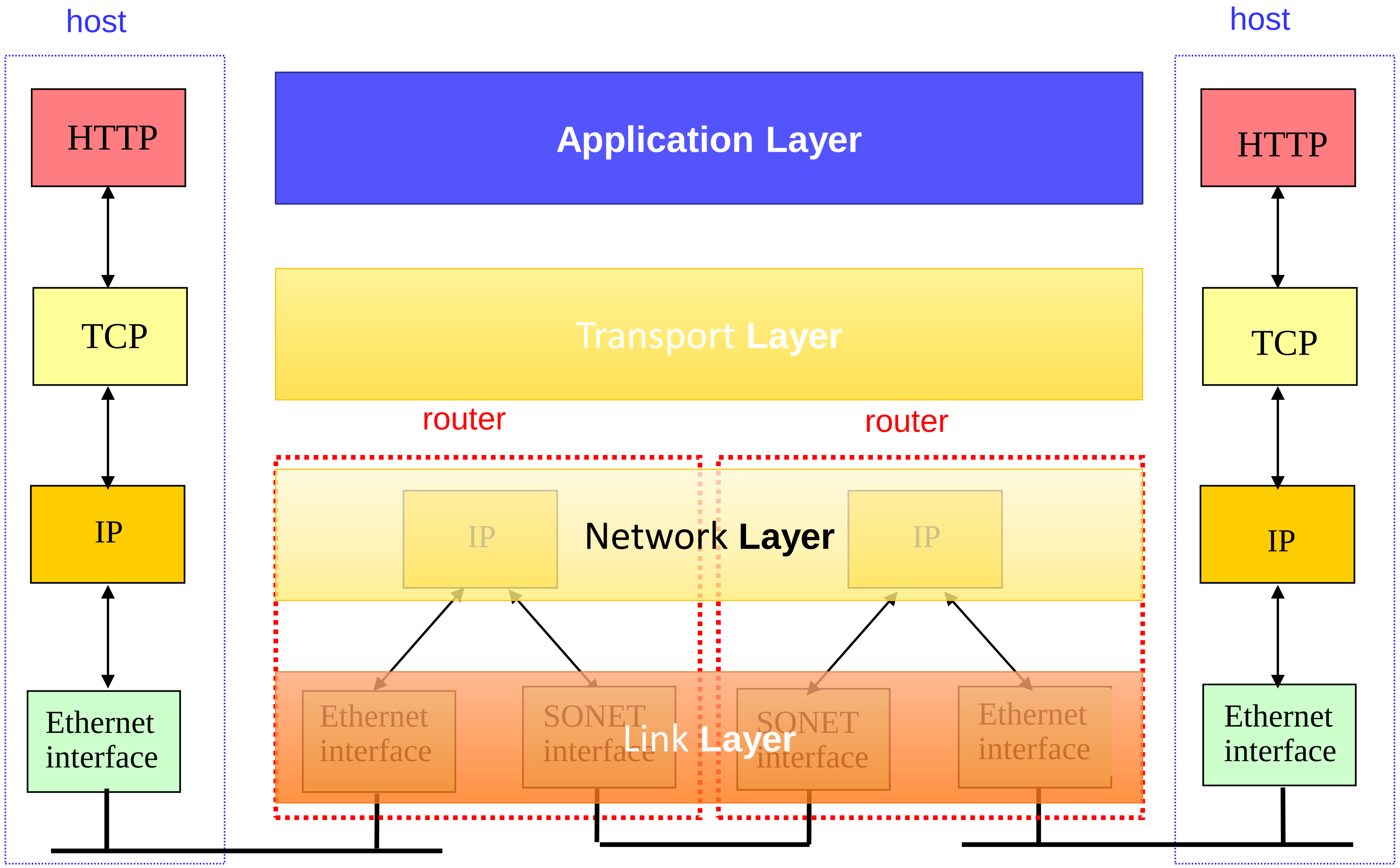
CS 43: Computer Networks
The Link Layer

Kevin Webb

Swarthmore College

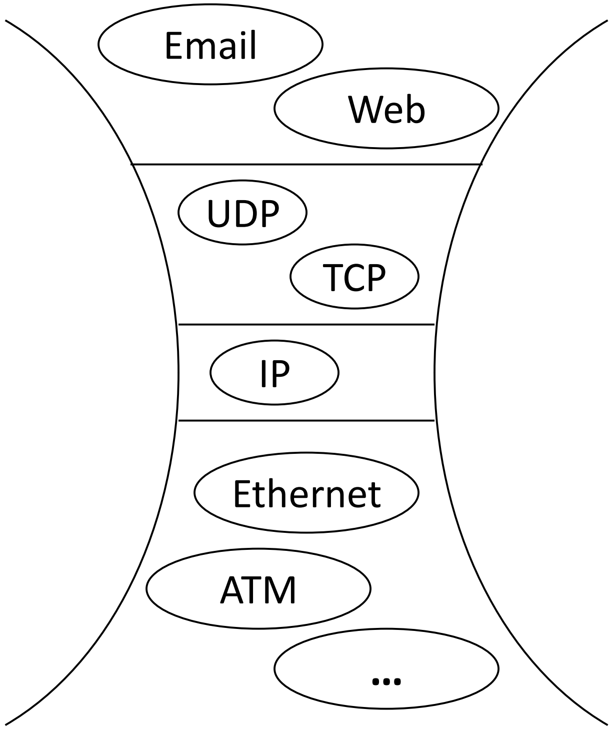
April 19, 2022

TCP/IP Protocol Stack



Internet Protocol Stack

- Application: Email, Web, …
- Transport: TCP, UDP, …
- Network: IP
- Link: Ethernet, WiFi, SONET, …
- Physical: copper, fiber, air, …



- “Hourglass” model, “thin waist”, “narrow waist”

