Computer System Design: Laws, Principles, Trends - II

Presenter: Sandeep K. S. Gupta

Reference:

-Computer Networking: A Top-Down Approach, 5ed, Ross and Kurose.

-Computer Organization and Design: The Hardware/Software Interface, David A. Patterson and John L. Hennessy

- An Engineering Approach to Computer Networking, S. Keshav

Arizona State University

School of computing, informatics, and decision systems engineering

Agenda

- Abstraction
- Layered Architecture
  - Example TCP/IP Protocol Stack
- Layering: Pros and Cons
- End-to-End Principle – Guiding Layering
- Hierarchical Decomposition – Improving Scalability; Fault-tolerance
  - Example: Domain Name System (DNS)
- Multiplexing & Batching – Sharing Resources; Reducing Cost
- TCP/IP Networking
  - Example: A day in the life of web page request

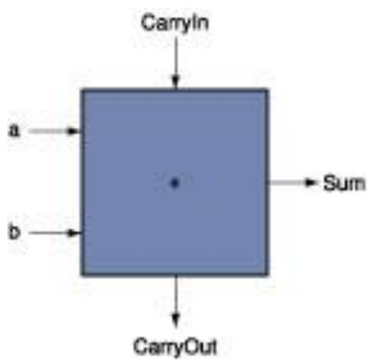
Mastering Complexity – Abstraction

- **Abstraction**: is a mechanism and practice to reduce and factor out details so that one can focus on a few concepts at a time. (wikipedia)
  - Control abstraction: abstraction of action
  - Data abstraction: abstraction for handling data in meaningful manner
- Layered Architecture:
  - Each layer provides some services to layer above it and uses some services below it
  - In distributed system – layers act as "peers"
  - Each layer hides (abstracts) some information

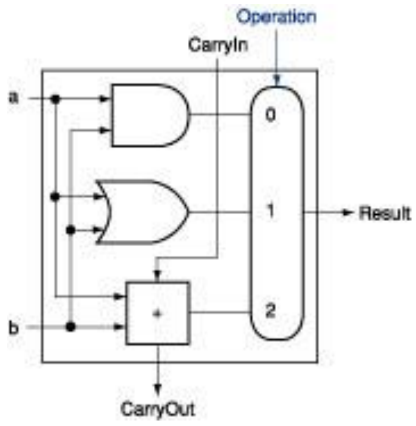
Layer - Level of Abstraction

- Level of abstraction consists of
  - an interface (outside view of what it does), and
  - an implementation (inside view of how it works)

