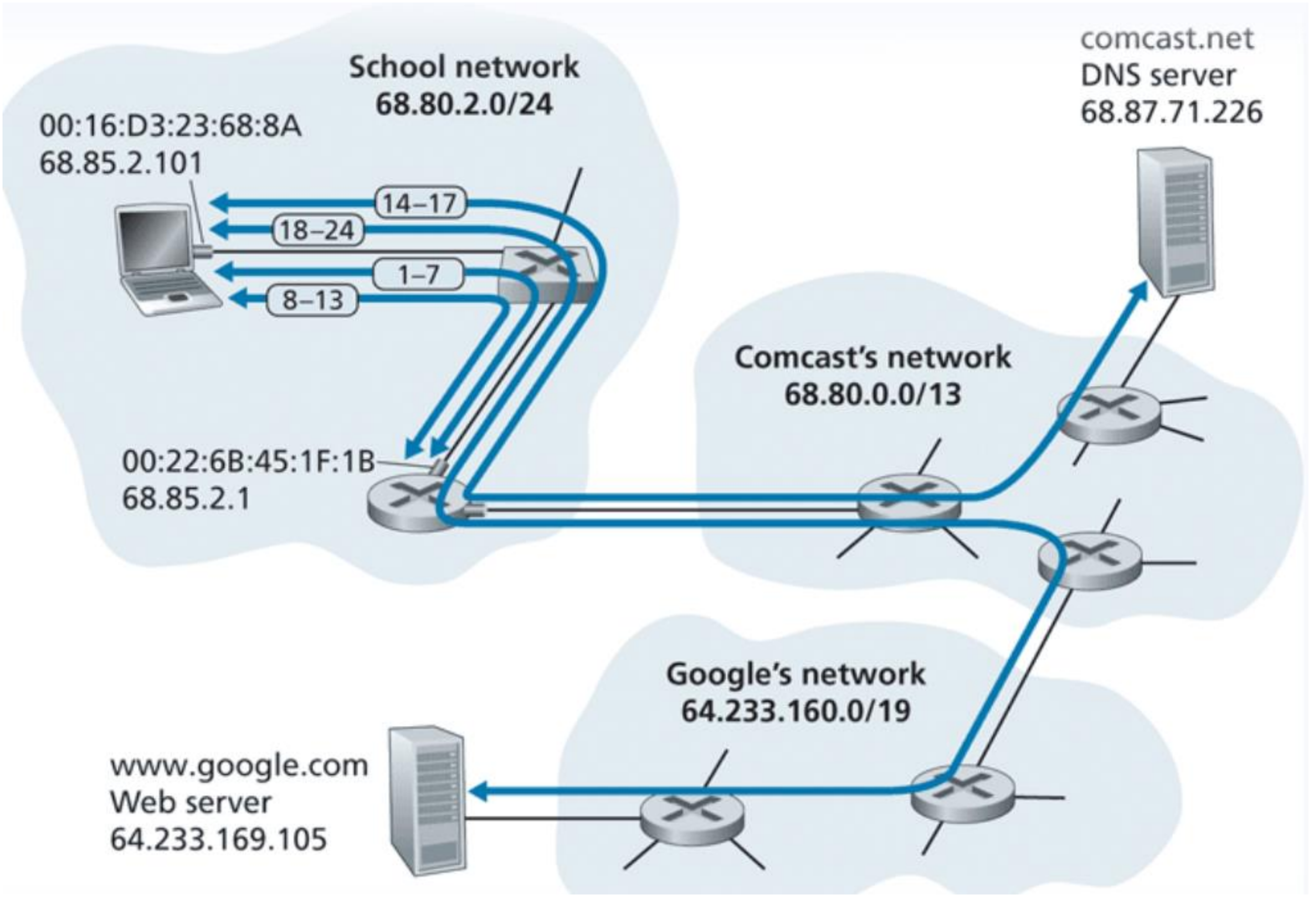
20. Eventually, the datagram containing the TCP SYN arrives at www.google.com. The TCP SYN message is extracted from the datagram and demultiplexed to the welcome socket associated with port 80. A connection socket (**Section 2.7**) is created for the TCP connection between the Google HTTP server and Bob's laptop. A TCP SYNACK (**Section 3.5.6**) segment is generated, placed inside a datagram addressed to Bob's laptop, and finally placed inside a link-layer frame appropriate for the link connecting www.google.com to its first-hop router.

21. The datagram containing the TCP SYNACK segment is forwarded through the Google, Comcast, and school networks, eventually arriving at the Ethernet controller in Bob's laptop. The datagram is demultiplexed within the operating system to the TCP socket created in step 18, which enters the connected state.



21. The datagram containing the TCP SYNACK segment is forwarded through the Google, Comcast, and school networks, eventually arriving at the Ethernet controller in Bob's laptop. The datagram is demultiplexed within the operating system to the TCP socket created in step 18, which enters the connected state.

