DIPARTIMENTO DI INFORMATICA
Via Salaria 113 - 00198, Roma

SAPIENZA UNIVERSITÀ DI ROMA

# Chapter 5
# Data Link Layer

Reti di Elaboratori
Corso di Laurea in Informatica
Università degli Studi di Roma "La Sapienza"
Canale A-L
Prof.ssa Chiara Petrioli

Parte di queste slide sono state prese dal materiale associato al libro *Computer Networking: A Top Down Approach* , 5th edition.

Thanks also to Antonio Capone, Politecnico di Milano, Giuseppe Bianchi and Francesco LoPresti, Un. di Roma Tor Vergata

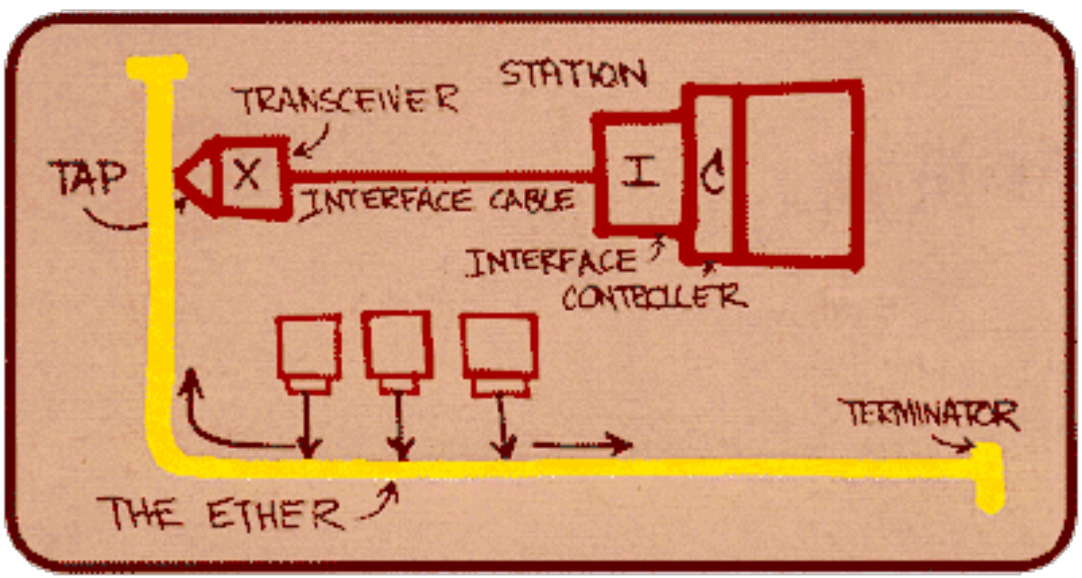
# Link Layer

- 5.1 Introduction and services
- 5.2 Error detection and correction
- 5.3Multiple access protocols
- 5.4 Link-Layer Addressing
- 5.5 Ethernet
- 5.6 Link-layer switches
- 5.7 PPP
- 5.8 Link virtualization: MPLS
- 5.9 A day in the life of a web request

# Ethernet

"dominant" wired LAN technology:

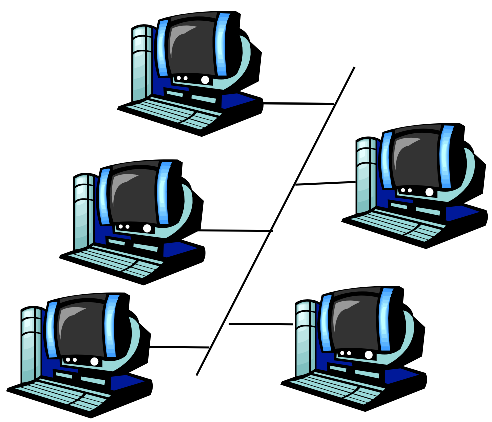
- cheap $20 for NIC
- first widely used LAN technology
- simpler, cheaper than token LANs and ATM
- kept up with speed race: 10 Mbps – 10 Gbps



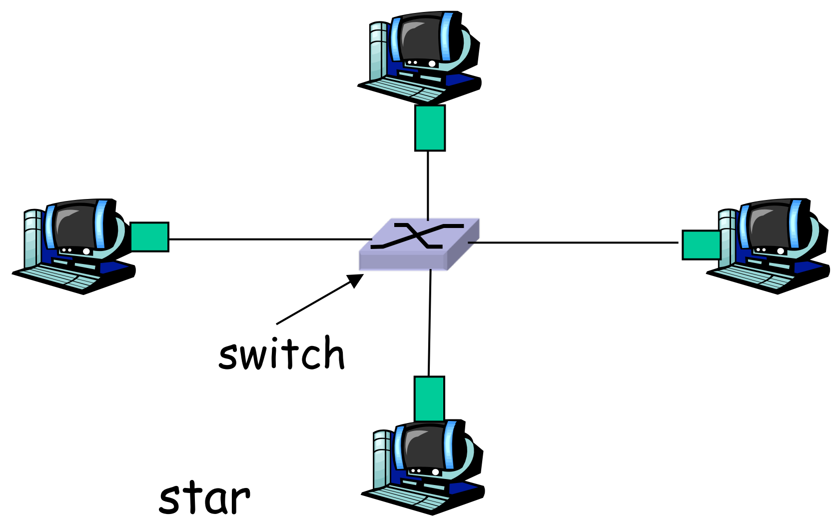

Metcalfe's Ethernet sketch

# Star topology

- bus topology popular through mid 90s
  - all nodes in same collision domain (can collide with each other)
- today: star topology prevails
  - active *switch* in center
  - each "spoke" runs a (separate) Ethernet protocol (nodes do not collide with each other)



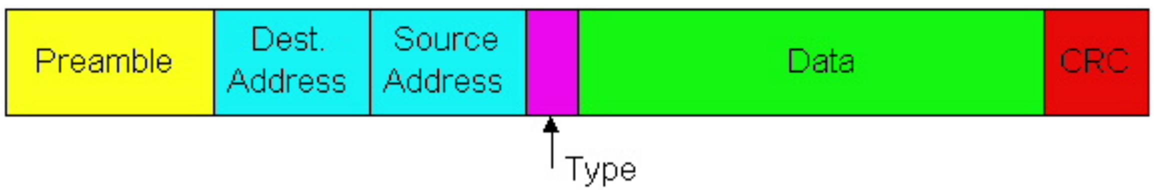
bus: coaxial cable




star

# Ethernet Frame Structure

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

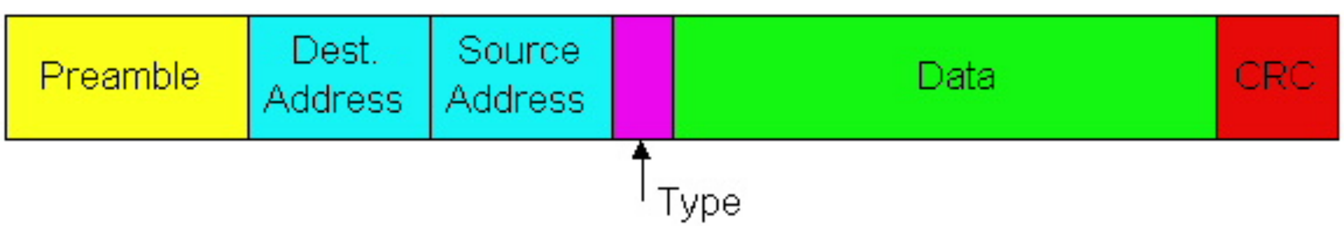



Preamble:

- 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 bytes
  - if adapter receives frame with matching destination address, or with broadcast address (eg 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:** checked at receiver, if error is detected, frame is dropped



# Ethernet: Unreliable, connectionless

- connectionless: No handshaking between sending and receiving NICs
- unreliable: receiving NIC doesn't send acks or nacks to sending NIC
  - stream of datagrams passed to network layer can have gaps (missing datagrams)
  - gaps will be filled if app is using TCP
  - otherwise, app will see gaps
- Ethernet's MAC protocol: unslotted CSMA/CD


5: DataLink Layer 5a-7

# 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 and sends jam signal
5. After aborting, NIC enters **exponential backoff**: after *m*th collision, NIC chooses *K* at random from $\{0,1,2,\ldots,2^m-1\}$. NIC waits $K \cdot 512$ bit times, returns to Step 2


5: DataLink Layer 5a-8

# Ethernet's CSMA/CD (more)

**Jam Signal:** make sure all other transmitters are aware of collision; 48 bits

**Bit time:** .1 microsec for 10 Mbps Ethernet ;
for K=1023, wait time is about 50 msec

**Exponential Backoff:**

- *Goal*: adapt retransmission attempts to estimated current load
  - heavy load: random wait will be longer
- first collision: choose K from {0,1}; delay is K· 512 bit transmission times
- after second collision: choose K from {0,1,2,3}…
- after ten collisions, choose K from {0,1,2,3,4,…,1023}


5: DataLink Layer 5a-9

# CSMA/CD efficiency

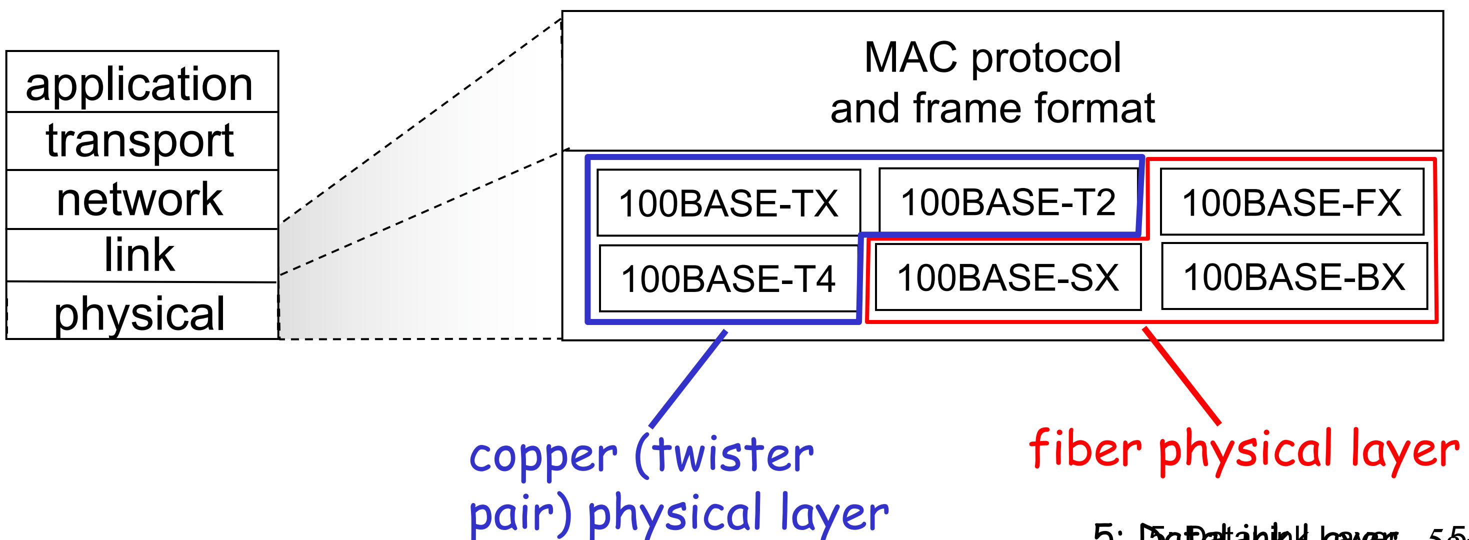
- $T_{prop}$ = max prop delay between 2 nodes in LAN
- $t_{trans}$ = time to transmit max-size frame

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

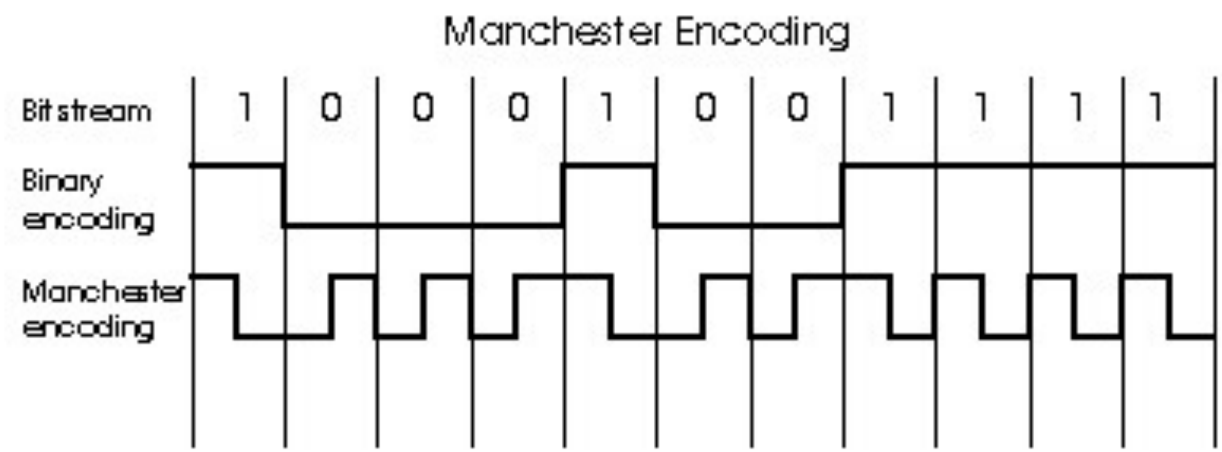
- efficiency goes to 1
  - as $t_{prop}$ goes to 0
  - as $t_{trans}$ goes to infinity
- better performance than ALOHA: and simple, cheap, decentralized!

# 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, 10G bps
  - different physical layer media: fiber, cable



# Manchester encoding




- used in 10BaseT
- each bit has a transition
- allows clocks in sending and receiving nodes to synchronize to each other
  - no need for a centralized, global clock among nodes!
- Hey, this is physical-layer stuff!


