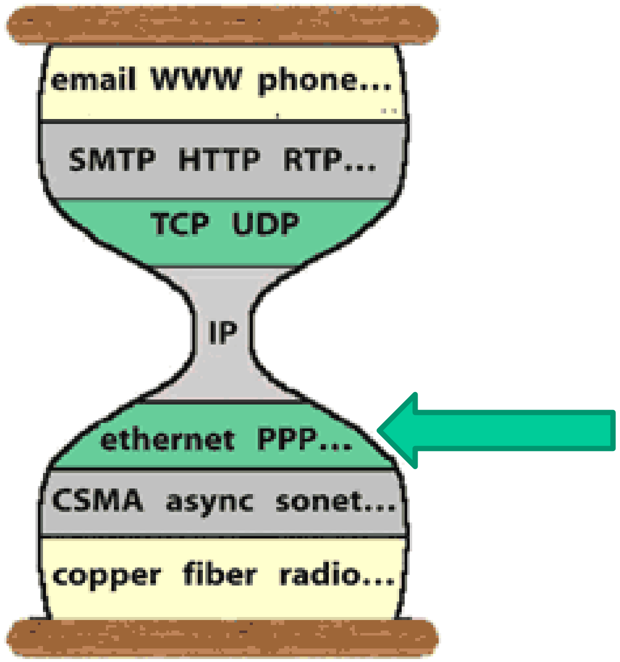
Lecture 16: LANs and Switches



6.1 Introduction, services

6.2 Error detection, correction

6.3 Multiple access protocols

6.4 LANs

- **6.4.1 Addressing, ARP**
- 6.4.2 Ethernet
- **6.4.3 Switches**
  - VLANS

6.5 Link virtualization: MPLS

6.6 Data center networking

6.7 A day in the life of a web request

MAC addresses

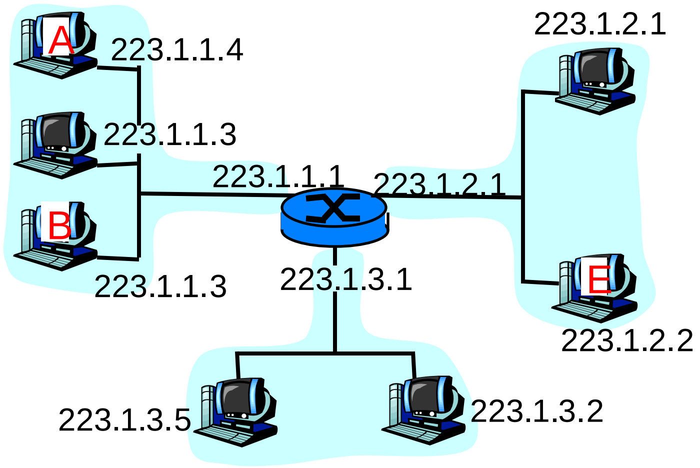
- 32-bit IP address:
  - *network-layer* address for interface
  - used for layer 3 (network layer) forwarding
  - e.g.: 128.119.40.136
- MAC (or LAN or physical or Ethernet) address:
  - function: used “locally” to get frame from one interface to another physically-connected interface (same subnet,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

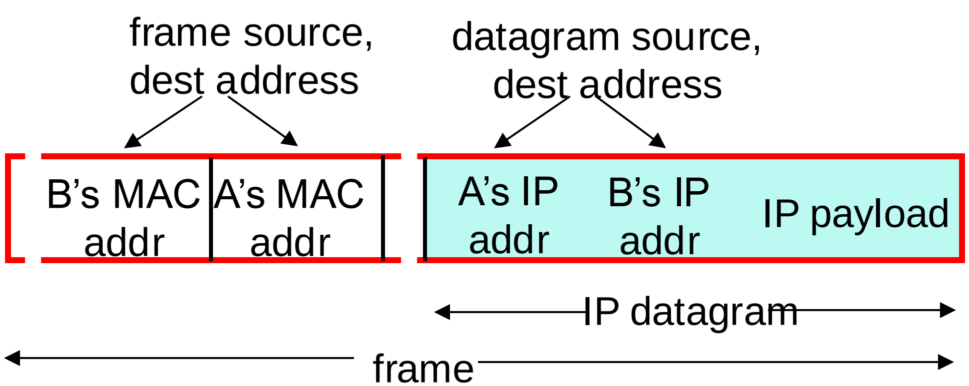
Connecting Link and IP layers

When node A sends IP packet to B:

- A looks up IP address of B, find B is on the same subnet as A
  - Apply subnet mask to both source and destination IP addresses: do A and B have the same subnet ID?
    - Yes: they are on the same subnet
    - No: they must communicate via router
- A sends to B using link layer protocol
  - Put IP packet inside a link layer frame

Subnet mask: 255.255.255.0

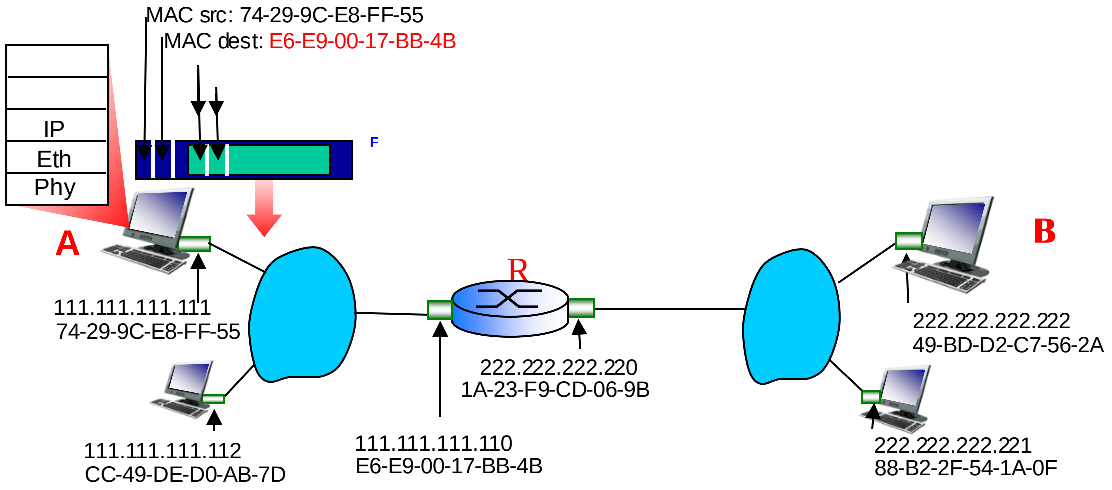




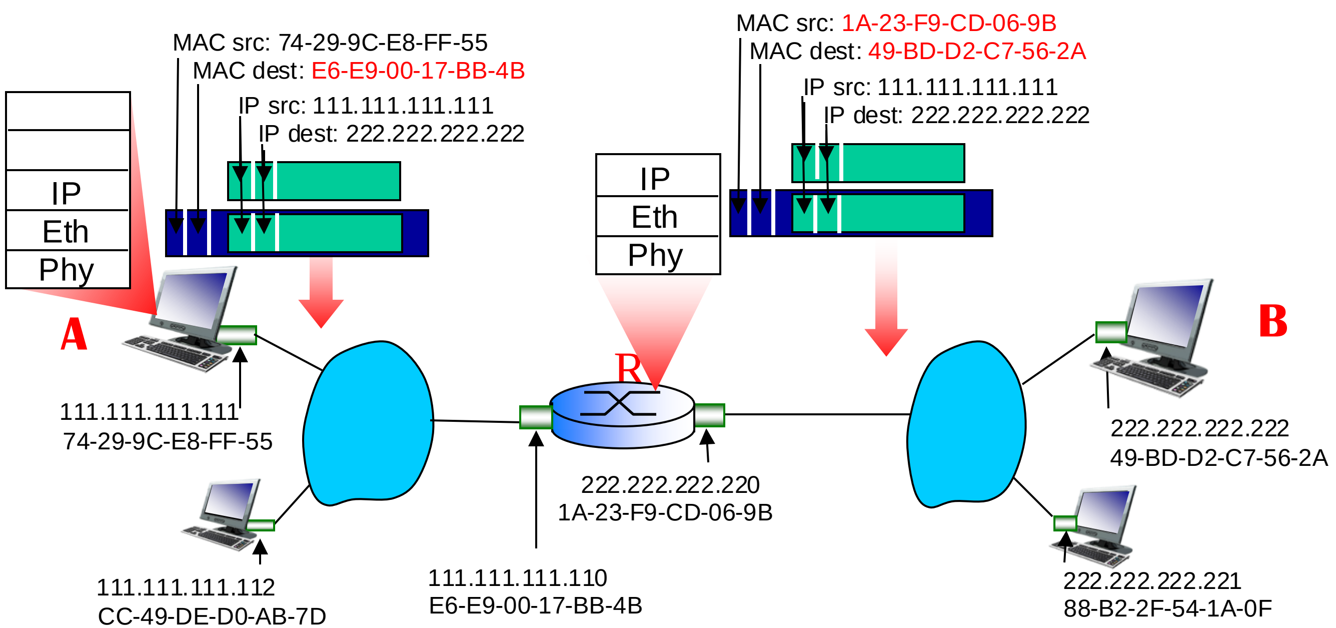
Given B's IP address, how to find B's MAC address?

