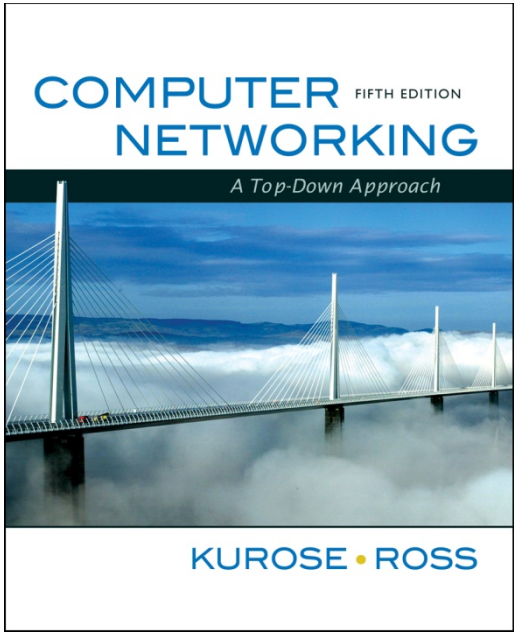
Chapter 5
Link Layer and LANs



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

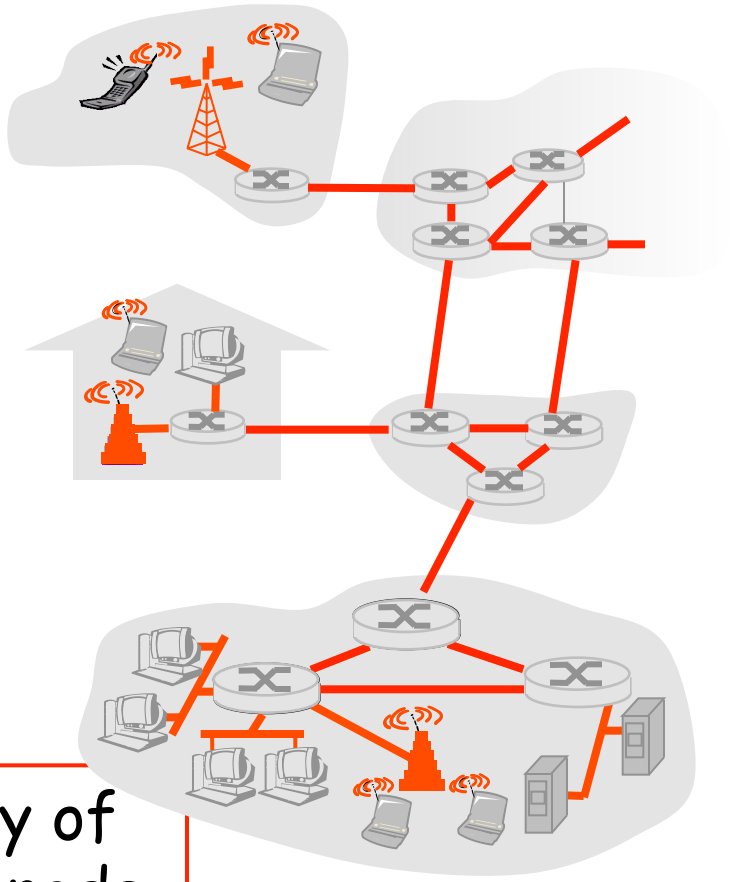
Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach 5th edition. Jim Kurose, Keith Ross Addison-Wesley, April 2009.

