

Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- **LANs**
 - **addressing, ARP**
 - Ethernet
 - switches
 - VLANs
- link virtualization: MPLS
- data center networking



- a day in the life of a web request

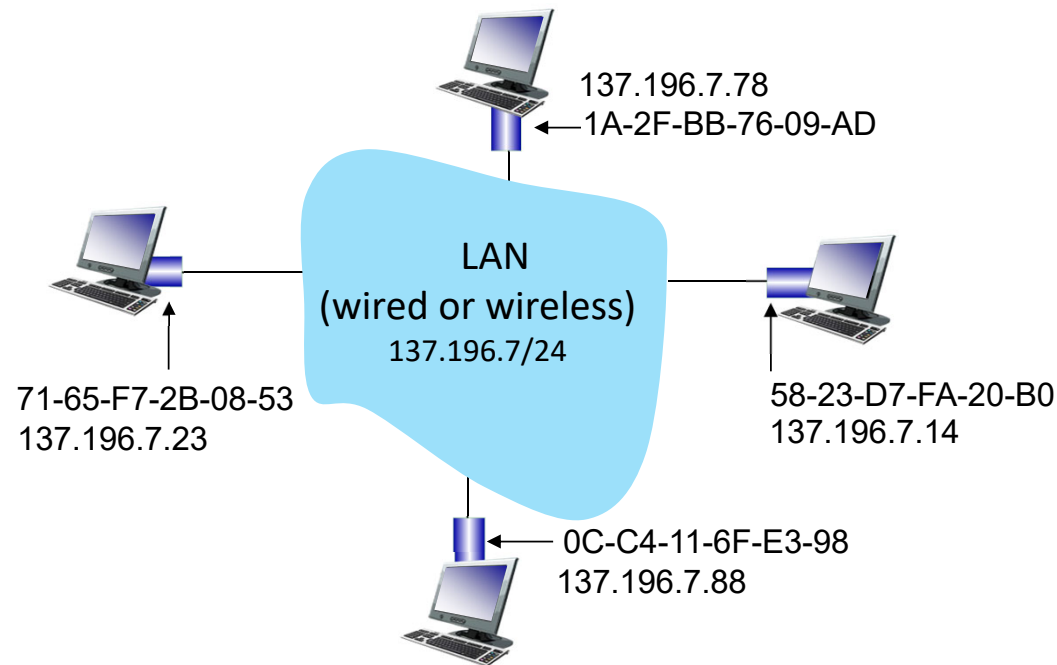
MAC addresses

- 32-bit IP address:
 - *network-layer* address for interface
 - used for layer 3 (network layer) forwarding
 - e.g.: 128.119.40.136
- MAC (or LAN or physical or Ethernet) address:
 - function: used “locally” to get frame from one interface to another physically-connected interface (same subnet, 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 interface on LAN

- has unique 48-bit **MAC** address
- has a locally unique 32-bit IP address (as we've seen)

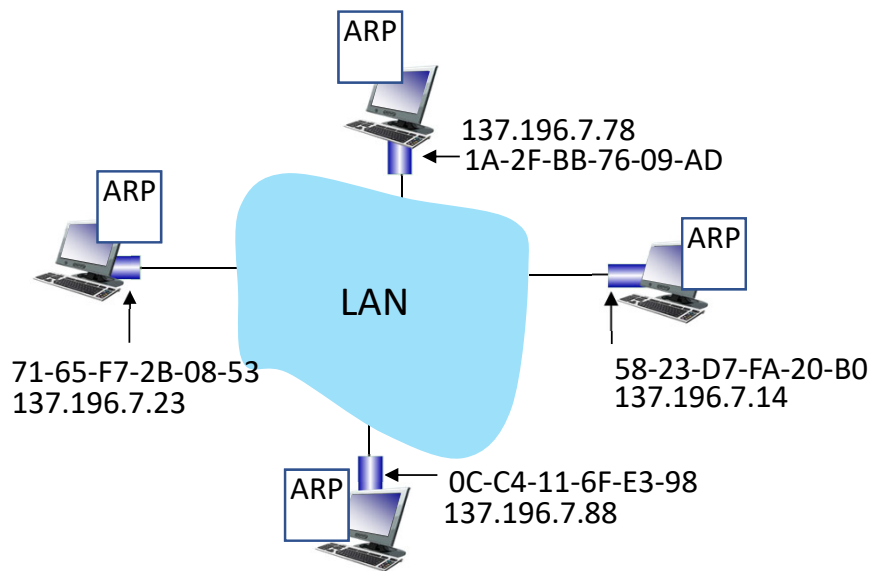


MAC addresses

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

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 in action

example: A wants to send datagram to B

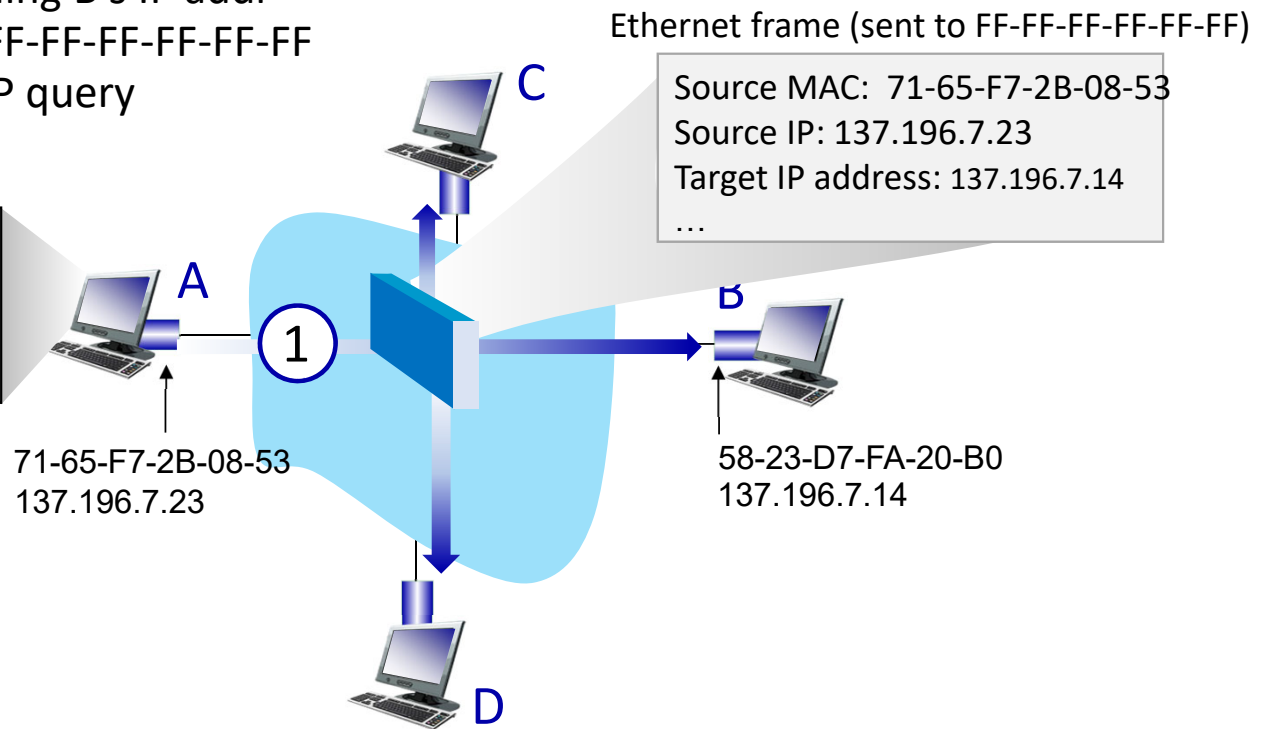
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address