Sending packets: $IP^A \rightarrow IP^B$

- Host A first uses subnet mask to find out whether Host B is on the same subnet
- If **yes**, packet can be sent directly: Lookup MAC for destination IP
- If **no**, packet should be set to the default router: Lookup MAC for the router's IP
- Create frame with the found MAC and original IP packet as a payload, send it



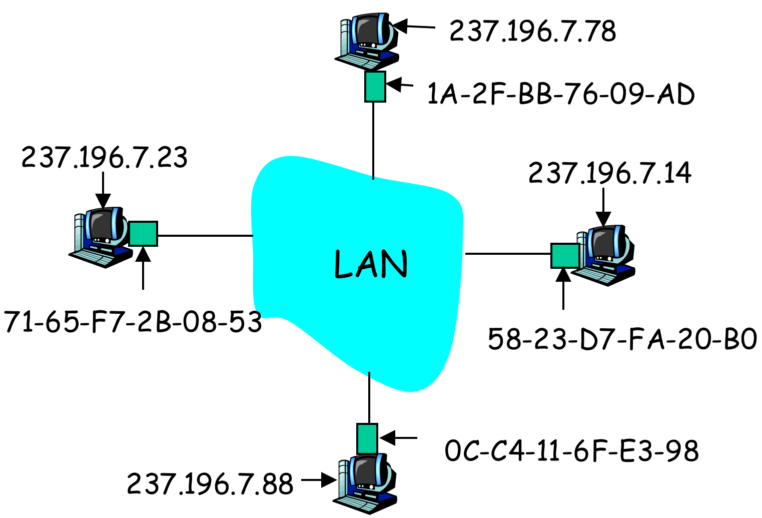
- Router or node receives the frame: Removes Ethernet header, finds IP destination address
- If IP is "self", deliver to transport, which will deliver to the app
- If IP is not self and node is router, repeat the previous steps (lookup routing table, lookup MAC, ...)



ARP: Address Resolution Protocol

- Every IP node (host & router) on LAN runs ARP to build an ARP table
- Each table entry: < IP address; MAC address; TTL>
  - Every time an entry is looked up, reset the TTL value
- *Soft-state* design: information deletes itself after certain time *unless being refreshed*

Nodes create their ARP tables without intervention from network administrator



ARP protocol in action: send ARP request

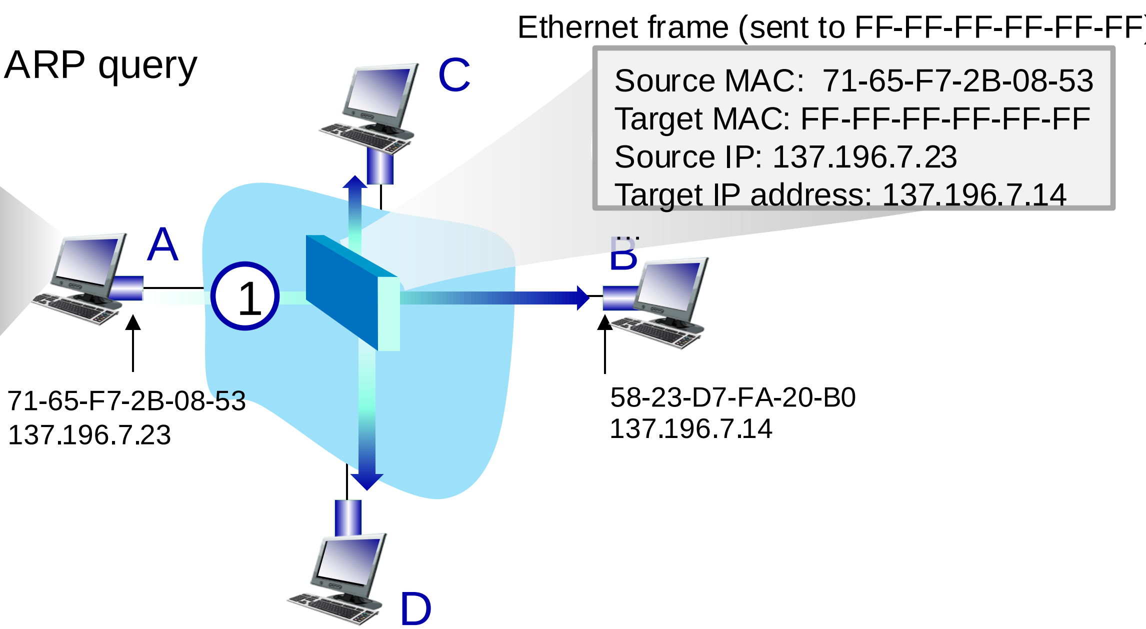
example: A wants to send datagram to B

- assume that A knows B’s IP address
- B’s MAC address not in A’s ARP table, so A uses ARP to find B’s MAC address

1 A broadcasts ARP query, containing B's IP addr
- 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: send ARP reply

example: A wants to send datagram to B

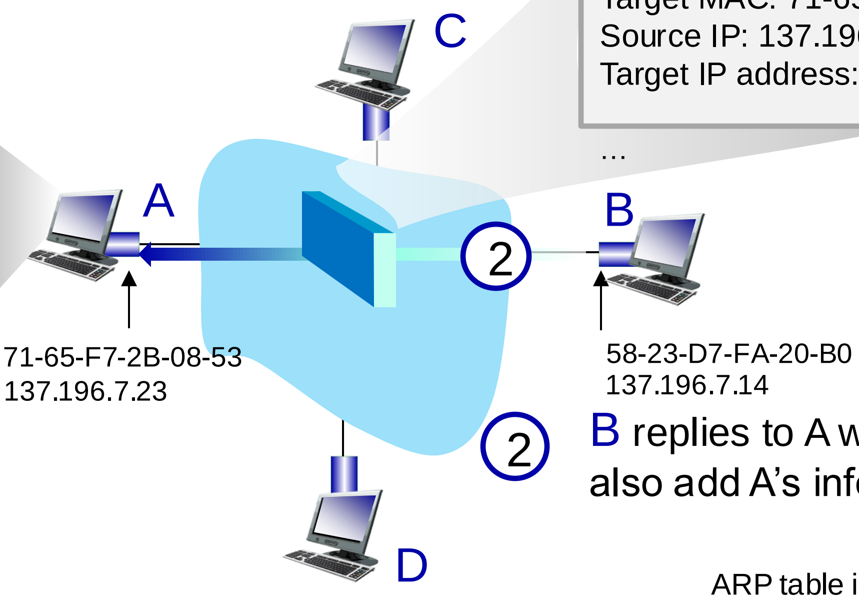
- assume that A knows B's IP address
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address

ARP reply Ethernet frame (sent to 71-65-F7-2B-08-53)

Source MAC address: 58-23-D7-FA-20-B0
Target MAC: 71-65-F7-2B-08-53
Source IP: 137.196.7.14
Target IP address: 137.196.7.23
...

ARP table in A

IP addr	MAC addr	TTL



2 B replies to A with ARP response, also add A's info to its ARP table

ARP table in B

IP addr	MAC addr	TTL
137.196.7.23	71-65-F7-2B-08-53	500

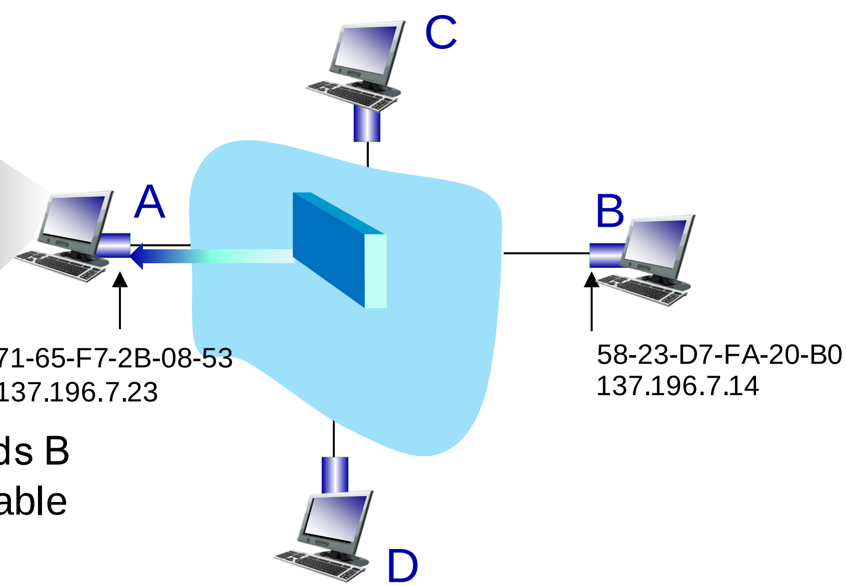
ARP protocol in action: add entry to ARP table