Link Layer: Introduction

Some terminology:

- hosts and routers are **nodes**
- communication channels that connect adjacent nodes along communication path are **links**
  - wired links
  - wireless links
  - LANs
- layer-2 packet is a **frame**, encapsulates datagram



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

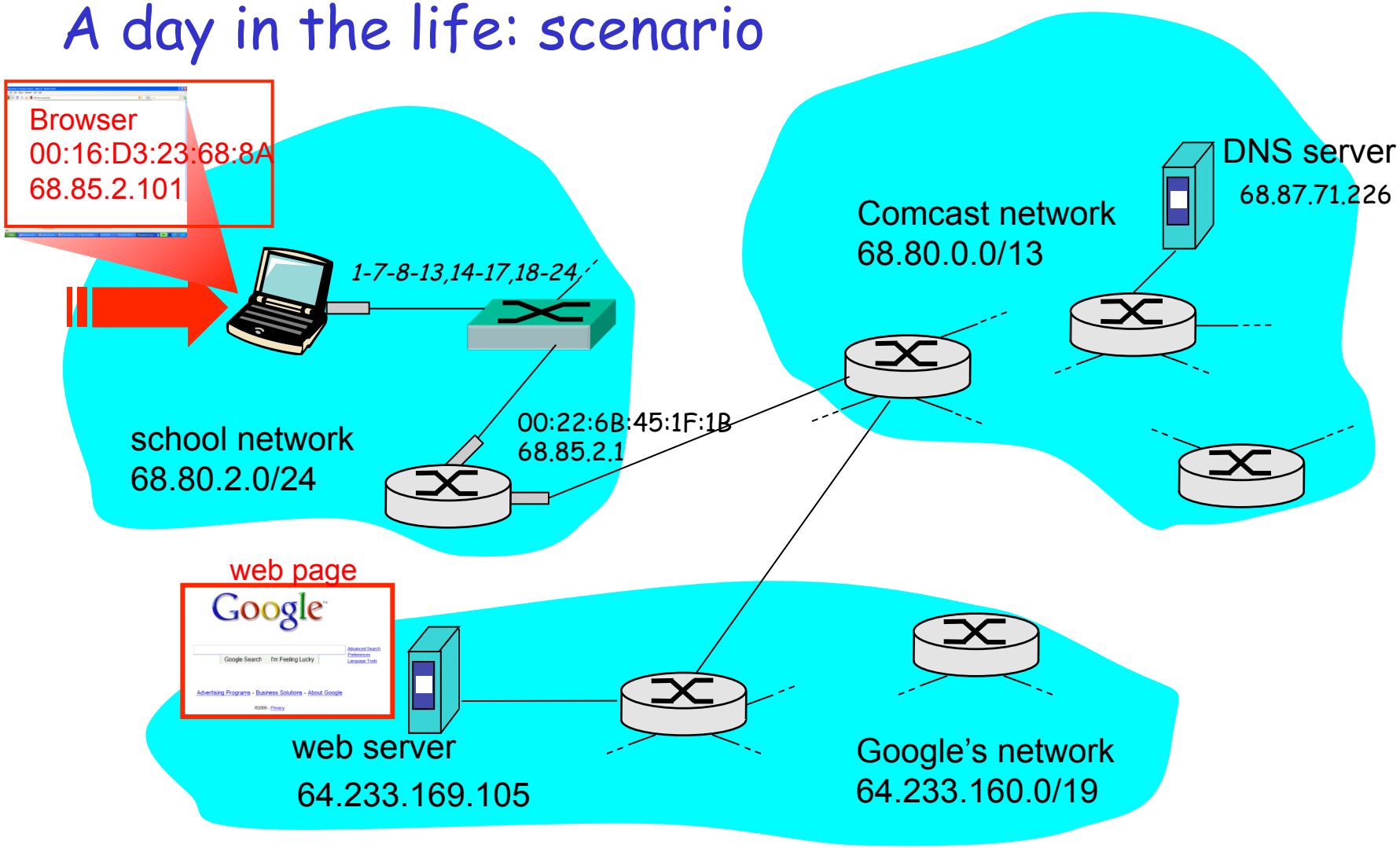
Link Layer

- 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

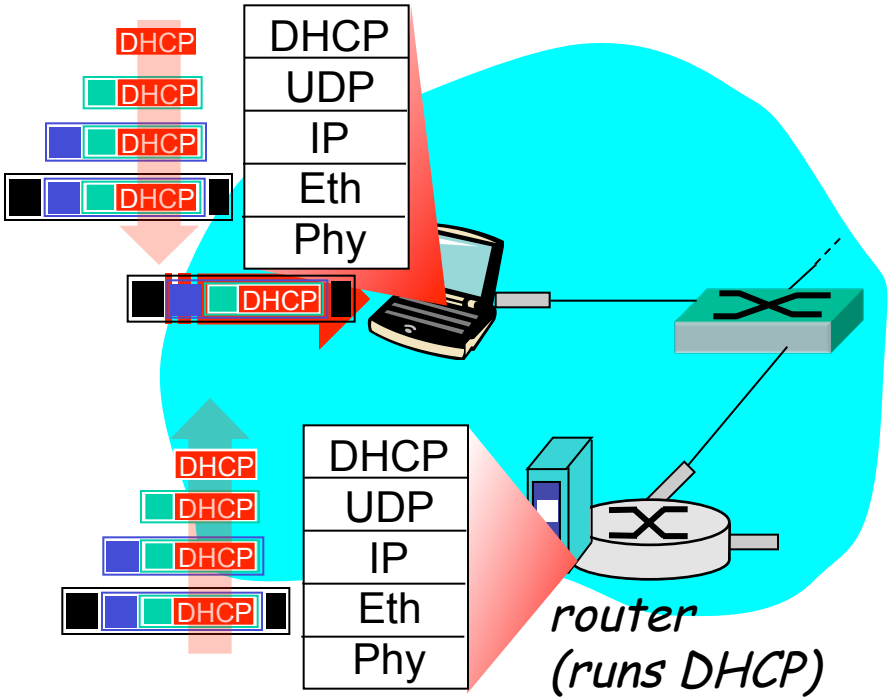