A broadcasts ARP query, containing B's IP addr

- ①
- destination MAC address = FF-FF-FF-FF-FF-FF
 - all nodes on LAN receive ARP query

ARP table in A

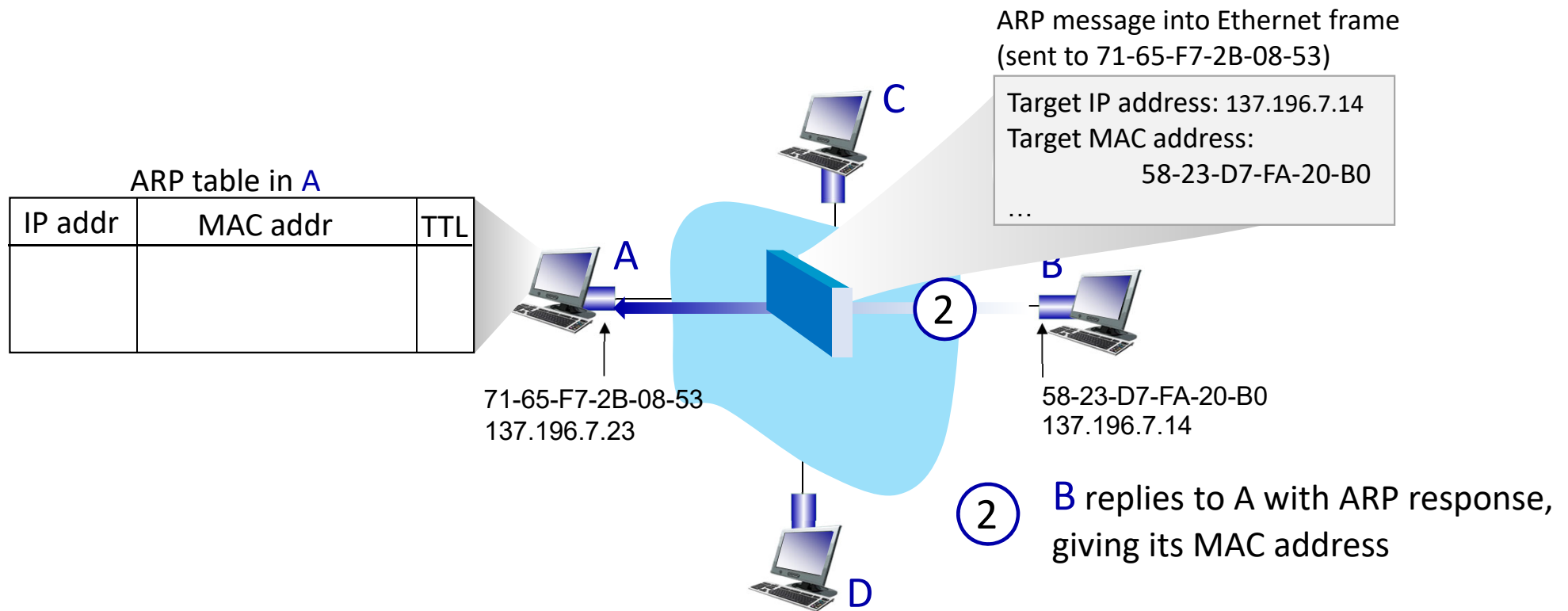
IP addr	MAC addr	TTL



ARP protocol in action

example: A wants to send datagram to B

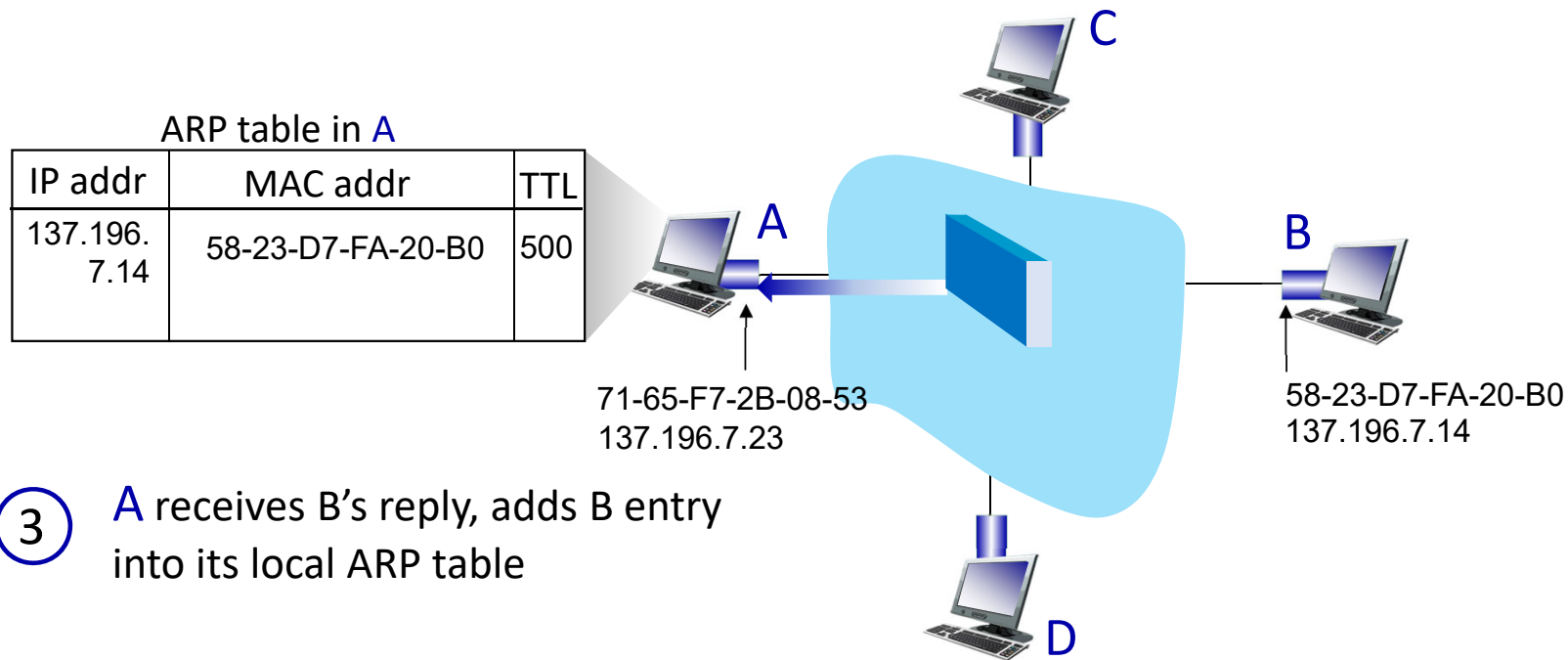
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