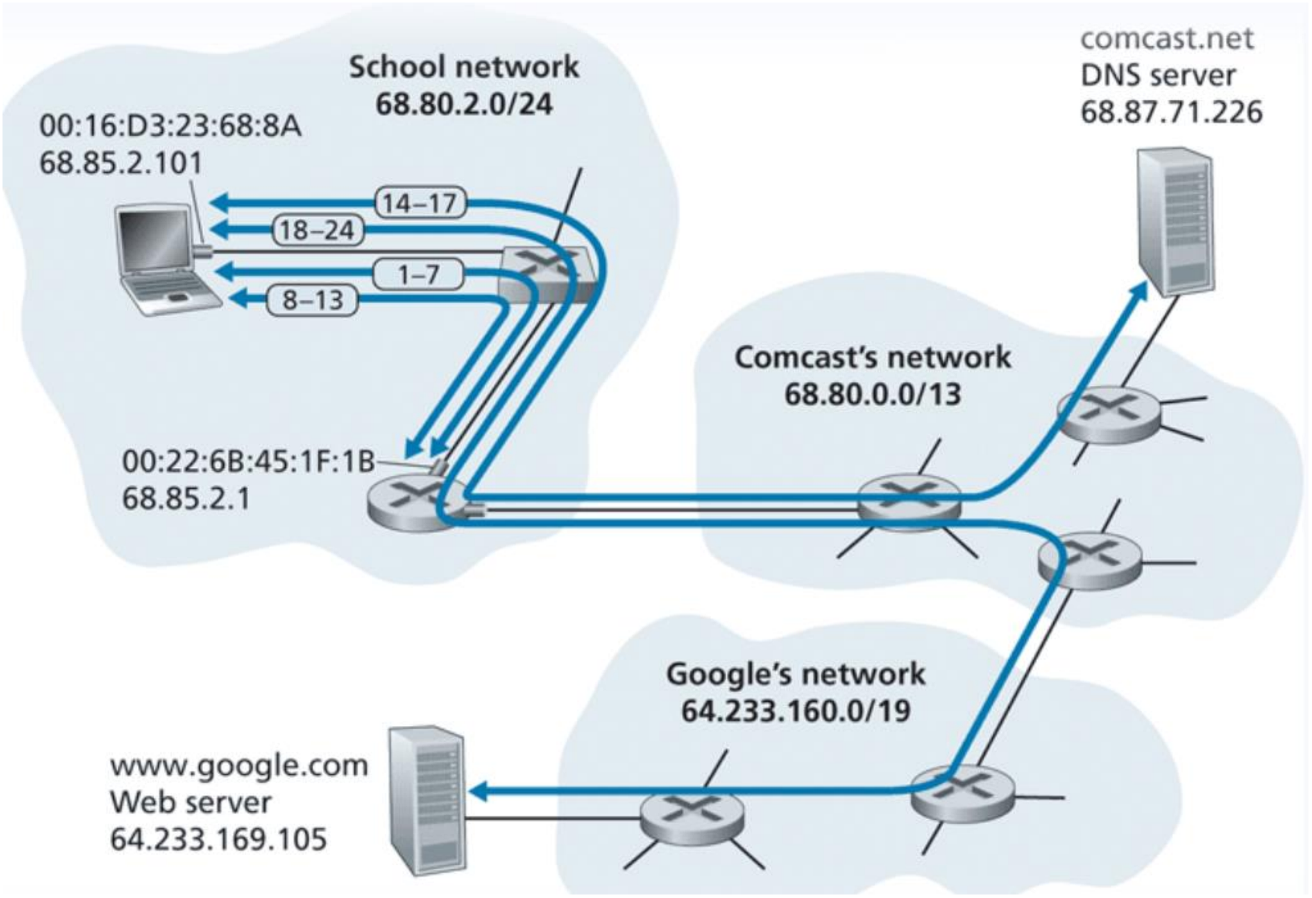
22. With the socket on Bob's laptop now (*finally!*) ready to send bytes to www.google.com, Bob's browser creates the HTTP GET message (**Section 2.2.3**) containing the URL to be fetched. The HTTP GET message is then written into the socket, with the GET message becoming the payload of a TCP segment. The TCP segment is placed in a datagram and sent and delivered to www.google.com as in steps 18–20 above.



21. The datagram containing the TCP SYNACK segment is forwarded through the Google, Comcast, and school networks, eventually arriving at the Ethernet controller in Bob's laptop. The datagram is demultiplexed within the operating system to the TCP socket created in step 18, which enters the connected state.

22. With the socket on Bob's laptop now (*finally!*) ready to send bytes to www.google.com, Bob's browser creates the HTTP GET message (**Section 2.2.3**) containing the URL to be fetched. The HTTP GET message is then written into the socket, with the GET message becoming the payload of a TCP segment. The TCP segment is placed in a datagram and sent and delivered to www.google.com as in steps 18–20 above.

23. The HTTP server at www.google.com reads the HTTP GET message from the TCP socket, creates an **HTTP response** message (**Section 2.2**), places the requested Web page content in the body of the HTTP response message, and sends the message into the TCP socket.



21. The datagram containing the TCP SYNACK segment is forwarded through the Google, Comcast, and school networks, eventually arriving at the Ethernet controller in Bob's laptop. The datagram is demultiplexed within the operating system to the TCP socket created in step 18, which enters the connected state.

22. With the socket on Bob's laptop now (*finally!*) ready to send bytes to www.google.com, Bob's browser creates the HTTP GET message (**Section 2.2.3**) containing the URL to be fetched. The HTTP GET message is then written into the socket, with the GET message becoming the payload of a TCP segment. The TCP segment is placed in a datagram and sent and delivered to www.google.com as in steps 18–20 above.

23. The HTTP server at www.google.com reads the HTTP GET message from the TCP socket, creates an **HTTP response** message (**Section 2.2**), places the requested Web page content in the body of the HTTP response message, and sends the message into the TCP socket.

24. The datagram containing the HTTP reply message is forwarded through the Google, Comcast, and school networks, and arrives at Bob's laptop. Bob's Web browser program reads the HTTP response from the socket, extracts the html for the Web page from the body of the HTTP response, and finally (*finally!*) displays the Web page!