Interface

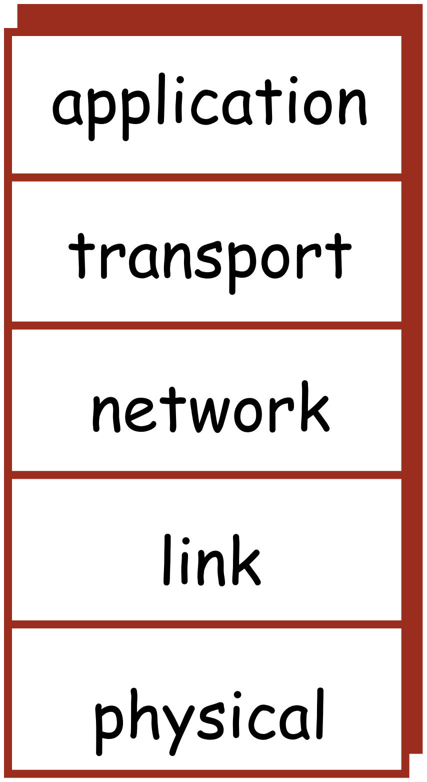


Implementation



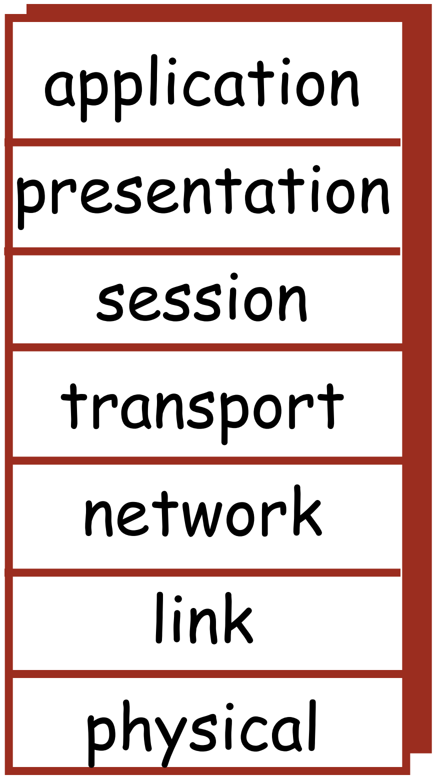
Example: Internet protocol stack

- application: supporting network applications
  - FTP, SMTP, HTTP
- transport: process-process data transfer
  - TCP, UDP
- network: routing of datagrams from source to destination
  - IP, routing protocols
- link: data transfer between neighboring network elements
  - PPP, Ethernet
- physical: bits "on the wire"



ISO/OSI reference model

- presentation: allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
- *session:* synchronization, checkpointing, recovery of data exchange
- Internet stack "missing" these layers!
  - these services, *if needed,* must be implemented in application
  - needed?



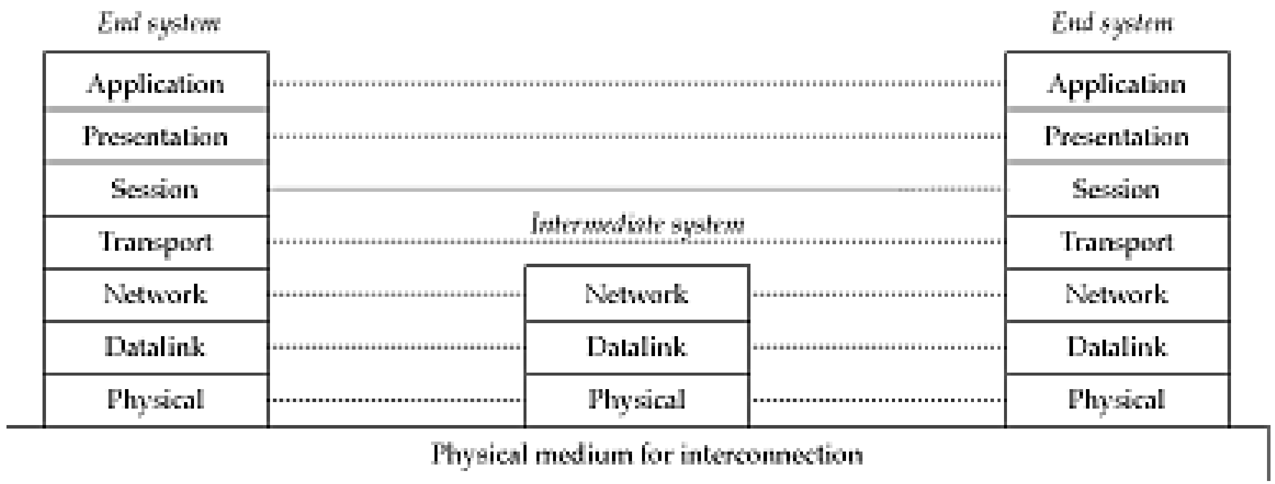
Layering – Pros or Cons

Dealing with complex systems:

- explicit structure allows identification, relationship of complex system's pieces
  - layered reference model for
- modularization eases maintenance, updating of system
  - change of implementation of layer's service transparent to rest of system
  - e.g., change in CPU scheduler doesn't affect rest of system
- layering considered harmful?

End to end design principle

- Information-hiding – may get in way of achieving good performance
- *Functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level*
- Example: Reliable data transmission
  - Transport-layer is still needed even if data-link layer provided reliable link level transmission



  - What is advantage of reliable data link? Improved performance.

[J. H. Saltzer, D. P. Reed, and D. D. Clark. 1984. End-to-end arguments in system design. *ACM Trans. Comput. Syst.* 2, 4]

End to end design principle (cont.)