ARP protocol in action

example: A wants to send datagram to B

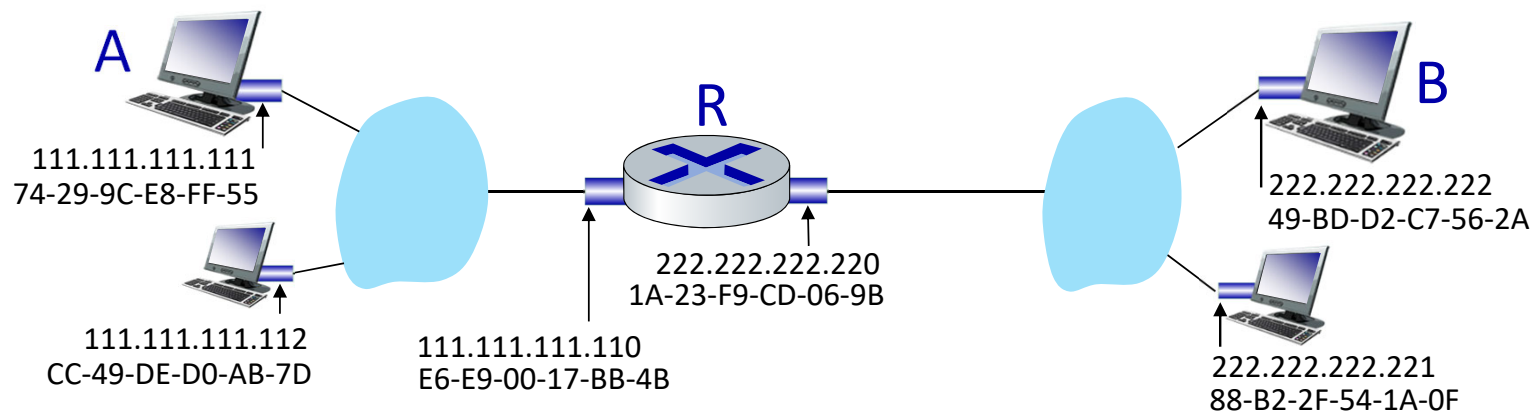
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



Routing to another subnet: addressing

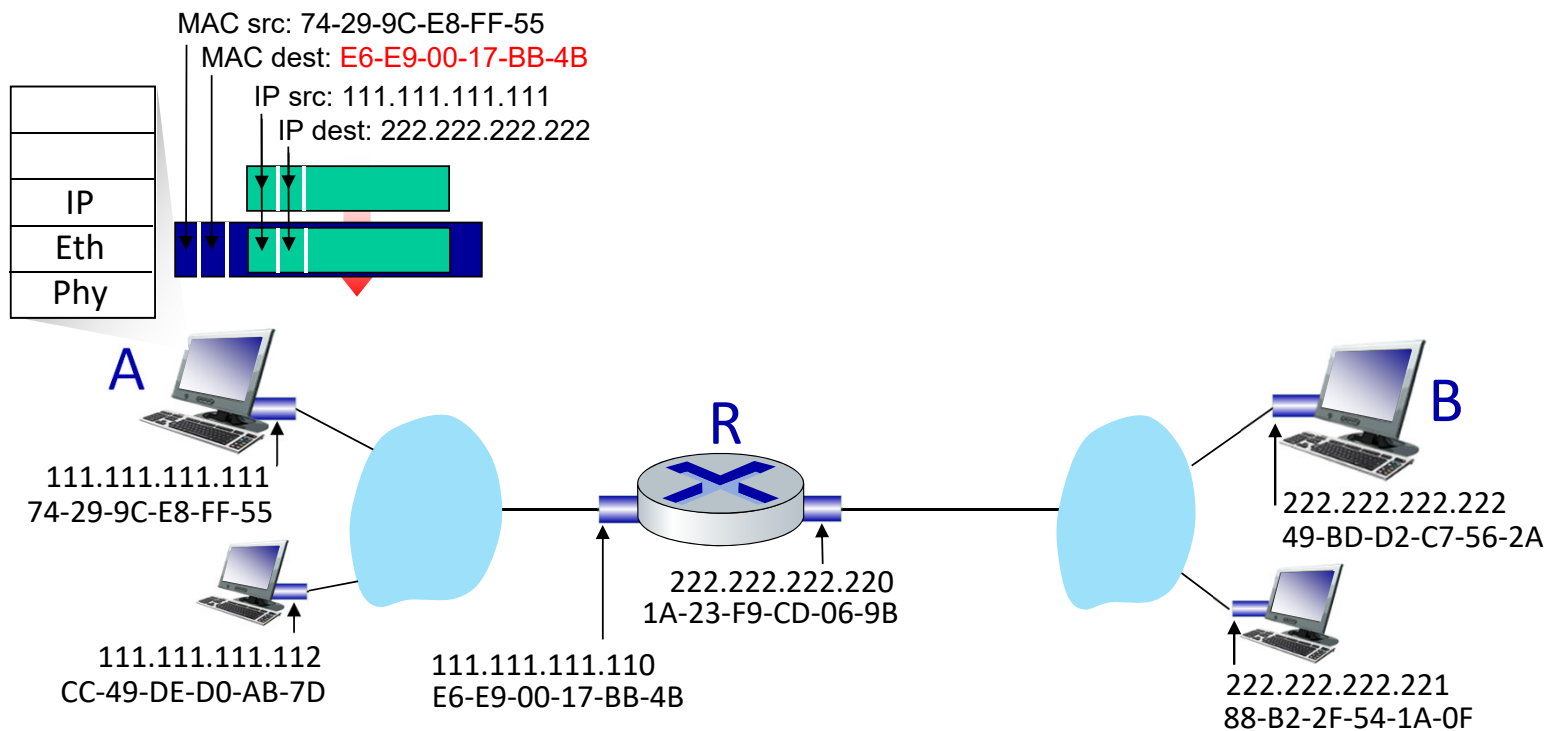
walkthrough: **sending a datagram from A to B via R**