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 table in A

IP addr	MAC addr	TTL
137.196.7.14	58-23-D7-FA-20-B0	500

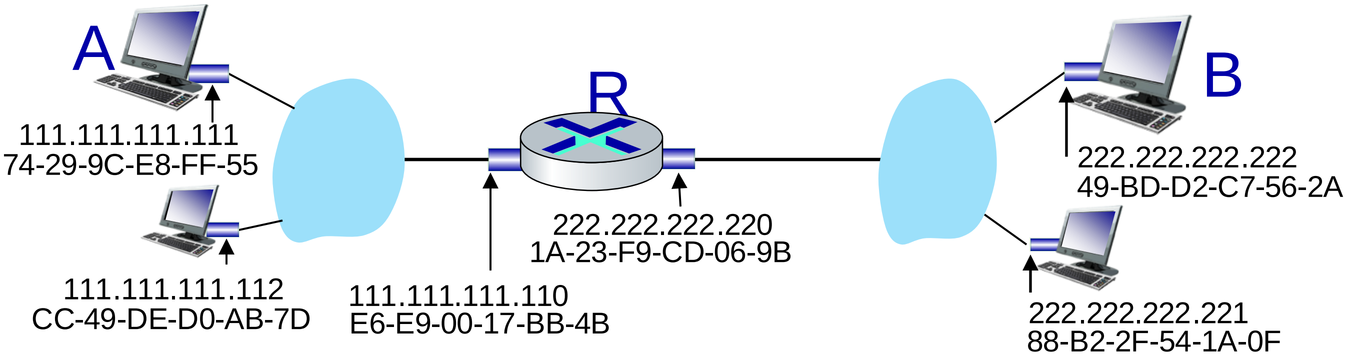


3 A receives B's reply, adds B entry into its local ARP table

Routing to another subnet: addressing

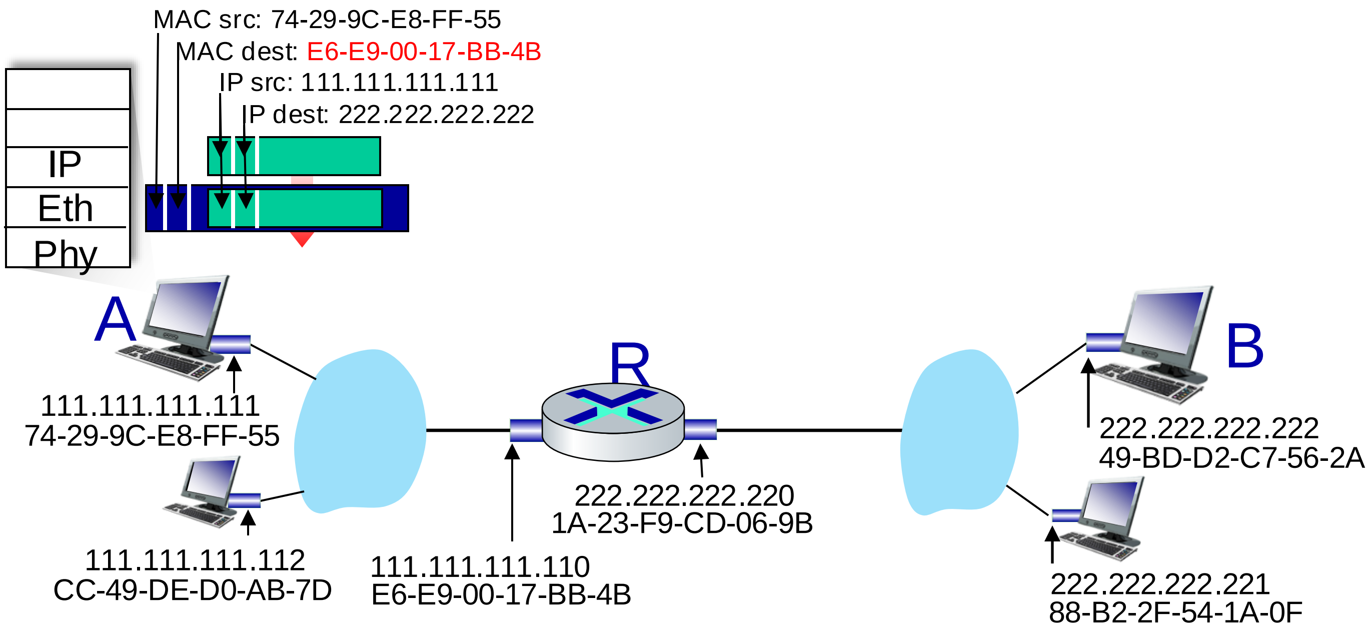
example: A wants to send datagram to B via R

- 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?)