Synthesis: a day in the life of a web request

- 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
  - Bob connects a laptop on his school's Ethernet switch and downloads a web page, say the home page of www.google.com

A day in the life: scenario

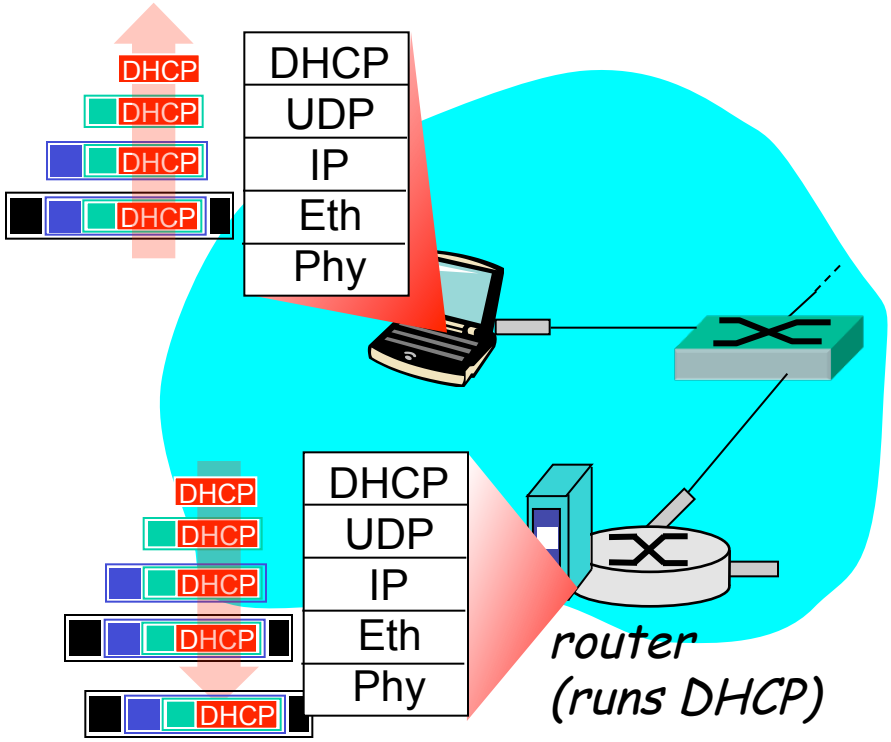


A day in the life… connecting to the Internet



- 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 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