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



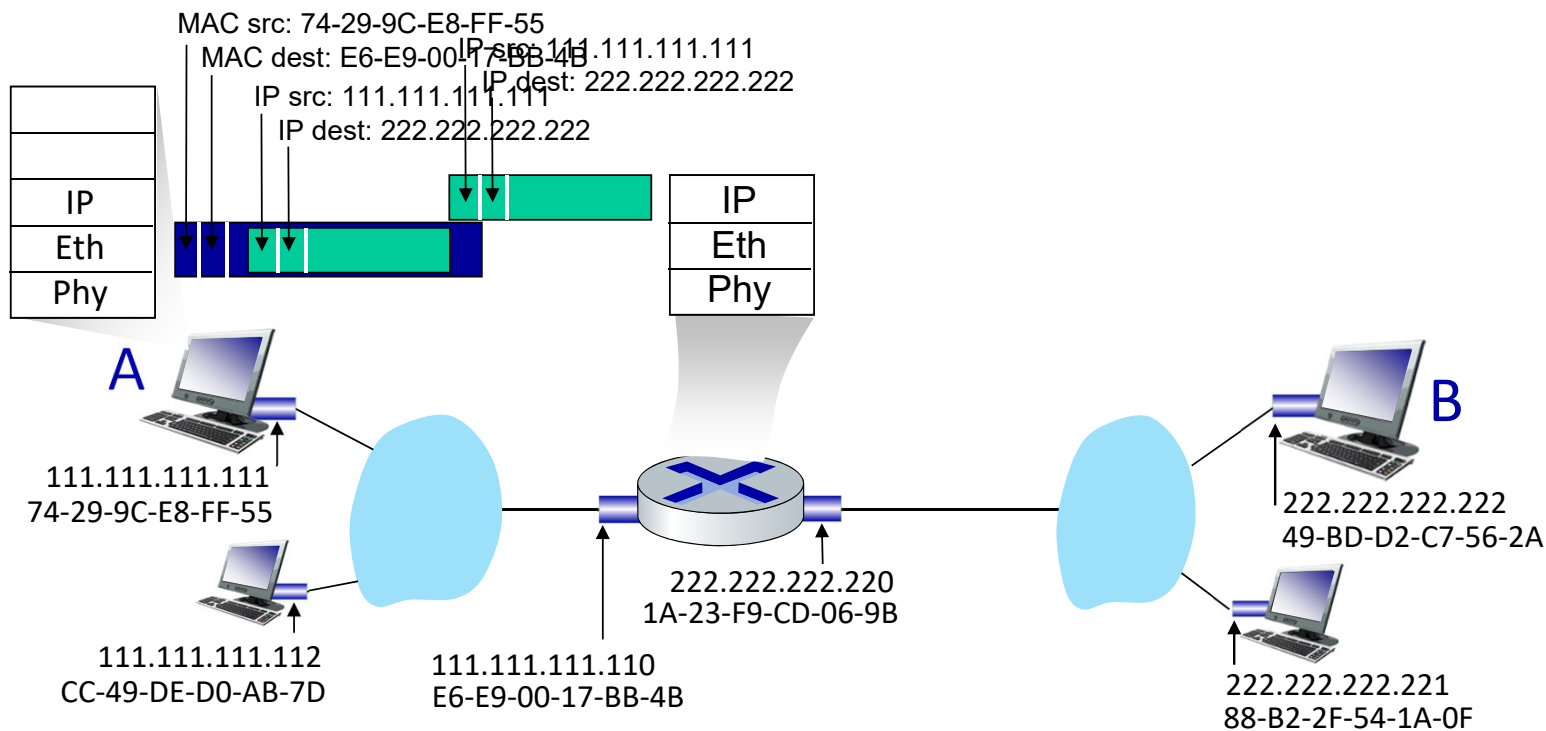
Routing to another subnet: addressing

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



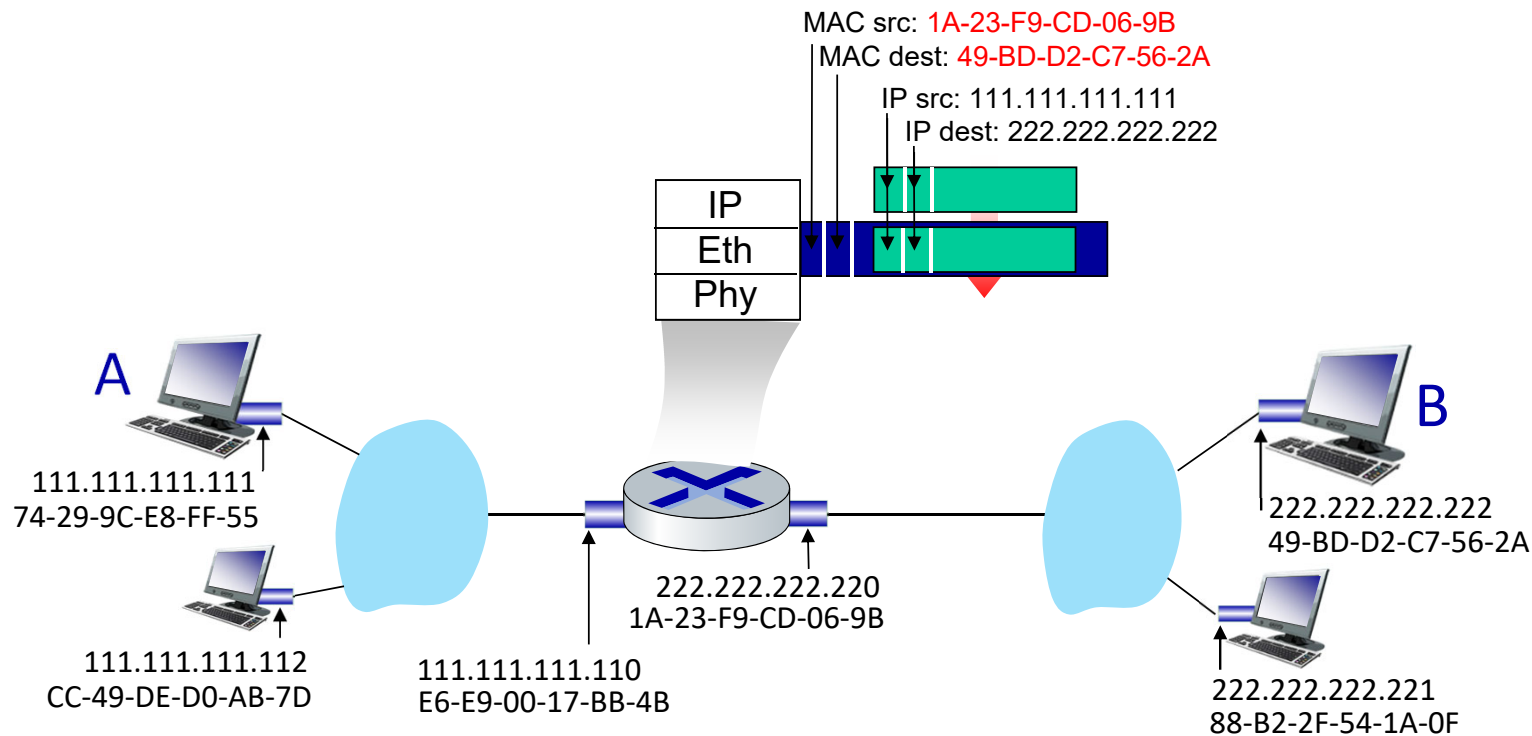
Routing to another subnet: addressing

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



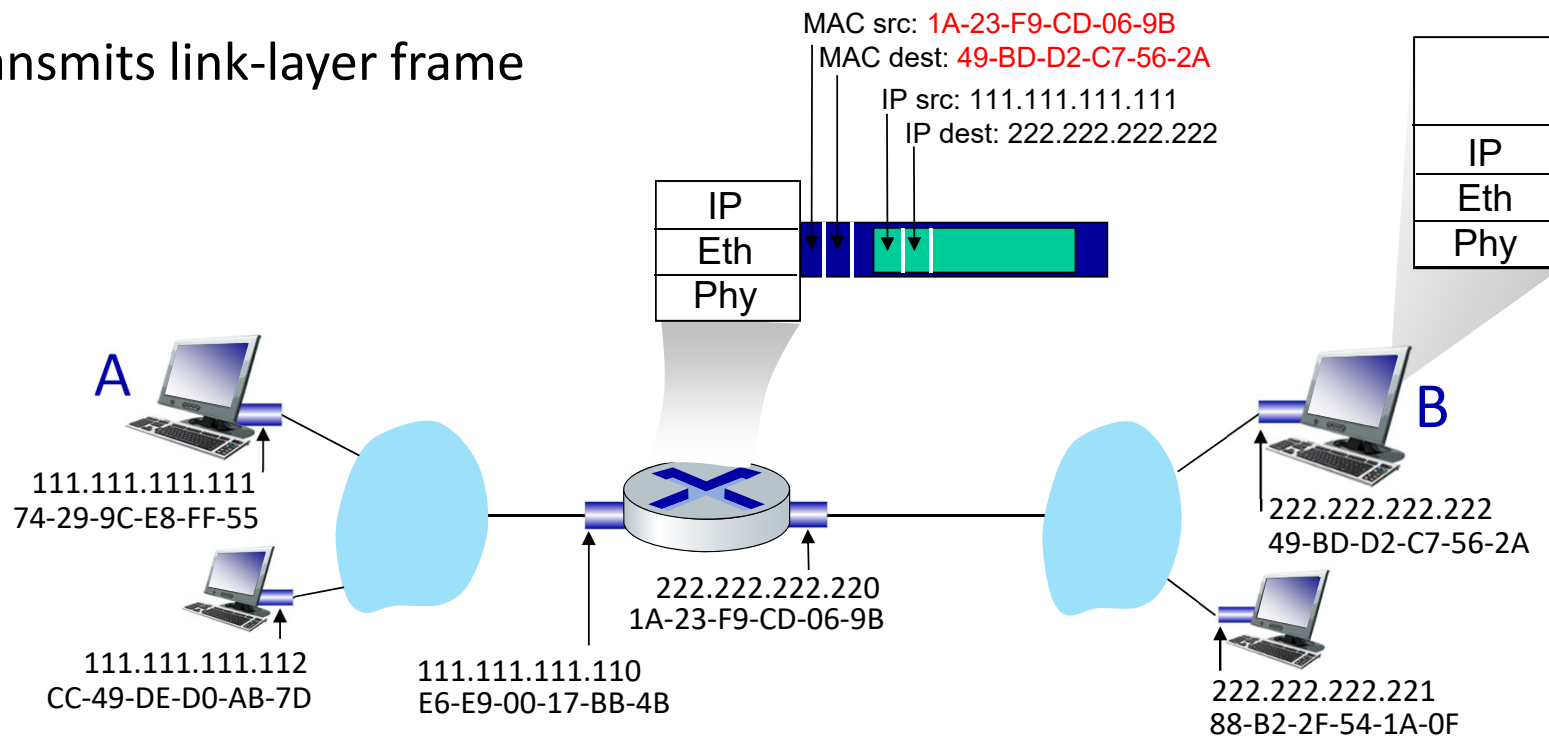
Routing to another subnet: addressing

- R determines outgoing interface, passes datagram with IP source A, destination B to link layer
- R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address