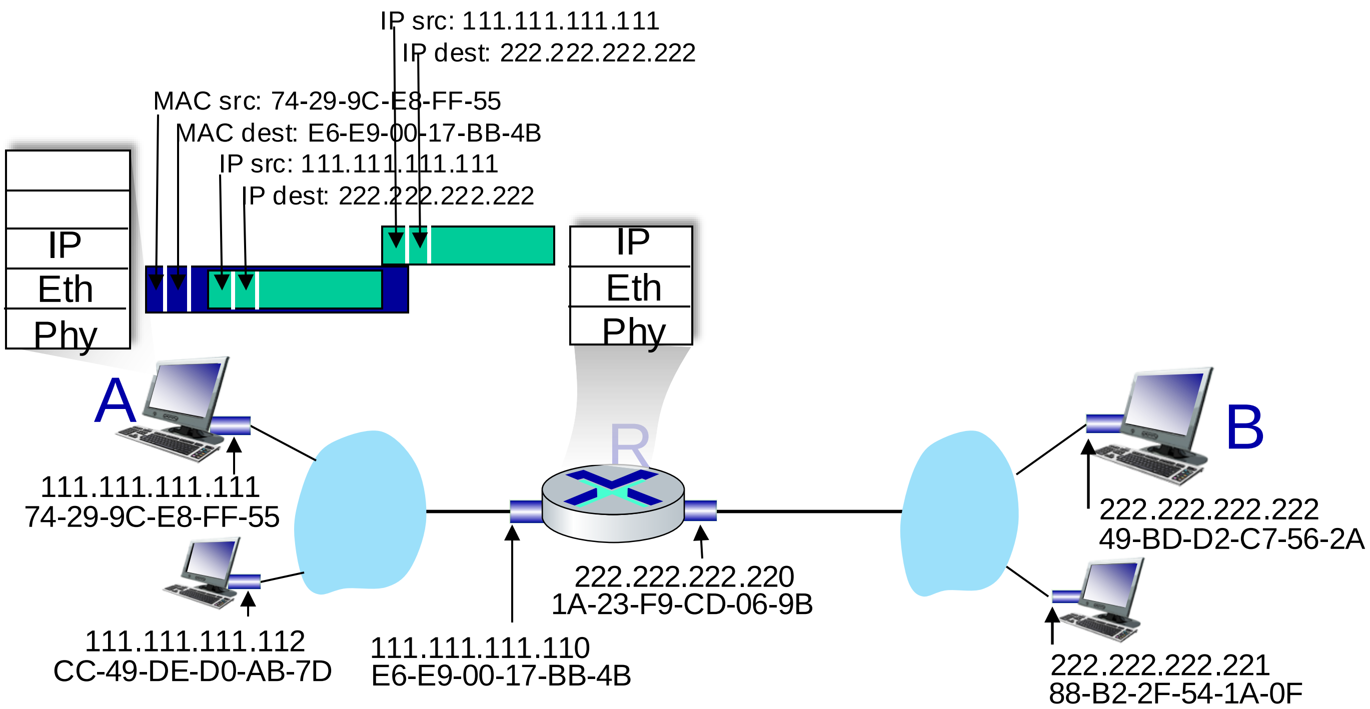
Routing to another subnet: addressing

- 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



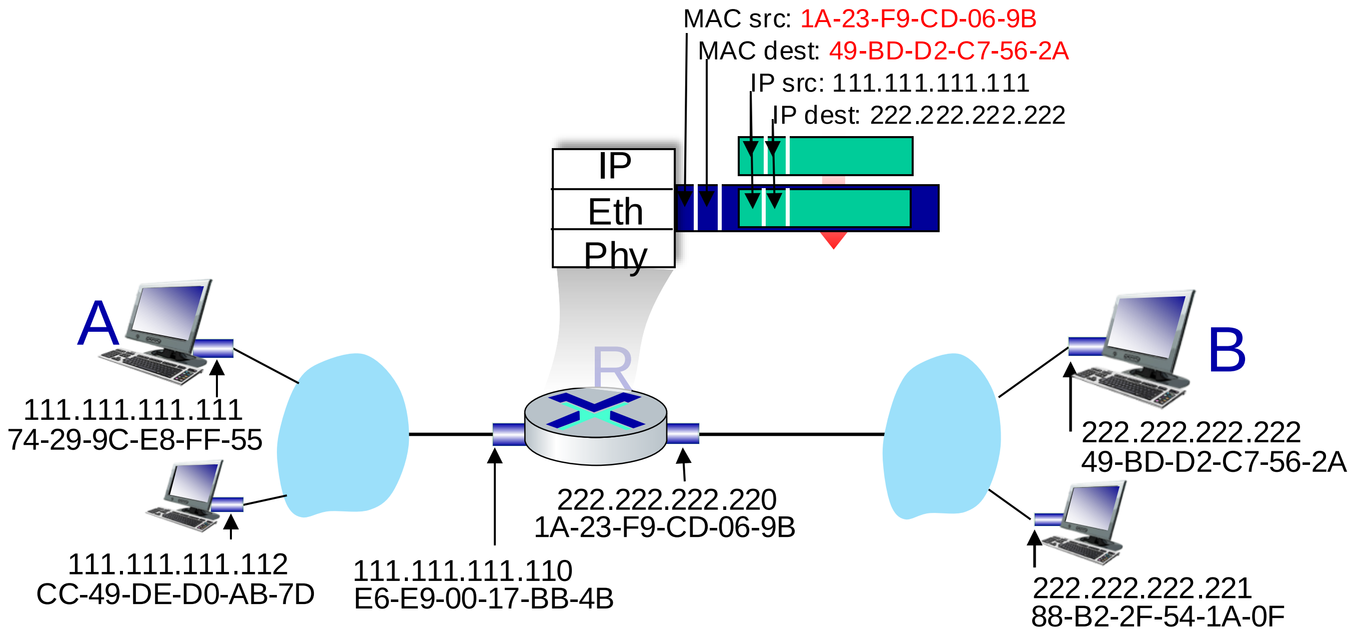
Routing to another subnet: addressing

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



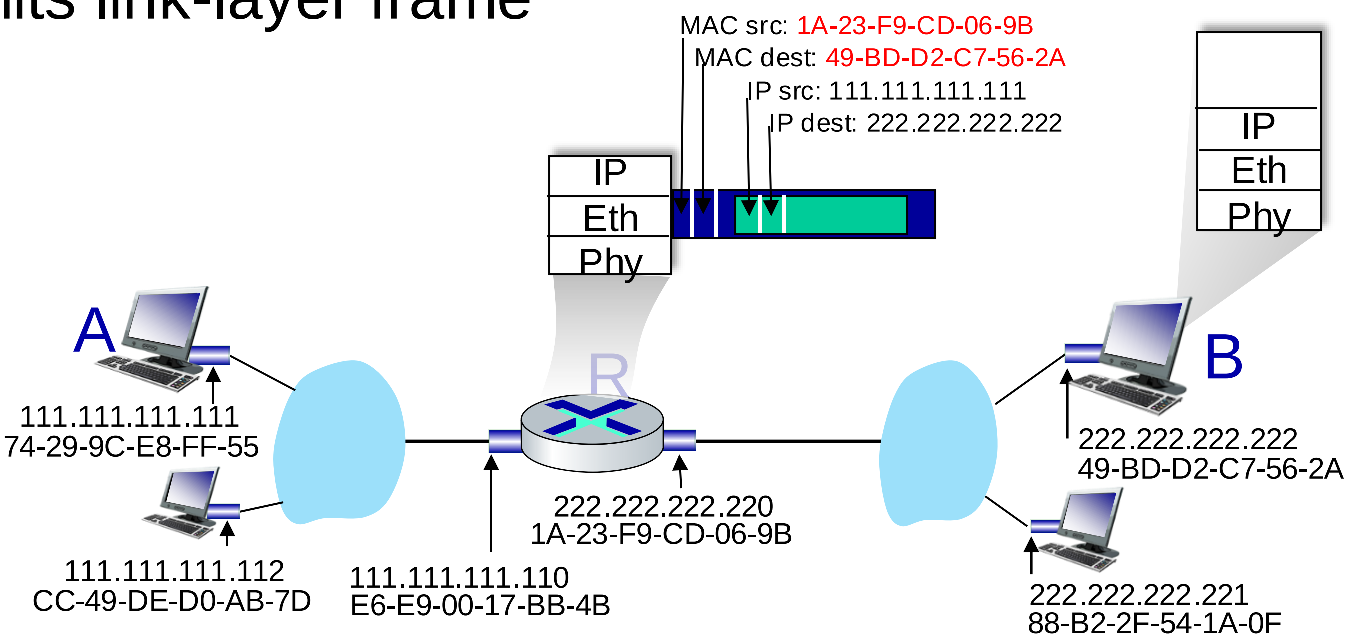
Routing to another subnet: addressing

- 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



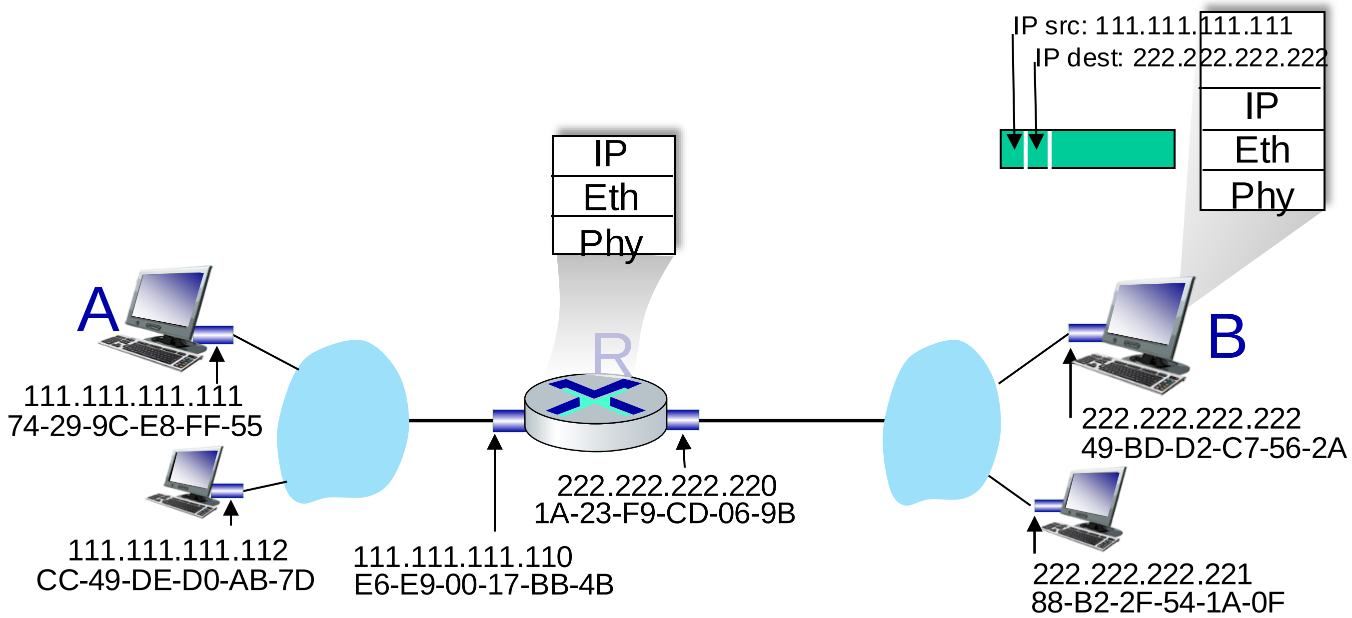
Routing to another subnet: addressing

- 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



Address Resolution Protocol: summary

- Node-A knows node-B's IP address: *broadcast* an ARP query msg which contains B's IP address
  - Destination MAC address: FF-FF-FF-FF-FF-FF
    - Received by all IP nodes on the same LAN
- B recognizes its own IP address in the ARP query, sends a reply:
  - Source MAC: B’s MAC address
  - Destination MAC: A’s MAC address
- As the result of ARP query-reply exchange, both A and B learned each other’s MAC address

review

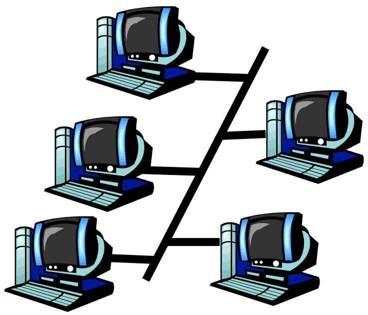
Ethernet CSMA/CD Algorithm

1. NIC receives datagram from network layer, creates frame
2. If channel idle: starts frame transmission. If channel busy, waits until channel idle, then transmits ("*1-persistent*")
3. If entire frame sent without detecting collision done
4. If detects collision while transmitting, aborts transmission, sends *jam signal* for a short time period
5. After aborting, NIC enters binary exponential backoff:
   - after n_{th} collision, NIC chooses a value K at random from