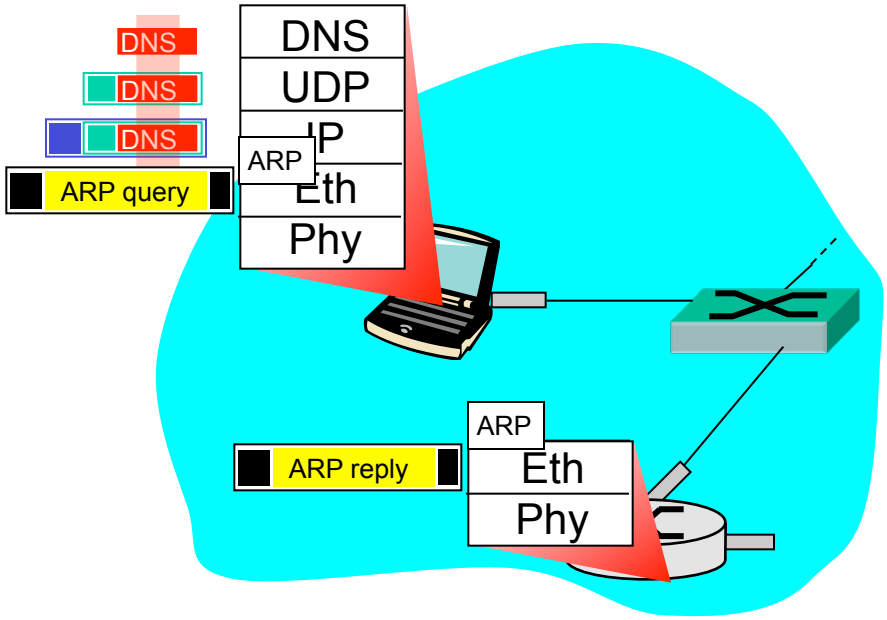
Getting Started DHCP, UDP, IP, and Ethernet (1-7)

1. OS of Bob's laptop creates a DHCP request, put it within a UDP segment with dest. port 67 (DHCP server) and source port 68 (DHCP client). Then, it is placed within an IP with broadcast addr. of 255.255.255.255 and source addr. of 0.0.0.0.
2. The IP datagram is then placed within an Ethernet frame with source addr, of FF:FF:FF:FF:FF:FF and source addr. of 00:16:D3:23:68:8A.
3. The broadcast Ethernet frame is sent to the Ethernet switch. It in turn broadcasts on all outgoing ports.
4. The router receives the broadcast frame on its interface with MAC 00:22:6B:45:1F:1B. The IP datagram is de-multiplexed up to UDP, so the DHCP server has the DHCP request message.

Getting Started DHCP, UDP, IP, and Ethernet (1-7)

5. Suppose the DHCP sever running within the router can allocate IP addr. in the CIDR block 68.85.2.0/24. The DHCP server allocates 68.85.2.101 to Bob's laptop, creates a DHCP ACK containing this IP address and DNF's IP (68.87.71.226), default gateway's IP (68..85.2.1), and the subnet block (68.85.2.0/24). The message is encapsulated through IP, Ethernet, with frame source MAC address of 00:22:68:45:F :1B and destination MAC of 00:16:D3:23:68:8A.
6. The Ethernet frame is sent by the router to the switch, which is self-learning and knows to forward to 00:16:D3:23:68:8A.
7. Bob's laptop receives the frame and de-multiplexes up to DHCP ACK message. It installs the address of the default gateway into it IP forwarding table. (hence, it will send all datagrams with destination address outside the subnet to the gateway.)