- Kind of:
  - "Occam's razor" (law of parsimony): "other things being equal, a simpler explanation is better than a more complex one".
  - KISS: Keep it simple, Stupid!
- Motivation for "Open" Systems (Lampson) in which the whole of OS consists of replaceable routines.
- *Separation of Mechanism and Policy design principle* – "mechanisms should not restrict the policies according to which decisions are made" – Birch Hansen, the evolution of operating systems.
  - Decoupling mechanism implementation from policy specification allows different applications to use the same mechanism
  - Example:
    - OS: scheduling policy versus scheduling mechanism
    - Security: authentication mechanism vs. authorization policy

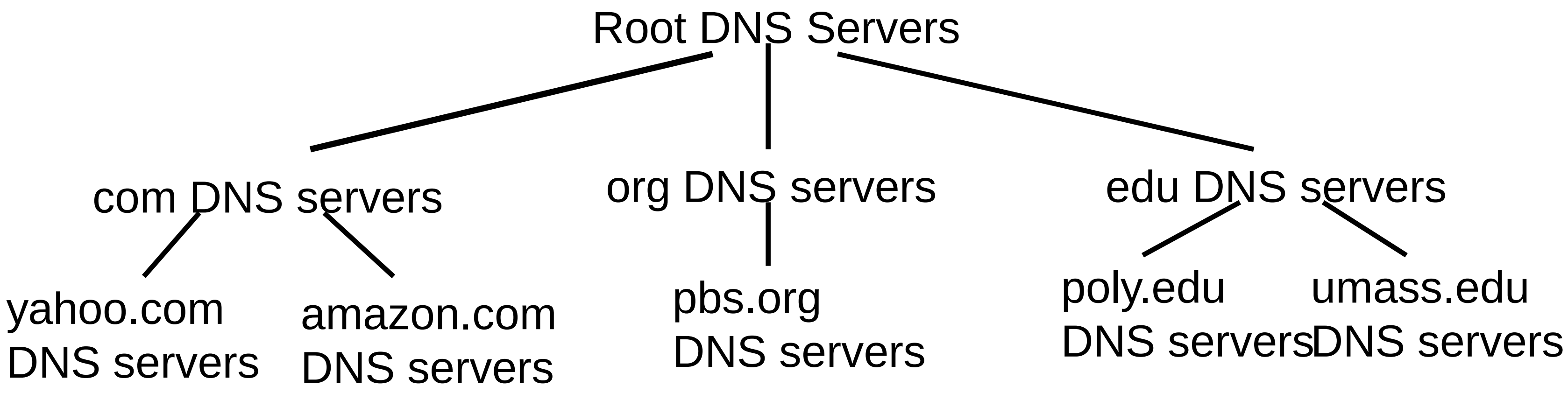
Hierarchical Decomposition – Improving Scalability and Fault Resilience

- Recursive decomposition of a system into smaller pieces that depend only on parent for proper execution
- No single point of control (failure)
- Highly scaleable and fault-tolerance; Specially with use of
  - distributed processing,
  - caching,
  - soft-state (as opposed to hard state): state (cached) information has time-out associated; on expiry of timer state info. needs to be reacquired.
- Leaf-to-leaf communication can be expensive
  - shortcuts help
- Example: Domain Name System (DNS): maintains mapping between domain names and IP address (and much more ..)

Soft state

- State: memory in the system that influences future behavior
  - for instance, VCI translation table
- State is created in many different ways
  - signaling
  - network management
  - routing
- How to delete it?
- Soft state => delete on a timer
- If you want to keep it, refresh
- Automatically cleans up after a failure
  - but increases bandwidth requirement

DNS: Distributed, Hierarchical Database



Client wants IP for www.amazon.com:

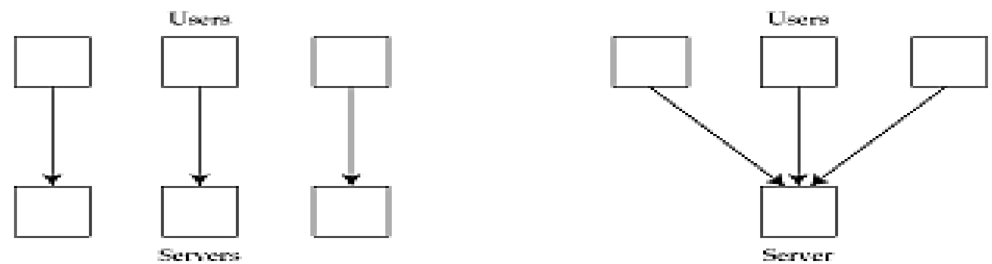
- client queries a root server to find com DNS server
- client queries com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS Caching