     $\{0, 1, 2, \ldots, 2^n - 1\}$

     - NIC waits K slots, returns to Step 2

       (1 time slot= transmission time for 512 bits)



Shared cable Ethernet

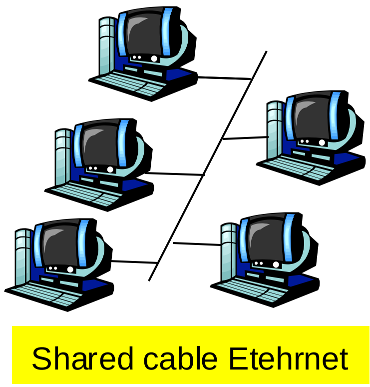
Ethernet: from shared cable to switches

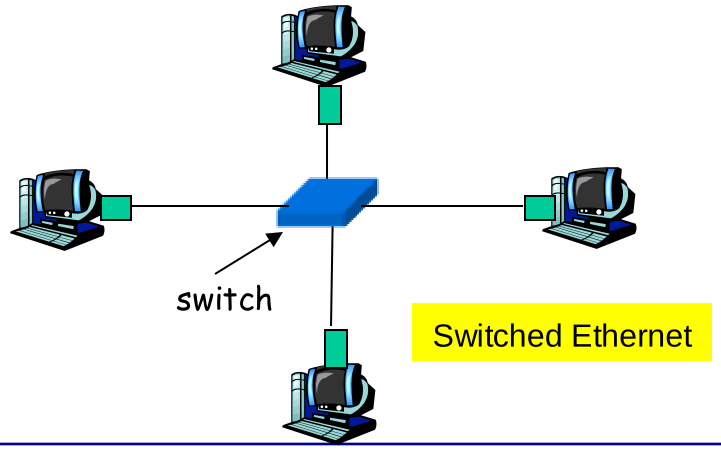
- Kept up with speed race: 10 Mbps – 100 Gbps
- CSMA/CD efficiency
  - T_{prop} = maximum propagation delay between any 2 nodes
  - T_{trans} = time to transmit a maximum-sized frame

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

T_{prop} stays the same, T_{trans} went *down* with time (speed goes up) → efficiency goes down

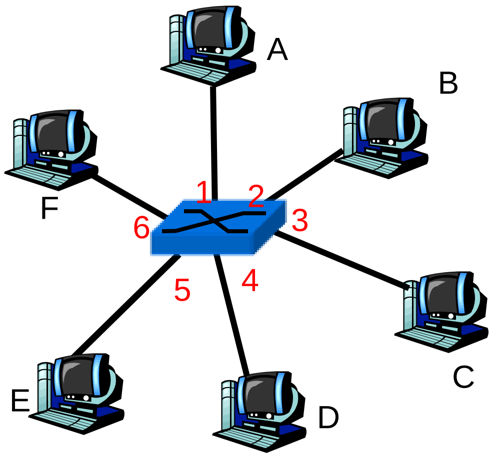
- Moved to switched Ethernet
  - The same Ethernet protocol runs on each wire





Ethernet switch

- ***A link layer device***: behave as a host on each connected LAN
  - speak Ethernet protocol at each interface
  - *transparent*: hosts are unaware of presence of switches
- Store-and-forward: examine each incoming frame's MAC address, forward to the destination LAN if dest. host is on a different LAN
  - A→D and B→E can go simultaneously
  - buffer frames temporarily if next LAN busy
- *NO configuration needed:* Plug-and-play

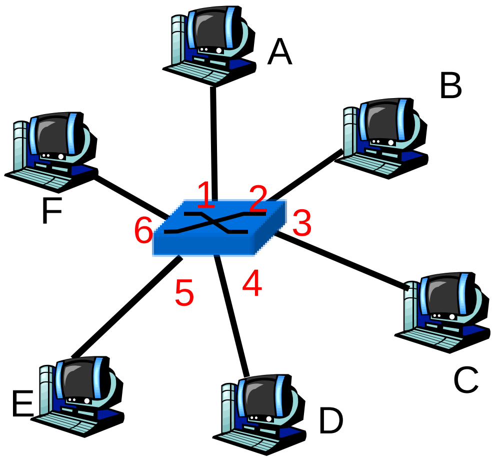


Switch needs a forwarding table

Q1: how does switch know D is reachable via interface 4, E is reachable via interface 5?

A1: each switch has a forwarding table, each table entry contains:

- MAC address of host
- interface to reach that MAC address
- TTL (time to live, a count-down timer)



Q2: how are the entries in the forwarding table created and maintained?

Building a forwarding table by *self-learning*

- Each table entry contains: MAC address, Interface, TTL
  - When a timer expires, the associated entry is dropped
- When receive a data frame:

1. look at *source* MAC address:

   if a matching table entry is found for the source MAC
   then reset the TTL
   else add the source MAC to the forwarding table

2. look up destination MAC address:

   if a matching table entry is found for **the** destination MAC

   {if the frame arrives on the interface to be used for forwarding
   then drop frame
   else forward frame to the interface indicated by the entry
   }

   else flood /* forward to all other interfaces except the arriving interface */

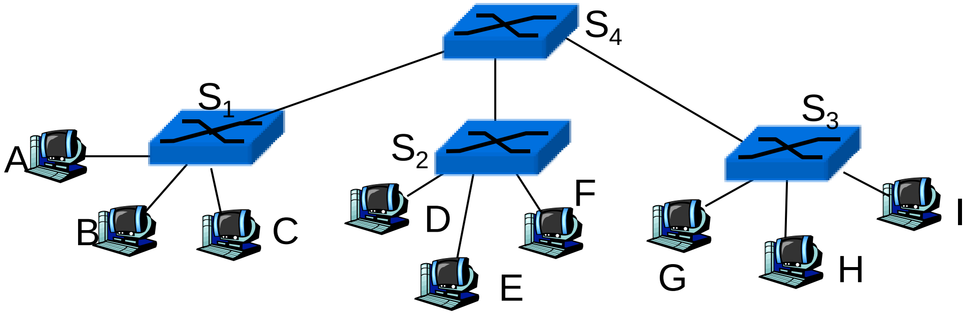
Interconnecting switches

- Switches can be connected together

Q: sending from A to F -- how does S_1 know to forward frame destined to F via S_4 and S_2?

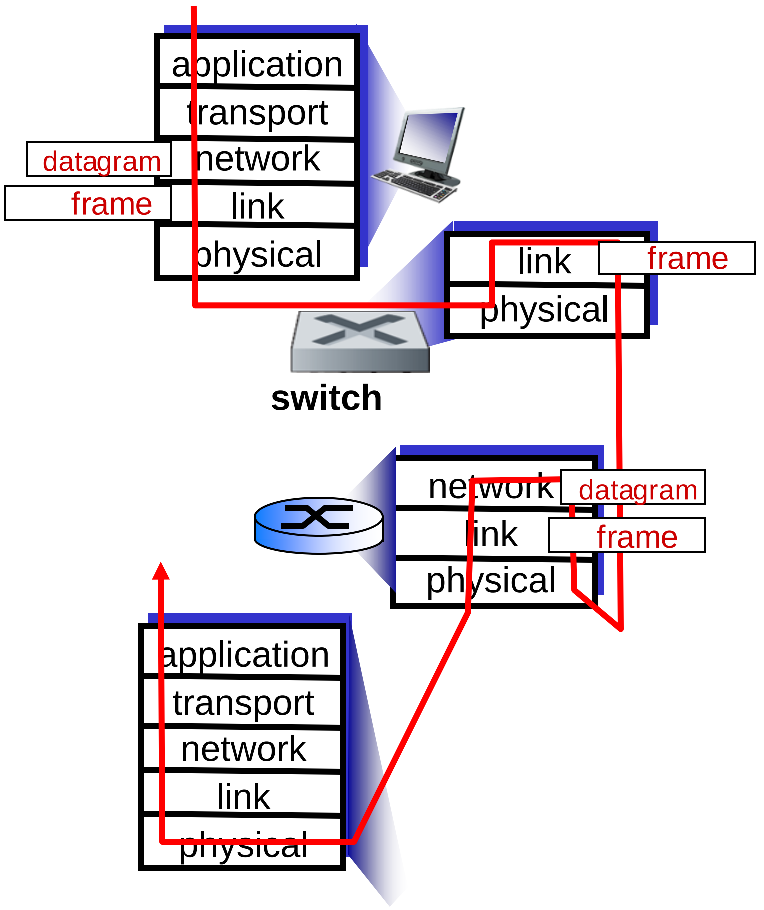
A: self learning! (works exactly the same as in single-switch case)