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

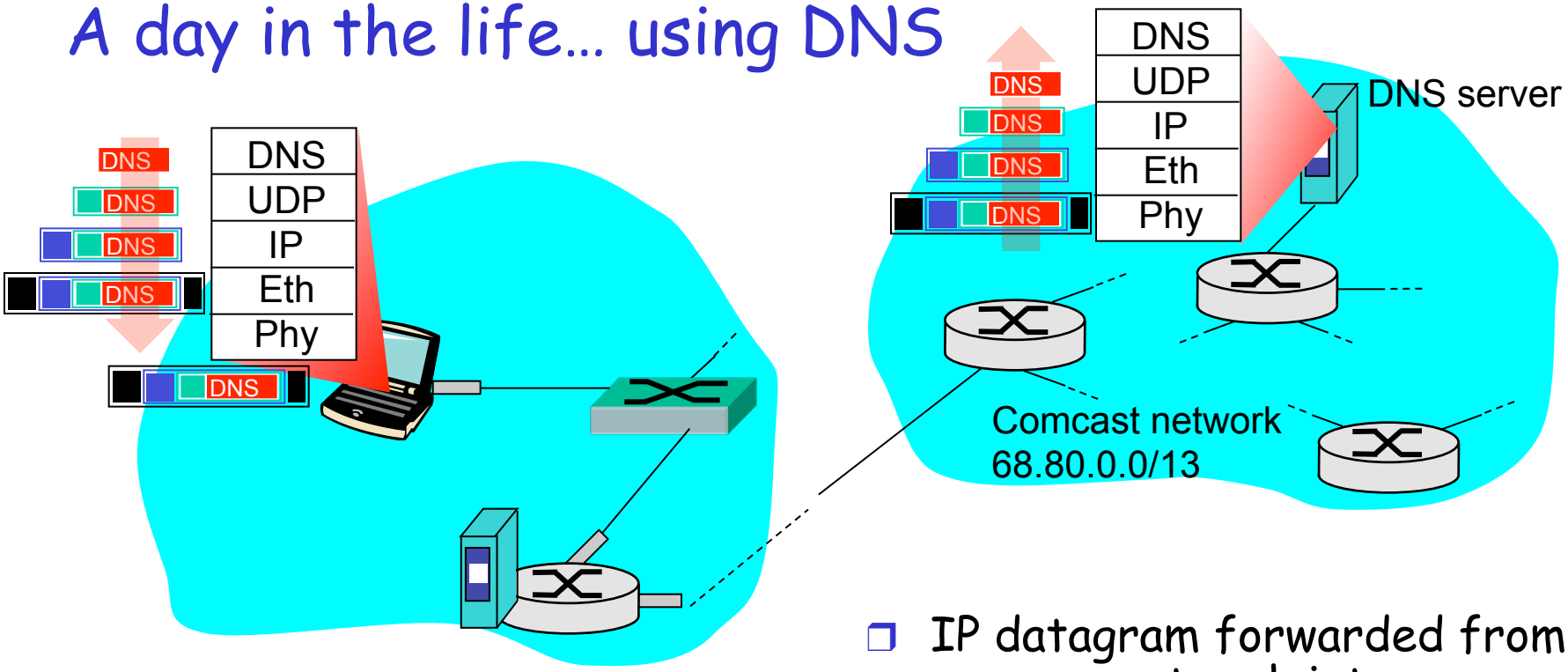
Getting Started DNS, ARP (8-13)

8. When Bob types the URL for www.google.com, the OS of Bob's laptop creates DNS query, putting the URL in the question section of the DNS message. The DNS is placed within a UDP segment with a dest. port of 53 (DNS server), in turn it is placed within an IP with dest. address of 68.87.71.226. and source address of 68.85.2.101.
9. Bob's laptop then places the datagram within an Ethernet frame. The frame is sent to the gateway router. Howver, Bob's laptop doesn't know the gateway router's MAC.
10. Bob's laptop creates an ARP query with the target IP of 68.85.2.1, places the ARP message within an Ethernet frame with a broadcast destination (FF:FF:FF:FF:FF:FF), and sends to the switch.