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time

Multiplexing (Sharing) Resources

- Trades time and space for money
- Response time↑, and space↑ (buffer space for queued jobs); costs↓
  - economies of scale



- Examples: multiplexed links; shared memory
- From job's perspective: shared resource is *unshared virtual resource*
- *Server* controls access to the shared resource
  - may use a *schedule* to resolve contention
    - choice of scheduling critical in proving quality of service (QoS) guarantees
  - may **batch** jobs together to amortize overhead across multiple jobs
    - De-multiplexing is needed to unpack the results of the batched job and returned to appropriate user.
    - In distributed system e.g. Internet – multiplexing and demultiplexing are done at different "ends" .

Batching

- Group tasks together to amortize overhead
- Only works when overhead for N tasks < N time overhead for one task (i.e. *nonlinear*)
- Also, time taken to accumulate a batch shouldn't be too long
- We're trading off reduced overhead for a longer worst case response time and increased throughput

(Statistical) Multiplexing Gain

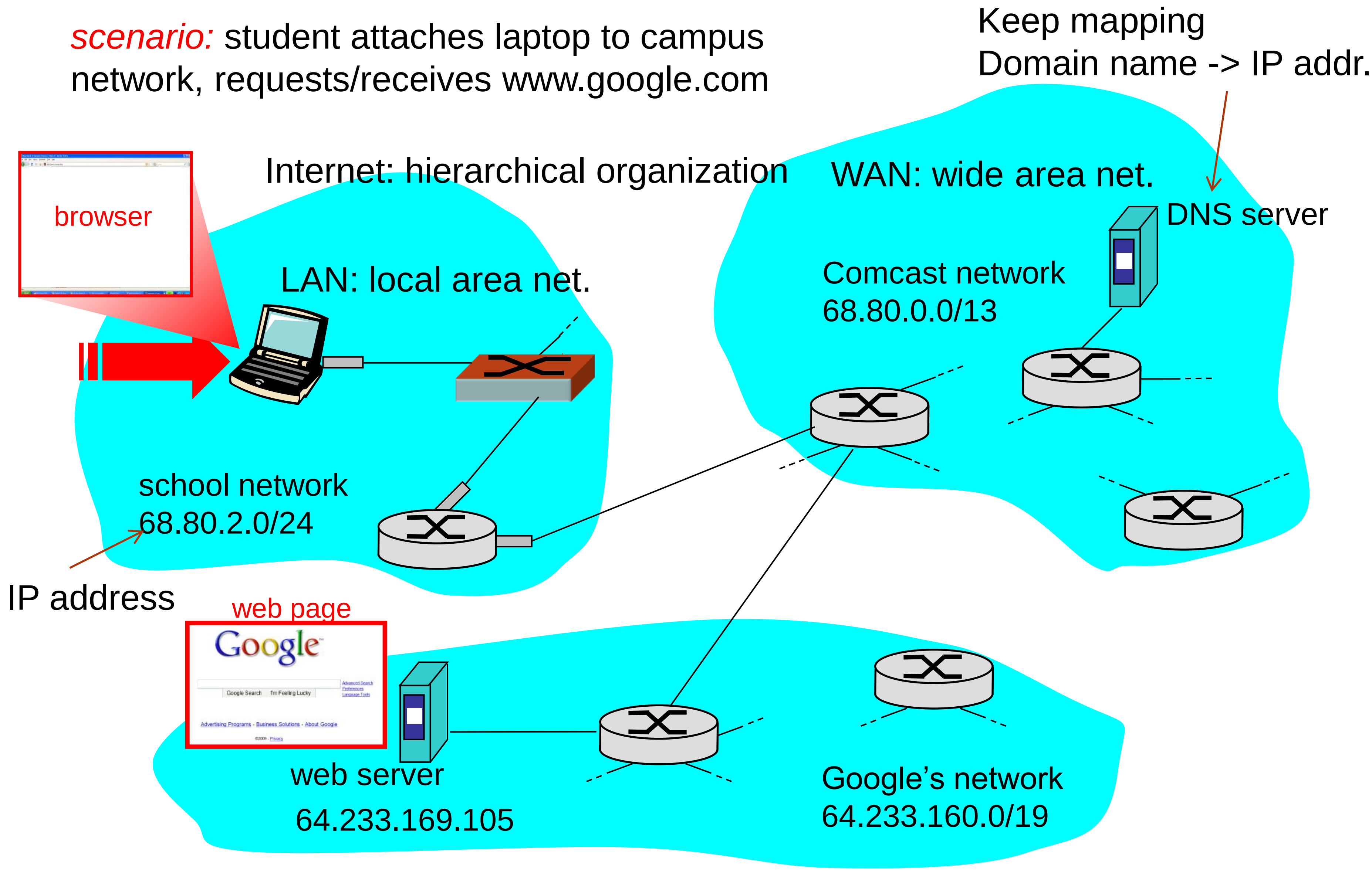
- Suppose resource has **capacity** C
  - E.g. bandwidth, total memory
- Shared by N identical tasks
  - E.g. packet, process
- Each task requires capacity b
  - E.g. packet size, process memory foot print
- If Nb <= C, then the *resource is underloaded (system is overprovisioned)*
- If at most 20% of tasks active, then C' >= 0.2*Nb is enough
  - we have used statistical knowledge of users to reduce system cost
  - *statistical multiplexing gain*: ability to reduce overall system requirement (cost) from C to C'