Recall IP Motivation

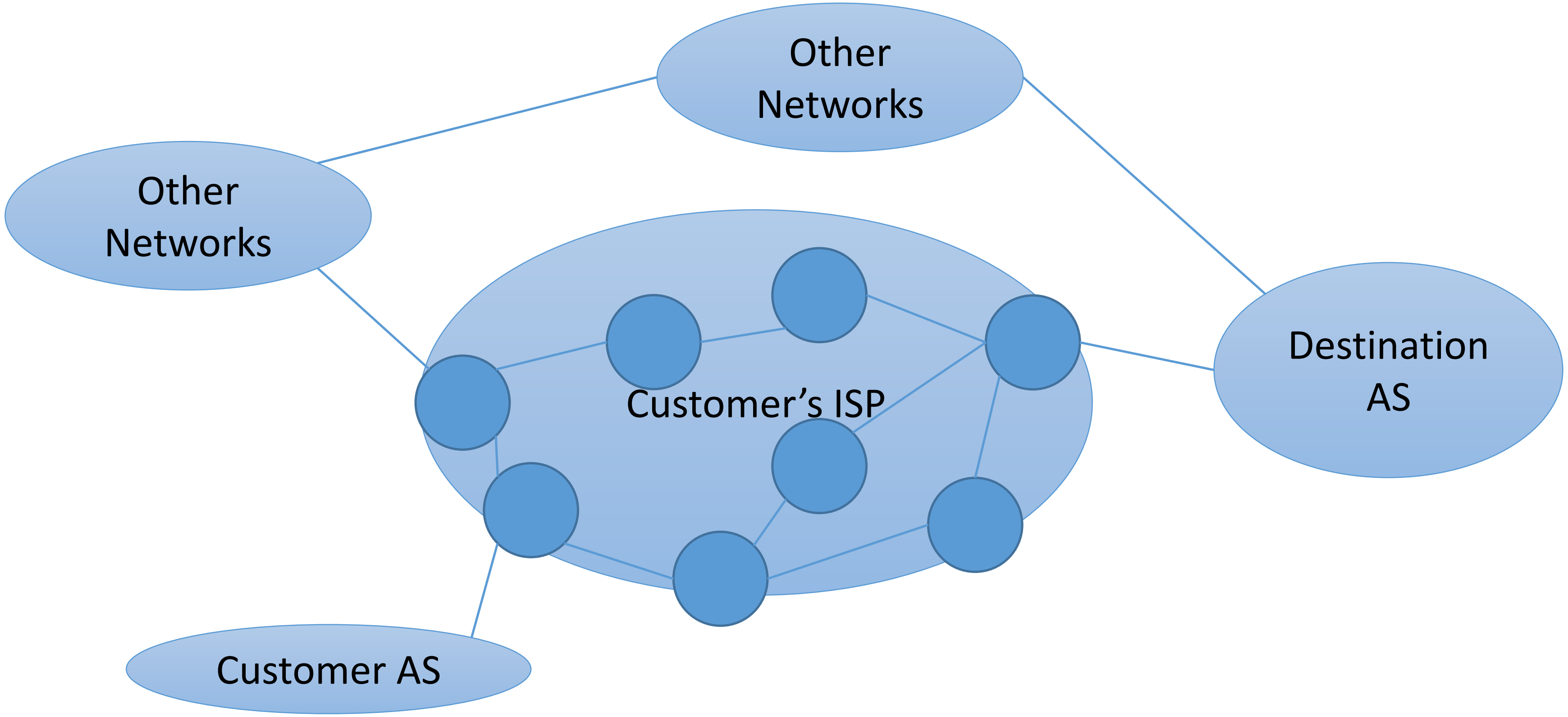
- 1970’s: new network technologies emerge
  - SATNet, Packet Radio, Ethernet
  - All “islands” to themselves – didn’t work together

- IP question: how to connect these networks?

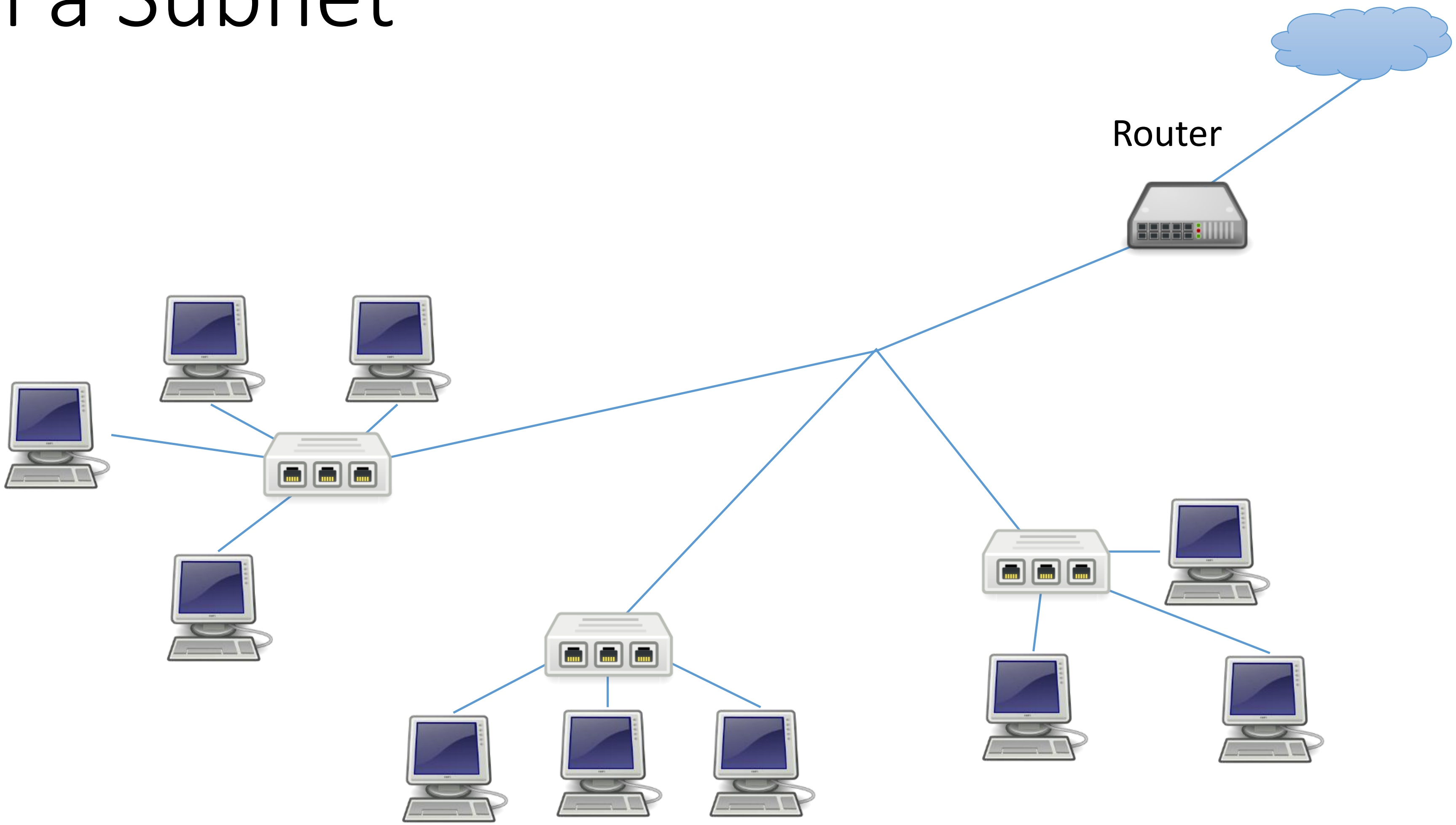
- This implies: These networks do all the stuff networks need to do, without IP or routers.
  - Solves some of the same problems as IP
  - Often in a different way (smaller scale)

From Macro- to Micro-

- Previously, we looked at Internet scale…



Within a Subnet



Link Layer Goal

- Get from one node to it's nearby neighbor on the same IP network.
- Abstract the details of the underlying network technology from the protocols above it (IP).
- Lots of media with different characteristics:
  - Copper cable
  - Fiber optic cable
  - Radio/electromagnetic broadcast
  - Satellite

Challenges

- Even with one medium:
    - Potentially many ways to format & signal data.
    - Multiple users may contend to transmit.
    - How do we address endpoints?
    - How do we locate destinations?

Link Layer Functions

1. Addressing: identifying endpoints

- Must be able to uniquely identify each host on the network. Can’t assume IP.