Getting Started DNS, ARP (8-13)

11. The gateway router receives the frame containing the ARP query message and finds that the target IP address of 68.85.2.1 matches its interface. It responds an ARP reply, indicates that its MAC 00:22:6B;45:1F:1B, with a destination address of 00:16:D3:23:68:8A(Bob's laptop).
12. Bob's laptop receives the frame and extracts the MAC address of the gateway router (00:22:6B:45:1f:1b) from the ARP reply message.
13. Bob's laptop now addresses the Ethernet frame containing the DNS query to the gateway router's MAC address. Note that the IP datagram in this frame has a destination 68.87.71.226 (DNS server), while the frame has a destination address of 00:22:6B:45:1F:1B (the gateway router). Bob's laptop sends this frame to the switch, which delivers the frame to the gateway router.

A day in the life… using DNS

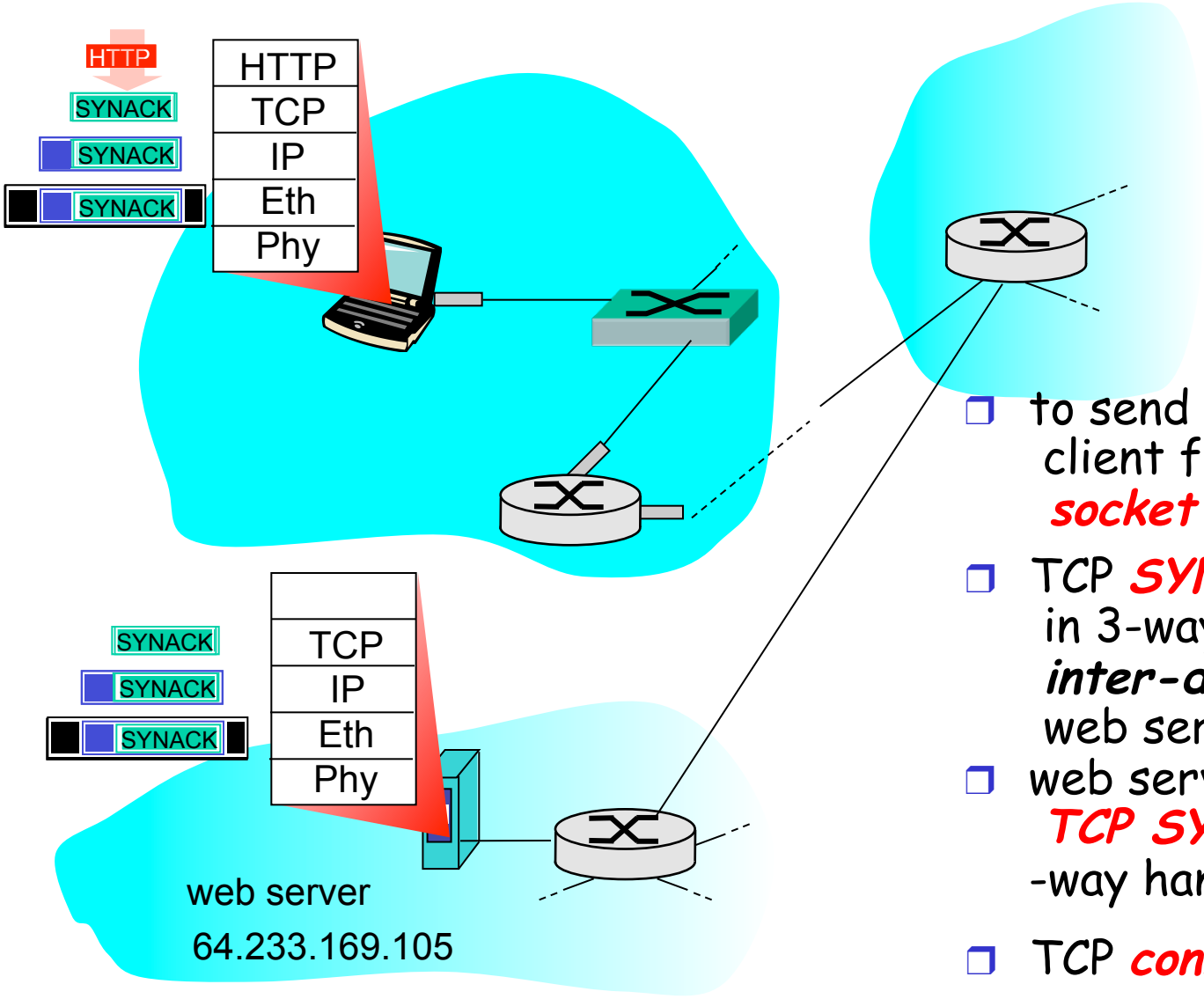


- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
- IP datagram forwarded from campus network into comcast network, routed (tables created by ***RIP, OSPF, IS-IS*** and/or ***BGP*** routing protocols) to DNS server
