

Link Layer

CMPS 4750/6750: Computer Networks

Outline

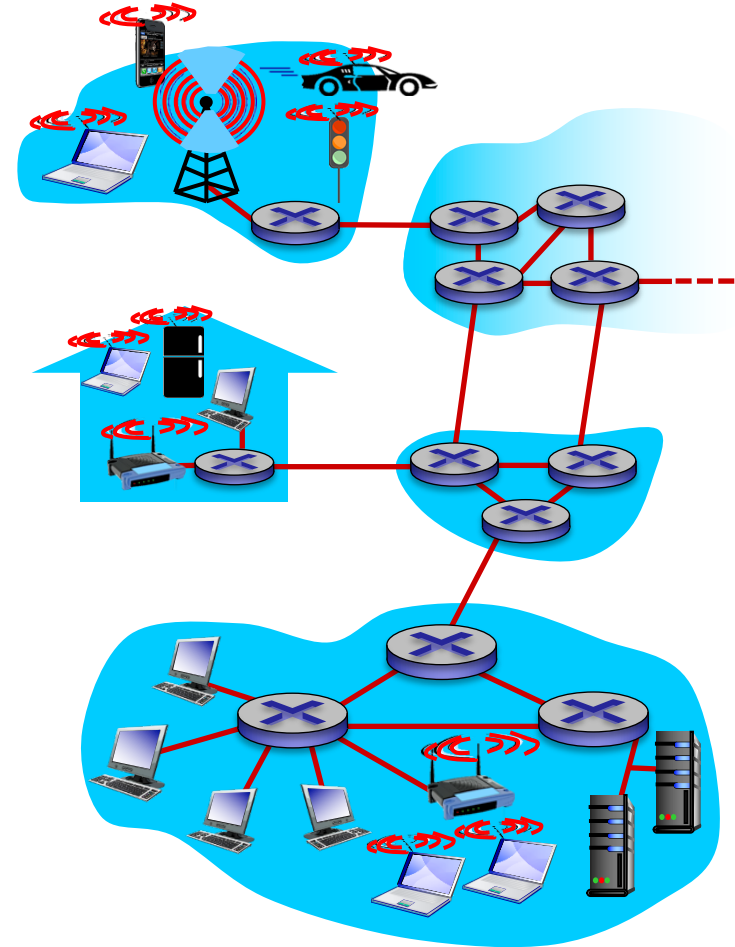
- **overview**
- error detection & correction
- multiple access
- local area networks
- a day in the life of a web request

Link layer: introduction

terminology:

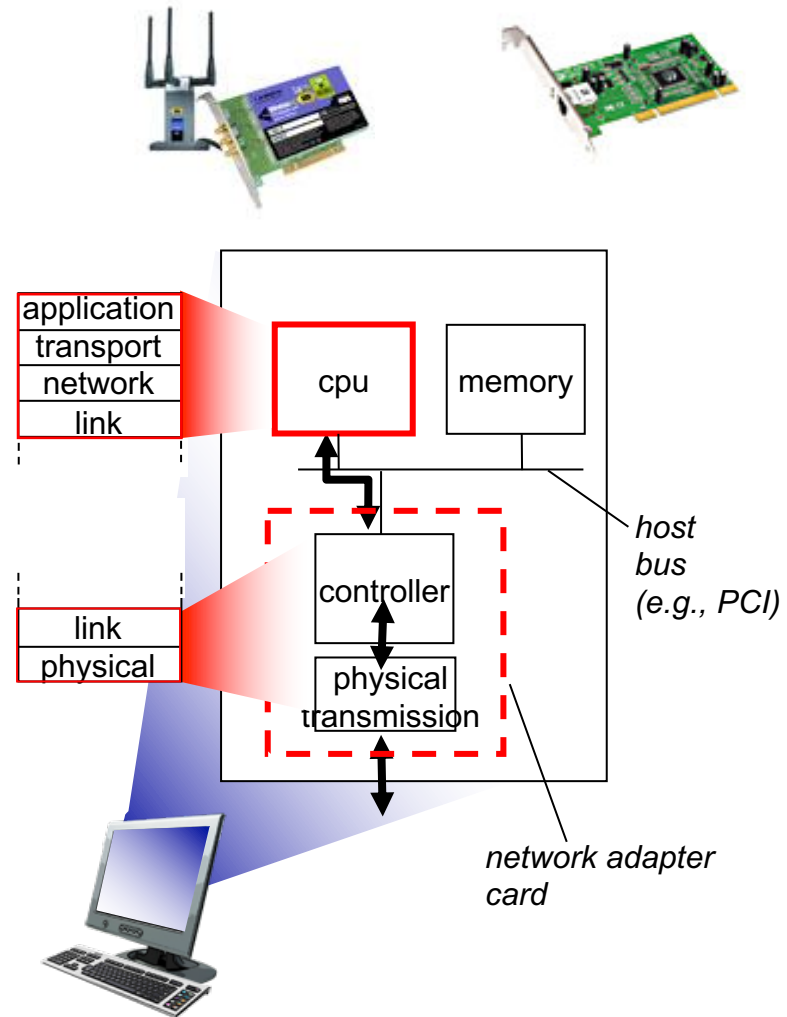
- hosts, switches, and routers: **nodes**
- communication channels that connect adjacent nodes along communication path: **links**
 - wired links
 - wireless links
- layer-2 packet: **frame**, encapsulates datagram

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

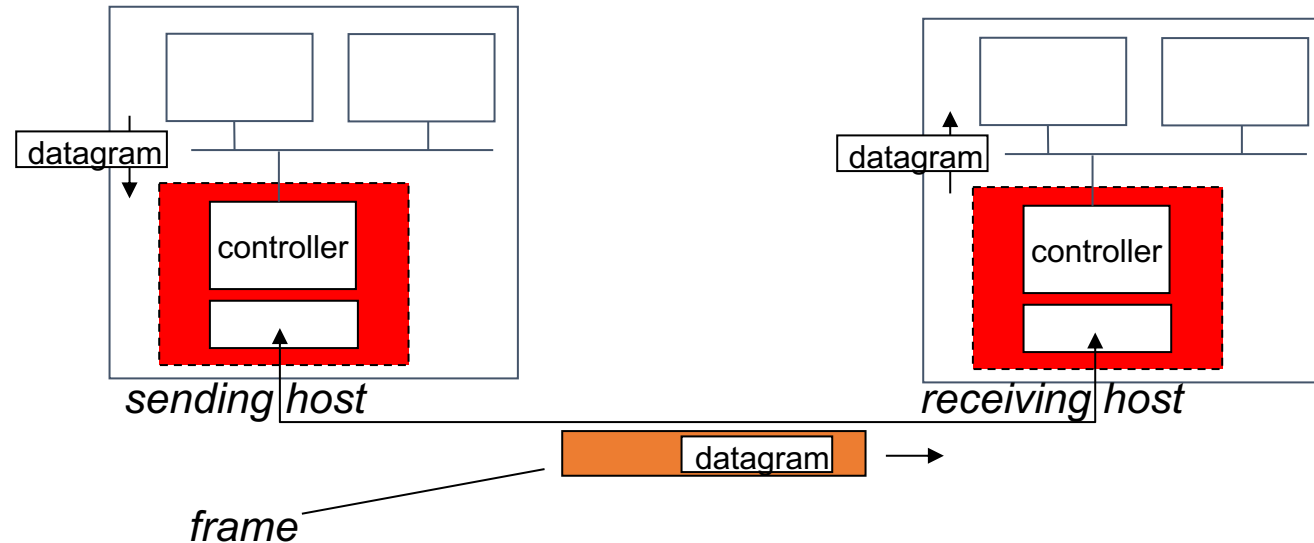


Where is the link layer implemented?

- in each and every host
- link layer implemented in “adaptor” (aka *network interface card* NIC) or on a chip
 - Ethernet card, 802.11 card; Ethernet chipset
 - implements link, physical layer
- attaches into host’s system buses
- combination of hardware, software, firmware



Adaptors communicating



▪ 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

Link layer services

▪ *framing*

- encapsulate datagram into frame, adding header, trailer
- “MAC” addresses used in frame headers to identify source, destination
 - different from IP address!

▪ *link access*

- channel access if shared medium

▪ *reliable delivery between adjacent nodes*

- we learned how to do this already (chapter 3)!
- seldom used on low bit-error link (fiber, some twisted pair)
- wireless links: high error rates
 - *Q*: why both link-level and end-end reliability?

Link layer services (more)

■ *error detection:*

- errors caused by signal attenuation, noise.
- receiver detects presence of errors:
 - signals sender for retransmission or drops frame

■ *error correction:*

- receiver identifies *and corrects* bit error(s) without resorting to retransmission

■ *half-duplex and full-duplex*

- with half duplex, nodes at both ends of link can transmit, but not at same time

Outline

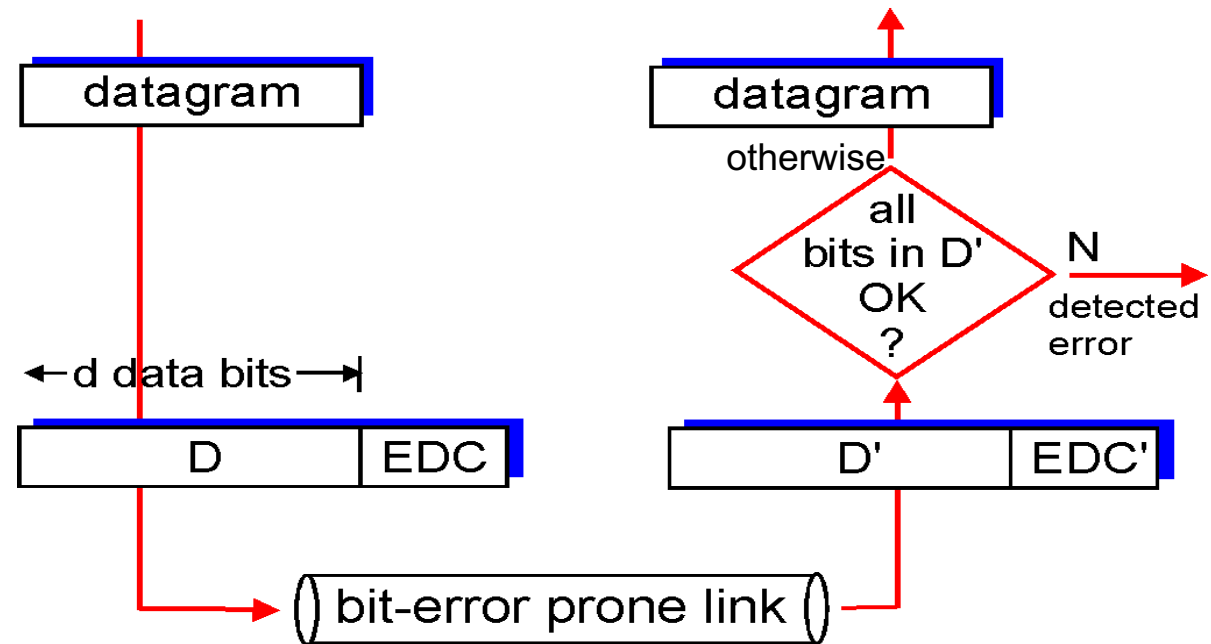
- overview
- error detection & correction
- multiple access
- local area networks
- a day in the life of a web request

Error detection

D = Data protected by error checking, may include header fields

EDC = Error Detection and Correction bits (redundancy)

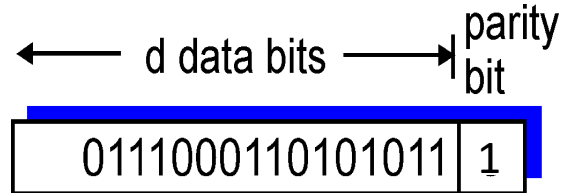
- Error detection not 100% reliable!
 - protocol may miss some errors, but rarely
 - larger EDC field yields better detection and correction



Parity checking

single bit parity:

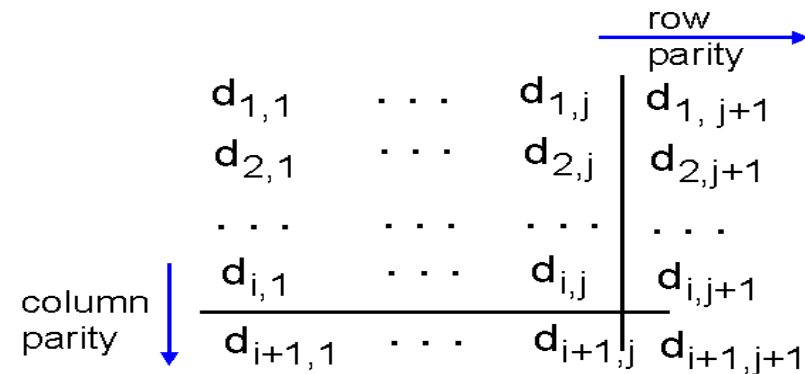
- detect single bit errors



One-bit **even** parity

two-dimensional bit parity:

- detect and correct single bit errors



1	0	1	0	1	1
1	1	1	1	0	0
0	1	1	1	0	1
0	0	1	0	1	0

no errors

1	0	1	0	1	1
1	0	1	1	0	0
0	1	1	1	0	1
0	0	1	0	1	0

parity error

*correctable
single bit error*

Internet checksum (review)

- Used by both TCP, UDP, and IPv4 (header)

sender:

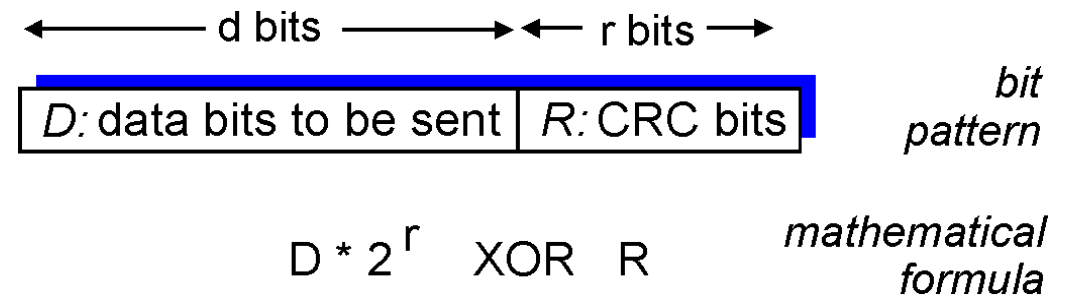
- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into packet checksum field

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*

Cyclic redundancy check (CRC)

- more powerful error-detection coding
 - widely used in practice: Ethernet, WiFi
- view data bits, **D**, as a binary number
- choose $r+1$ bit pattern (generator), **G**, known to sender and receiver
- **sender**: choose r CRC bits, **R**, such that $\langle D, R \rangle$ exactly divisible by G (modulo 2)
- **receiver**: divides $\langle D, R \rangle$ by G . If non-zero remainder: error detected!
- can detect all burst errors less than $r+1$ bits



Cyclic redundancy check (CRC)

want:

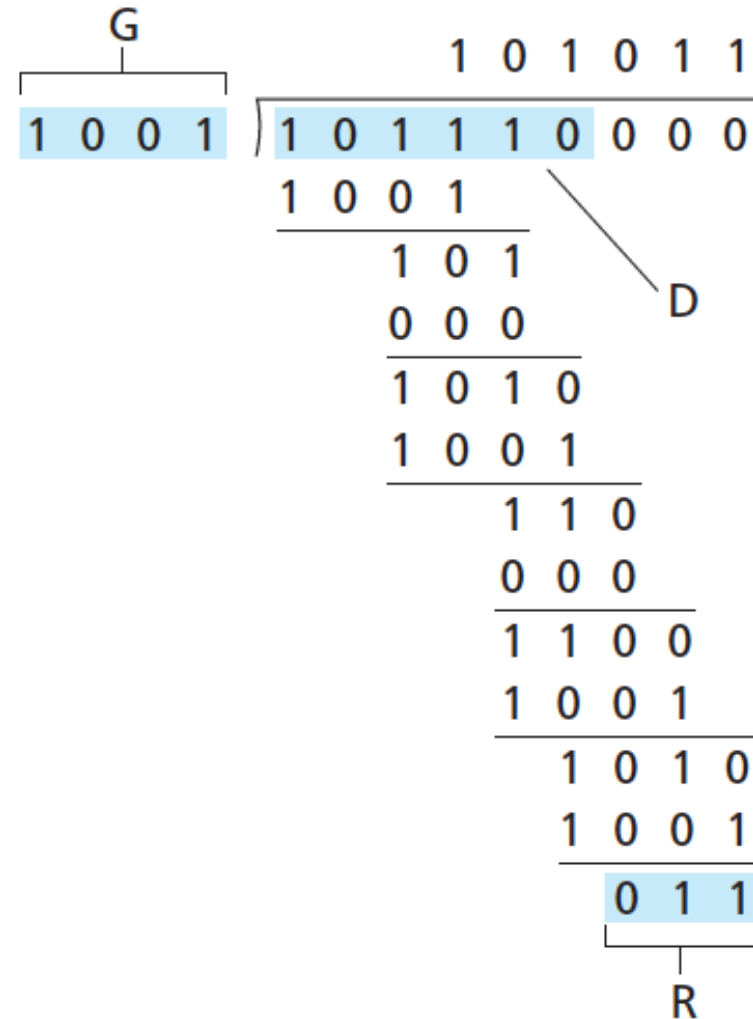
$$D \cdot 2^r \text{ XOR } R = nG$$

equivalently:

$$D \cdot 2^r = nG \text{ XOR } R$$

equivalently:

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



Outline

- overview
- error detection & correction
- **multiple access**
- local area networks
- a day in the life of a web request

Multiple access links, protocols

two types of “links”:

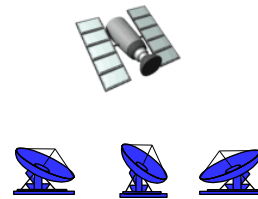
- point-to-point
 - point-to-point link for dial-up access
 - point-to-point link between Ethernet switch, host
- *broadcast (shared wire or wireless medium)*



shared wire (e.g.,
cabled Ethernet)



shared RF
(e.g., 802.11 WiFi)



shared RF
(satellite)



humans at a
cocktail party
(shared air, acoustical)

Multiple access protocols

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: interference
 - *collision* if node receives two or more signals at the same time

multiple access protocol (MAC)

- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
 - no out-of-band channel for coordination

An ideal multiple access protocol

given: broadcast channel of rate R bps

desiderata:

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

MAC protocols: taxonomy

three broad classes:

- *channel partitioning*

- divide channel into smaller “pieces” (time slots, frequency, code)
- allocate piece to node for exclusive use

- *random access*

- channel not divided, allow collisions
- “recover” from collisions

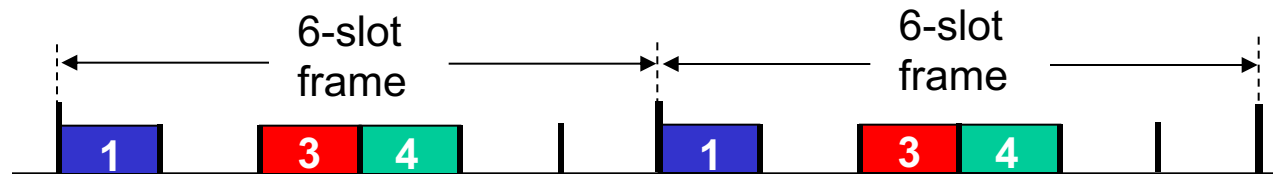
- *“taking turns”*

- nodes take turns, but nodes with more to send can take longer turns

Channel partitioning MAC protocols: TDMA

TDMA: time division multiple access

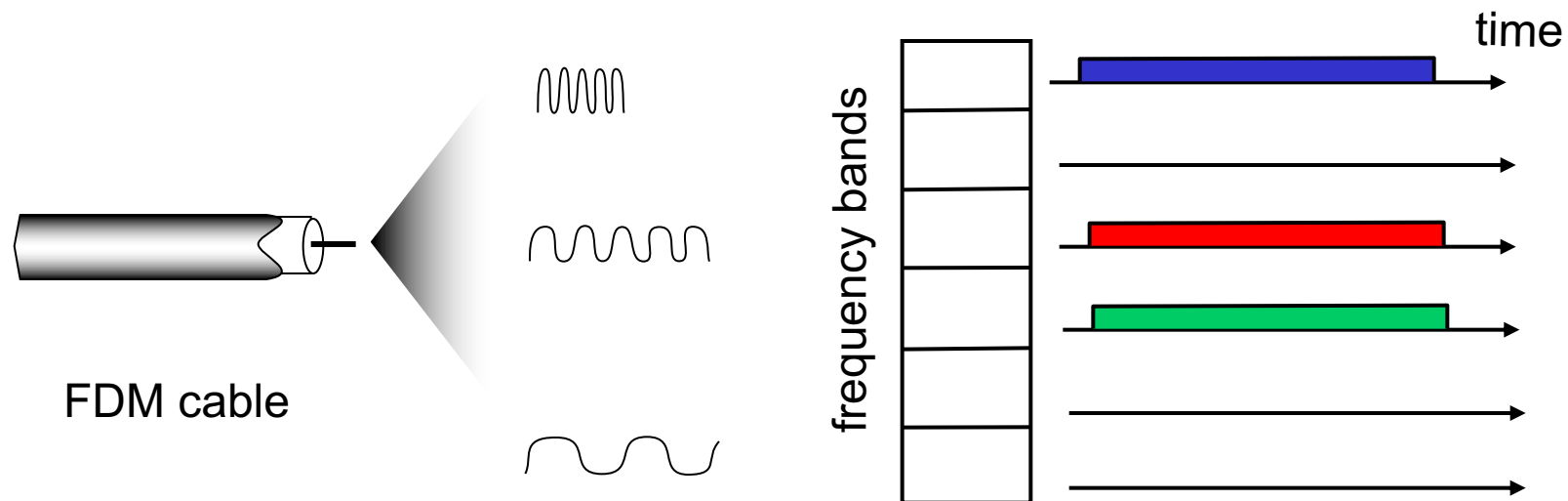
- access to channel in "rounds"
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



Channel partitioning MAC protocols: FDMA

FDMA: frequency division multiple access

- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



Random access protocols

- when node has packet to send
 - transmit at full channel data rate R
 - no *a priori* coordination among nodes
- two or more transmitting nodes → “collision”
- **random access MAC protocol** specifies:
 - how to detect collisions
 - how to recover from collisions
- examples:
 - slotted ALOHA, ALOHA
 - CSMA, CSMA/CD, CSMA/CA

Slotted ALOHA

- Part of ALOHAnet, a pioneering computer networking system developed at the University of Hawaii in 1970s

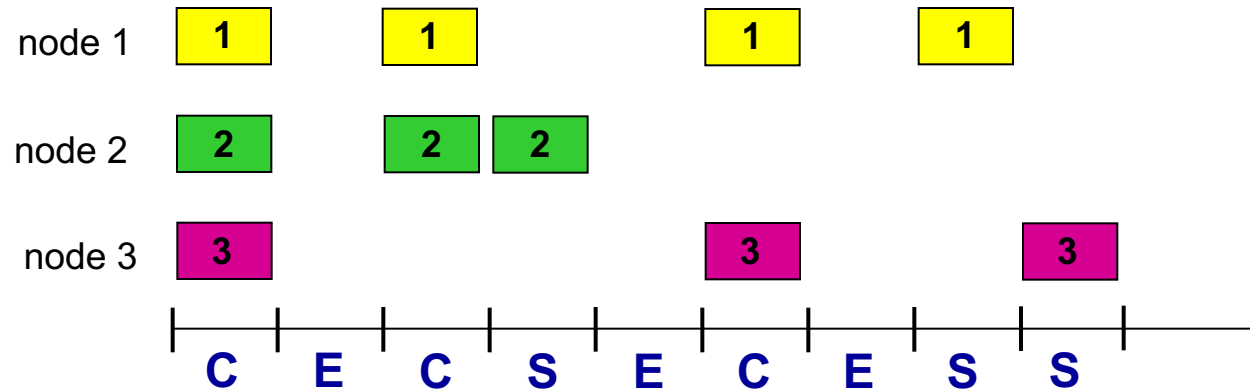
assumptions:

- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

operation:

- when node obtains fresh frame, transmits in next slot
 - *if no collision:* node can send new frame in next slot
 - *if collision:* node retransmits frame in each subsequent slot with prob. p until success

Slotted ALOHA



Pros:

- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

Cons:

- collisions, wasting slots
- idle slots
- nodes may be able to detect collision in less than time to transmit packet
- clock synchronization

Slotted ALOHA: efficiency

efficiency: long-run fraction of successful slots (assuming: many nodes, all with many frames to send)

- *suppose*: N nodes with many frames to send, each transmits in slot with probability p
- prob that given node has success in a slot = $p(1 - p)^{N-1}$
- prob that *any* node has a success = $Np(1 - p)^{N-1}$

- max efficiency: find p^* that maximizes $Np(1 - p)^{N-1}$

$$\Rightarrow p^* = \frac{1}{N}$$

- for many nodes, take limit of $Np^*(1 - p^*)^{N-1}$ as N goes to infinity, gives:

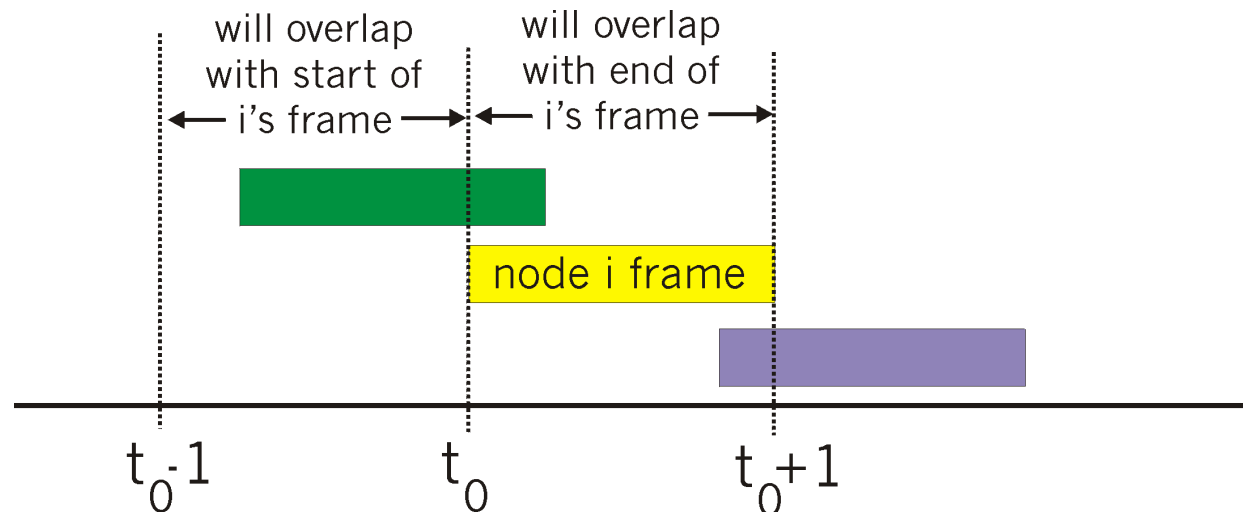
$$\text{max efficiency} = 1/e = .37$$

at best: channel used for useful transmissions 37% of time!