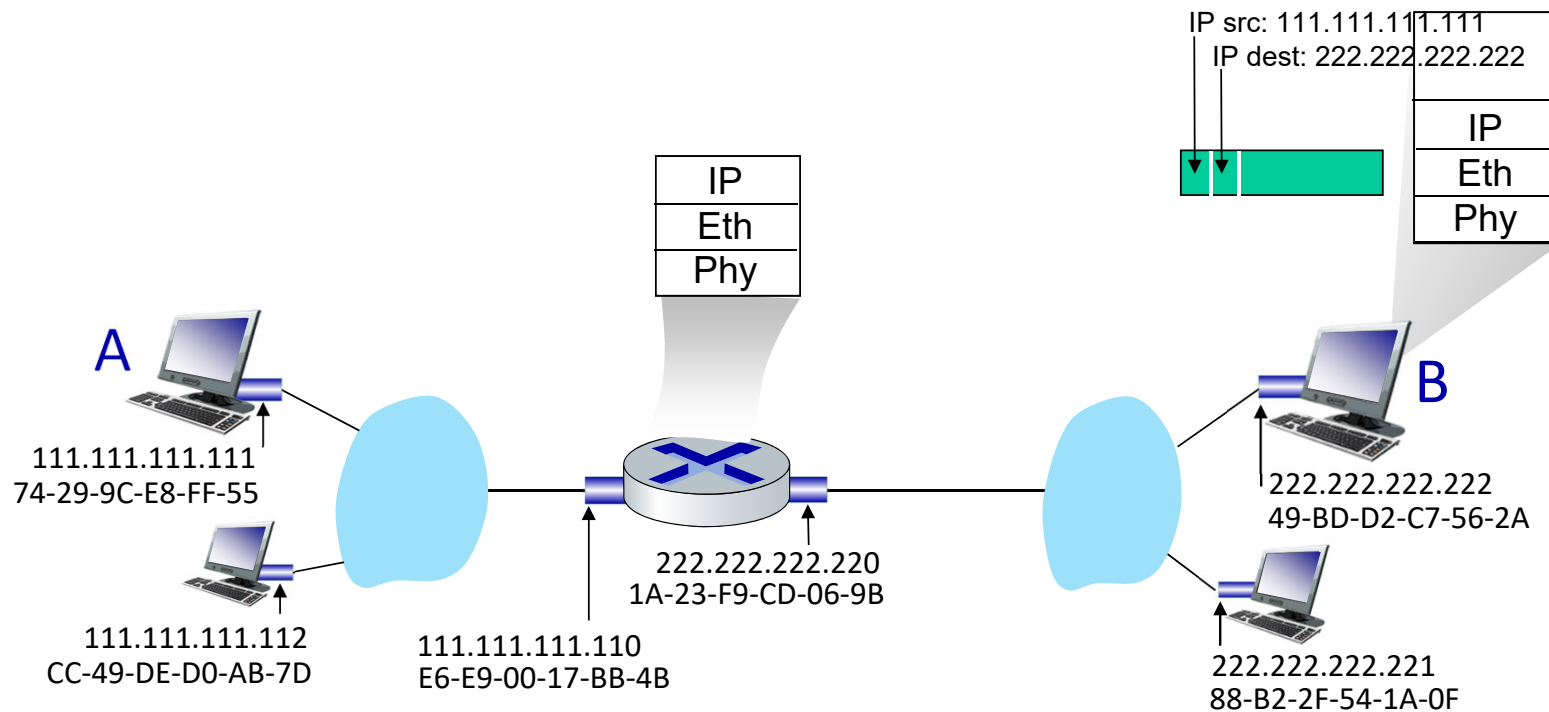
Routing to another subnet: addressing

- R determines outgoing interface, passes datagram with IP source A, destination B to link layer
- R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address
- transmits link-layer frame



Routing to another subnet: addressing

- B receives frame, extracts IP datagram destination B
- B passes datagram up protocol stack to IP



Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- **LANs**
 - addressing, ARP
 - **Ethernet**
 - switches
 - VLANs
- link virtualization: MPLS
- data center networking