- demux'ed to DNS server
- DNS server replies to client with IP address of www.google.com

Intra-Domain routing to the DNS router (14-17)

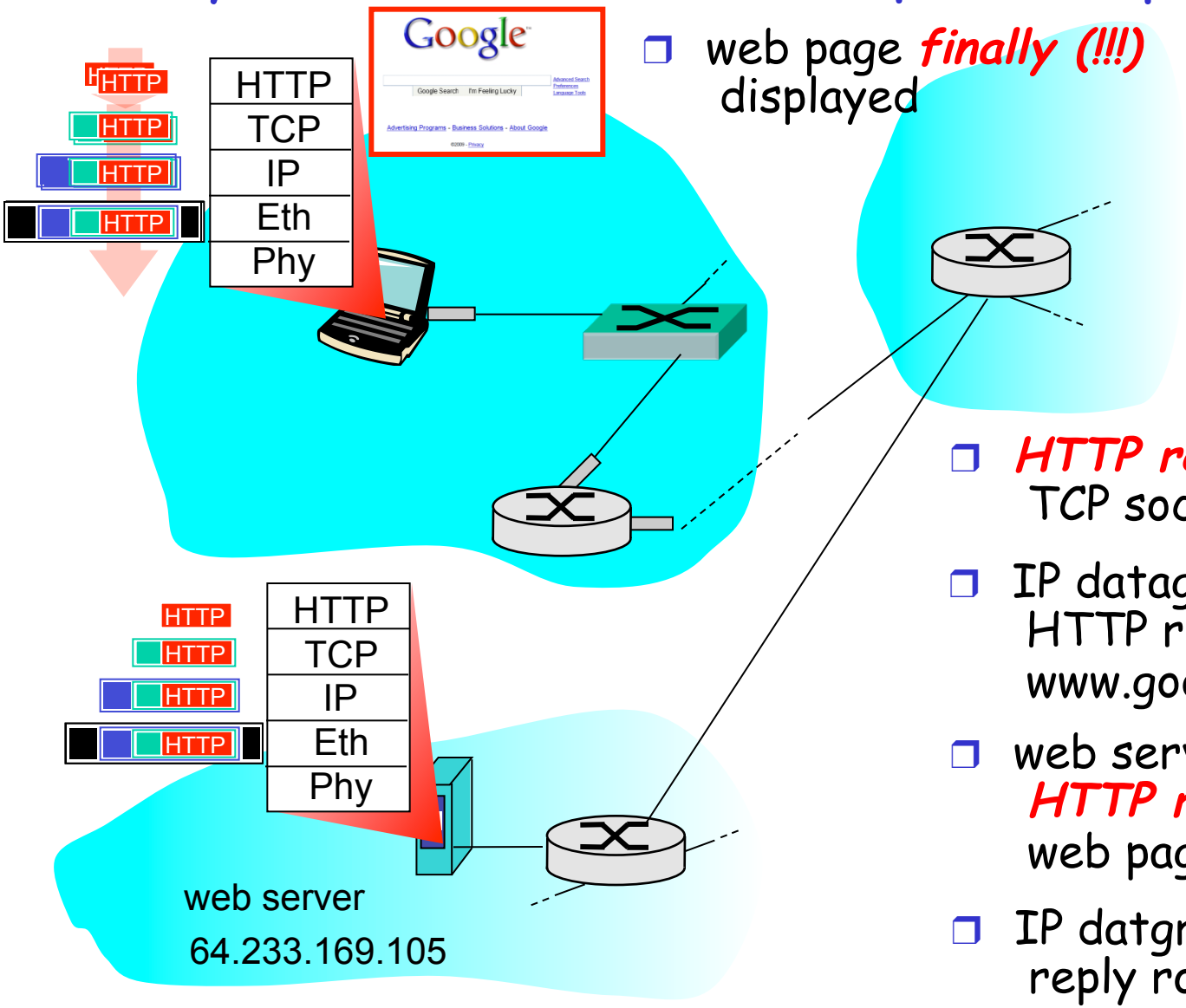
14. The gateway router receives the frame and extract the IP datagram containing the DNS query. It looks up the destination address (68.87.2.1) and determines from the forwarding table that the datagram should be sent to the router in Comcast network.
15. The Comcast network router determines the outgoing interface on which to forward the datagram towards the DNS server, which was filled by Comcast intra-domain protocol (RIP, OSPF, IS-IS) and the Inter-domain protocol, BGP.
16. DNS server looks up the name www.google.com in its DNS database and finds the DNS resource record that contains the IP (64.233.169.105). The DNS server forms a DNS reply containing the hostname-to-IP address mapping, places it in UDP, IP to Bob's laptop.
17. Bob's laptop extracts the IP address of the server from the DNS message. Finally, Bob's laptop is ready to contact the www.google.com server.