Example of statistical multiplexing gain

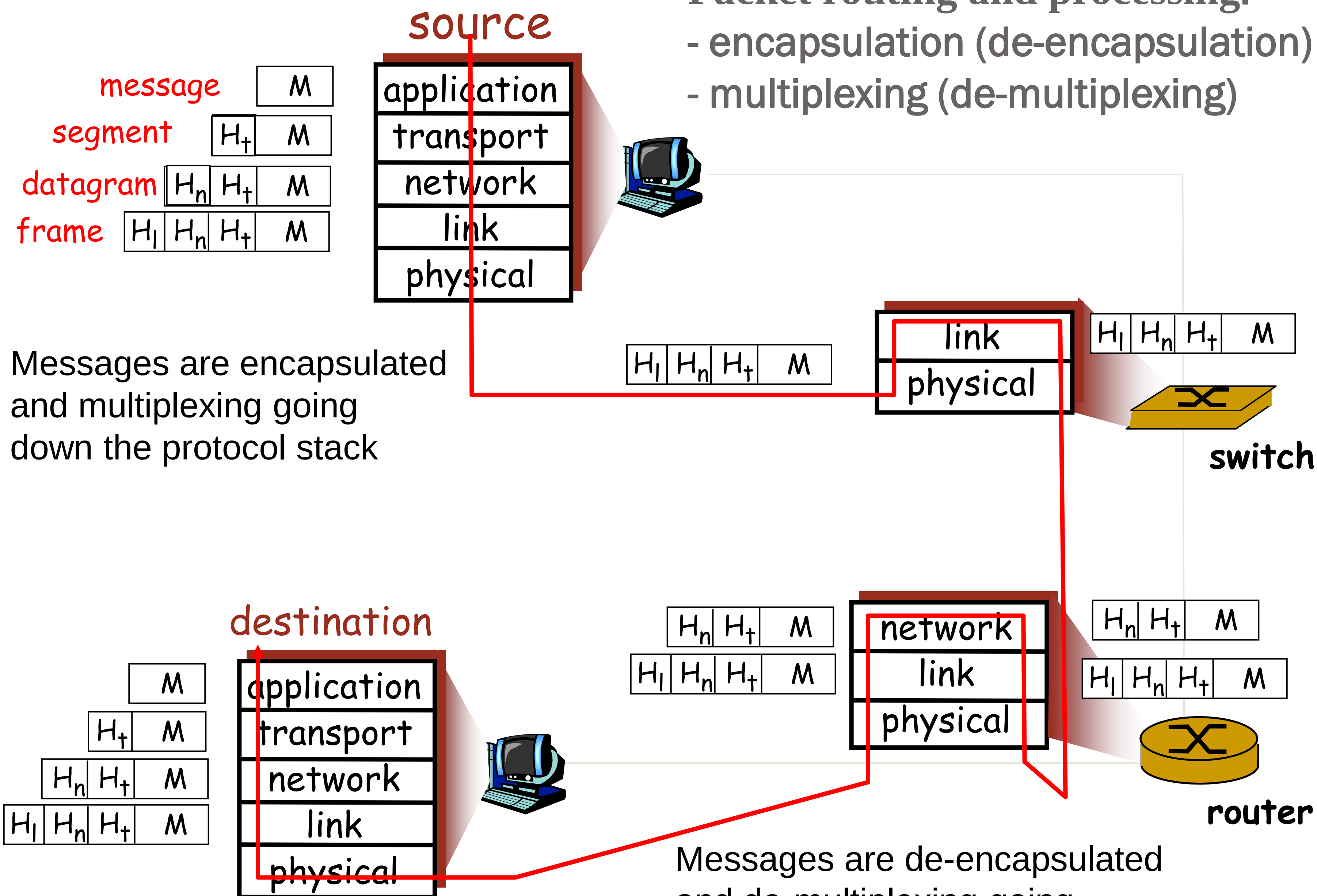
- Consider a 100 room hotel
- How many external phone lines does it need if only 20% of occupants make call at any time?
  - each line costs money to install and rent
  - Just 20 lines would be sufficient.
- What if a voice call is active only 50% of the time?
  - Can get by with only 10 lines (by using both spatial and temporal statistical multiplexing gain)
  - but only in a packet-switched network e.g. Internet (as opposed to connection-based networks e.g. telephone networks)
- Remember
  - to get SMG, we need good statistics!
  - if statistics are incorrect or change over time, we're in trouble
  - example: road system