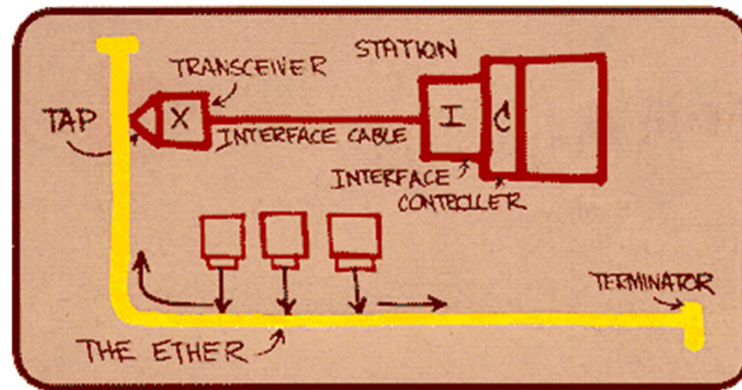


- a day in the life of a web request

Ethernet

“dominant” wired LAN technology:

- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps – 400 Gbps
- single chip, multiple speeds (e.g., Broadcom BCM5761)

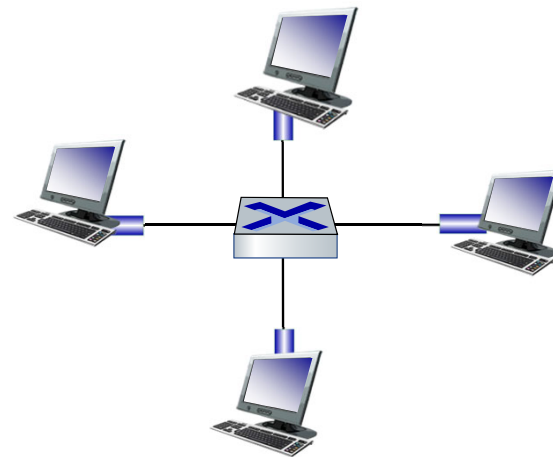
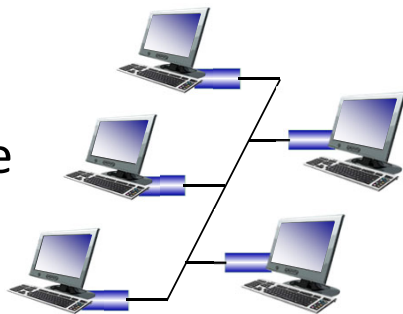


Metcalfe's Ethernet sketch

Ethernet: physical topology

- **bus:** popular through mid 90s
 - all nodes in same collision domain (can collide with each other)
- **switched:** prevails today
 - active link-layer 2 *switch* in center
 - each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)

bus: coaxial cable



switched

Ethernet frame structure

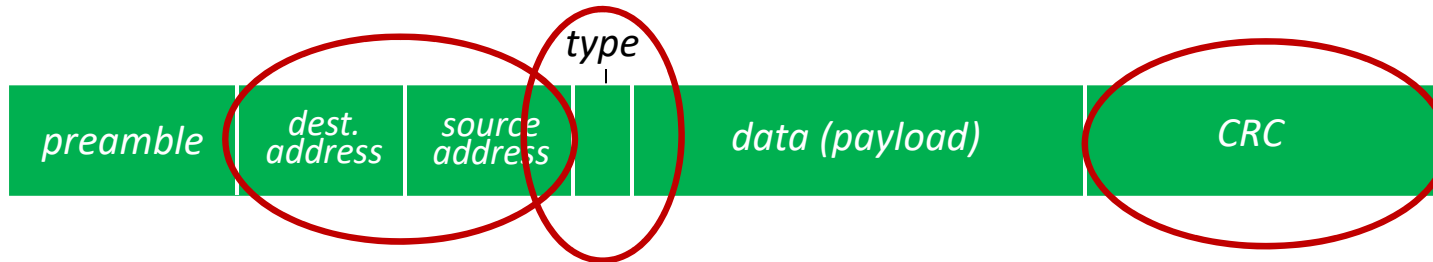
sending interface encapsulates IP datagram (or other network layer protocol packet) in **Ethernet frame**



preamble:

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

Ethernet frame structure (more)



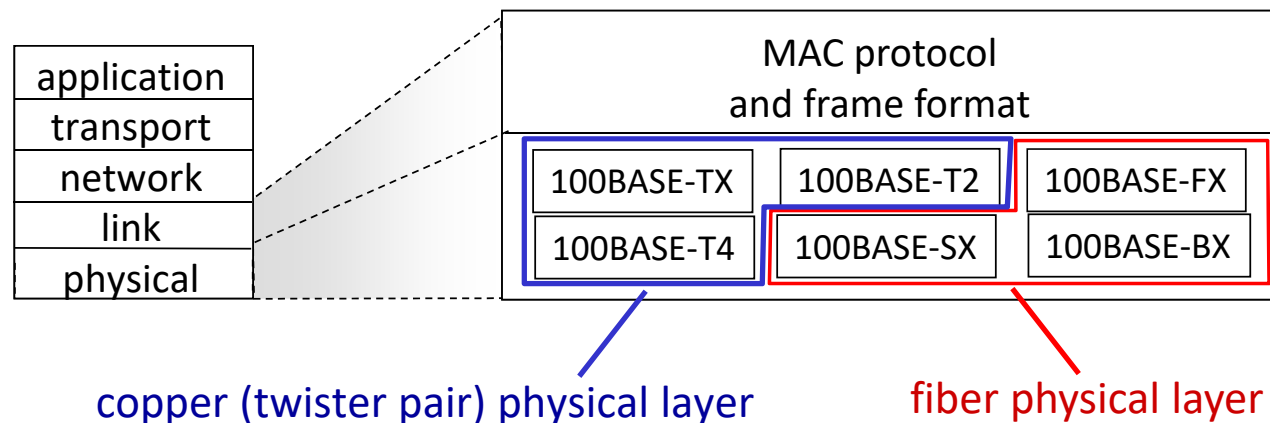
- **addresses:** 6 byte source, destination MAC addresses
 - if adapter receives frame with matching destination address, or with broadcast address (e.g., ARP packet), it passes data in frame to network layer protocol
 - otherwise, adapter discards frame
- **type:** indicates higher layer protocol
 - mostly IP but others possible, e.g., Novell IPX, AppleTalk
 - used to demultiplex up at receiver
- **CRC:** cyclic redundancy check at receiver
 - error detected: frame is dropped

Ethernet: unreliable, connectionless

- **connectionless**: no handshaking between sending and receiving NICs
- **unreliable**: receiving NIC doesn't send ACKs or NAKs to sending NIC
 - data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost
- Ethernet's MAC protocol: unslotted **CSMA/CD with binary backoff**

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



Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- **LANs**
 - addressing, ARP
 - Ethernet
 - **switches**
 - VLANs
- link virtualization: MPLS
- data center networking