Under the assumption that the connected network graph is a *tree*



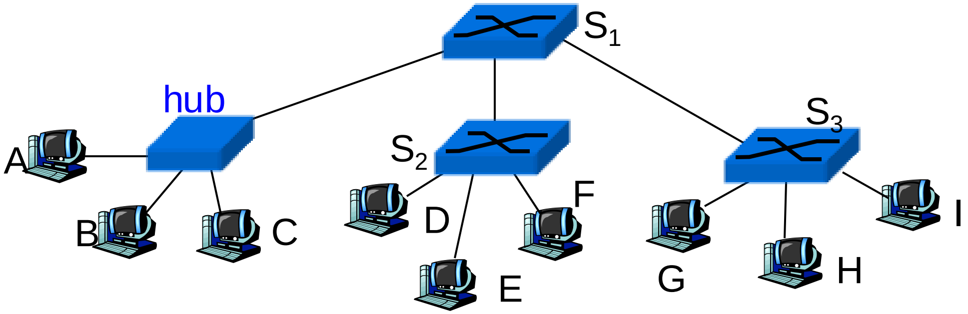
Switches vs. Routers

- Both are store-and-forward devices
  - Routers: network layer devices (examine IP headers)
  - Switches: link layer devices (examine Ethernet headers)
- Build forwarding table
  - Routers run routing protocols
  - Switches implement self-learning algorithms



Switches vs. Hubs

- Hub: pure signal amplification
  - When A sends a frame, B, C, and S1 hear it immediately
- Switches: buffer frame, then forward
  - If D sends a frame, E, F, and S1 cannot hear it immediately
  - Because of buffering, a switch can connect LANs of different speed

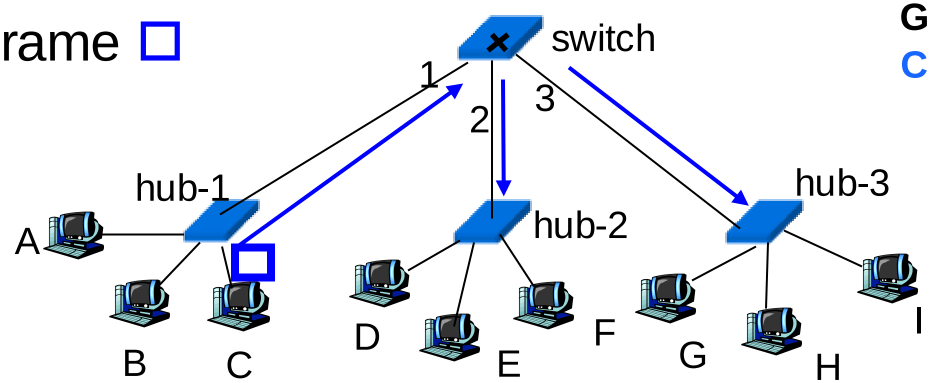


important

Ethernet switch example

- Suppose the switch's forwarding table already has entries for A, B, E, G
- C sends a data frame □ to D
  - A, B, and switch all hear frame □

address	interface
A	1
B	1
E	2
G	3
C	1



Switch receives frame □ from C

- Add to forwarding table: C is on interface 1
- D is not in table: forwards □ to interfaces 2 and 3

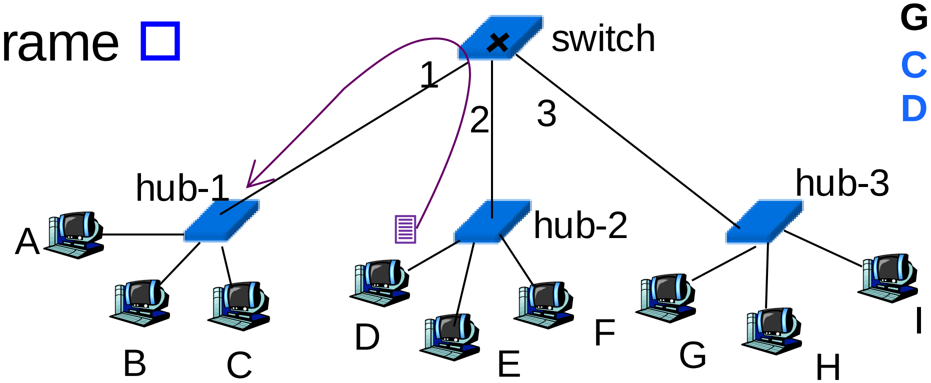
□ is received by D

important

Ethernet switch example

- Suppose the switch's forwarding table already has entries for A, B, E, G
- C sends a data frame □ to D
  - A, B, and switch all hear frame □

address	interface
A	1
B	1
E	2
G	3
C	1
D	2



Switch receives frame □ from C

- Add to forwarding table: C is on interface 1
- D is not in table: forwards □ to interfaces 2 and 3

□ is received by D

When D replies back with a frame ▤ to C

- E, F, and switch all hear frame ▤

Switch receives frame ▤ from D

- Add to forwarding table: D is on interface 2
- Forward ▤ to interface 1

Practice Question 1

- All switches start with empty forwarding table
- Consider the following frame transmission:

  $A \rightarrow B$, $B \rightarrow A$

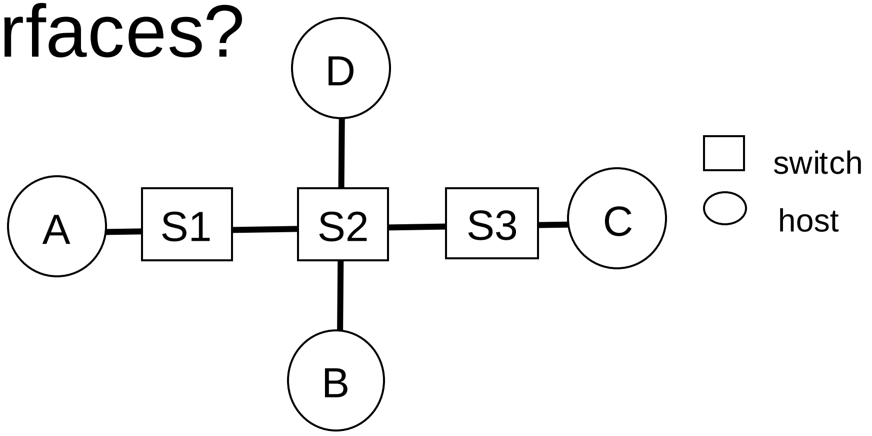
  $B \rightarrow C$, $C \rightarrow B$

  $D \rightarrow C$, $C \rightarrow D$

Q: How many times does S2 need to flood the data frame to all its interfaces?

A→B: S2 flood

B→C: S2 flood



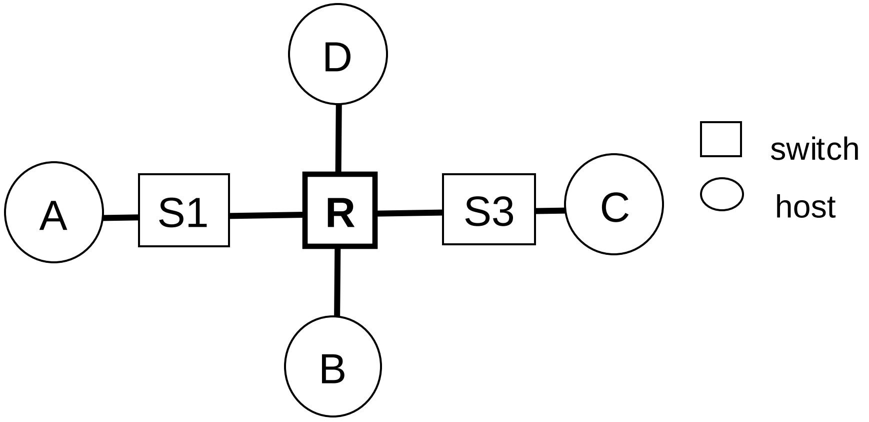
Practice Question 2

- All switches start with empty forwarding table
- All hosts are statically configured, start with empty ARP table, knowing others' IP
- Consider the following IP datagram transmission:

  $A \rightarrow B$, $B \rightarrow A$; $B \rightarrow C$, $C \rightarrow B$; $D \rightarrow C$, $C \rightarrow D$

Q: How many times does R need to flood frame to all its interfaces?

Q: S1, S3 table?