- Implication: each host on the Internet will have **two** addresses: IP & link-layer

Typically referred to as “MAC address”
Media Access Control

Addressing

- Typically, humans deal in IP addresses
  (or DNS names that resolve to them)

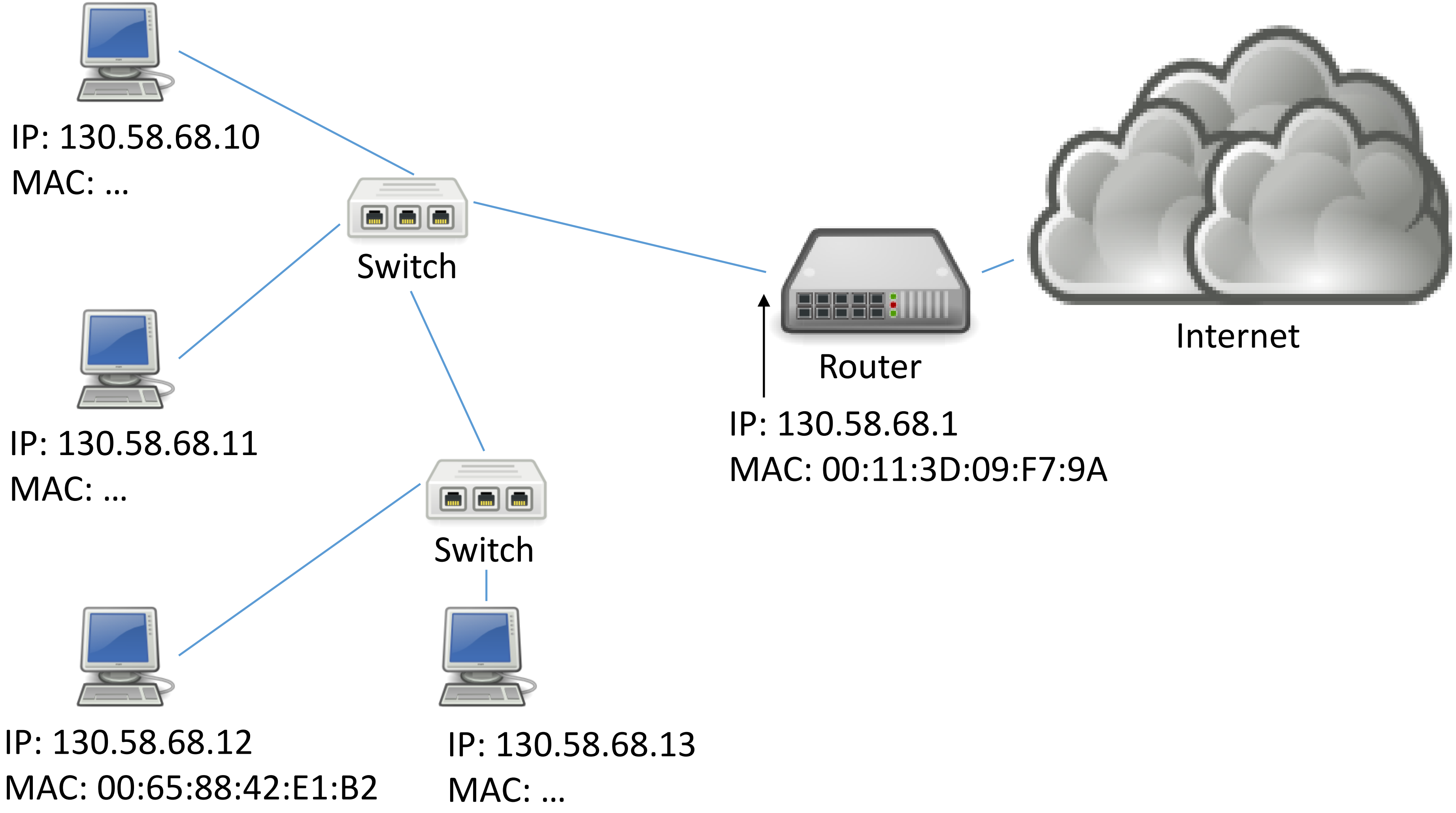
- Network needs a mechanism to determine corresponding MAC address for local sending

ARP: Address Resolution Protocol

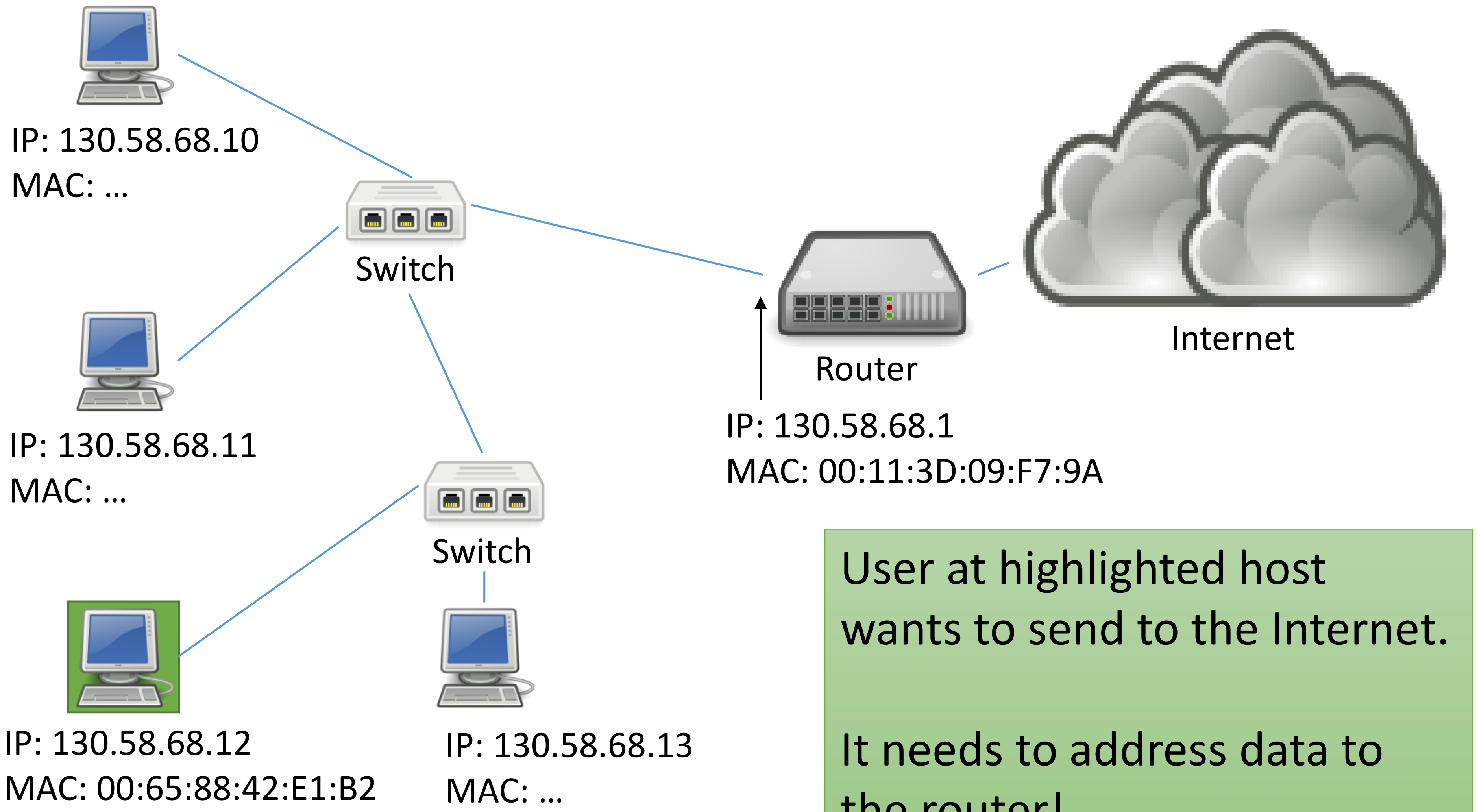
- Common in networks you use: Ethernet, WiFi
- Broadcast to entire local network:
    - "I'm looking for the MAC address of the host with IP address A.B.C.D. If you're out there, please respond to me!"