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

Web Client-Server Interaction: TCP & HTTP (18-24)

18. Bob's laptop is ready to create the TCP socket that will be used to send the HTTP GET to the web server. 3-way handshake must be first performed. Bob's laptop creates a TCP SYN with destination port 80, places it inside an IP with destination 64.233.169.105. Then, places the datagram inside a frame with MAC 00:22:6B:45:1F:1B, and sends the frame to the switch.
19. The router in the school network, Comcast's network, google's network forward the datagram, which are determined by BGP.
20. The web server receives the datagram, and a connected socket is created for the TCP connection between the google HTTP server and Bob's laptop. A TCP SYNACK is generated addressed to Bob's laptop and places inside a link-layer frame to its first-hop router.
21. The datagram containing the TCP SYNACK segment is forwarded through the Google, Comcast, and school networks, eventually arrives at the Ethernet card in Bob's laptop.

Web Client-Server Interaction: TCP & HTTP (18-24)

22. With the socket on Bob's laptop ready to send a byte to www.google.com. Bob's browser creates the HTTP GET containing the URL to be fetched. The message is then written into the socket, with the GET message becoming the payload of a TCP segment.
23. The HTTP server at Google reads the HTTP GET message from the TCP socket, creates an HTTP response message, and sends the message into the TCP socket.
24. The datagram containing the HTTP reply message is forwarded through the Google, Comcast, and school networks, and arrives at the Bob's laptop. Bob's web browser program reads the HTTP response from the socket, extracts the html file for the web page from the body of the HTTP response, and finally displays the web page.