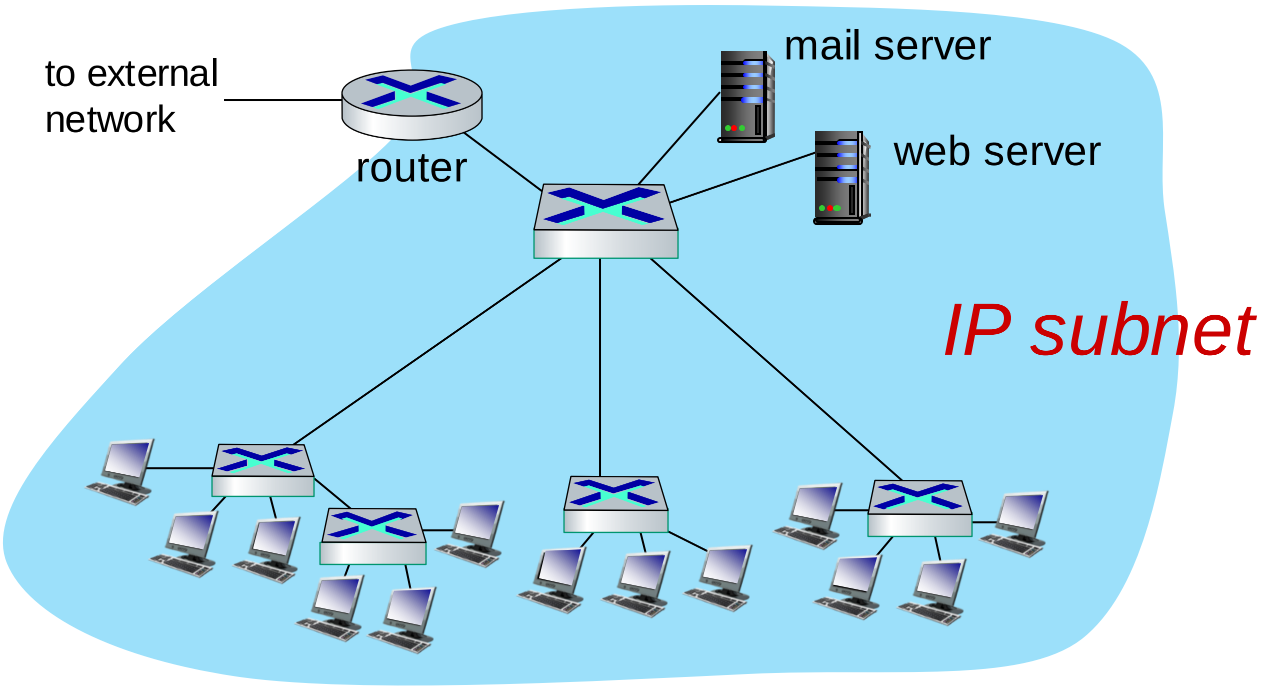
Switches: advantages and limitations

- Transparent: no change to hosts
- Isolates collision domains
  - resulting in higher total maximum throughput
- Can connect Ethernets of different speeds
  - because it is a store and forward device
- Constrained topology: ***tree only***
  - all inter-segment traffic concentrated on a single tree
  - (all multicast traffic forwarded to all LAN's)

Routers: advantages and limitations

- Support arbitrary topologies
- Require IP address configuration (not plug-and-play)
- More complex packet processing than switches
- Switches do well in small setting, routers are used in large networks

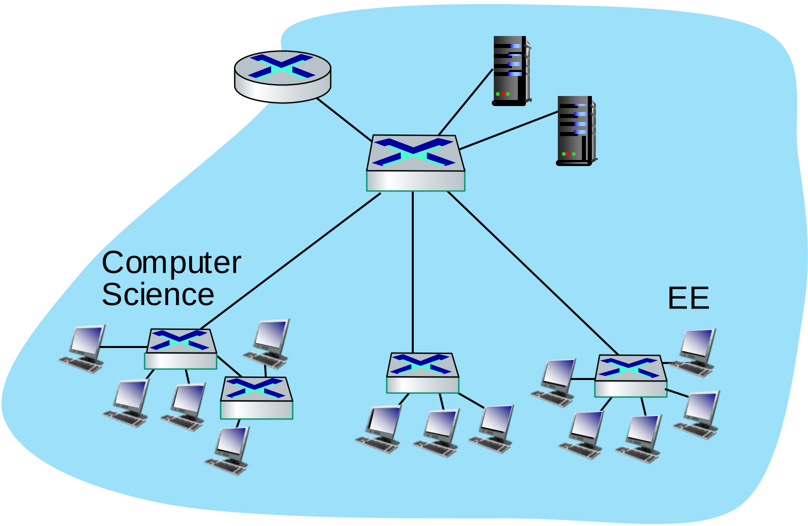
Small institutional network



Virtual LANs (VLANs): motivation

FYI

What happens as LAN sizes scale, users change location?



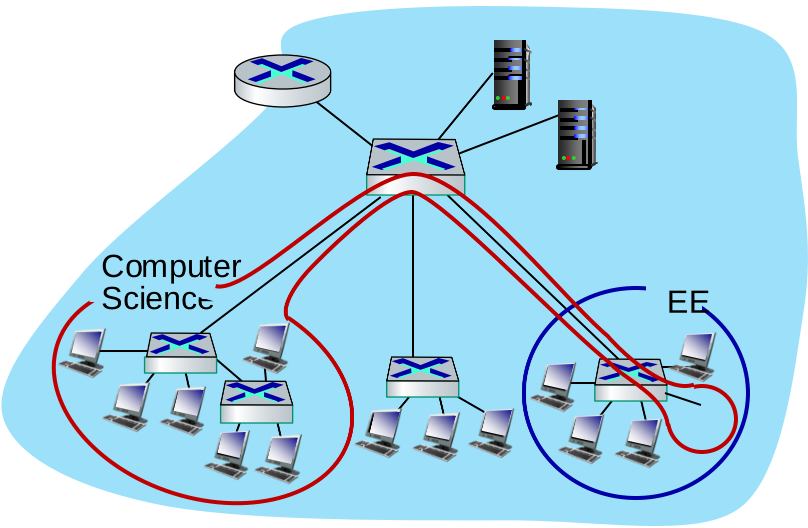
single broadcast domain:

- *scaling:* all layer-2 broadcast traffic (ARP, DHCP, unknown MAC) must cross entire LAN
- efficiency, security, privacy, efficiency issues

Virtual LANs (VLANs): motivation

FYI

What happens as LAN sizes scale, users change location?



single broadcast domain:

- *scaling:* all layer-2 broadcast traffic (ARP, DHCP, unknown MAC) must cross entire LAN
- efficiency, security, privacy, efficiency issues

administrative issues:

- CS user moves office to EE - *physically* attached to EE switch, but wants to remain *logically* attached to CS switch

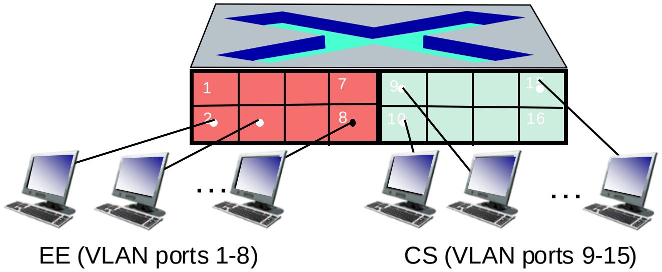
Port-based VLANs

FYI

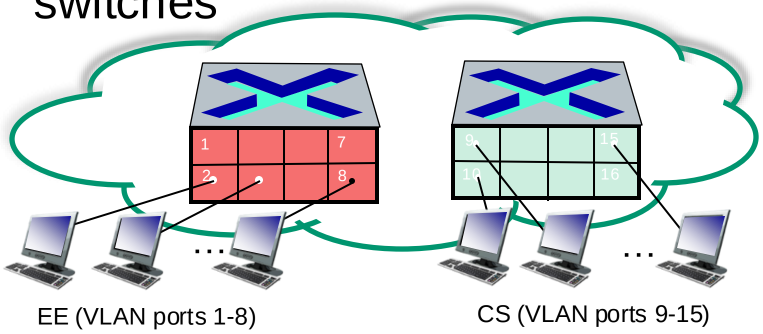
Virtual Local Area Network (VLAN)

switch(es) supporting VLAN capabilities can be configured to define multiple *virtual* LANS over single physical LAN infrastructure.

port-based VLAN: switch ports grouped (by switch management software) so that *single* physical switch ……



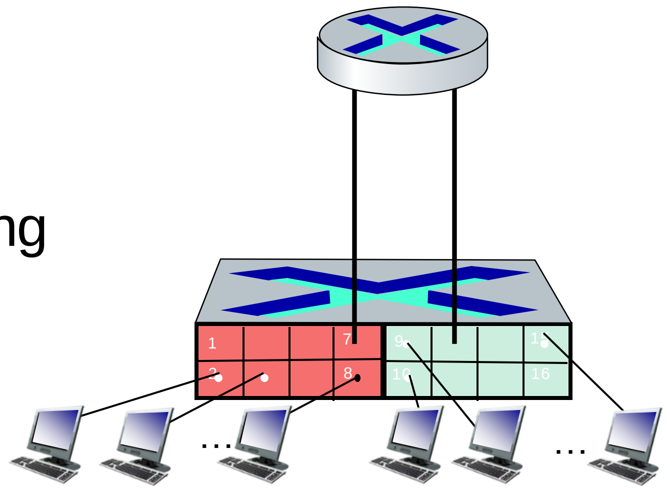
… operates as multiple virtual switches



Port-based VLANs

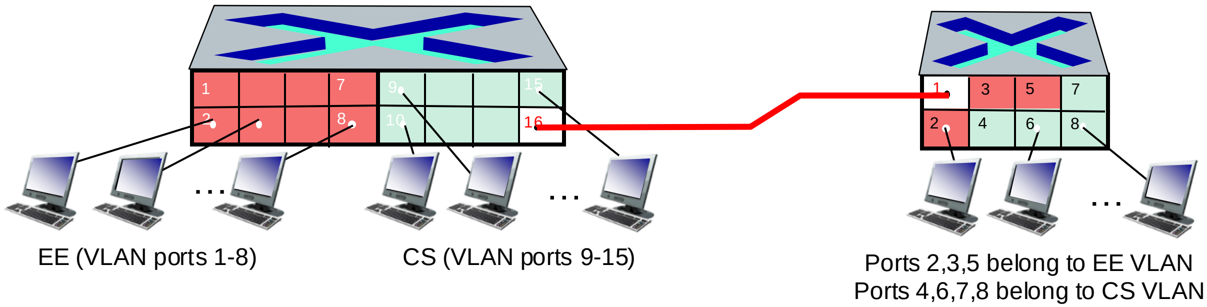
FYI

- Traffic isolation: frames to/from ports 1-8 can *only* reach ports 1-8
  - can also define VLAN based on MAC addresses of endpoints, rather than switch port
- Dynamic membership: ports can be dynamically assigned among VLANs
- Forwarding between VLANS: done via routing (just as with separate switches)
  - in practice vendors sell combined switches plus routers