5: DataLink Layer 5-12

# Ethernet: some numbers..

- Slot time 512 bit times (di riferimento, la tras missione NON e' slottizzata!!)
- Interframegap 9.6 micros
- Number of times max for retransmitting a frame 16
- Backoff limit ($2^{\text{backoff limit}}$ indicates max length of the backoff interval): 10
- Jam size: 48 bits
- Max frame size: 1518 bytes
- Min frame size 64 bytes (512 bits)
- Address size: 48 bits

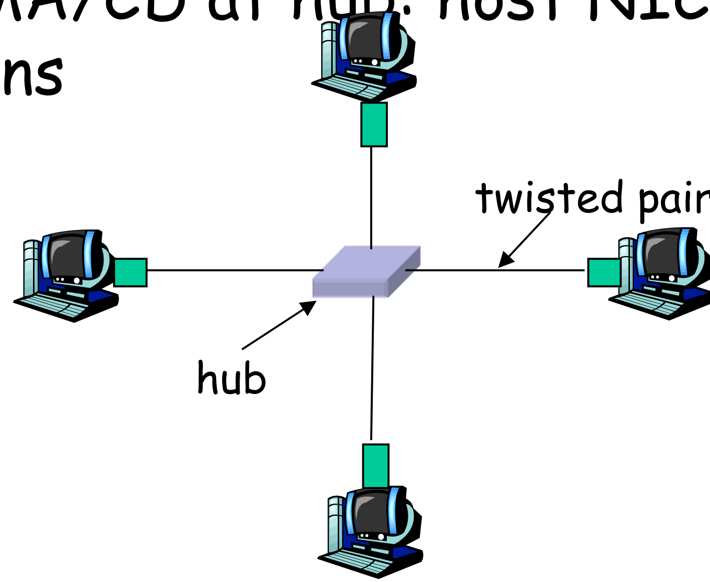
# Link Layer

- 5.1 Introduction and services
- 5.2 Error detection and correction
- 5.3 Multiple access protocols
- 5.4 Link-layer Addressing
- 5.5 Ethernet
- 5.6 Link-layer switches, LANs, VLANs
- 5.7 PPP
- 5.8 Link virtualization: MPLS
- 5.9 A day in the life of a web request

# Hubs

… physical-layer ("dumb") repeaters:

- bits coming in one link go out *all* other links at same rate
- all nodes connected to hub can collide with one another
- no frame buffering
- no CSMA/CD at hub: host NICs detect collisions





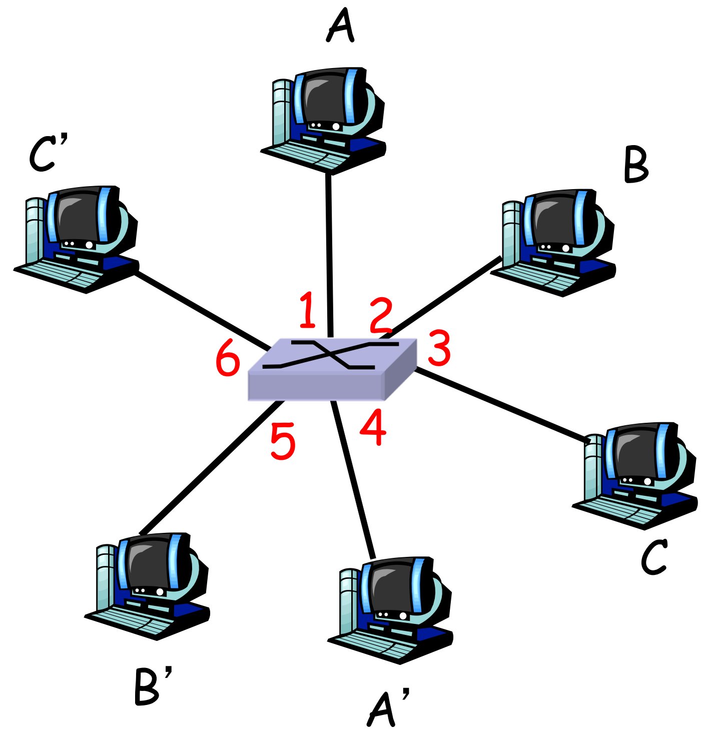
5: DataLink Layer 5-15

# Switch

- link-layer device: smarter than hubs, take *active* role
  - store, forward Ethernet frames
  - examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links
- *transparent*
  - hosts are unaware of presence of switches
- *plug-and-play, self-learning*
  - switches do not need to be configured

# Switch: allows *multiple* simultaneous transmissions

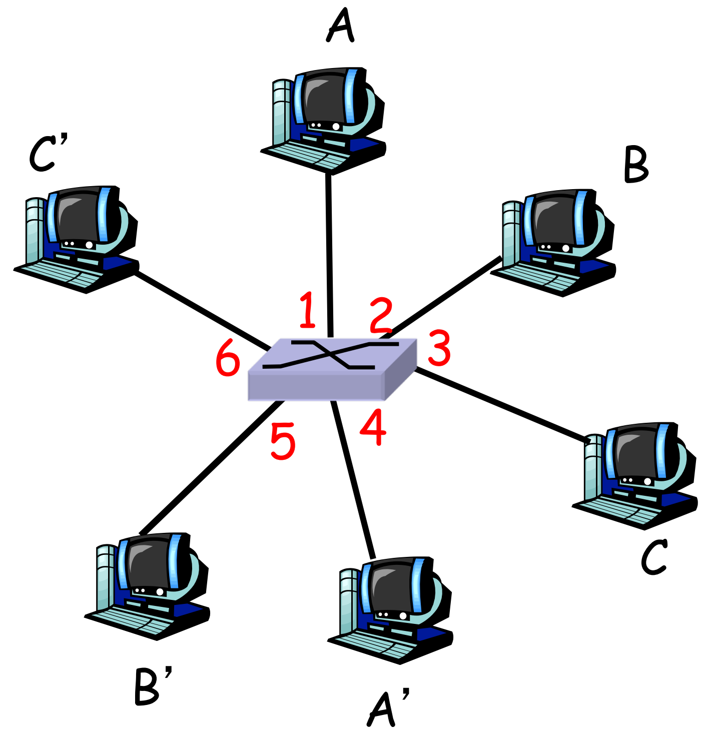
- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, but no collisions; full duplex
  - each link is its own collision domain
- *switching:* A-to-A' and B-to-B' simultaneously, without collisions
  - not possible with dumb hub




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

# Switch Table

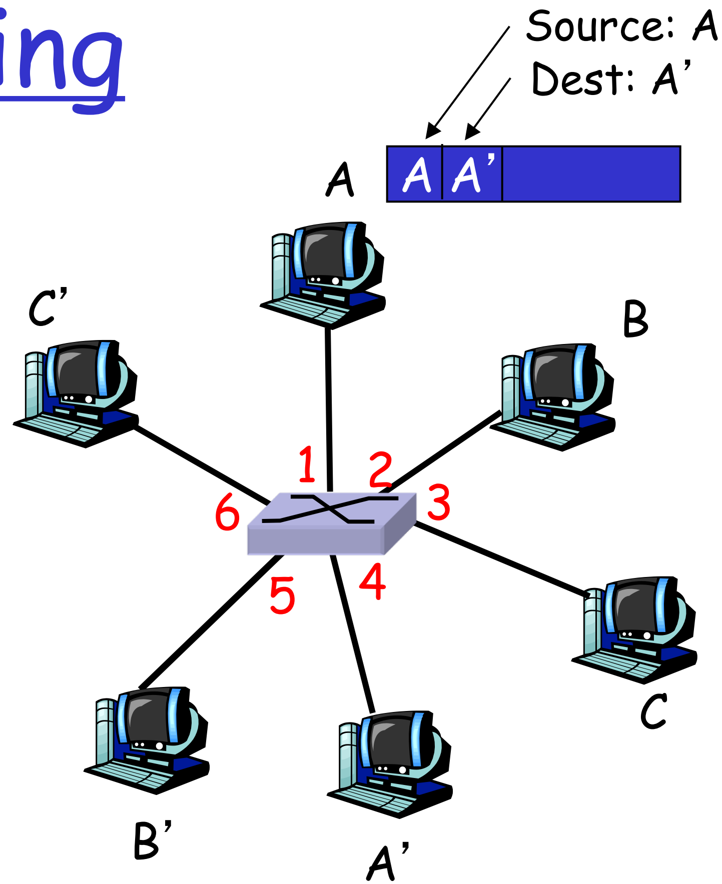
- *Q:* how does switch know that A' reachable via interface 4, B' reachable via interface 5?
- *A:* each switch has a switch table, each entry:
  - (MAC address of host, interface to reach host, time stamp)
- looks like a routing table!
- *Q:* how are entries created, maintained in switch table?
  - 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)*