Networking Example: A Day in Life of Web Page

scenario: student attaches laptop to campus network, requests/receives www.google.com



Packet routing and processing:
- encapsulation (de-encapsulation)
- multiplexing (de-multiplexing)



Messages are encapsulated and multiplexing going down the protocol stack

Messages are de-encapsulated and de-multiplexing going up the protocol stack

Networking Example: A Day in Life of Web Page

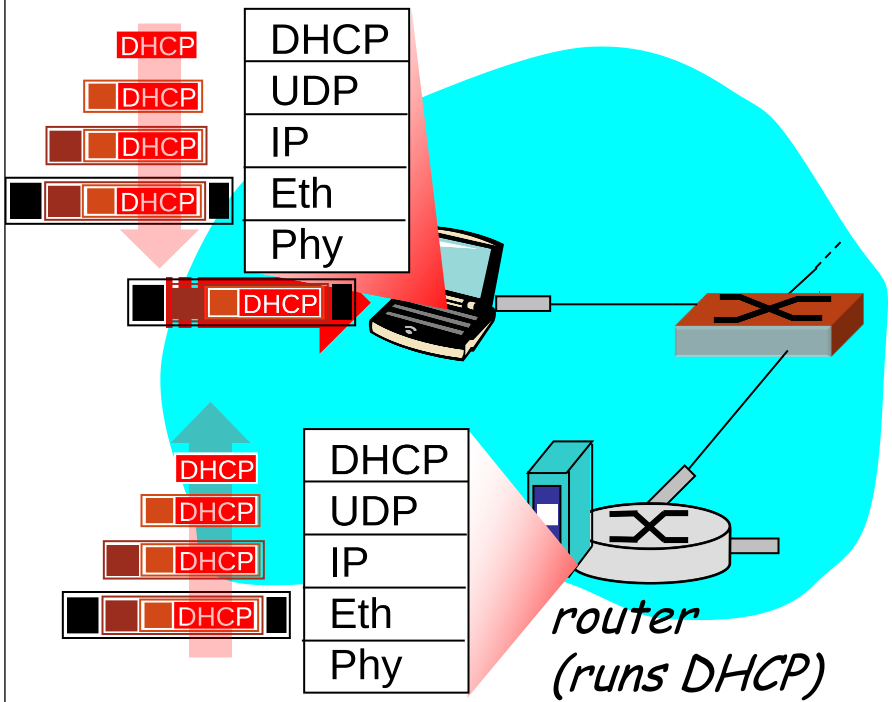
Background:

1. Each interface has network layer address (IP address) and link-layer address (MAC address)
2. Switches “learn” which MAC address are reachable from which of its interface
3. Address Resolution Protocol (ARP): like DNS for MAC layer - maintains mapping of IP address to MAC address
4. Two types of transport services: connection-oriented & reliable (TCP) and connection-less & unreliable (UDP)
5. DHCP service provides dynamic IP addresses

Steps:

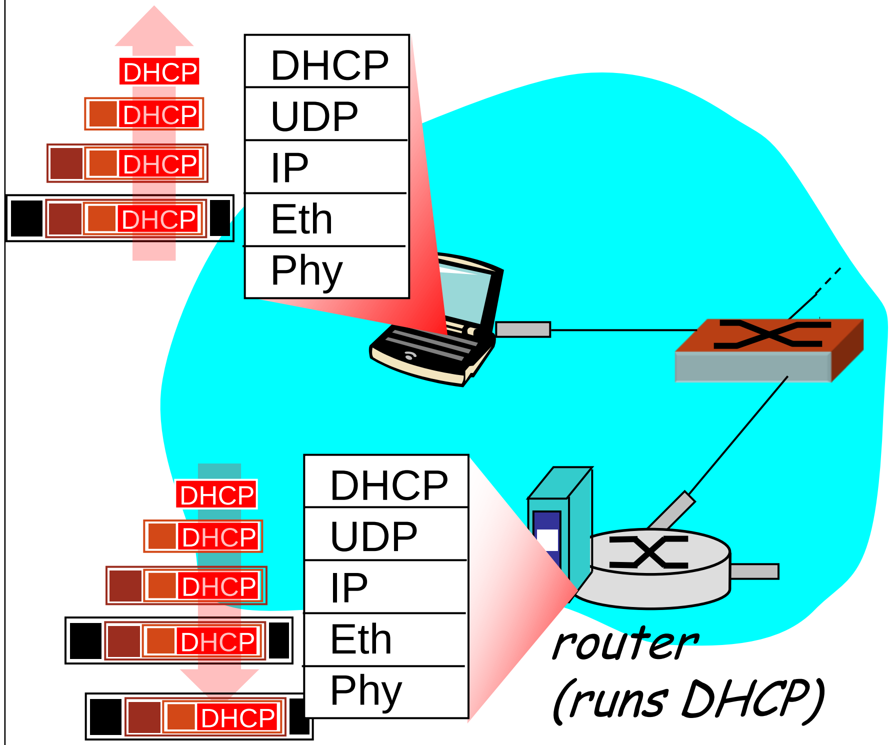
1. Laptop acquires IP address (connects to local network)
   - Using DHCP
2. Acquires IP address of the Web server
   - Using DNS
3. Establishes TCP connection with Web Server
4. Send HTTP request to Web Server
5. Web server sends requested Web page to Laptop

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

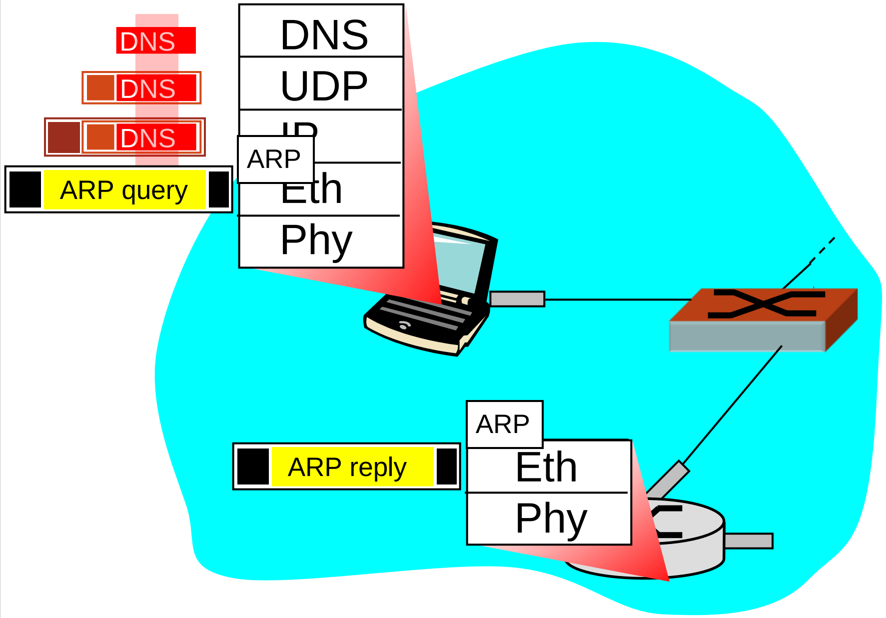
A day in the life... connecting to the Internet