- ~~You will implement this in lab 7!~~

ARP Example


IP: 130.58.68.10
MAC: …
Switch
IP: 130.58.68.11
MAC: …
Switch
IP: 130.58.68.12
MAC: 00:65:88:42:E1:B2
IP: 130.58.68.13
MAC: …
Router
IP: 130.58.68.1
MAC: 00:11:3D:09:F7:9A
Internet

ARP Example



IP: 130.58.68.10
MAC: …

Switch

IP: 130.58.68.11
MAC: …

Switch

IP: 130.58.68.12
MAC: 00:65:88:42:E1:B2

IP: 130.58.68.13
MAC: …

Router

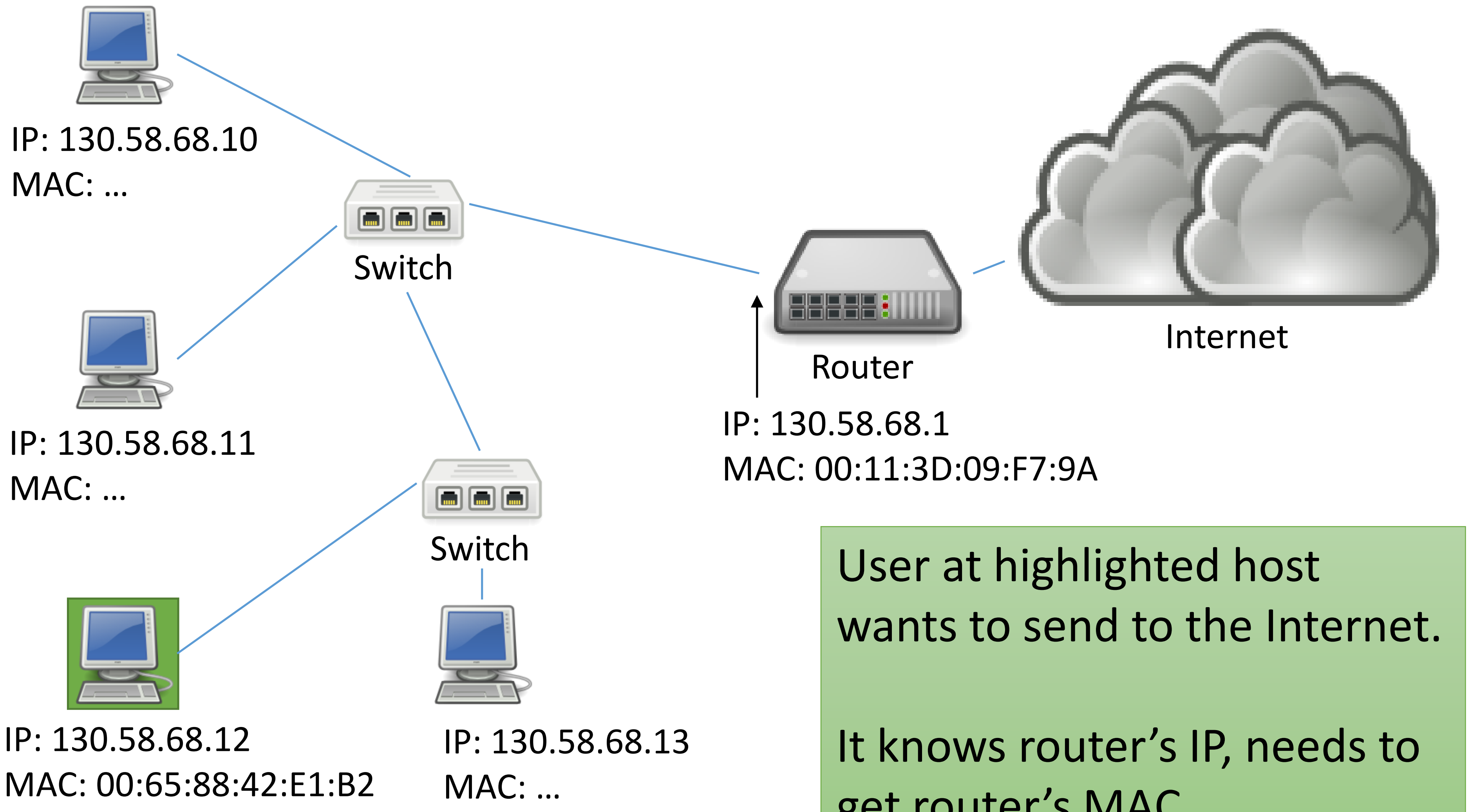
IP: 130.58.68.1
MAC: 00:11:3D:09:F7:9A

Internet

User at highlighted host wants to send to the Internet.

It needs to address data to the router!