# Switch: frame filtering/forwarding

When frame received:

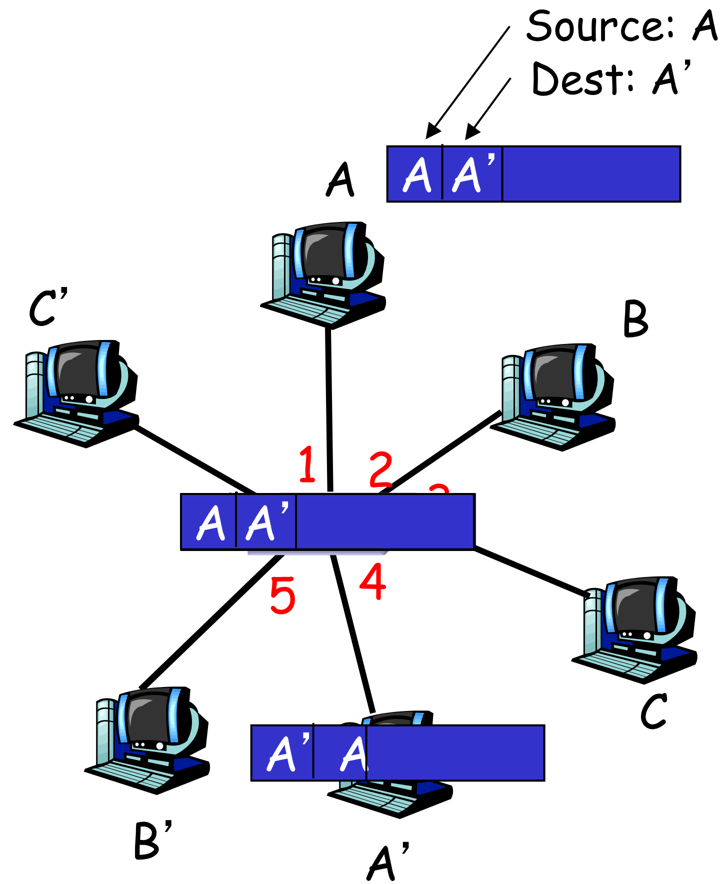
1. record link associated with sending host
2. index switch table using MAC dest address
3. **if** entry found for destination
   **then** {
   **if** dest on segment from which frame arrived
   **then** drop the frame
   **else** forward the frame on interface indicated
   }
   **else** flood

*forward on all but the interface on which the frame arrived*


5a-20

# Self-learning, forwarding: example

- frame destination unknown: *flood*
- destination A location known: *selective send*

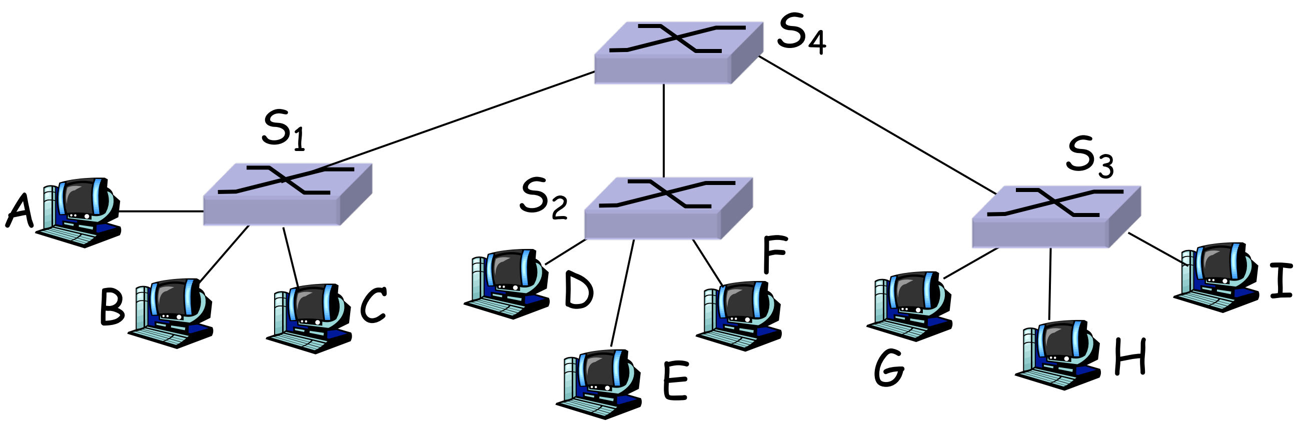



| MAC addr | interface | TTL |
|---|---|---|
| *A* | *1* | *60* |
| *A'* | *4* | *60* |

*Switch table (initially empty)*

# Interconnecting switches

- switches can be connected together

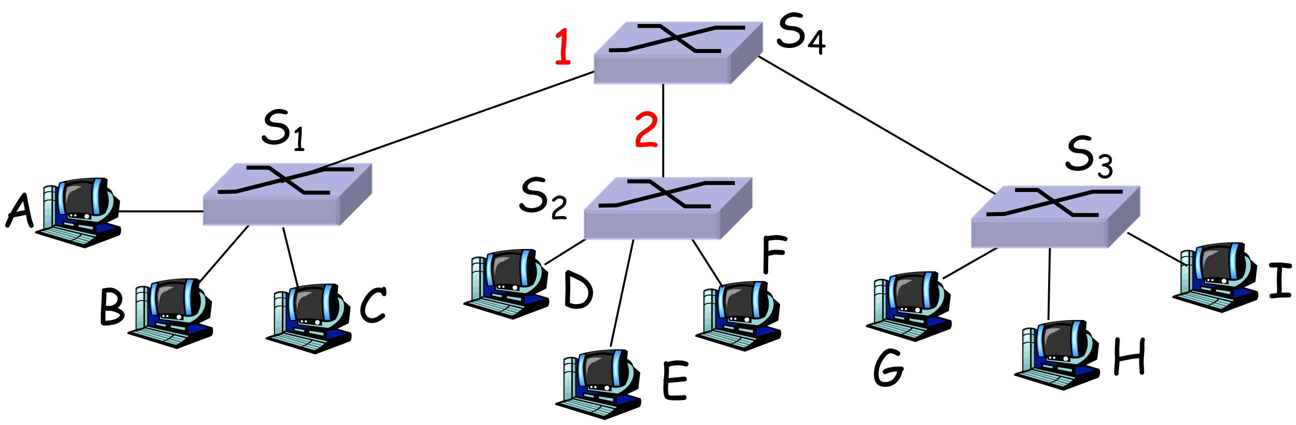



- *Q:* sending from A to G - how does $S_1$ know to forward frame destined to F via $S_4$ and $S_3$?
- *A:* self learning! (works exactly the same as in single-switch case!)