Pure (unslotted) ALOHA

- unslotted Aloha: simpler, no synchronization
- when frame first arrives
 - transmit immediately
- collision probability increases:
 - frame sent at t_0 collides with other frames sent in $[t_0-1, t_0+1]$



Pure ALOHA efficiency

$P(\text{success by given node}) = P(\text{node transmits}) \cdot$

$P(\text{no other node transmits in } [t_0-1, t_0]) \cdot$

$P(\text{no other node transmits in } [t_0, t_0+1])$

$$= p \cdot (1 - p)^{N-1} \cdot (1 - p)^{N-1}$$

$$= p \cdot (1 - p)^{2(N-1)}$$

... choosing optimum p and then letting $N \rightarrow \infty$

$$= 1/(2e) = .18$$

even worse than slotted Aloha!

CSMA (carrier sense multiple access)

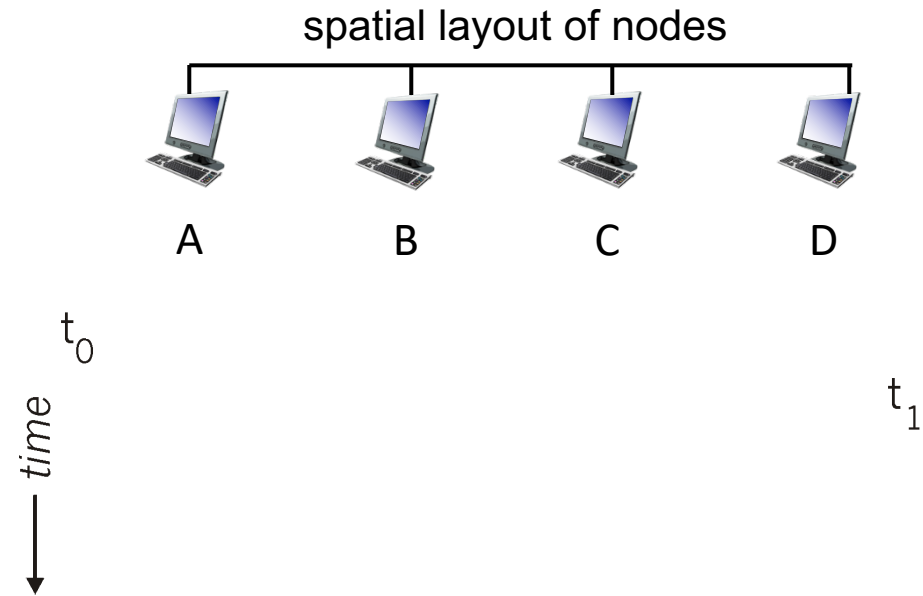
CSMA: listen before transmit:

- if channel sensed idle: transmit entire frame
- if channel sensed busy, defer transmission

- human analogy: don't interrupt others!

CSMA collisions

- collisions *can still occur*:
propagation delay means two nodes may not hear each other's transmission
- collision: entire packet transmission time wasted
 - distance & propagation delay play role in determining collision probability



CSMA/CD (collision detection)

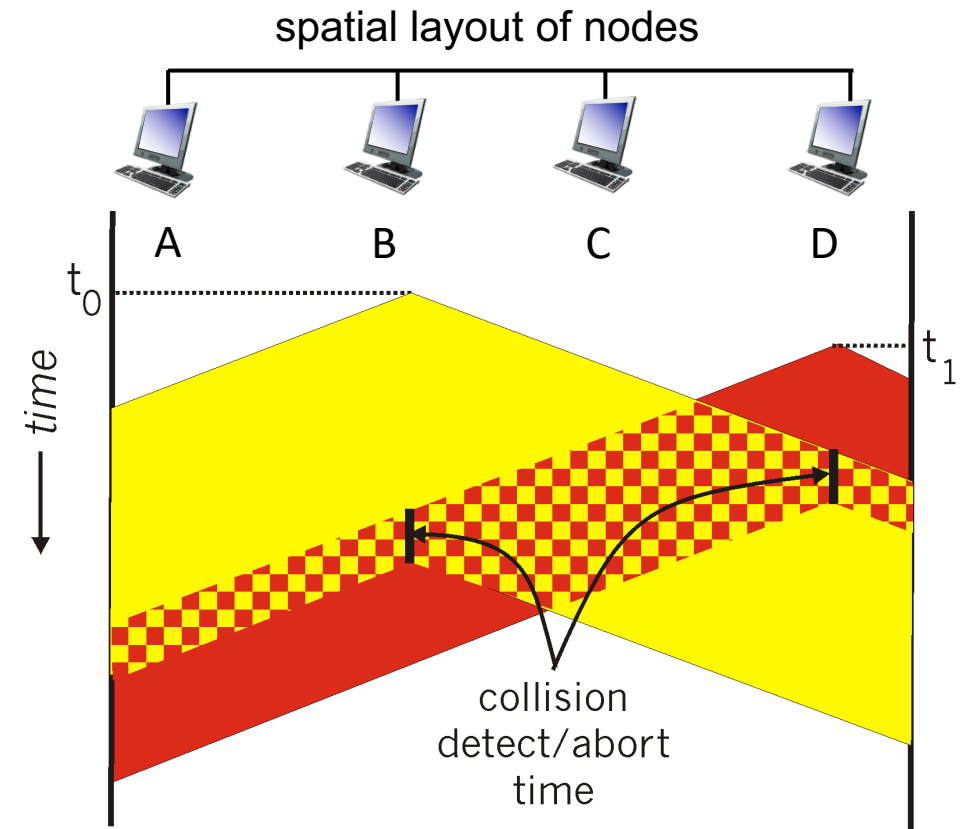
■ CSMA/CD

- carrier sensing, deferral as in CSMA
- collisions *detected* within short time
- colliding transmissions aborted, reducing channel wastage

■ collision detection:

- easy in wired LANs: measure signal strengths, compare transmitted, received signals
- difficult in wireless LANs: received signal strength overwhelmed by local transmission strength

■ human analogy: the polite conversationalist



Ethernet CSMA/CD algorithm

1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel idle, starts frame transmission. If NIC senses channel busy, waits until channel idle, then transmits.
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame !
4. If NIC detects another transmission while transmitting, aborts
5. After aborting, NIC enters *binary (exponential) backoff*:
 - after m th collision, NIC chooses K at random from $\{0, 1, 2, \dots, 2^m - 1\}$. NIC waits $K \cdot 512$ bit times, returns to Step 2
 - longer backoff interval with more collisions

CSMA/CD efficiency

- t_{prop} = max prop delay between 2 nodes in LAN
- t_{trans} = time to transmit max-size frame

$$efficiency = \frac{1}{1 + 5t_{prop}/t_{trans}}$$

- efficiency goes to 1
 - as t_{prop} goes to 0
 - as t_{trans} goes to infinity
- better performance than ALOHA: and simple, cheap, decentralized!

“Taking turns” MAC protocols

channel partitioning MAC protocols:

- share channel *efficiently* and *fairly* at high load
- inefficient at low load: delay in channel access, $1/N$ bandwidth allocated even if only 1 active node!

random access MAC protocols

- efficient at low load: single node can fully utilize channel
- high load: collision overhead

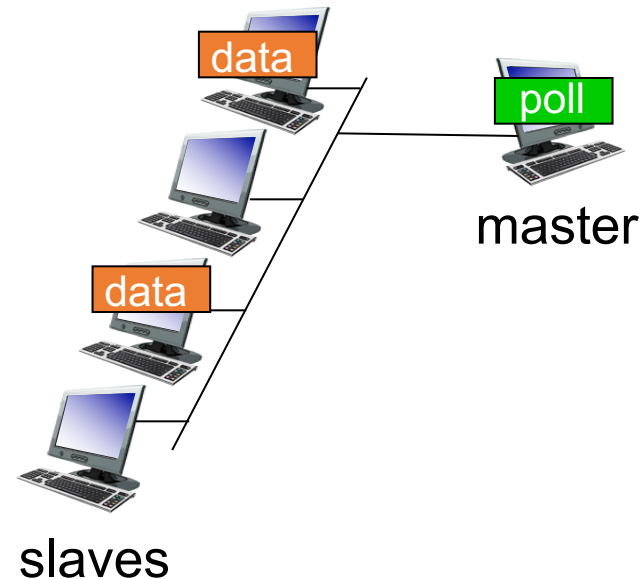
“taking turns” protocols

look for best of both worlds!

“Taking turns” MAC protocols

polling:

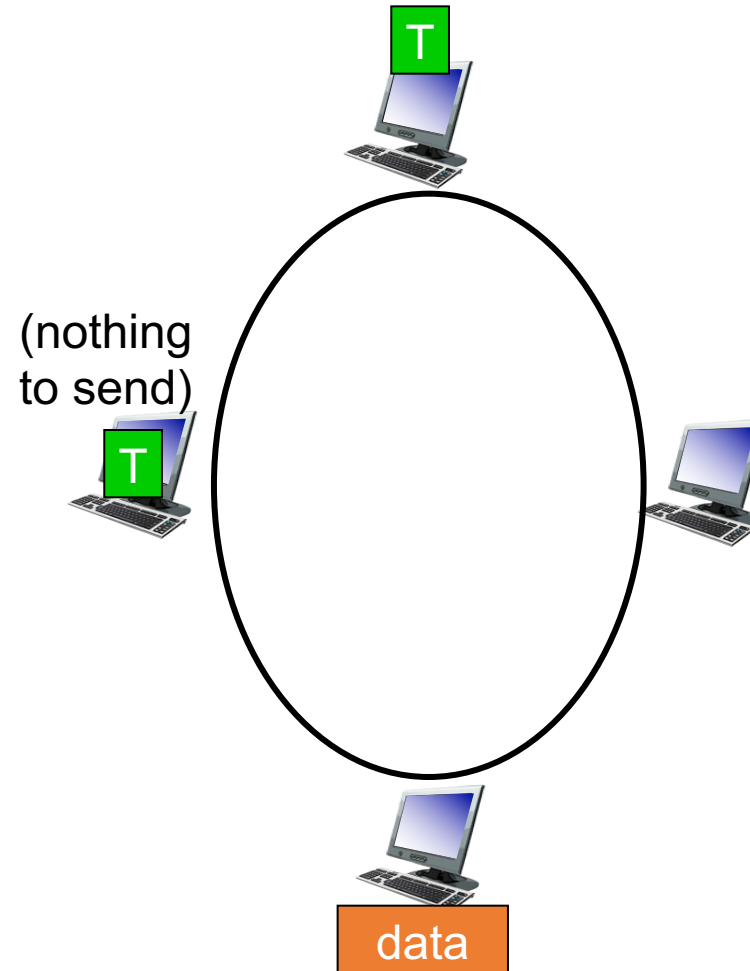
- master node “invites” slave nodes to transmit in turn
- typically used with “dumb” slave devices
- concerns:
 - polling overhead
 - latency
 - single point of failure (master)



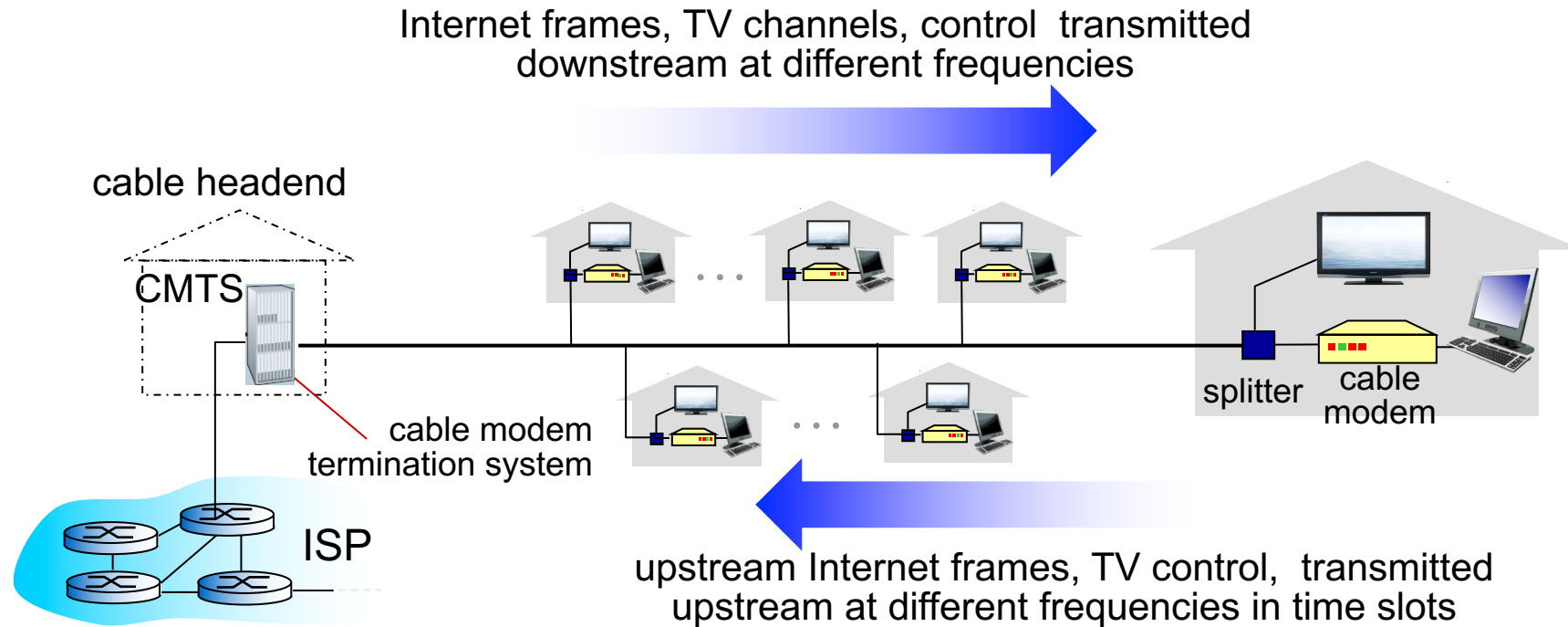
“Taking turns” MAC protocols

token passing:

- control *token* passed from one node to next sequentially.
- token message
- concerns:
 - token overhead
 - latency
 - single point of failure (token)

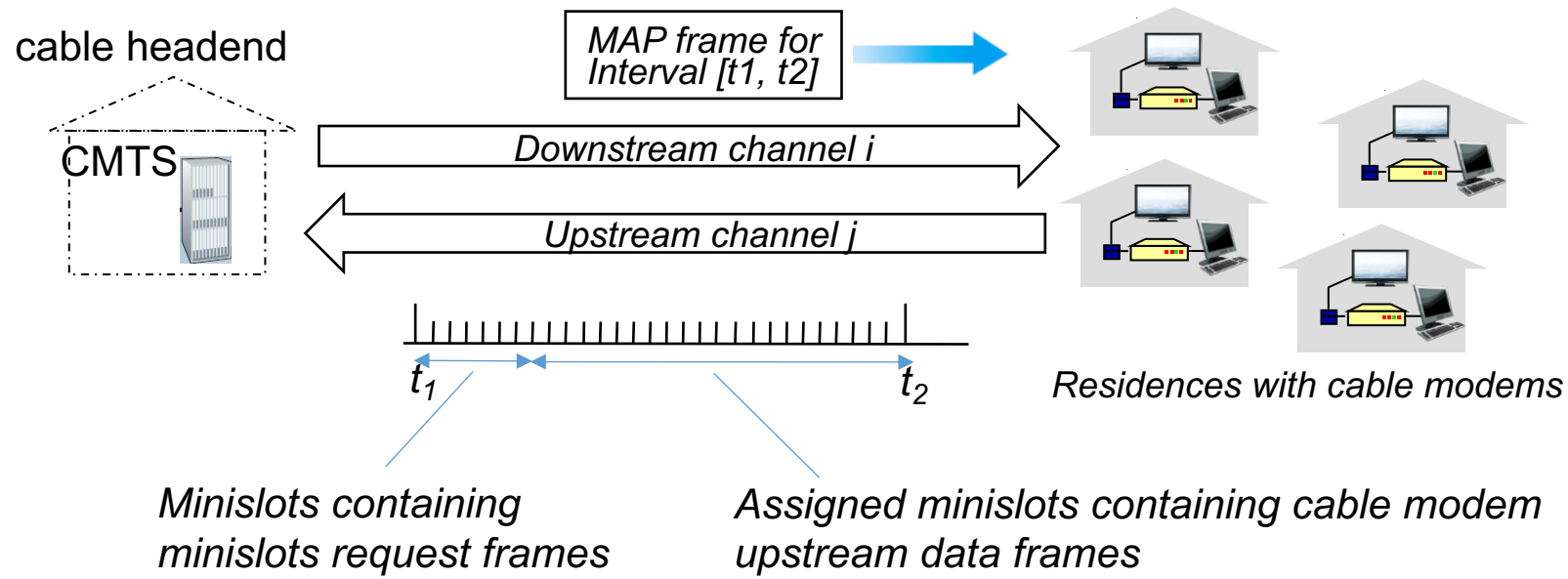


Cable access network



- **multiple** 40Mbps downstream (broadcast) channels: single CMTS transmits into channels
- **multiple** 30 Mbps upstream channels
 - **multiple access:** all users contend for certain upstream channel time slots (others assigned)

Cable access network



DOCSIS: data-over-cable service interface specifications

- FDM over upstream, downstream frequency channels
- TDM upstream: some slots assigned, some have contention
 - request for upstream slots transmitted via random access (binary backoff) in selected slots
 - downstream MAP frame: assigns upstream slots

Summary of MAC protocols

- *channel partitioning*, by time, frequency or code
 - Time Division, Frequency Division,
- *random access* (dynamic),
 - ALOHA, S-ALOHA, CSMA, CSMA/CD
 - CSMA/CD used in Ethernet
 - CSMA/CA used in 802.11 WiFi
- *taking turns*
 - polling from central site, token passing
 - Bluetooth, FDDI, token ring

Outline

- overview
- error detection & correction
- multiple access
- local area networks
 - MAC addresses
 - ARP
 - Ethernet
- a day in the life of a web request

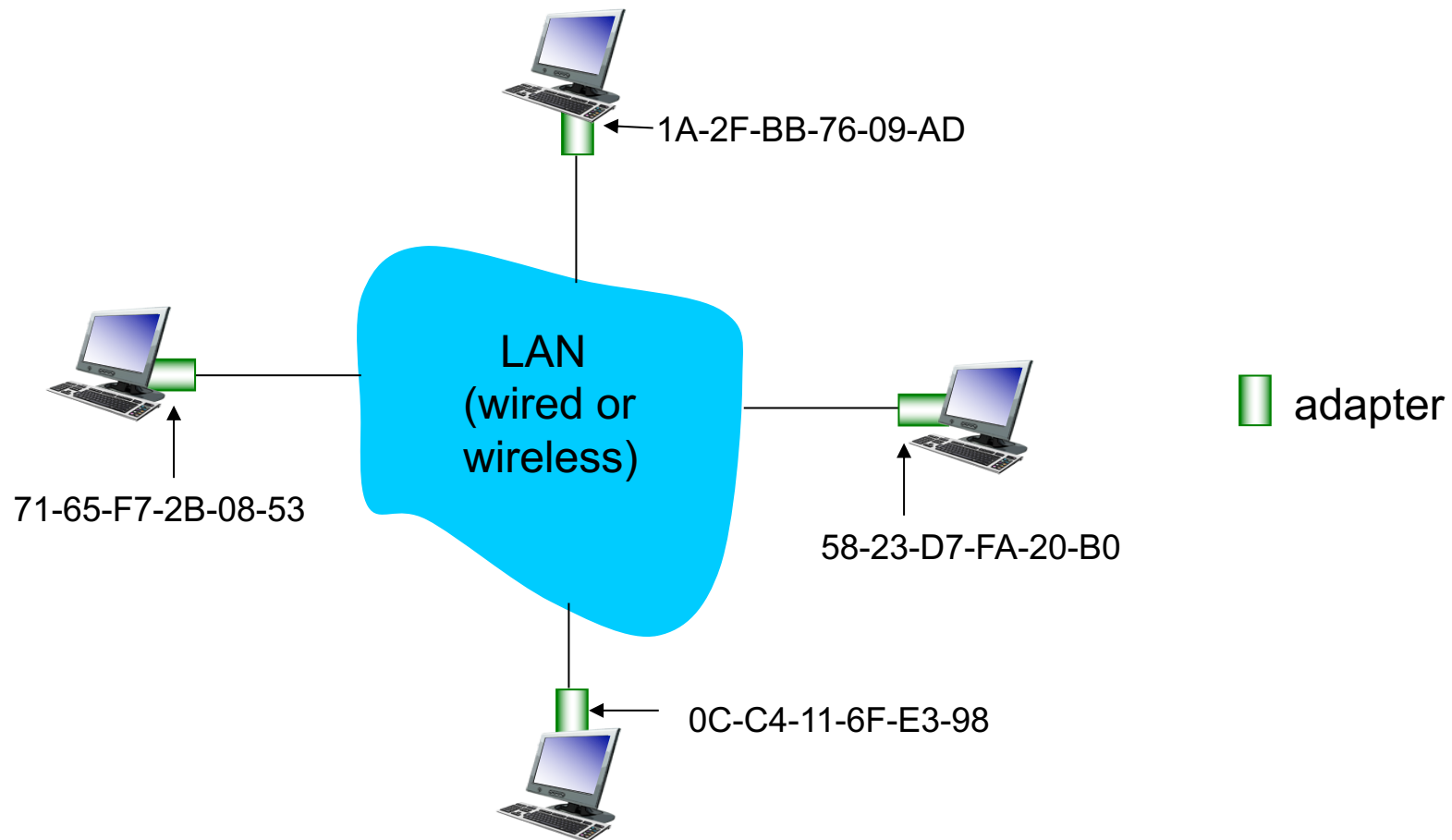
MAC addresses

- 32-bit IP address:
 - *network-layer* address for interface
 - used for layer 3 (network layer) forwarding
- MAC (or LAN or physical or Ethernet) address:
 - *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)

MAC addresses

each adapter has unique *MAC* address

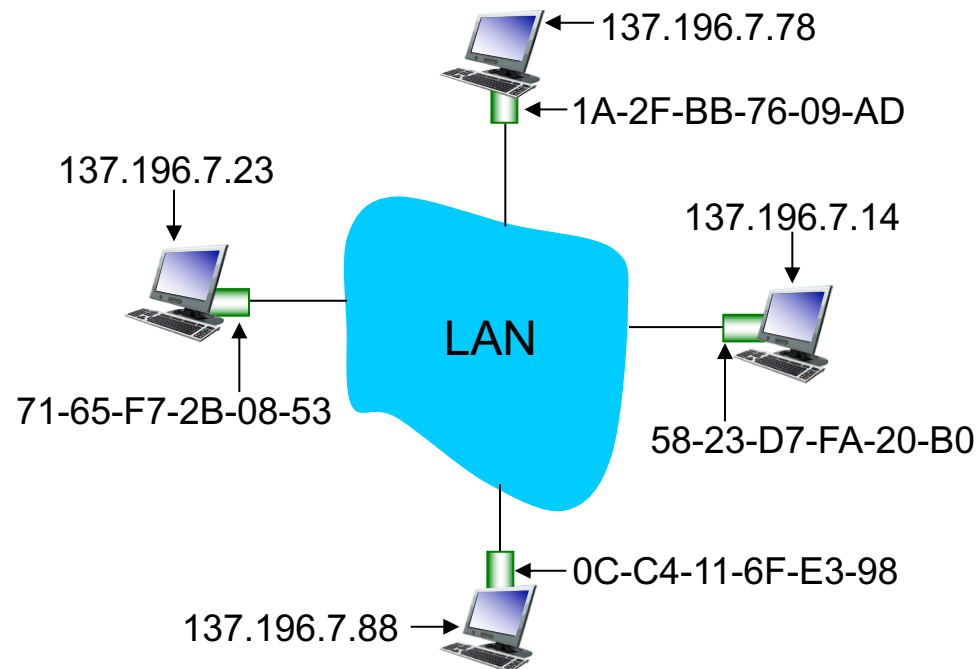


MAC addresses (more)

- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- MAC flat address → portability
 - can move LAN card from one LAN to another
- IP hierarchical address *not* portable
 - address depends on IP subnet to which node is attached
- analogy:
 - MAC address: like Social Security Number
 - IP address: like postal address

ARP: address resolution protocol

Question: how to determine interface's MAC address, knowing its IP address?



ARP table: each IP node (host, router) on LAN has table

- IP/MAC address mappings for some LAN nodes:
< IP address; MAC address; TTL >
- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

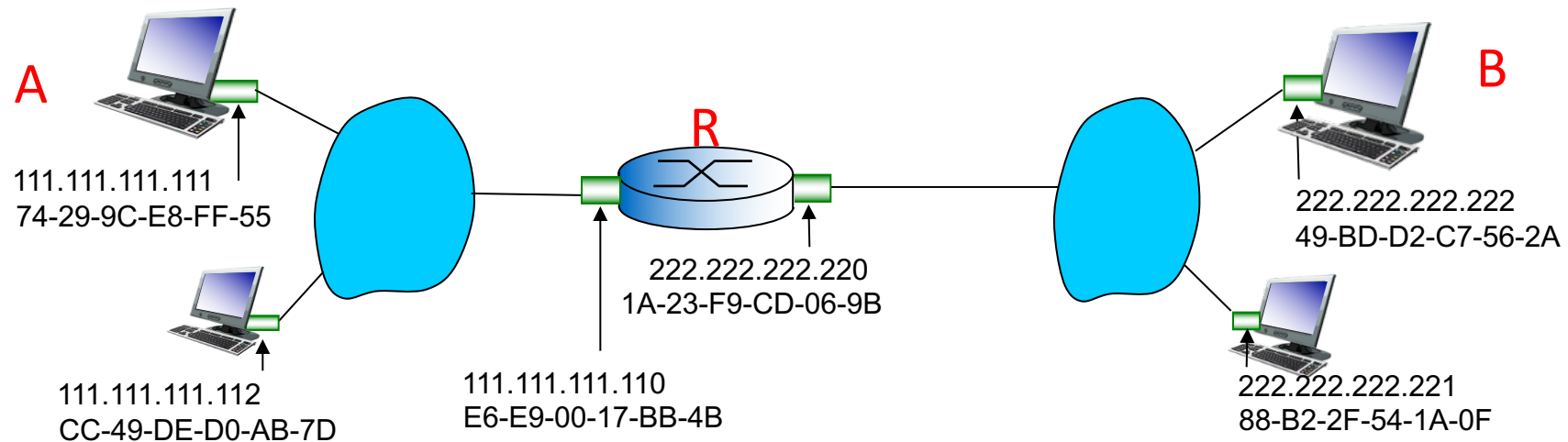
ARP protocol: same LAN

- A wants to send datagram to B
 - suppose 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)
- ARP is “plug-and-play”:
 - nodes create their ARP tables *without intervention from net administrator*