ARP Example



IP: 130.58.68.10
MAC: …

Switch

IP: 130.58.68.11
MAC: …

Switch

IP: 130.58.68.12
MAC: 00:65:88:42:E1:B2

IP: 130.58.68.13
MAC: …

Router

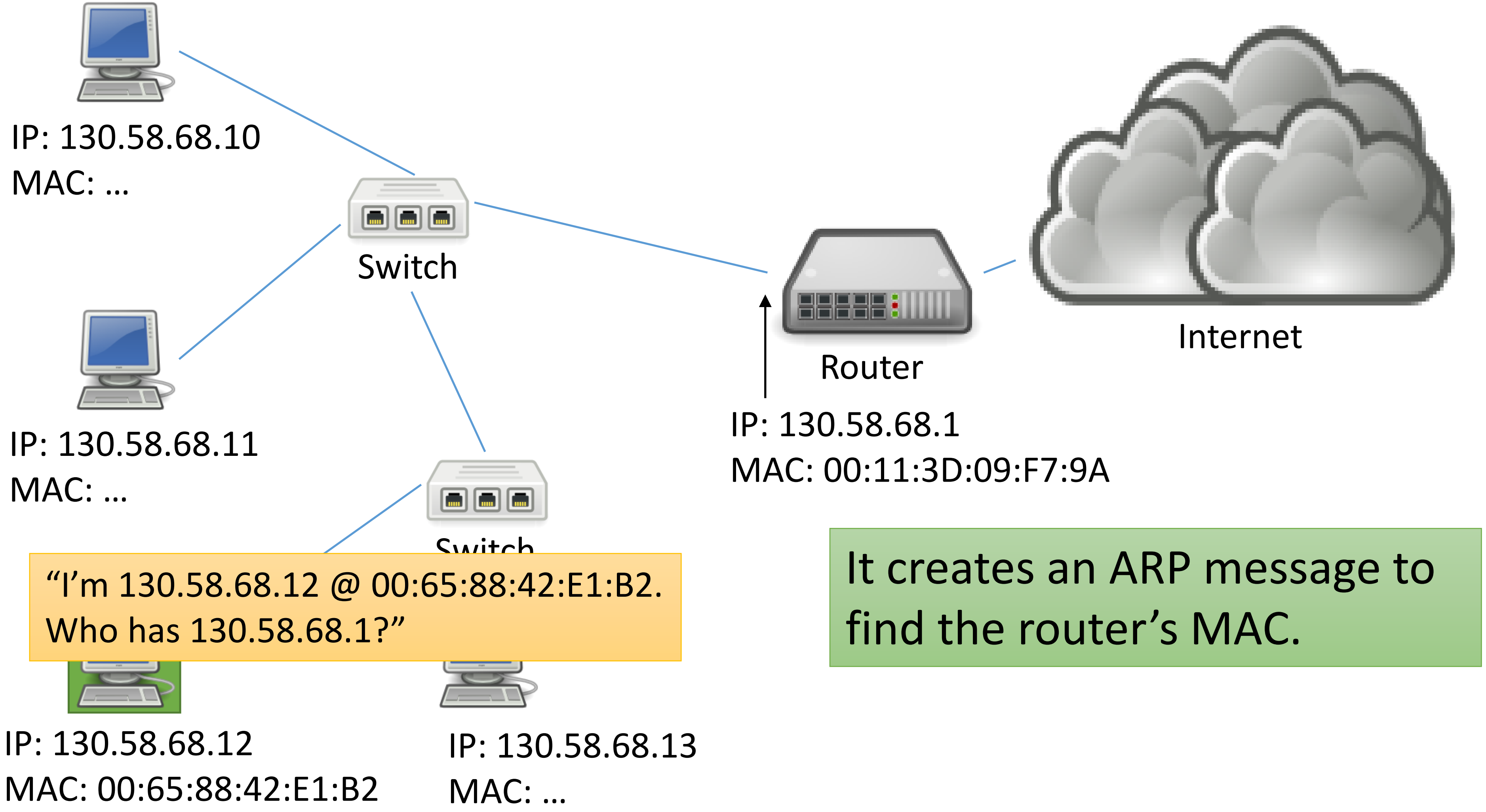
IP: 130.58.68.1
MAC: 00:11:3D:09:F7:9A

Internet

User at highlighted host wants to send to the Internet.

It knows router’s IP, needs to get router’s MAC.

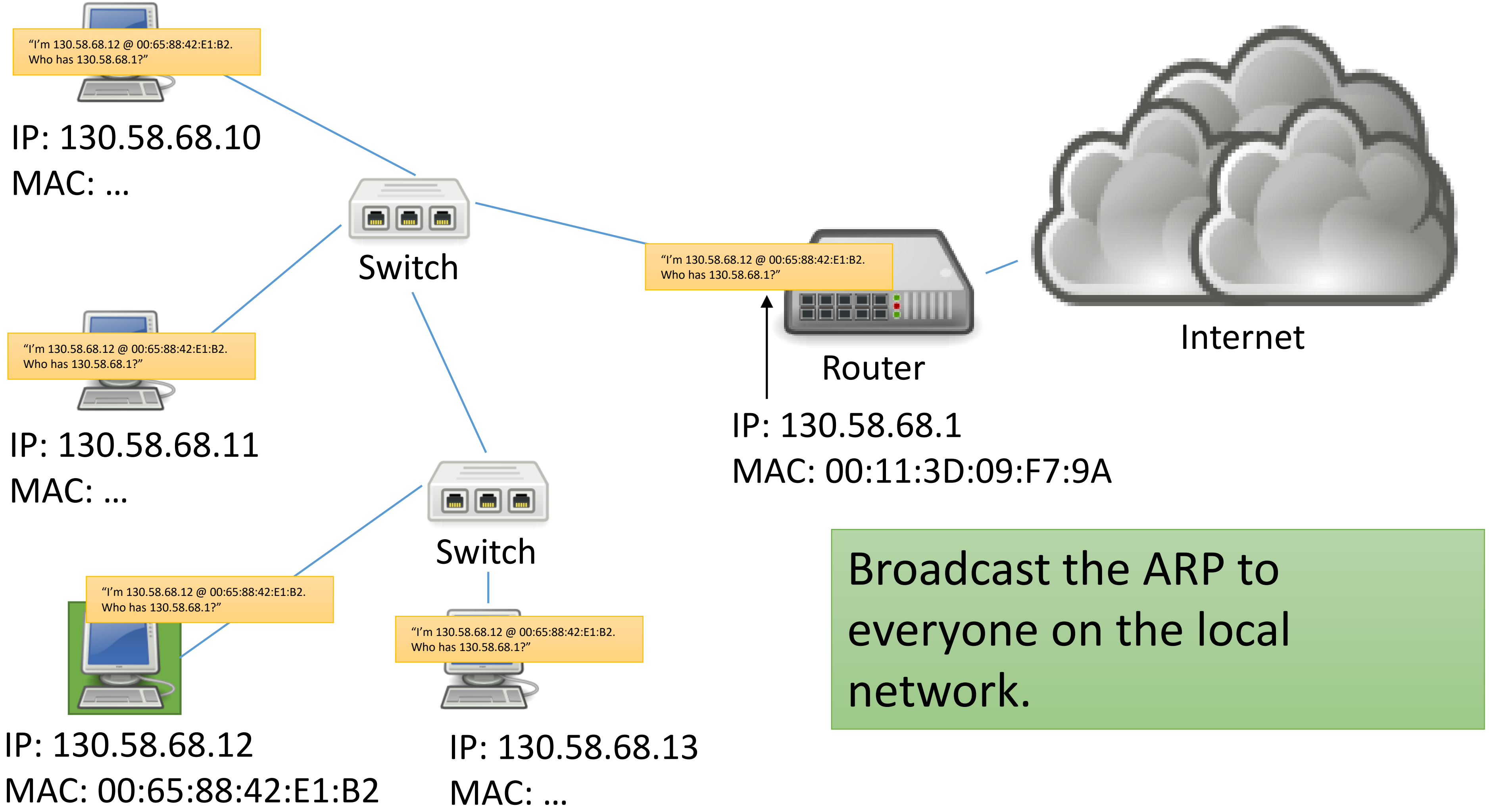
ARP Example



"I'm 130.58.68.12 @ 00:65:88:42:E1:B2.
Who has 130.58.68.1?"

It creates an ARP message to find the router's MAC.

ARP Example



"I'm 130.58.68.12 @ 00:65:88:42:E1:B2. Who has 130.58.68.1?"

IP: 130.58.68.10
MAC: …

Switch

"I'm 130.58.68.12 @ 00:65:88:42:E1:B2. Who has 130.58.68.1?"