VLANS spanning multiple switches

FYI

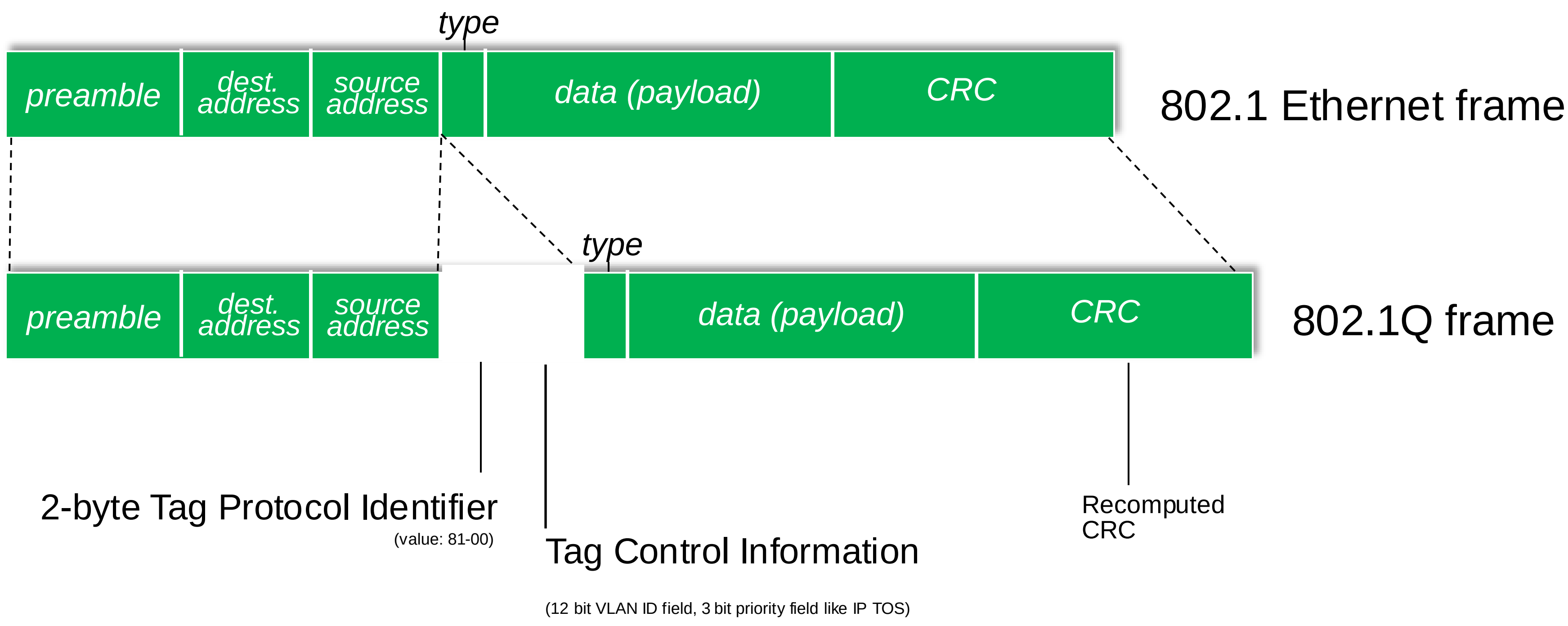


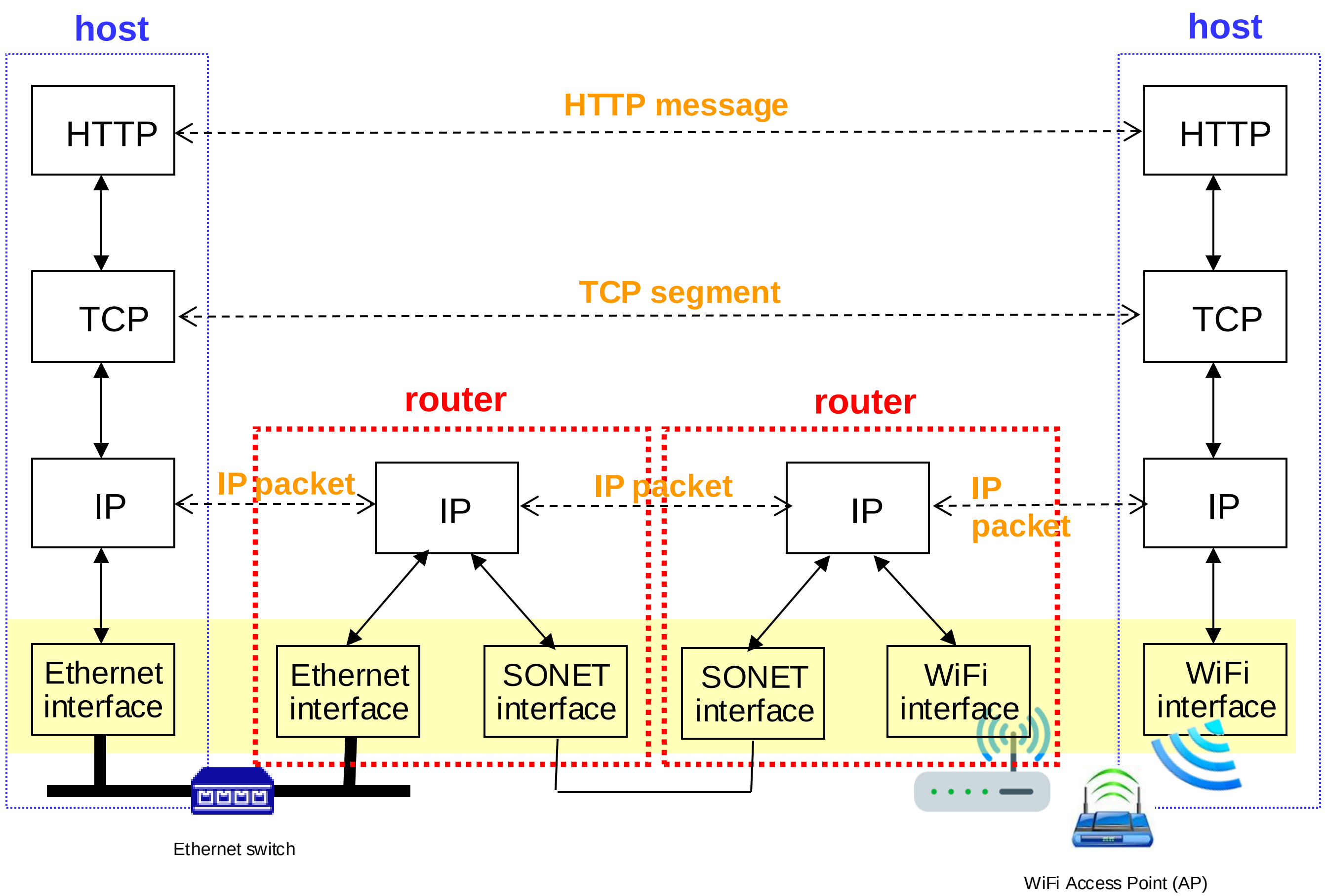
trunk port: carries frames between VLANS defined over multiple physical switches

- frames forwarded within VLAN between switches can't be vanilla 802.1 frames (must carry VLAN ID info)
- 802.1q protocol adds/removed additional header fields for frames forwarded between trunk ports

802.1Q VLAN frame format

FYI




host
host
HTTP message
HTTP
HTTP
TCP segment
TCP
TCP
router
router
IP packet
IP packet
IP packet
IP
IP
IP
IP
Ethernet interface
Ethernet interface
SONET interface
SONET interface
WiFi interface
WiFi interface
Ethernet switch
WiFi Access Point (AP)