5: DataLink Layer 5a-22

# Self-learning multi-switch example

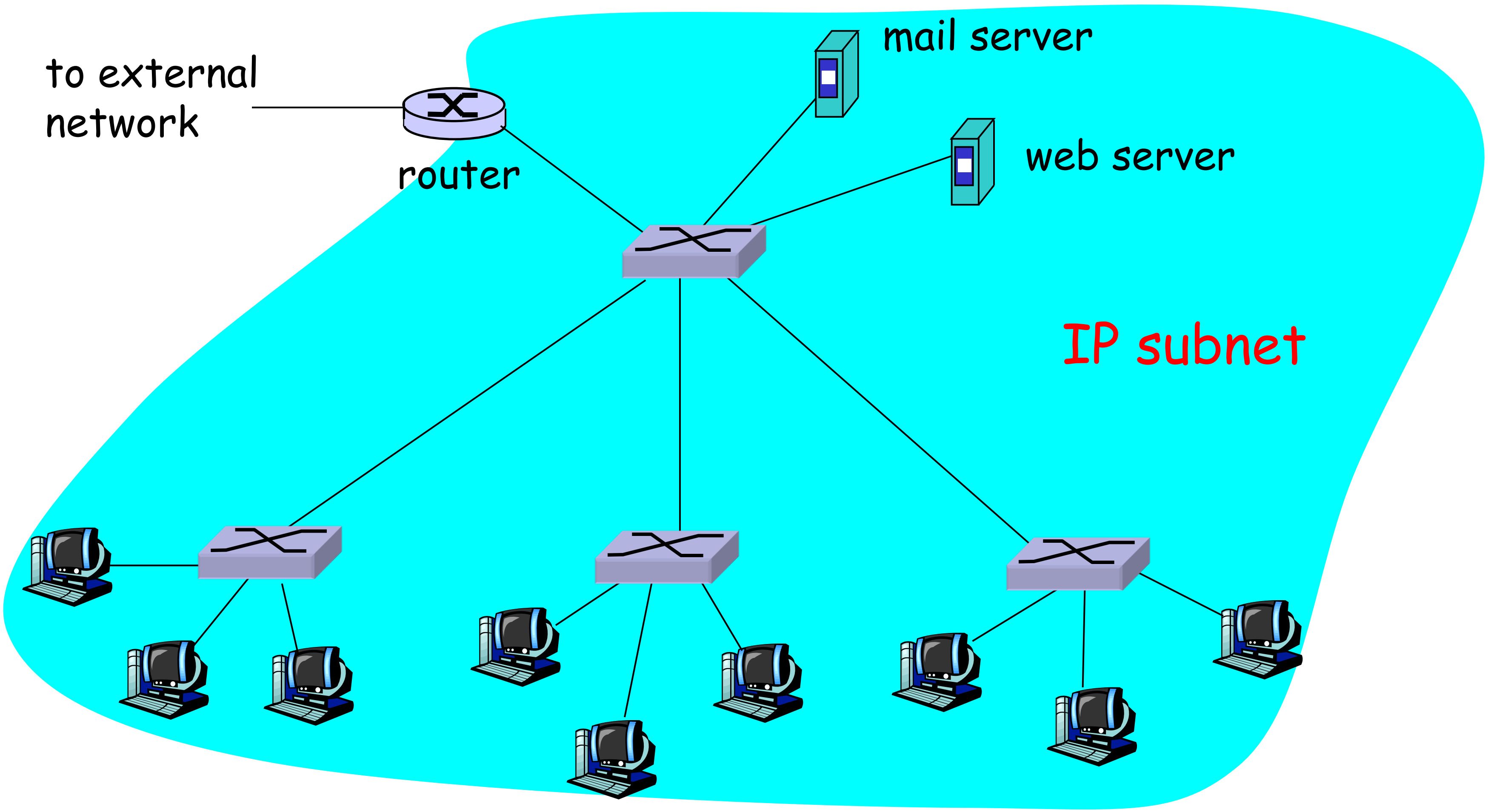
Suppose C sends frame to I, I responds to C




- *Q:* show switch tables and packet forwarding in $S_1$, $S_2$, $S_3$, $S_4$


5: DataLink Layer 5a-23

# Institutional network

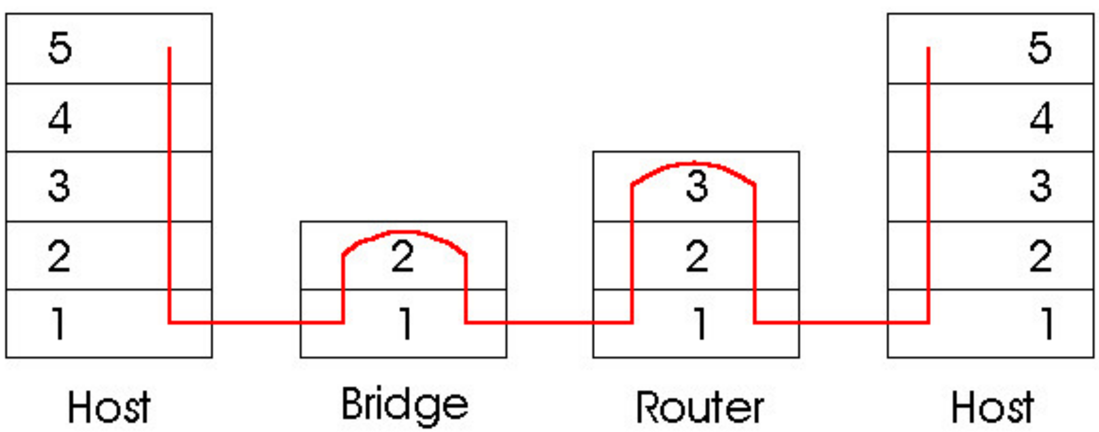




5: DataLink Layer 5a-24

# Switches vs. Routers

- both store-and-forward devices
  - routers: network layer devices (examine network layer headers)
  - switches are link layer devices
- routers maintain routing tables, implement routing algorithms
- switches maintain switch tables, implement filtering, learning algorithms