IP: 130.58.68.11
MAC: …

Switch

"I'm 130.58.68.12 @ 00:65:88:42:E1:B2. Who has 130.58.68.1?"

IP: 130.58.68.12
MAC: 00:65:88:42:E1:B2

"I'm 130.58.68.12 @ 00:65:88:42:E1:B2. Who has 130.58.68.1?"

IP: 130.58.68.13
MAC: …

"I'm 130.58.68.12 @ 00:65:88:42:E1:B2. Who has 130.58.68.1?"

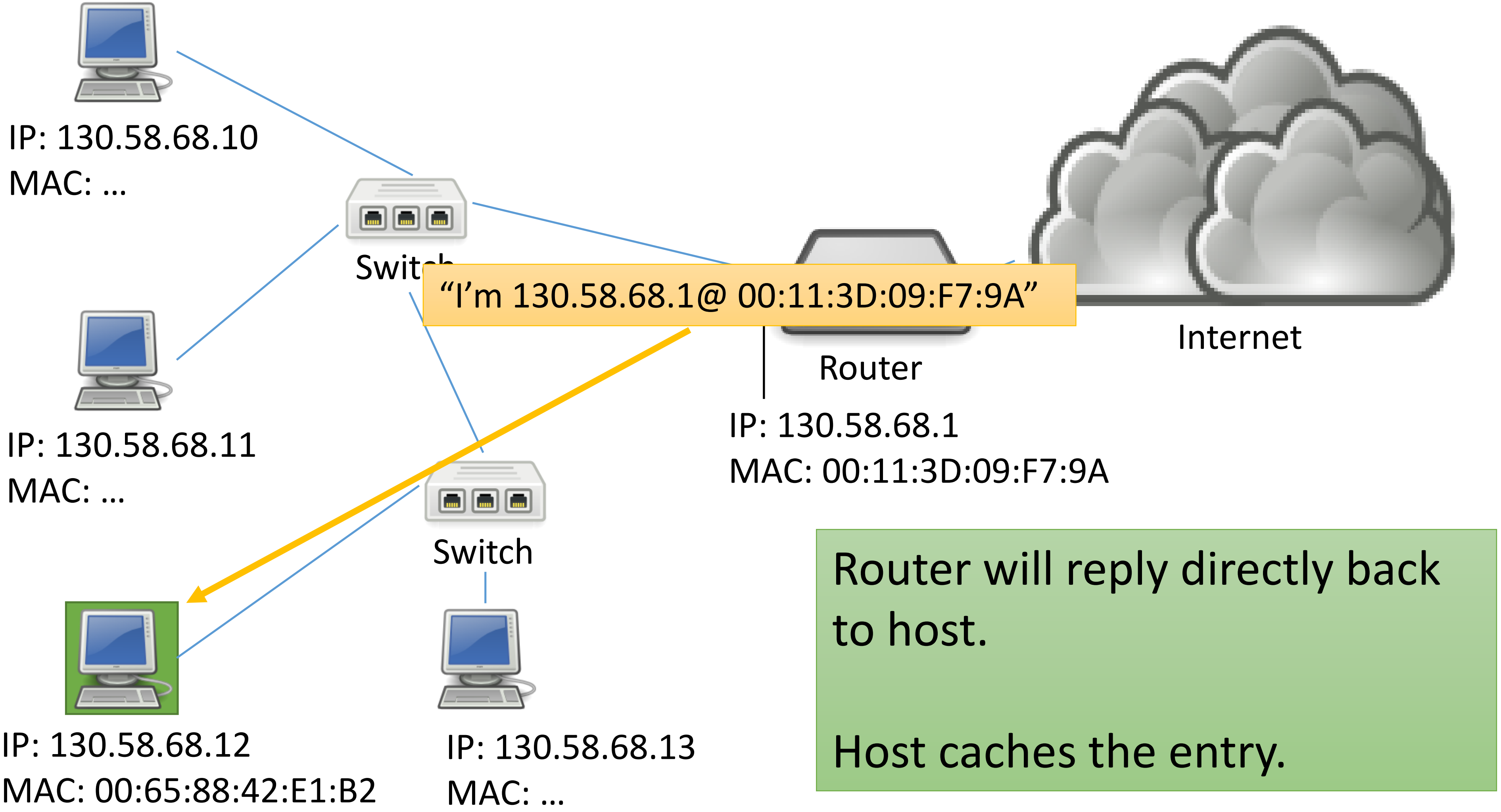
Router

IP: 130.58.68.1
MAC: 00:11:3D:09:F7:9A

Internet

Broadcast the ARP to everyone on the local network.

ARP Example



IP: 130.58.68.10
MAC: …

IP: 130.58.68.11
MAC: …

IP: 130.58.68.12
MAC: 00:65:88:42:E1:B2

IP: 130.58.68.13
MAC: …

Switch

Switch

“I’m 130.58.68.1@ 00:11:3D:09:F7:9A”

Router

IP: 130.58.68.1
MAC: 00:11:3D:09:F7:9A

Internet

Router will reply directly back to host.

Host caches the entry.

Link Layer Functions

1. Addressing: identifying endpoints

2. Framing: Dividing data into pieces that are sized for the network to handle.

- Data pieces:
  - Transport: Segments
  - Network: Datagrams (or packets)
  - Link: Frames
  - Physical: Bits

Link Layer Functions

1. Addressing: identifying endpoints

2. Framing: Dividing data into pieces that are sized for the network to handle.

- Data pieces:

• Transport:	Segments
• Network:	Datagrams (or packets)
• Link:	Frames
• Physical:	Bits

“Big freaking deal, Sherlock!”

Why do we put a limit on the size of a frame?

A. To keep one user from hogging the channel.

B. To make signaling message boundaries easier.

C. To achieve higher performance

D. Some other reason.

Link Layer Functions

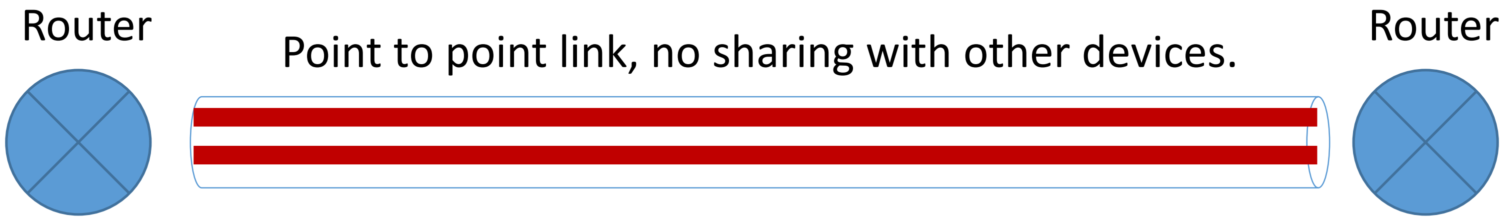
1. Addressing: identifying endpoints

2. Framing: Dividing data into pieces that are sized for the network to handle.

3. Link access: Determining how to share the medium, who gets to send, and for how long.

Link Access

- Some networks may not require much.