Addressing: routing to another LAN

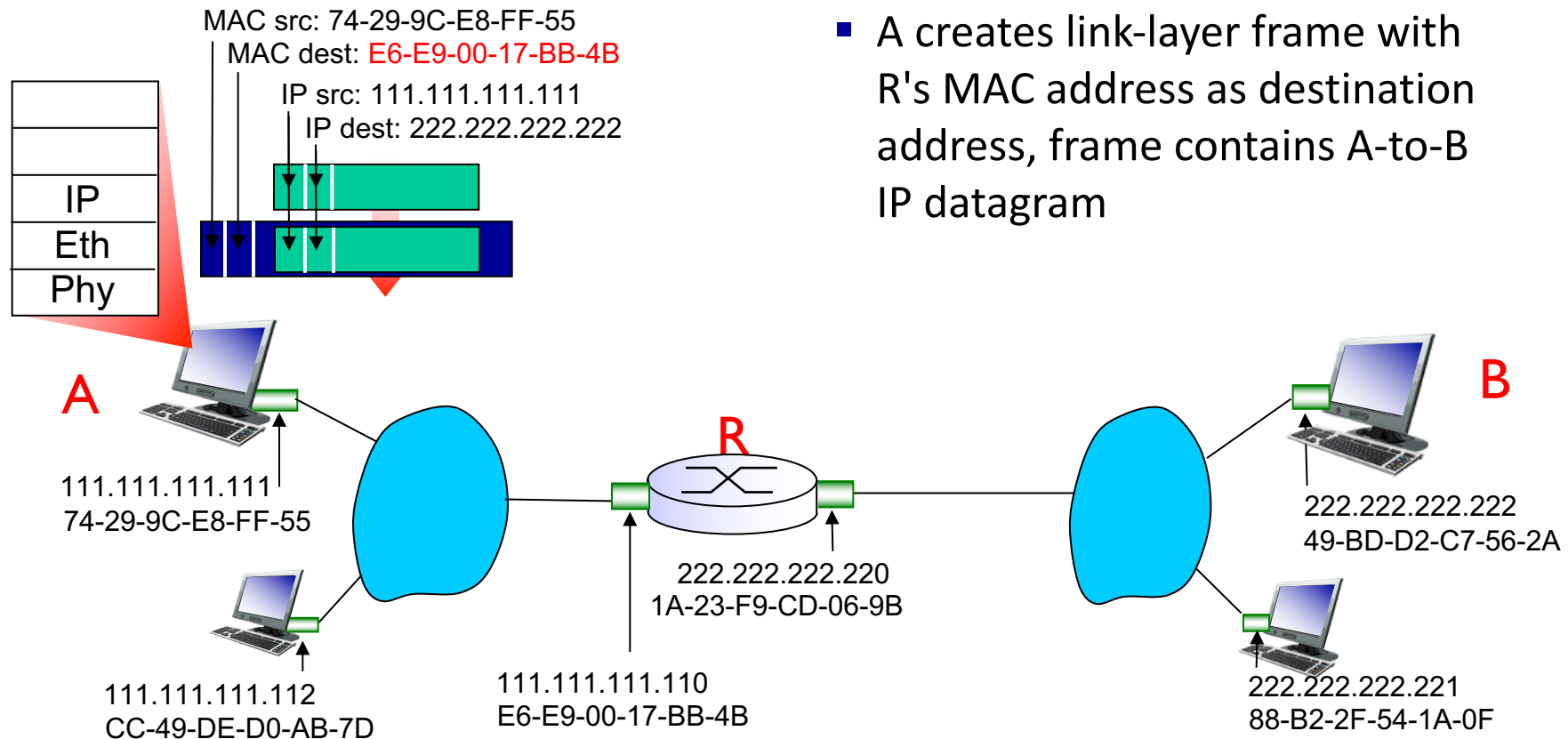
walkthrough: **send datagram from A to B via R**

- focus on addressing – at IP (datagram) and MAC layer (frame)
- assume A knows B's IP address
- assume A knows IP address of first hop router, R (how?)
- assume A knows R's MAC address (how?)



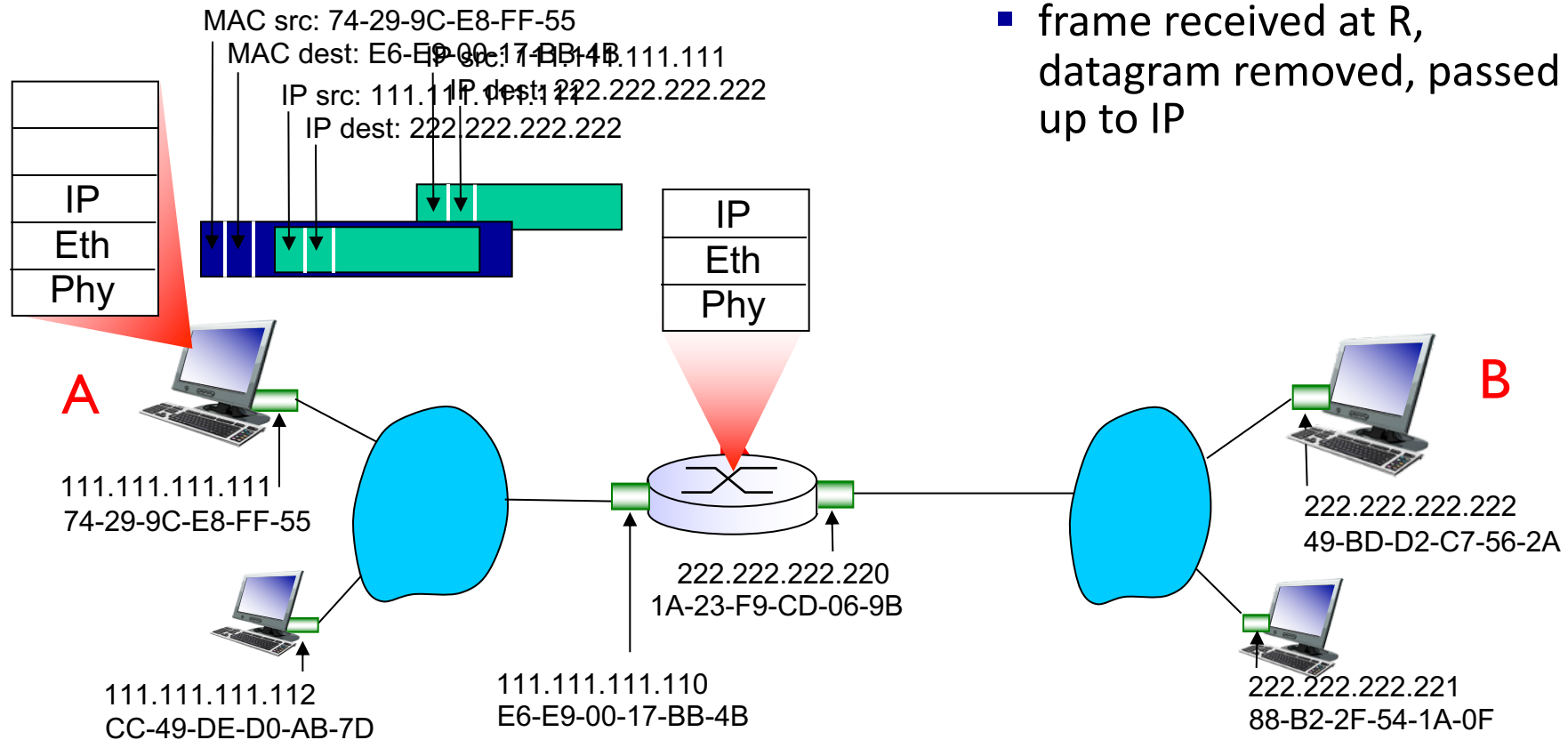
Addressing: routing to another LAN

- 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



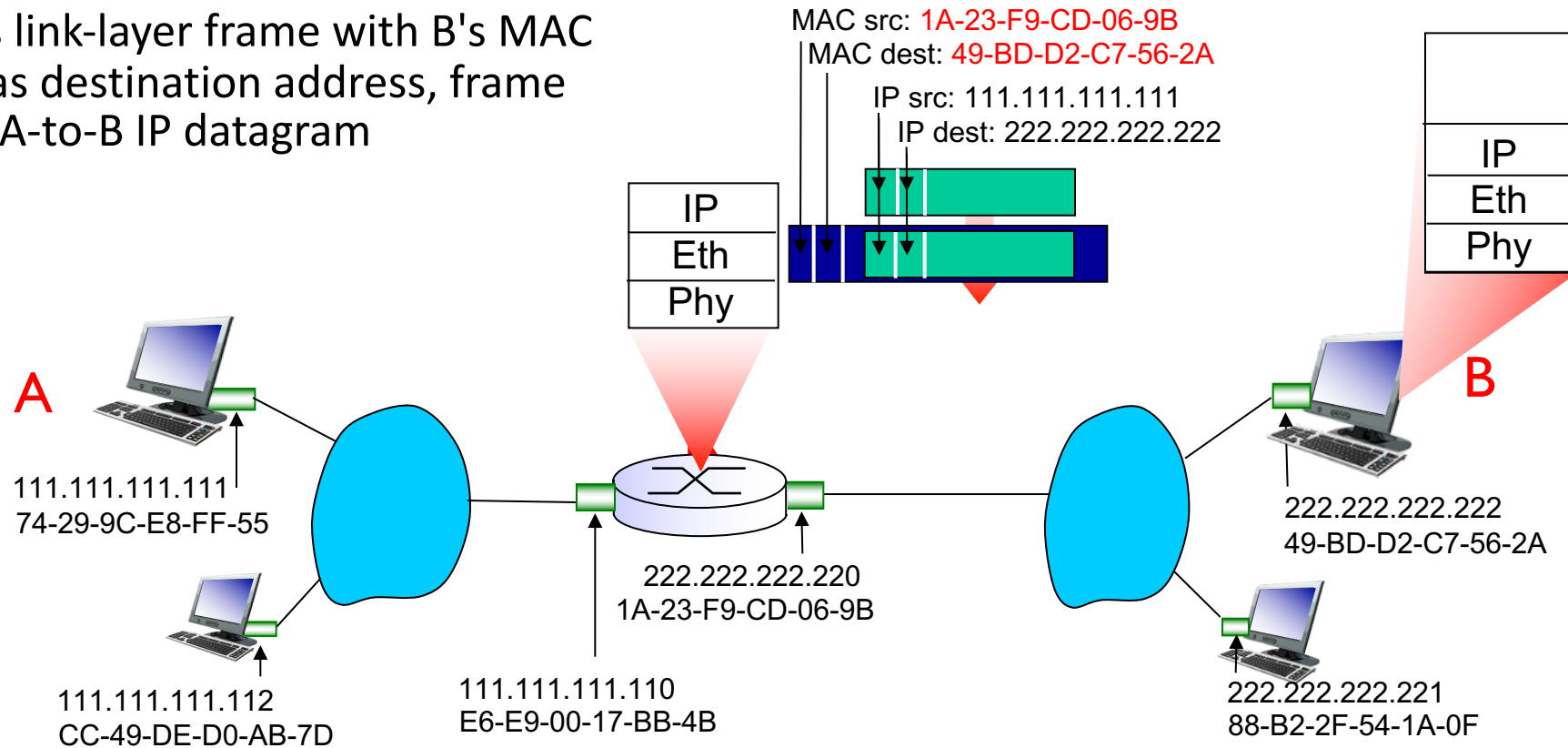
Addressing: routing to another LAN

- frame sent from A to R
- frame received at R, datagram removed, passed up to IP