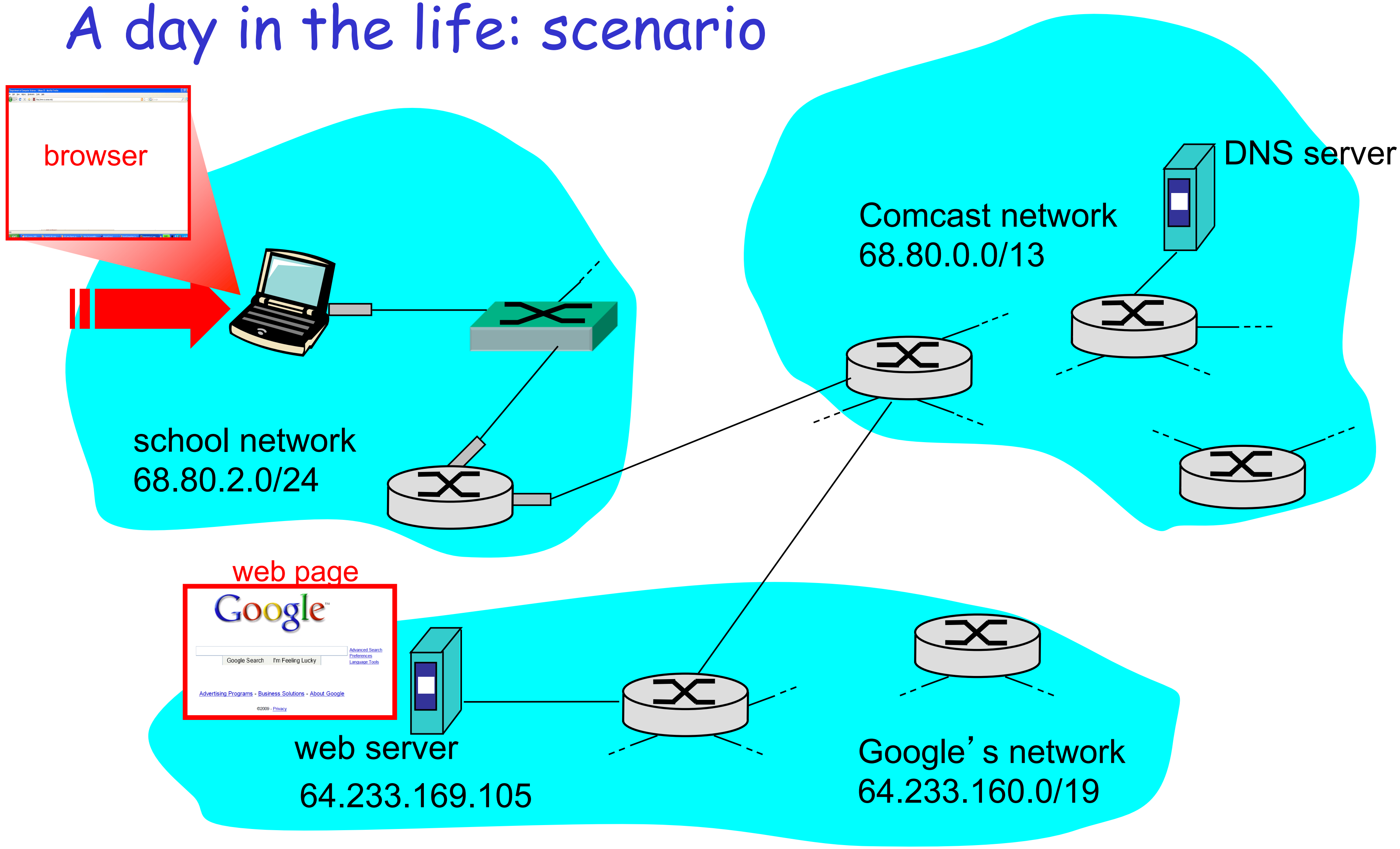


5a-25

# 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

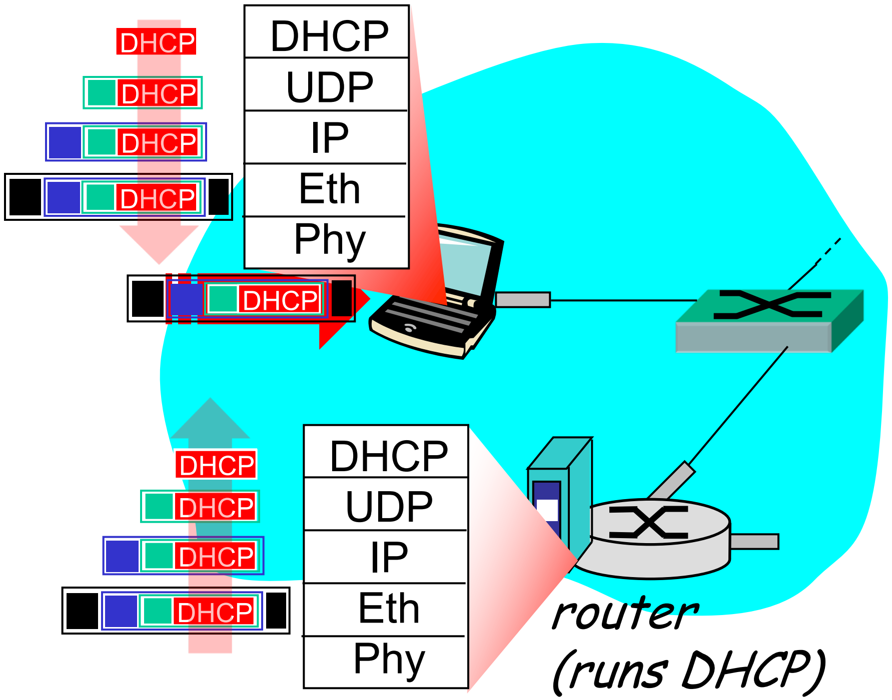
# A day in the life: scenario





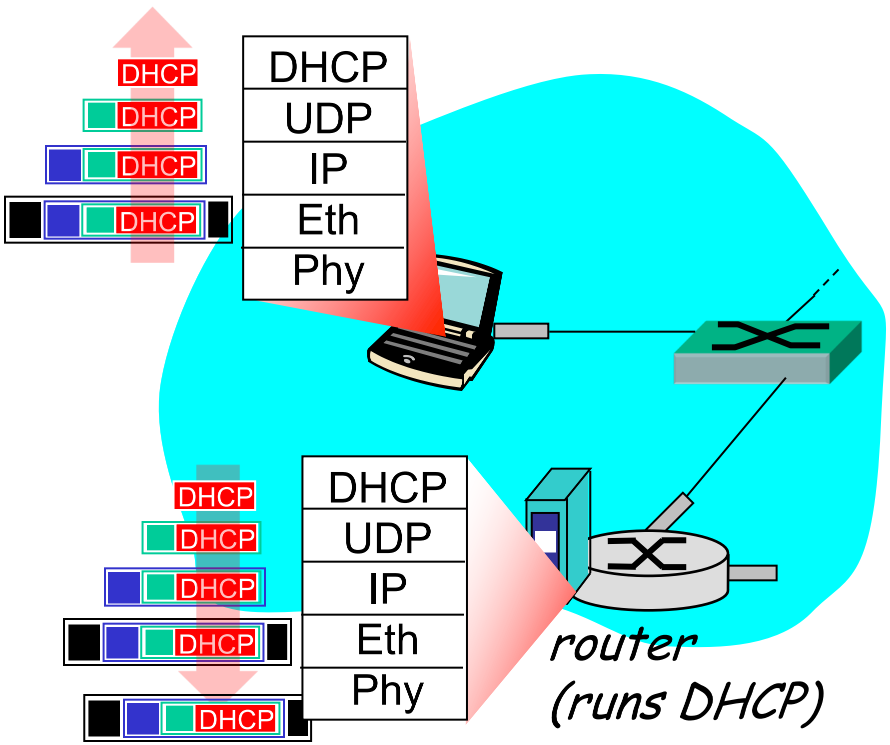
5a-27

# 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 *802.1* Ethernet
- Ethernet frame ***broadcast*** (dest: FFFFFFFFFFFF) on LAN, received at router running *DHCP* server
- Ethernet ***demux'ed*** to IP demux'ed, UDP demux'ed to DHCP