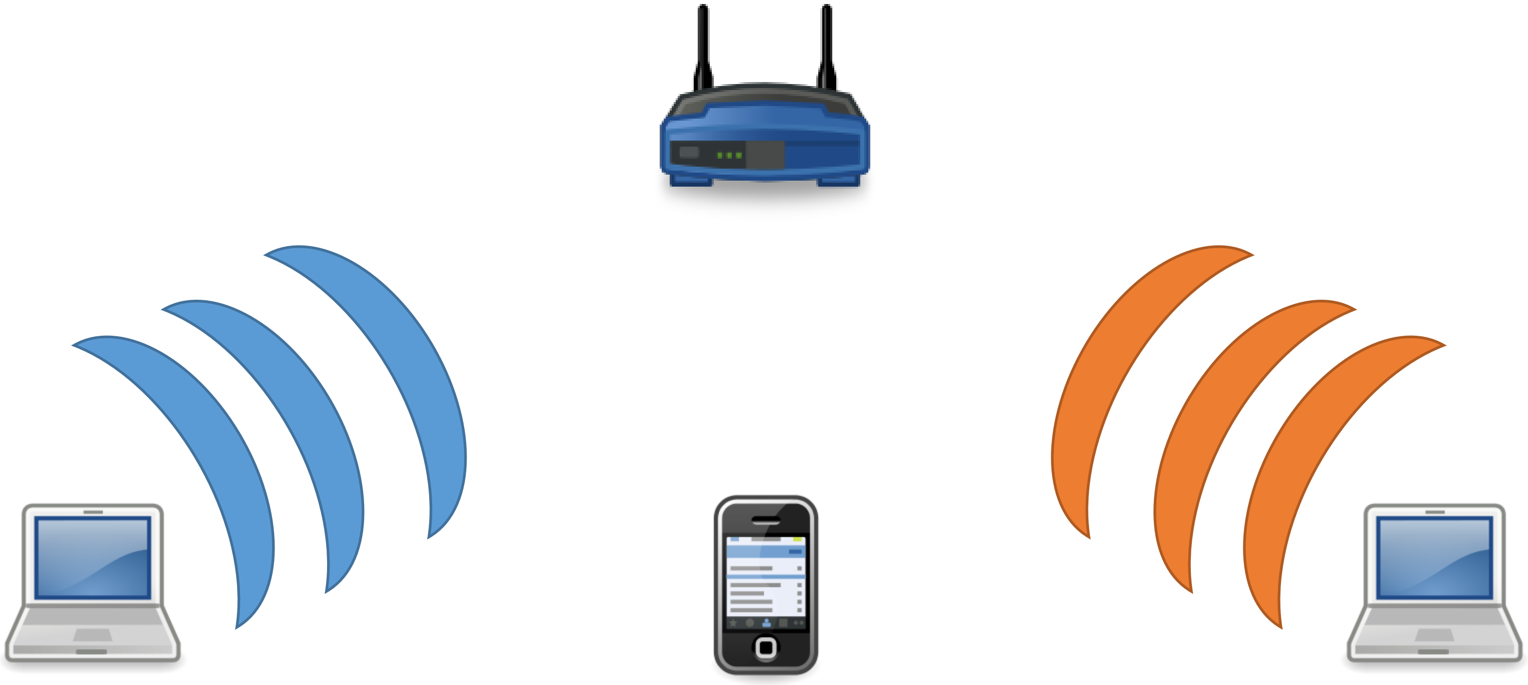


Example 1: Single copper wire, only one of them can send at a time.

Example 2: Two copper wires in cable, each can send on one simultaneously.

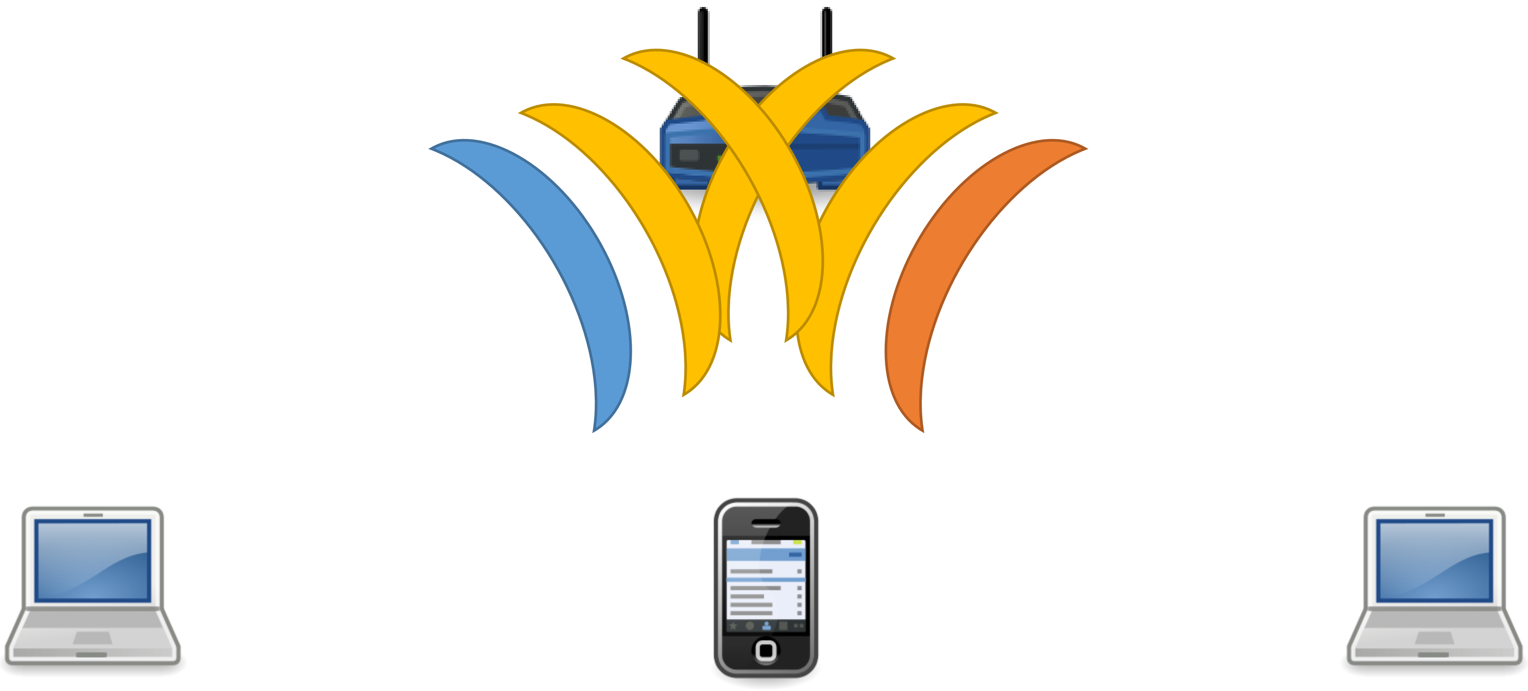
Link Access

- For other networks, this is a huge challenge.



Link Access

- For other networks, this is a huge challenge.



Collision!

How should we handle collisions in general (for WiFi and other link media)?