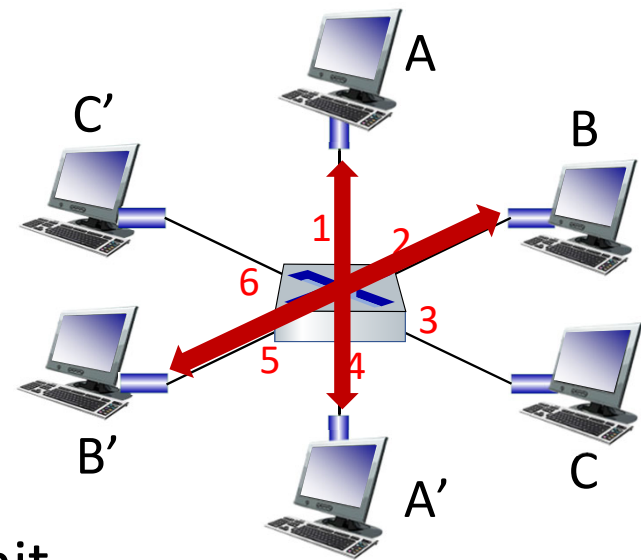
- a day in the life of a web request

Ethernet switch

- Switch is a **link-layer** device: takes an *active* role
 - store, forward Ethernet frames
 - examine incoming frame's MAC address, *selectively* forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- **transparent**: hosts *unaware* of presence of switches
- **plug-and-play, self-learning**
 - switches do not need to be configured

Switch: multiple simultaneous transmissions

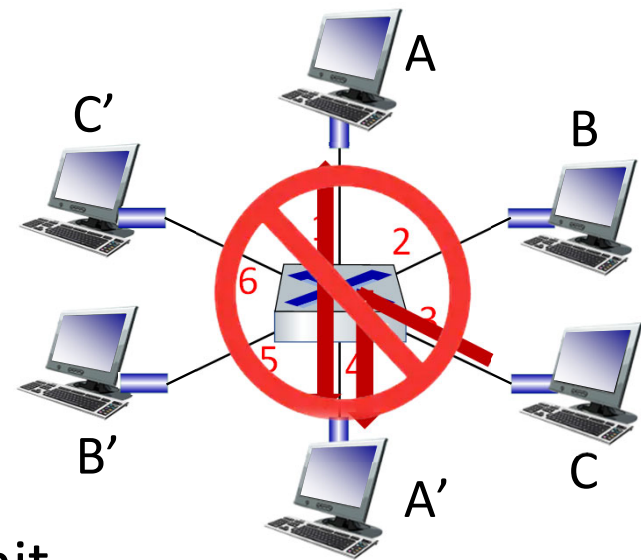
- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, so:
 - no collisions; full duplex
 - each link is its own collision domain
- **switching:** A-to-A' and B-to-B' can transmit simultaneously, without collisions



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

Switch: multiple simultaneous transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, so:
 - no collisions; full duplex
 - each link is its own collision domain
- **switching:** A-to-A' and B-to-B' can transmit simultaneously, without collisions
 - but A-to-A' and C to A' can *not* happen simultaneously



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

Switch forwarding table

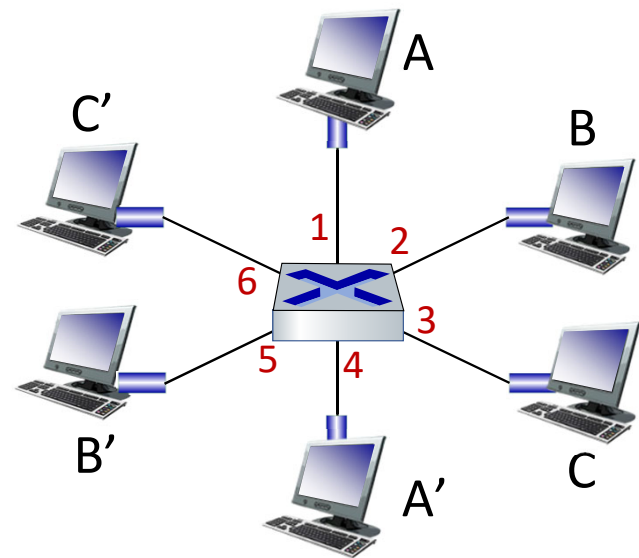
Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

A: each switch has a **switch table**, each entry:

- (MAC address of host, interface to reach host, time stamp)
- looks like a routing table!

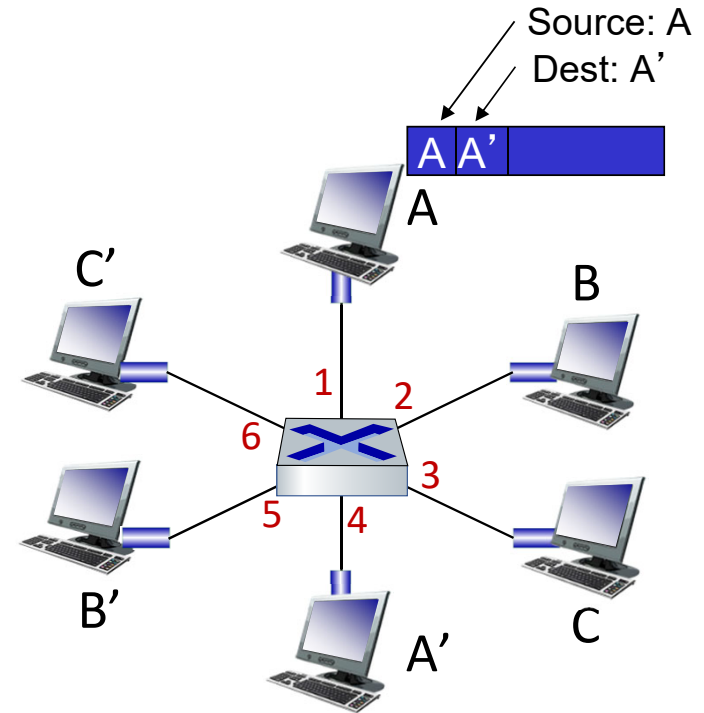
Q: how are entries created, maintained in switch table?

- something like a routing protocol?