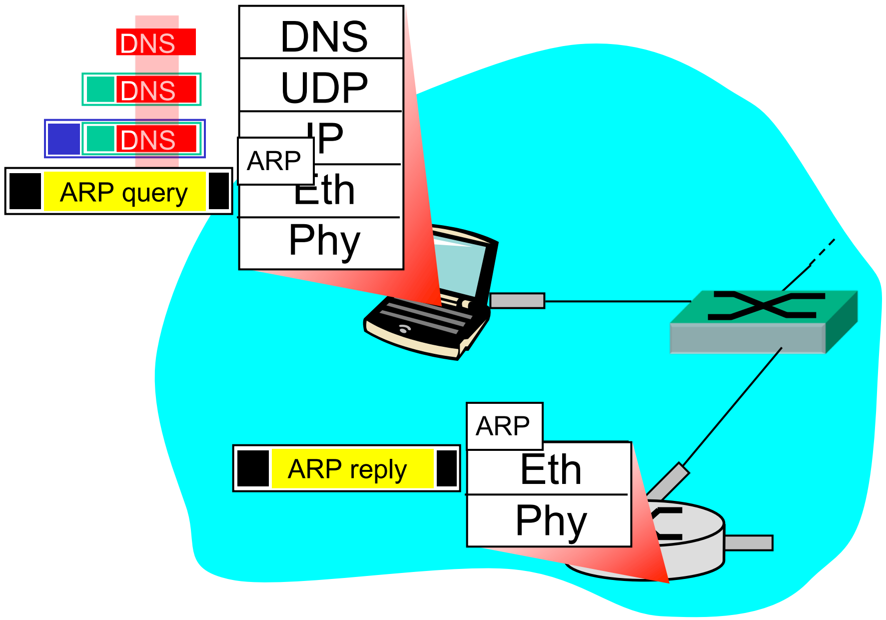
# A day in the life… connecting to the Internet




- 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

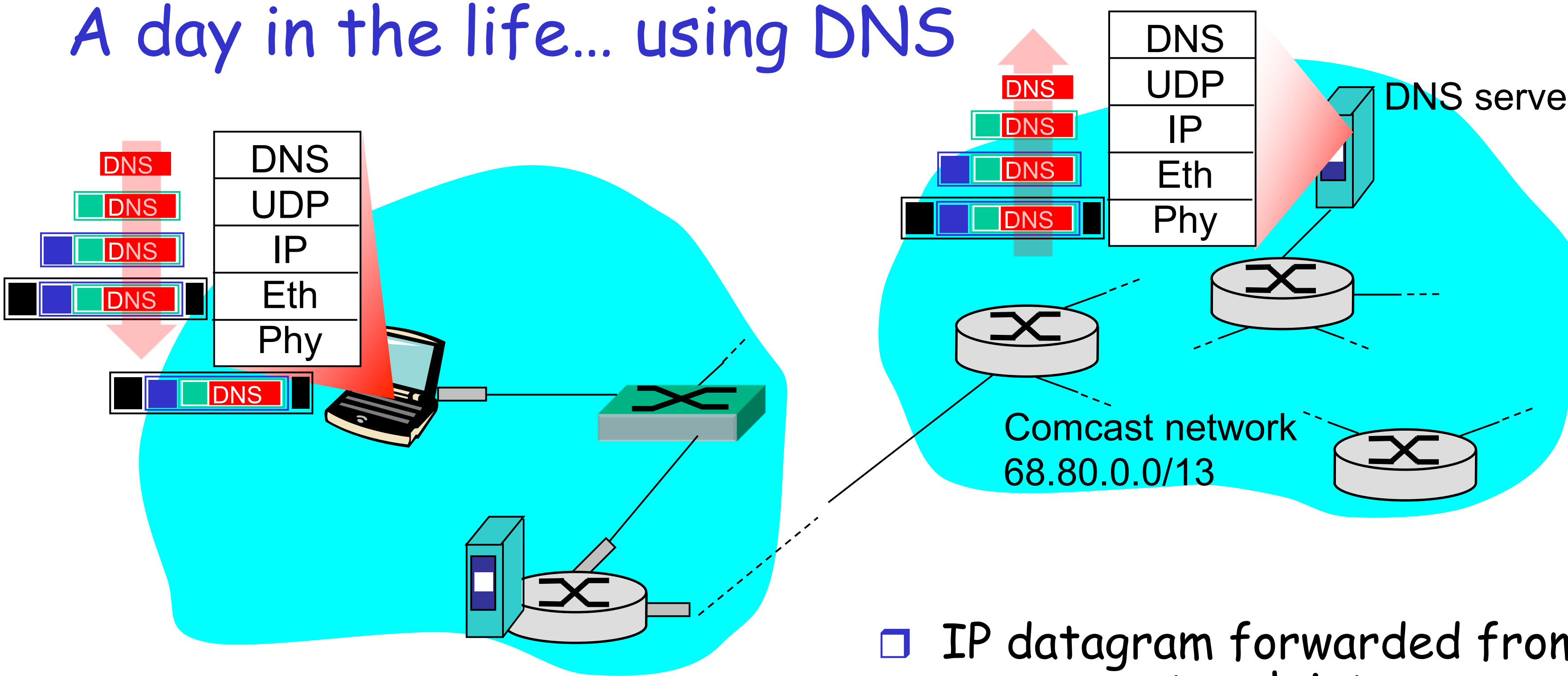
*Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router*

# 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, encasulated in Eth. In order 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