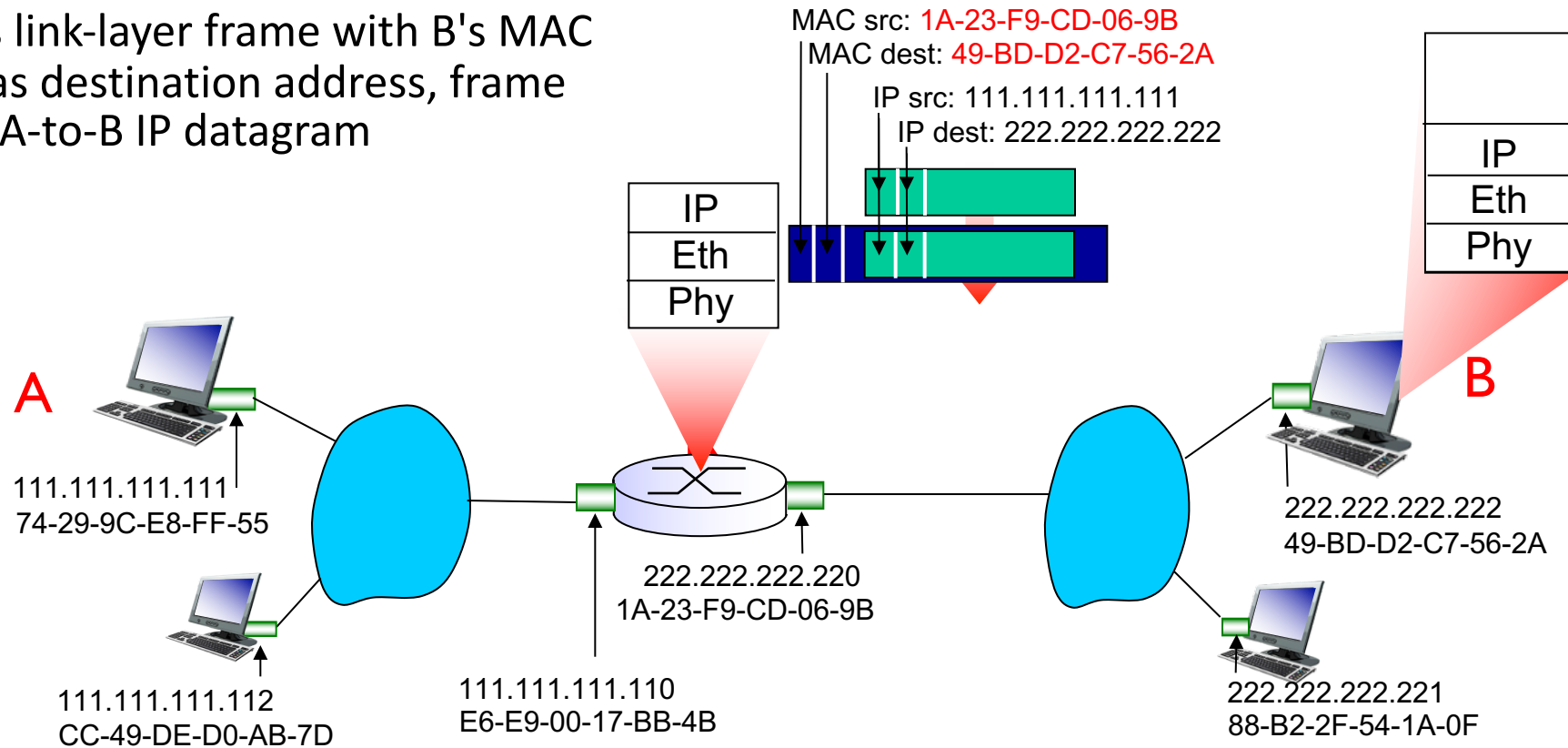
Addressing: routing to another LAN

- 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



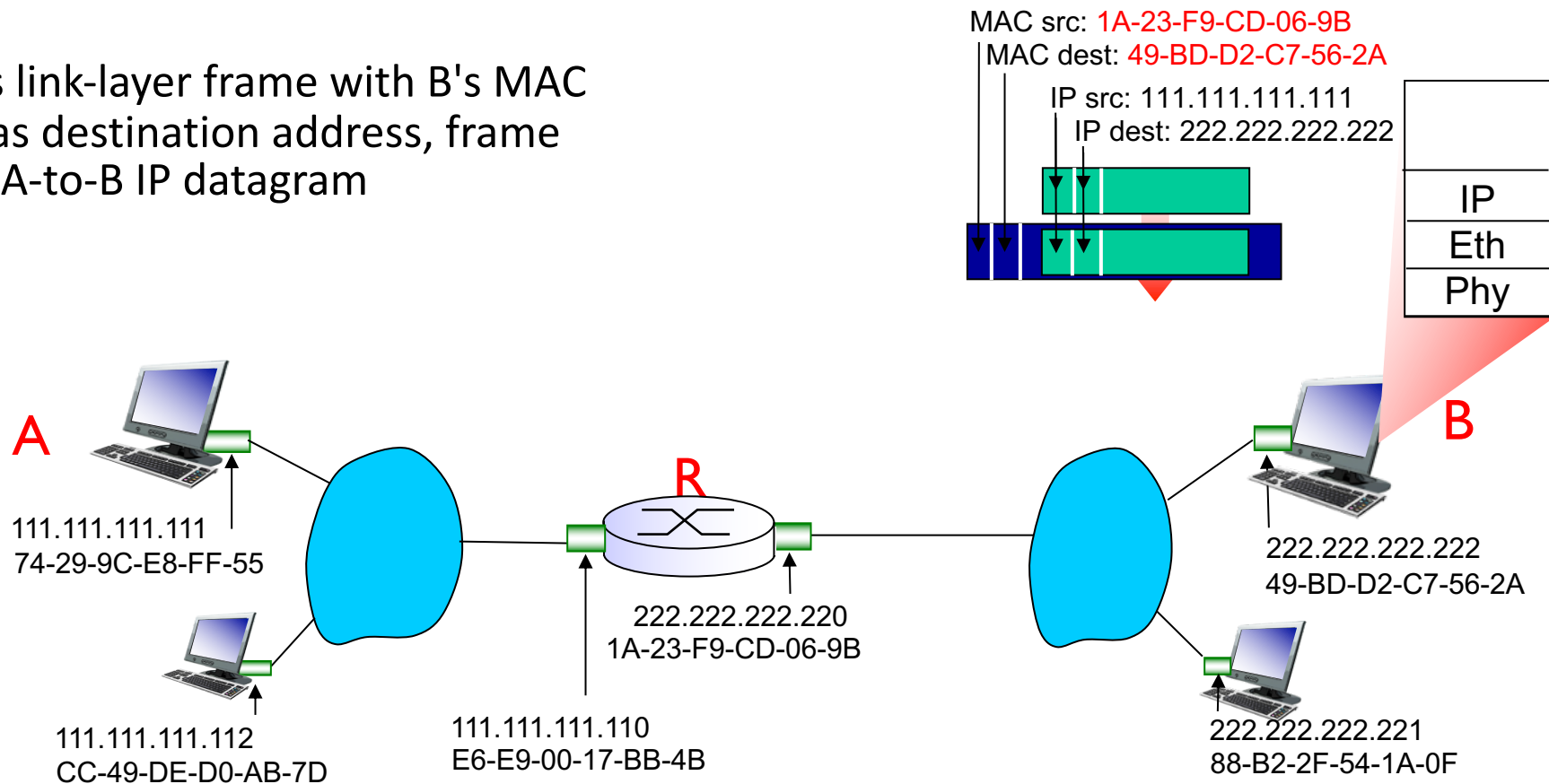
Addressing: routing to another LAN

- 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



Addressing: routing to another LAN

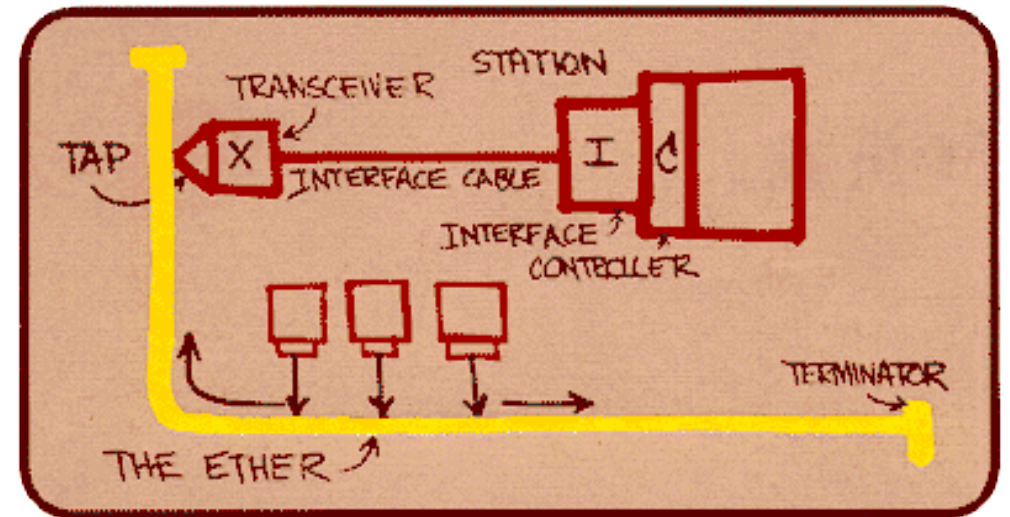
- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as destination address, frame contains A-to-B IP datagram



Ethernet

“dominant” wired LAN technology:

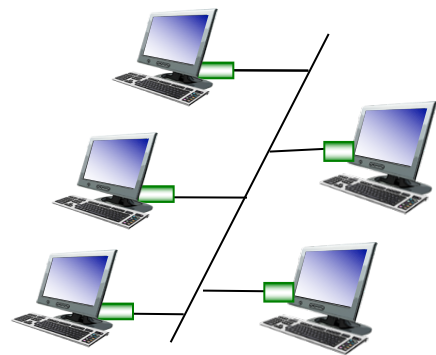
- single chip, multiple speeds (e.g., Broadcom BCM5761)
- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps - 10 Gbps



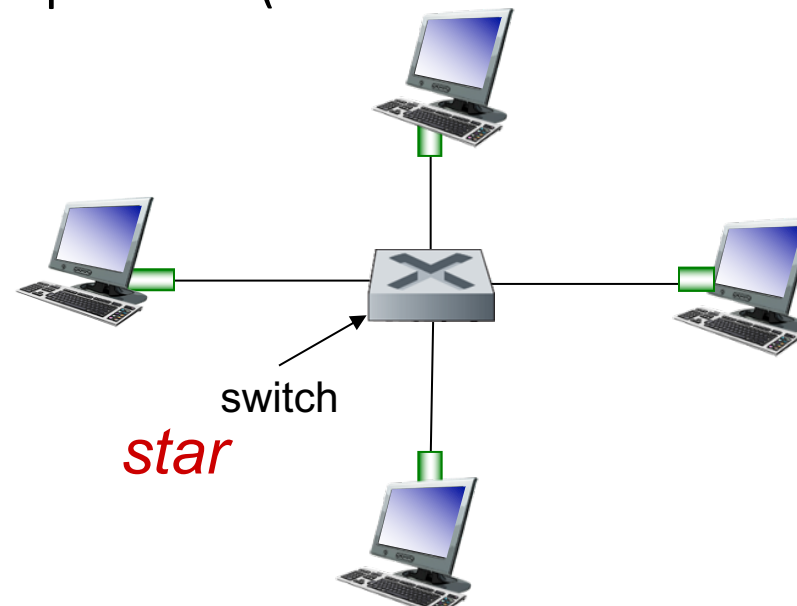
Metcalfe's Ethernet sketch

Ethernet: physical topology

- *bus*: popular through mid 90s
 - all nodes in same collision domain (can collide with each other)
 - CSMA/CD MAC with binary exponential backoff
- *star*: prevails today
 - active *switch* in center
 - each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)



bus: coaxial cable



Ethernet frame structure

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



Preamble (8 bytes):

- 7 bytes with pattern 10101010 followed by one byte with pattern 10101011
- used to synchronize receiver, sender clock rates

Ethernet frame structure (more)

- *addresses*: source (6 bytes), destination (6 bytes) 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 (2 bytes)*: indicates higher layer protocol (IP, ARP, etc.)



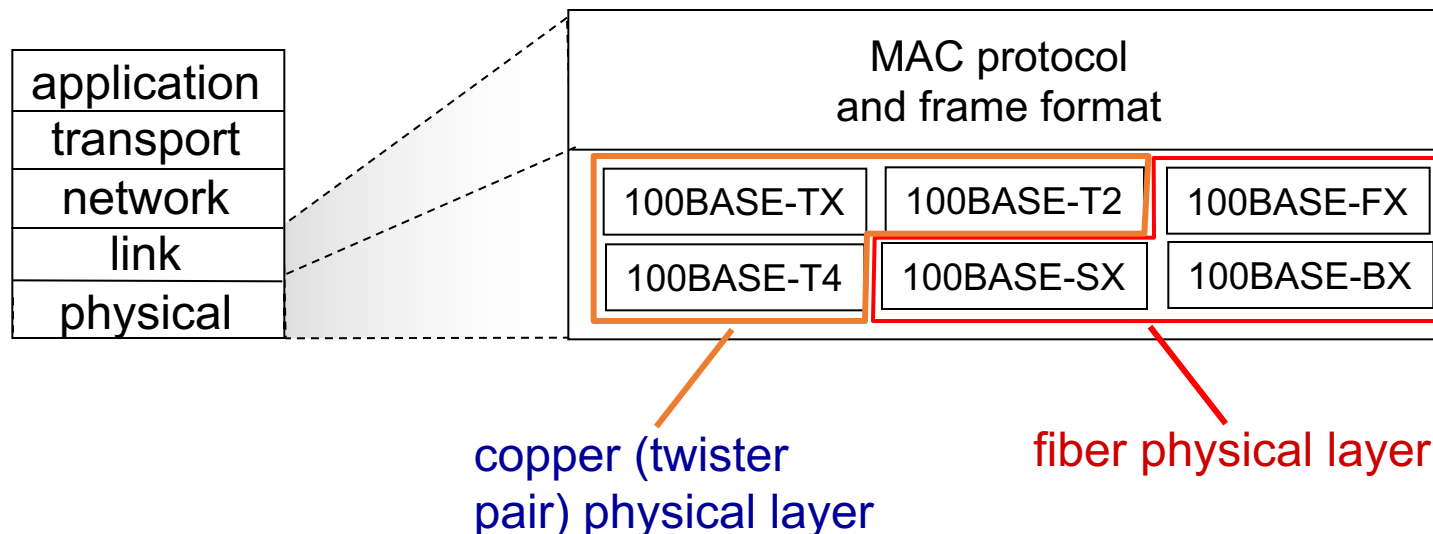
Ethernet: unreliable, connectionless

- *CRC (4 bytes)*: cyclic redundancy check at receiver
 - error detected: frame is dropped
- *connectionless*: no handshaking between sending and receiving NICs
- *unreliable*: receiving NIC doesn't send acks or nacks to sending NIC
 - data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost



802.3 Ethernet standards: link & physical layers

- *many* different Ethernet standards
 - common MAC protocol and frame format
 - different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1Gbps, 10 Gbps, 40 Gbps
 - different physical layer media: fiber, cable