A. Enforce at the end hosts that only one sender transmit at a time.

B. Enforce in the network that only one sender transmit at a time.

C. Detect collisions and retransmit later.

D. Something else.

Link Layer Functions

1. Addressing: identifying endpoints

2. Framing: Dividing data into pieces that are sized for the network to handle.

3. Link access: Determining how to share the medium, who gets to send, and for how long.

4. Error detection/correction and reliability.

Reliability in the link layer seems at odds with the E2E principle. Why would we add reliability here?

A. Legacy reasons: reliability was done at the link layer first, E2E came later.

B. It improves performance.

C. It’s necessary for correctness.

D. Some other reason.

E. It’s completely unnecessary.

Link Layer Functions

1. Addressing: identifying endpoints

2. Framing: Dividing data into pieces that are sized for the network to handle.

Not so complex…

3. Link access: Determining how to share the medium, who gets to send, and for how long.

Next time

4. Error detection/correction and reliability.

Recall: Internet Checksum

Goal: detect “errors” (e.g., flipped bits) in transmitted packet (note: used at transport layer only)

Sender:

- treat segment contents as sequence of 16-bit integers
- checksum: 1’s complement sum of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

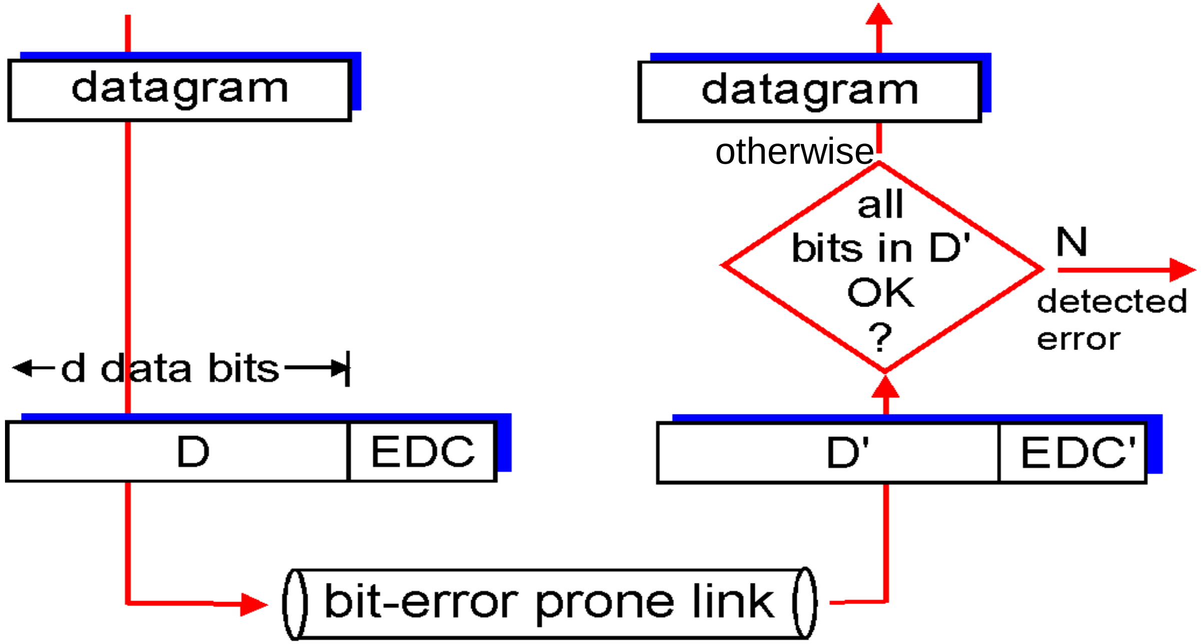
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?*

Error Detection

EDC= Error Detection and Correction bits (redundancy)

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



Simple Parity - Sender

- Suppose you want to send the message:
  - 001011011011000110010
- For every *d* bits (e.g., *d* = 7), add a parity bit:
  - 1 if the number of one's is odd
  - 0 if the number of one's is even

Message chunk	Parity bit
0010110	1
1101100	0
0110010	1

- 001011011101100001100101

Simple Parity - Sender

- Suppose you want to send the message:
  - 0010110   1101100   0110010
- For every *d* bits (e.g., *d* = 7), add a parity bit:
  - 1 if the number of one’s is odd
  - 0 if the number of one’s is even

Message chunk	Parity bit
0010110	1
1101100	0
0110010	1

- $0010110\underline{1}1101100\underline{0}0110010\underline{1}$

Simple Parity - Receiver

- For each block of size *d*:
  - Count the number of 1’s and compare with following parity bit.

- If an odd number of bits get flipped, we’ll detect it (can’t do much to correct it).

- Cost: One extra bit for every *d*
  - In this example, 21 -> 24 bits.

Two-Dimensional Parity

- Suppose you want to send the same message:
  - 001011011011000110010
- Add an extra parity byte, compute parity on “columns” too.
- Can detect 1, 2, 3-bit (and some 4-bit) errors

	Message chunk	Parity bit
	0010110	1
	1101100	0
	0110010	1
Parity byte:	1001000	0

Forward Error Correction

- With two-dimensional parity, we can even *correct* single-bit errors.

								Parity bits
	0	0	1	0	1	1	0	1
	1	0	1	0	0	0	1	0
	1	0	0	1	0	1	1	0
	1	1	1	0	1	1	0	1
Parity byte	1	1	1	1	1	1	0	0

Exactly one bit has been flipped. Which is it?

In practice…

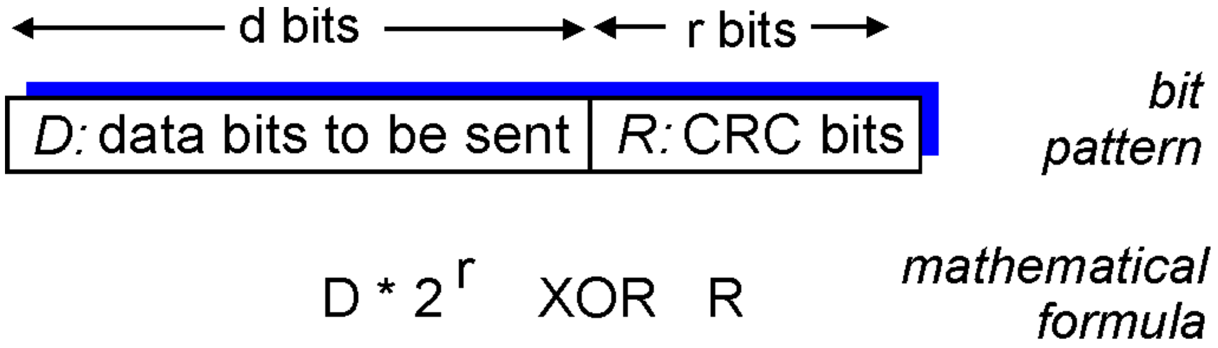
- Bit errors might occur in bursts.

- We’re willing to trade computational complexity for space efficiency.
    - Make the detection routine more complex, to detect error bursts, without tons of extra data

- Insight: We need hardware to interface with the network, do the computation there!

Cyclic redundancy check

- 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)



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

Summary

- The link layer provides lots of functionality:
    - addressing, framing, media access, error checking
    - *could* be used independently of IP!
    - typically only small scale

- Many different technologies out there.
    - copper wires, optics, wireless, satellite
    - differing challenges for each