Switch: 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)*

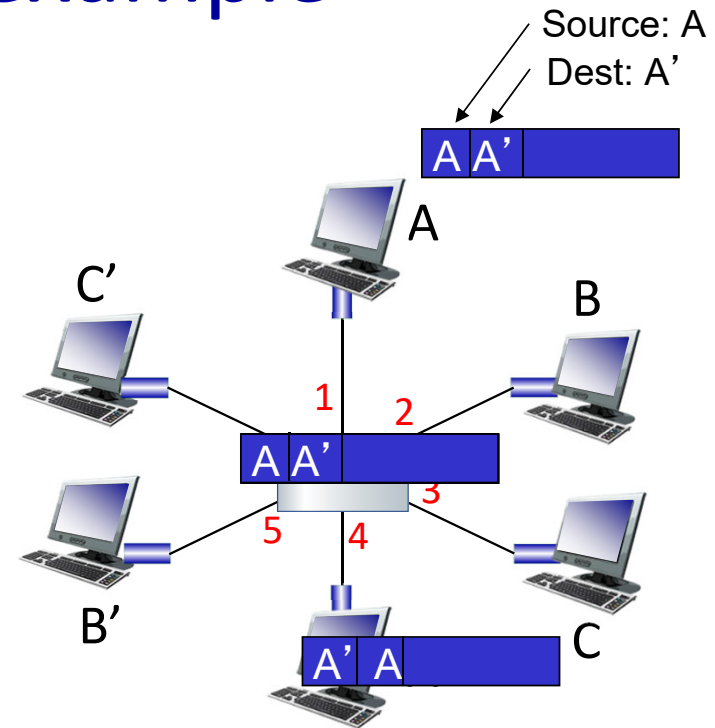
Switch: frame filtering/forwarding

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 */

Self-learning, forwarding: example

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

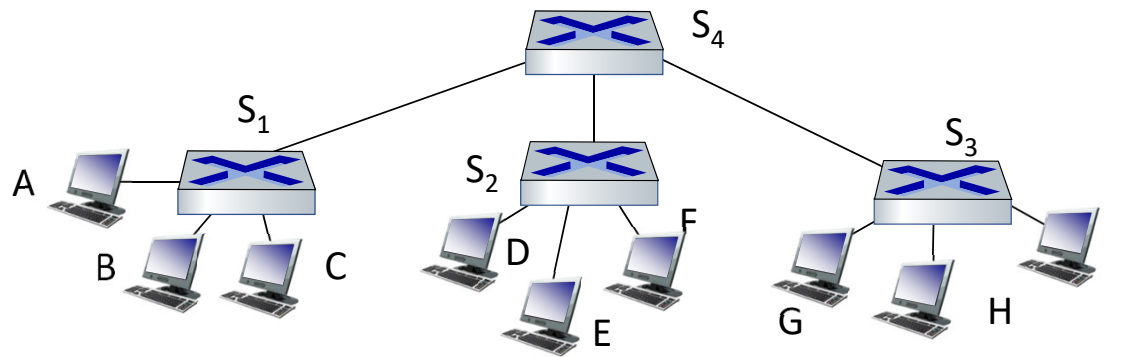


MAC addr	interface	TTL
A	1	60
A'	4	60

*switch table
(initially empty)*

Interconnecting switches

self-learning switches can be connected together:

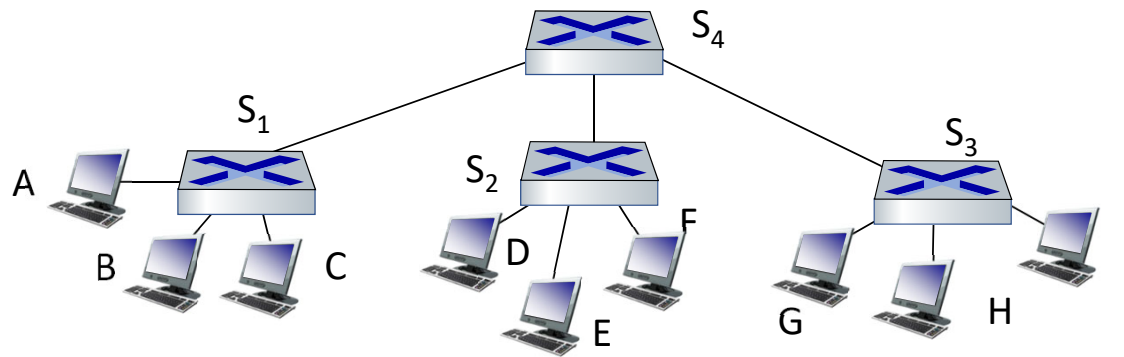


Q: sending from A to G - how does S₁ know to forward frame destined to G via S₄ and S₃?

- A: self learning! (works exactly the same as in single-switch case!)

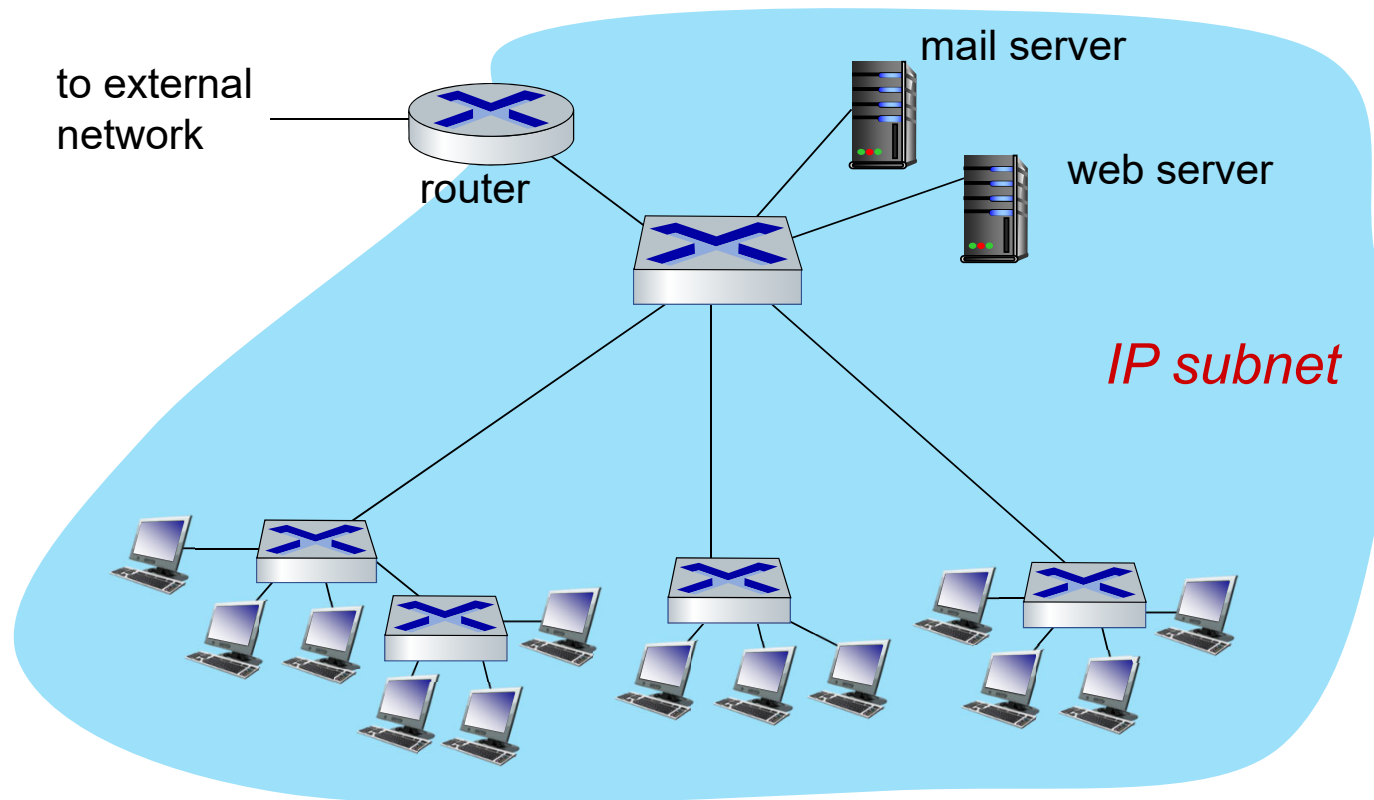
Self-learning multi-switch example

Suppose C sends frame to I, I responds to C



Q: show switch tables and packet forwarding in S₁, S₂, S₃, S₄

Small institutional network



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