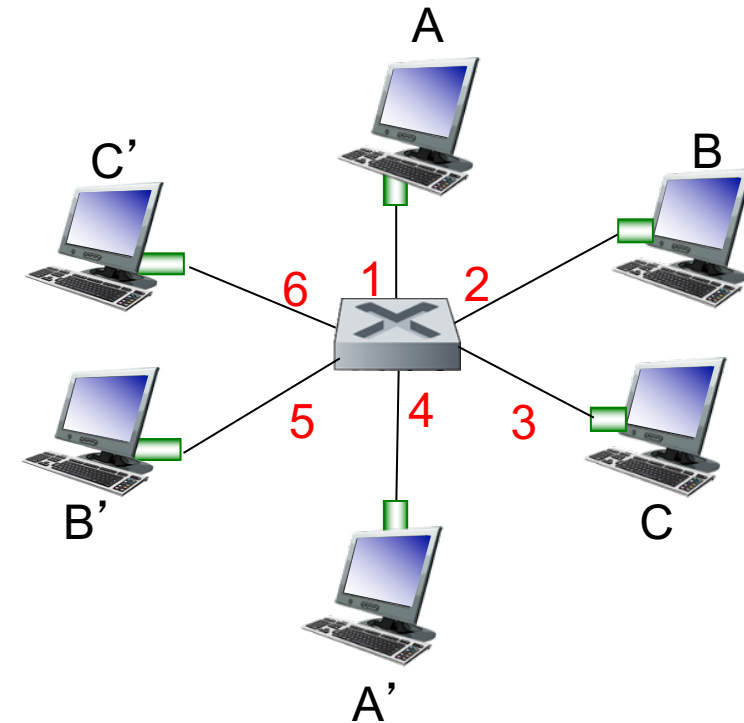


Ethernet switch

- link-layer device: takes an *active* role
 - store, forward Ethernet frames
- *transparent*
 - hosts are unaware of presence of switches
- *plug-and-play, self-learning*
 - switches do not need to be configured

Switch: *multiple* simultaneous transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, but no collisions; full duplex
 - each link is its own collision domain
- **switching**: A-to-A' and B-to-B' can transmit simultaneously, without collisions



*switch with six interfaces
(1,2,3,4,5,6)*

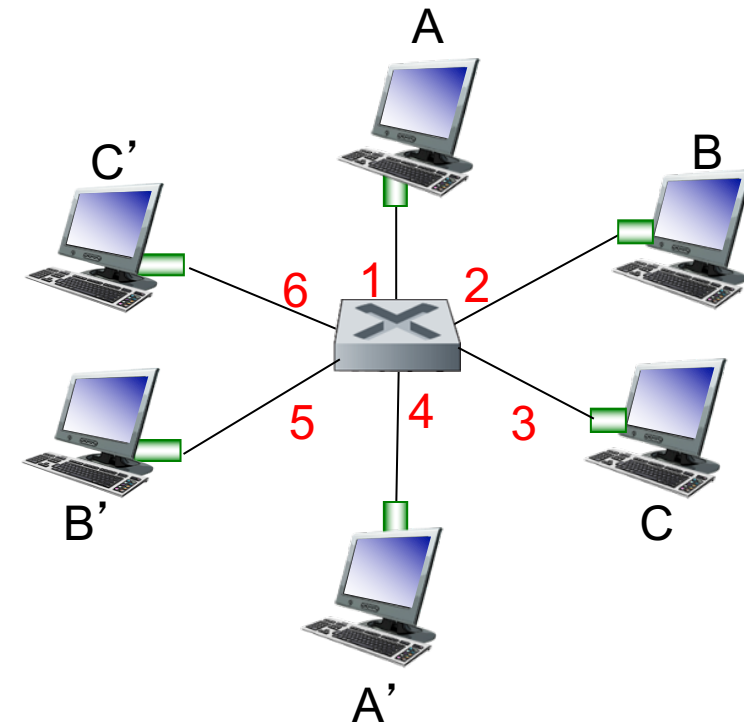
Switch forwarding table

Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

- A: each switch has a **switch table**, each entry:
 - (MAC address of host, interface to reach host, time stamp)
 - looks like a routing table!

Q: how are entries created, maintained in switch table?

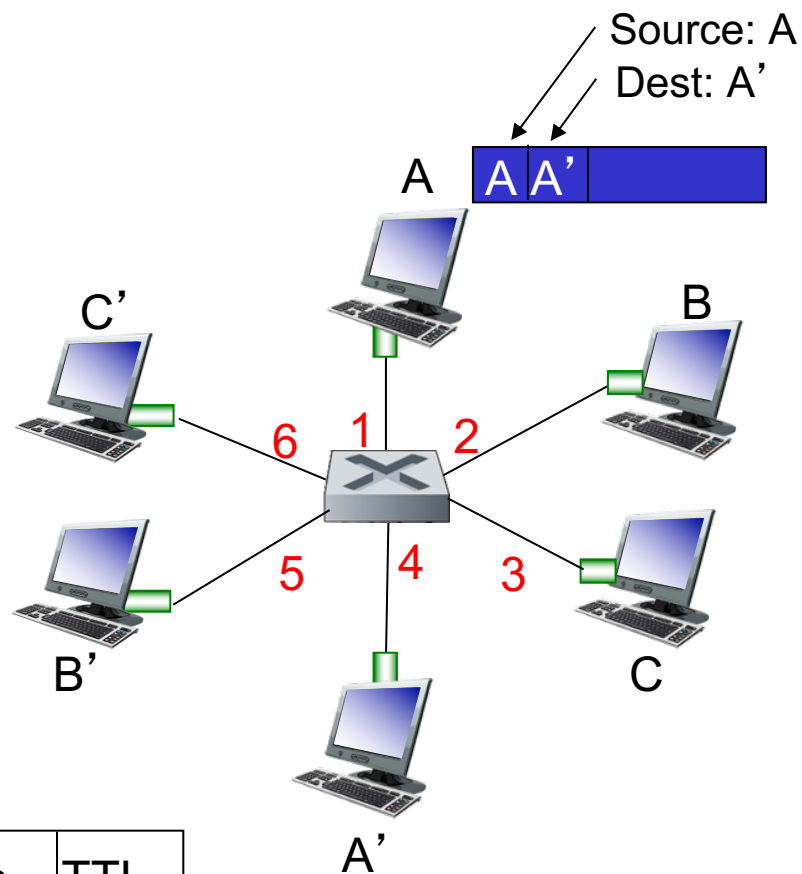
- something like a routing protocol?



*switch with six interfaces
(1,2,3,4,5,6)*

Switch: self-learning

- switch *learns* which hosts can be reached through which interfaces
 - when frame received, switch “learns” location of sender: incoming LAN segment
 - records sender/location pair in switch table



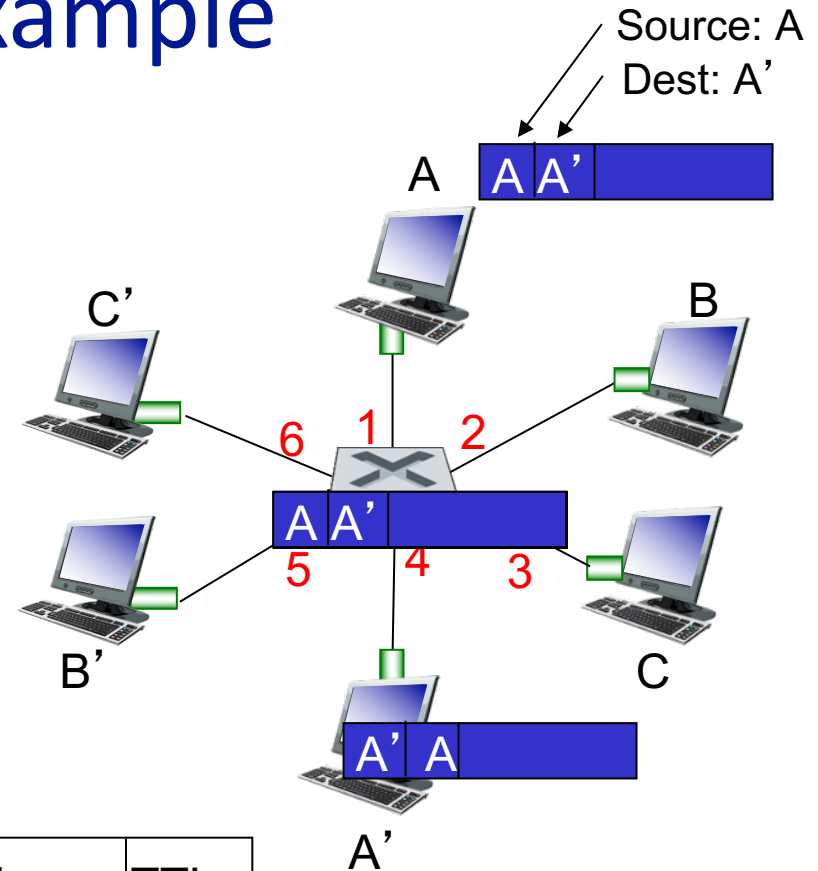
MAC addr	interface	TTL
A	1	60

*Switch table
(initially empty)*

Self-learning, forwarding: example

- frame destination, A', location unknown:
- destination A location known **selectively send on just one link**

flood



MAC addr	interface	TTL
A	1	60
A'	4	60

*switch table
(initially empty)*

Switch: frame filtering/forwarding

when frame received at switch:

1. record incoming link, MAC address of sending host

2. index switch table using MAC destination address

3. if entry found for destination

then {

 if destination on segment from which frame arrived

 then drop frame

 else forward frame on interface indicated by entry

}

else flood /* forward on all interfaces except arriving interface */

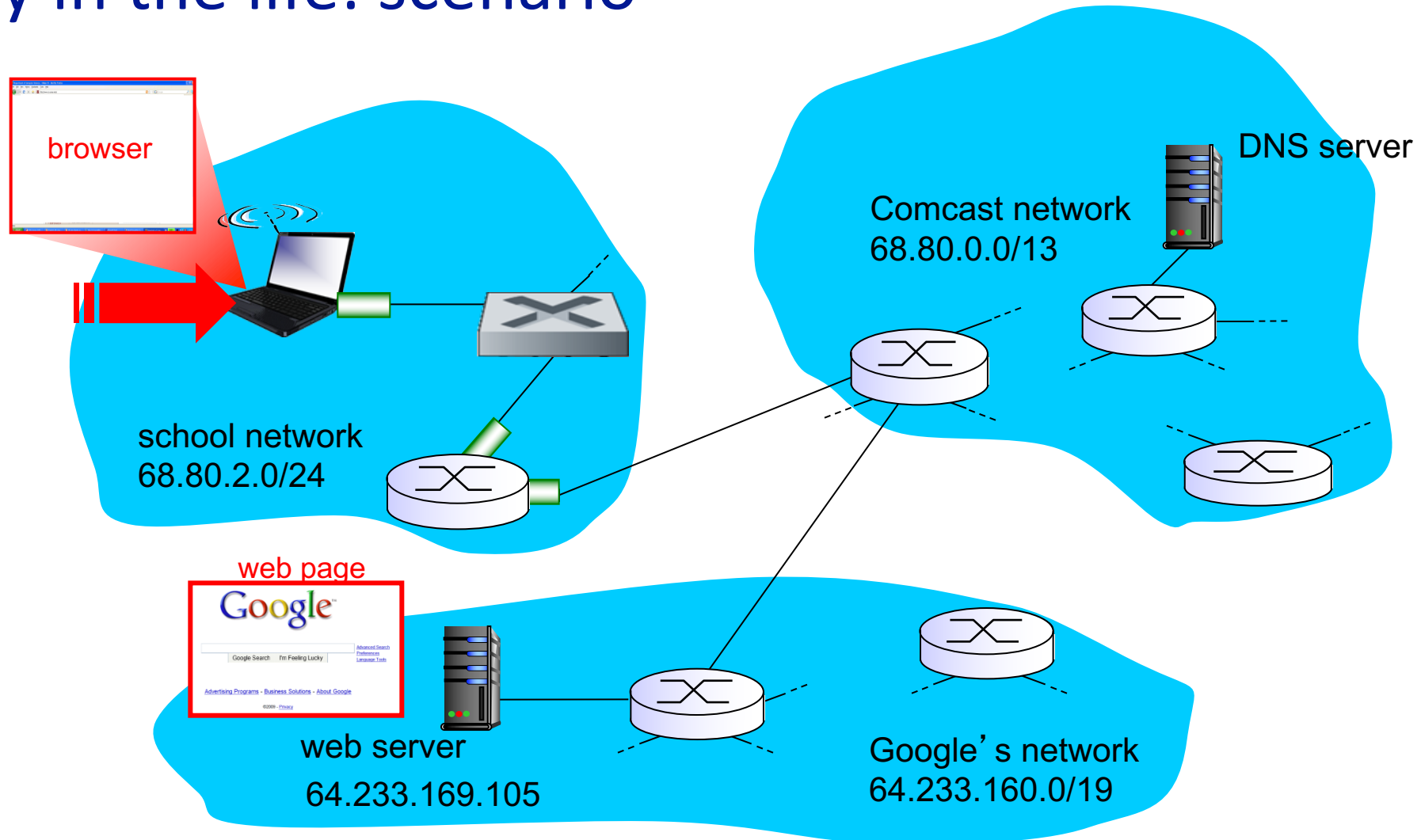
Outline

- overview
- error detection & correction
- multiple access
- local area networks
- a day in the life of a web request

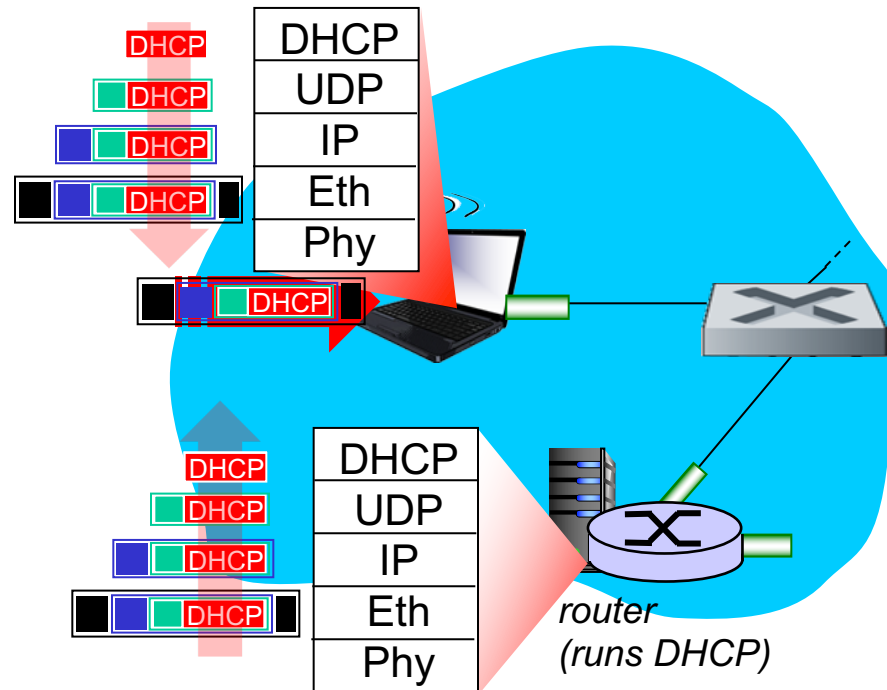
Synthesis: a day in the life of a web request