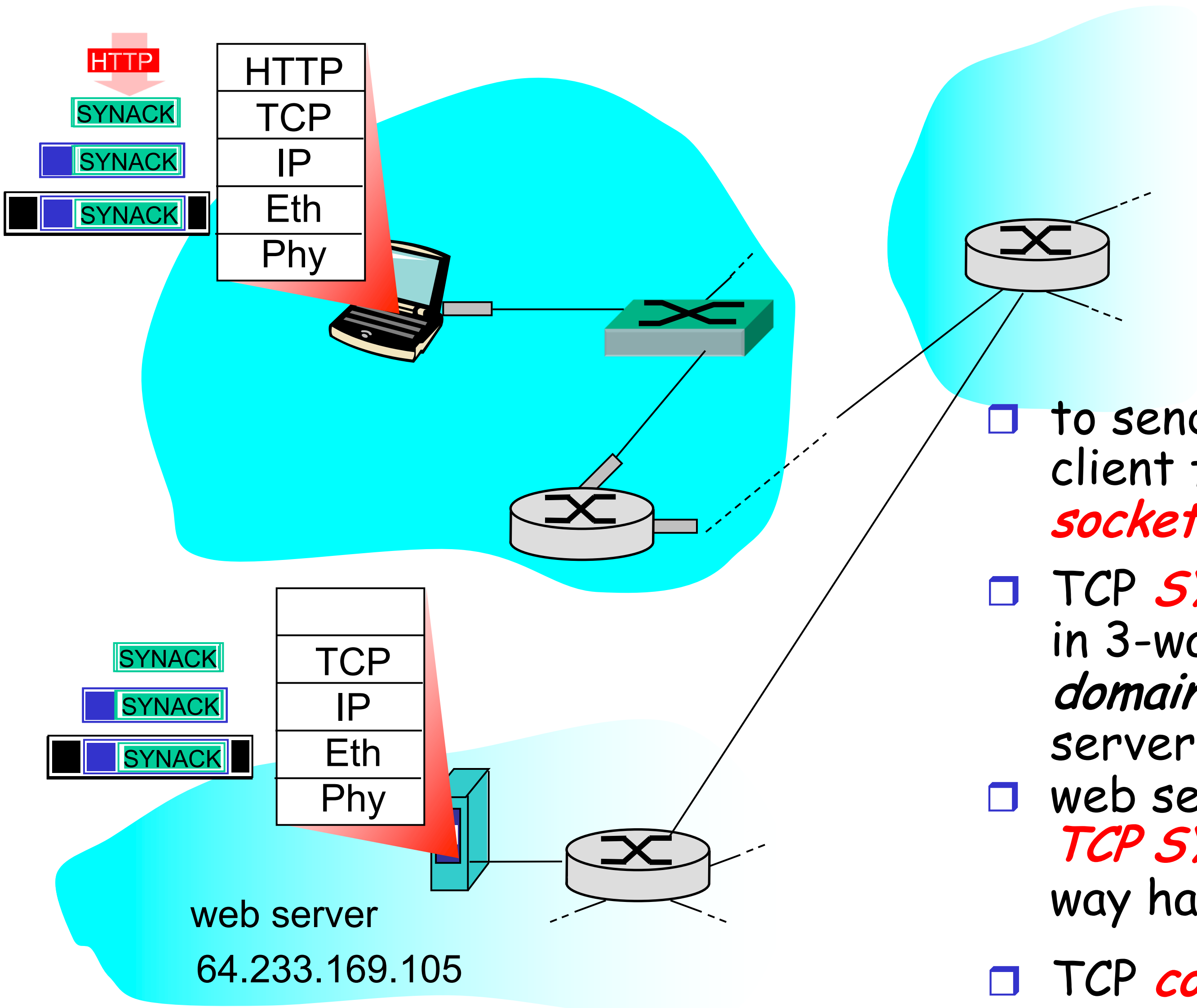
# A day in the life… using DNS




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

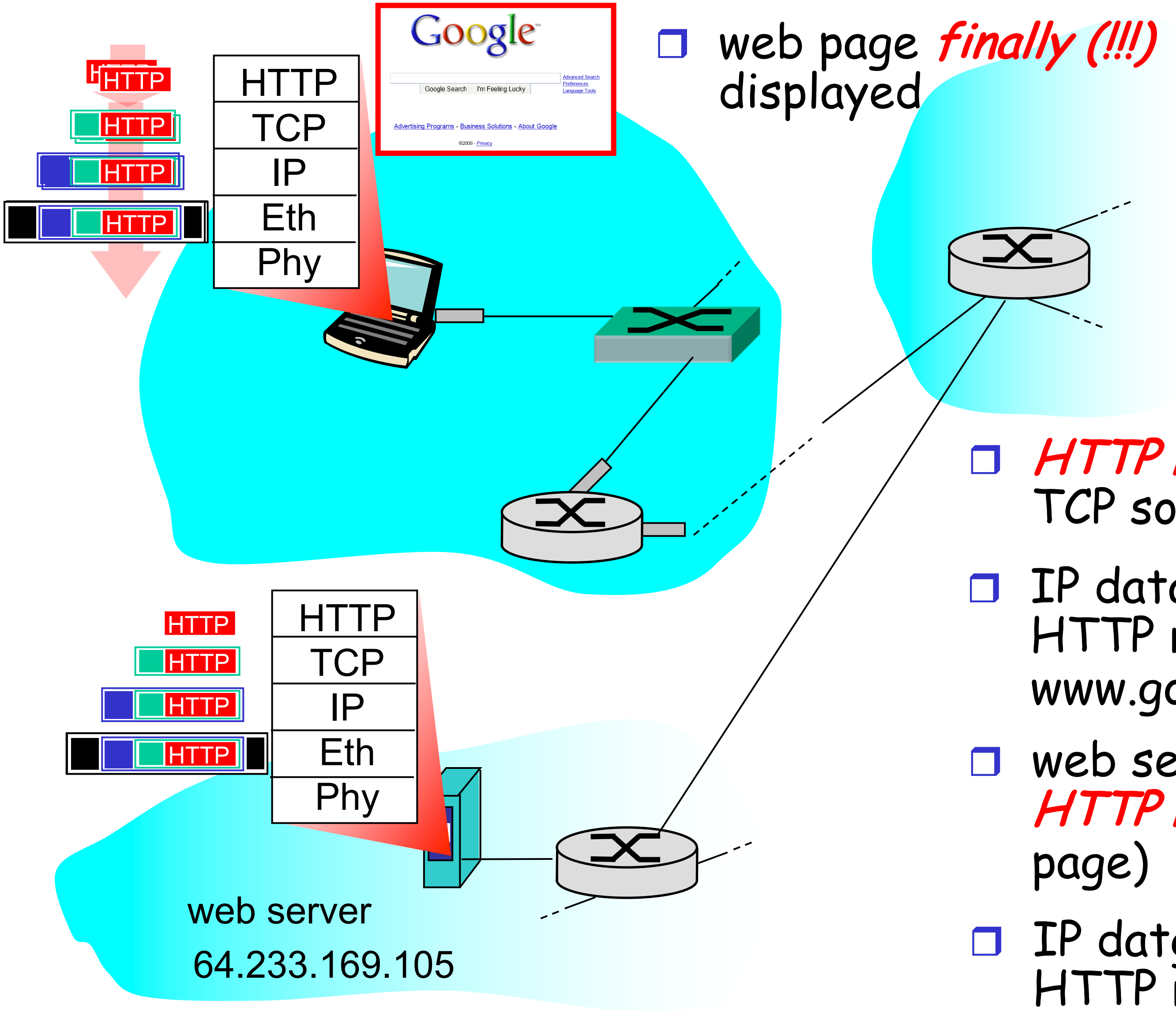
- IP datagram forwarded from campus network into comcast network, routed (tables created by ***RIP, OSPF, IS-IS*** and/or ***BGP*** routing protocols) to DNS server
- demux'ed to DNS server
- DNS server replies to client with IP address of www.google.com

# A day in the life… TCP connection carrying HTTP




- to send HTTP request, client first opens ***TCP socket*** to web server
- TCP ***SYN segment*** (step 1 in 3-way handshake) ***inter-domain routed*** to web server
- web server responds with ***TCP SYNACK*** (step 2 in 3-way handshake)
- TCP ***connection established!***

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




- web page *finally (!!!)* displayed

- *HTTP request* sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with *HTTP reply* (containing web page)
- IP datgram containing HTTP reply routed back to client

# Chapter 5 outline

- 5.1 Introduction and services
- 5.2 Error detection and correction
- 5.3Multiple access protocols
- 5.4 LAN addresses and ARP
- 5.5 Ethernet
- 5.6 Hubs, bridges, and switches
- 5.7 Wireless links and LANs
- 5.8 PPP
- 5.9 ATM
- 5.10 Frame Relay