- journey down protocol stack complete!
 - application, transport, network, link
- putting-it-all-together: synthesis!
 - *goal:* identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
 - *scenario:* student attaches laptop to campus network, requests/receives `www.google.com`

A day in the life: scenario

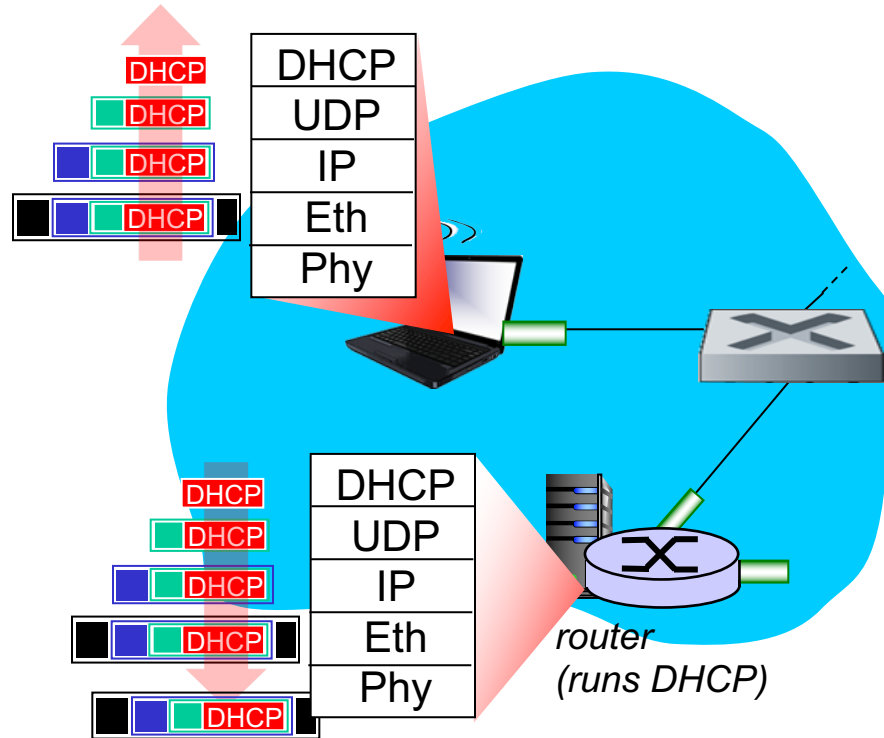


A day in the life... connecting to the Internet



- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use *DHCP*
- DHCP request **encapsulated** in **UDP**, encapsulated in **IP**, encapsulated in **802.3 Ethernet**
- Ethernet frame **broadcast** (dest: FFFFFFFF) on LAN, received at router running **DHCP** server
- Ethernet **demuxed** to IP demuxed, UDP demuxed to DHCP

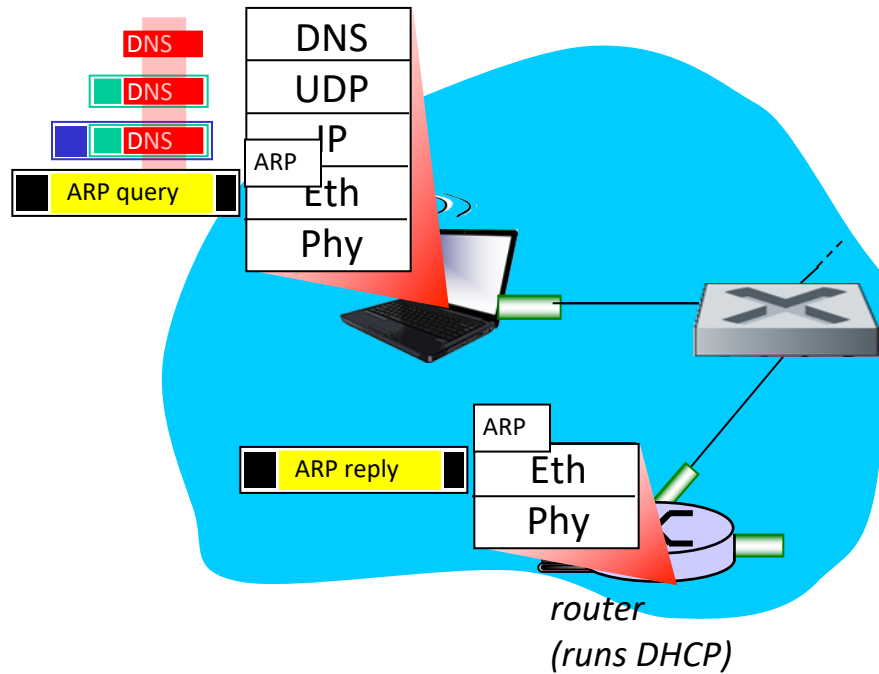
A day in the life... connecting to the Internet



- DHCP server formulates *DHCP ACK* containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation at DHCP server, frame forwarded (*switch learning*) through LAN, demultiplexing at client
- DHCP client receives DHCP ACK reply

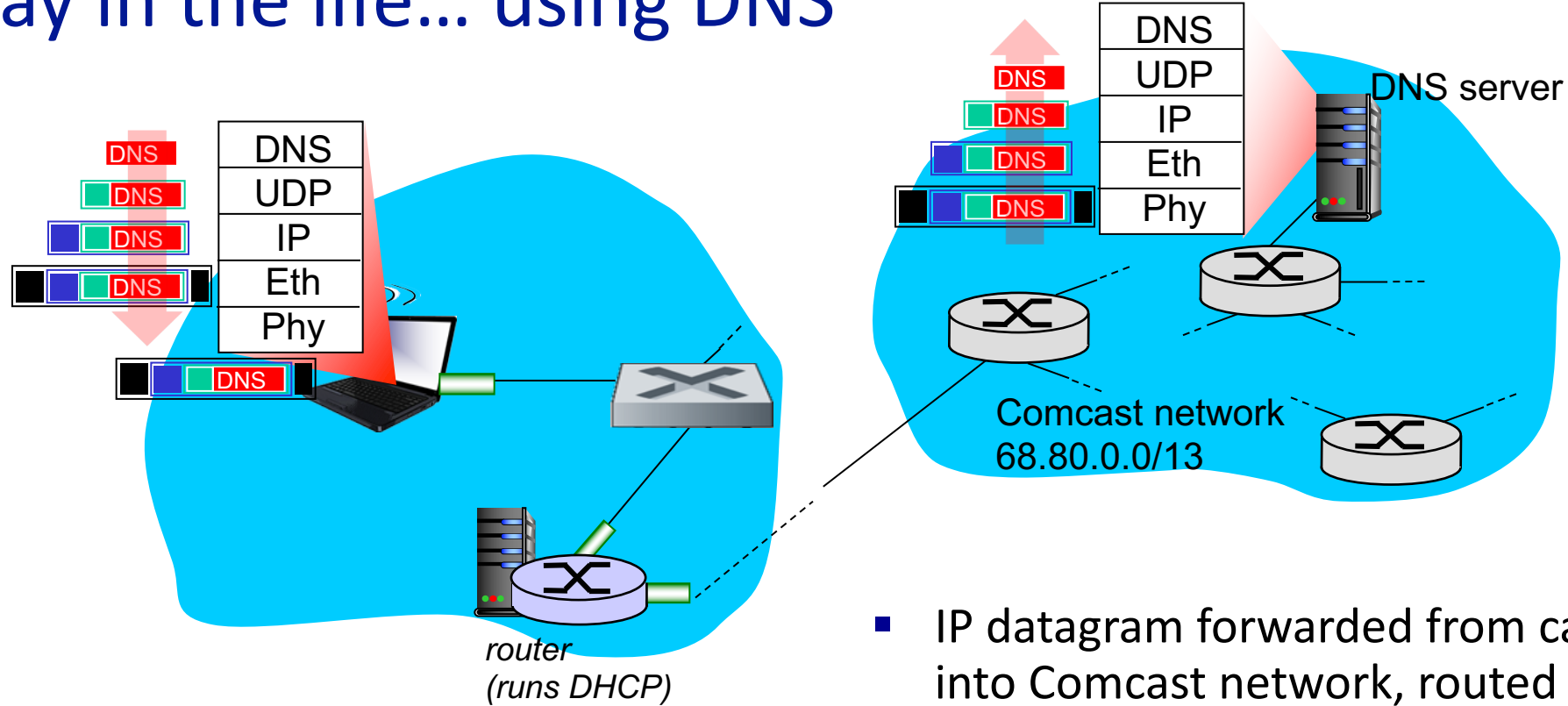
Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

A day in the life... ARP (before DNS, before HTTP)



- before sending *HTTP* request, need IP address of `www.google.com`: *DNS*
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: *ARP*
- *ARP query* broadcast, received by router, which replies with *ARP reply* giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

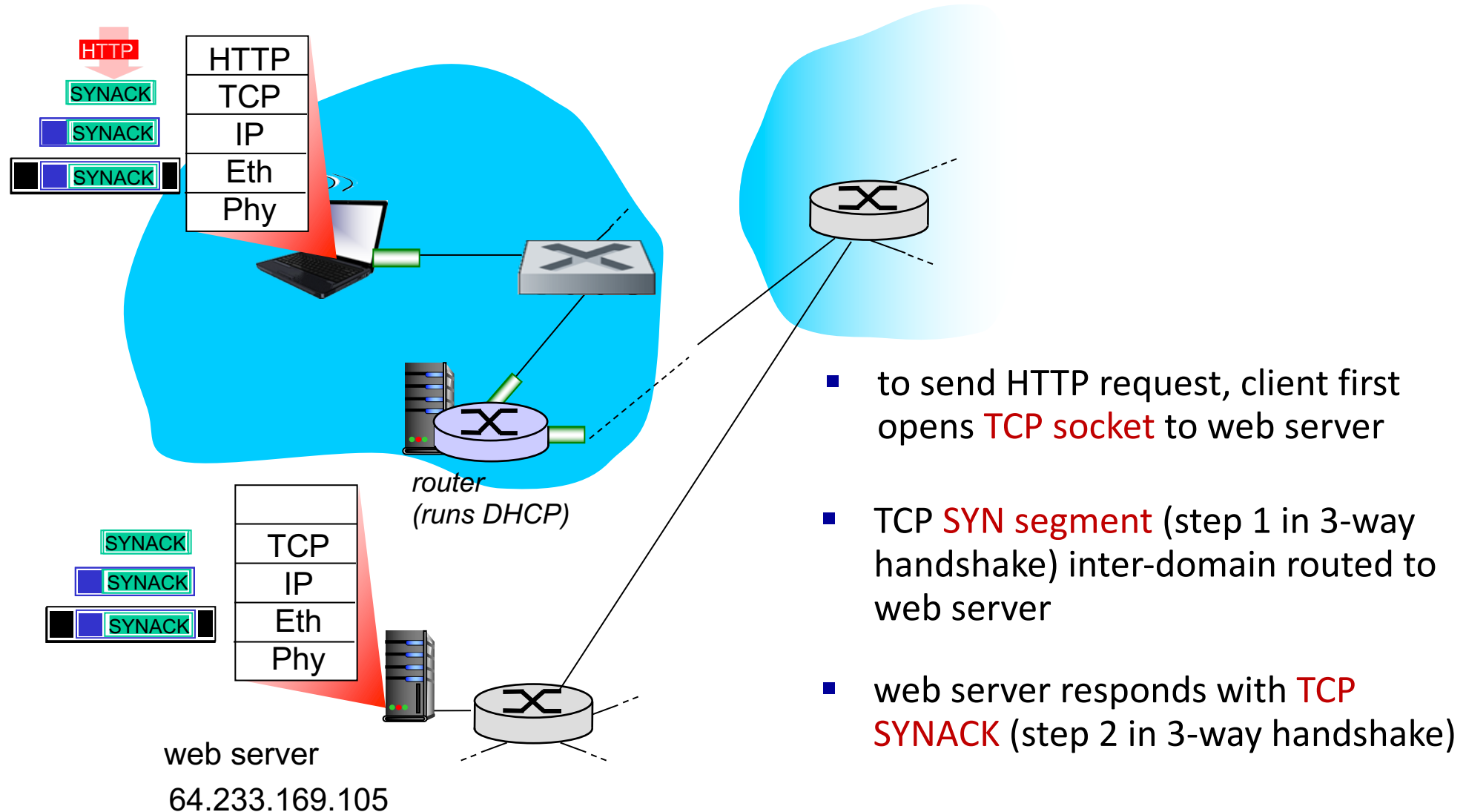
A day in the life... using DNS



- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router

- IP datagram forwarded from campus network into Comcast network, routed (tables created by **RIP**, **OSPF**, **IS-IS** and/or **BGP** routing protocols) to DNS server
- demuxed to DNS server
- DNS server replies to client with IP address of www.google.com

A day in the life...TCP connection carrying HTTP



A day in the life... HTTP request/reply