- 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

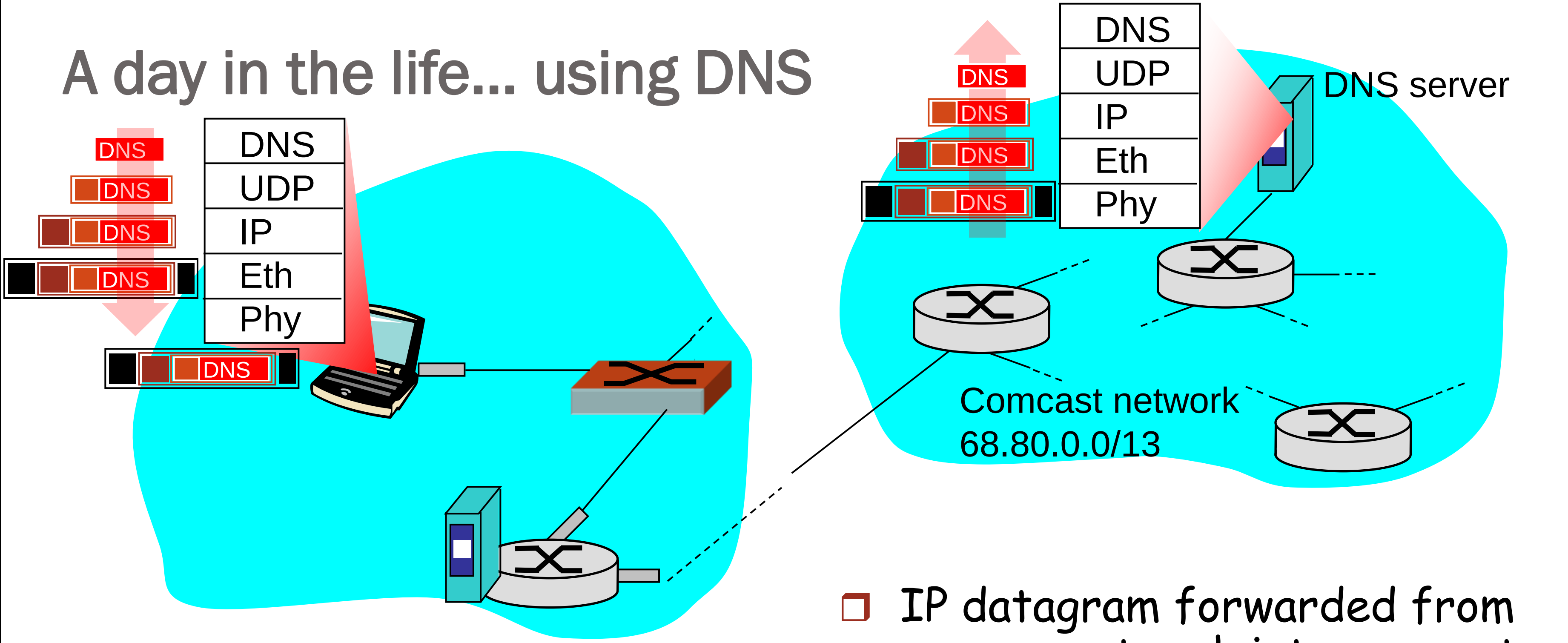
Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

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



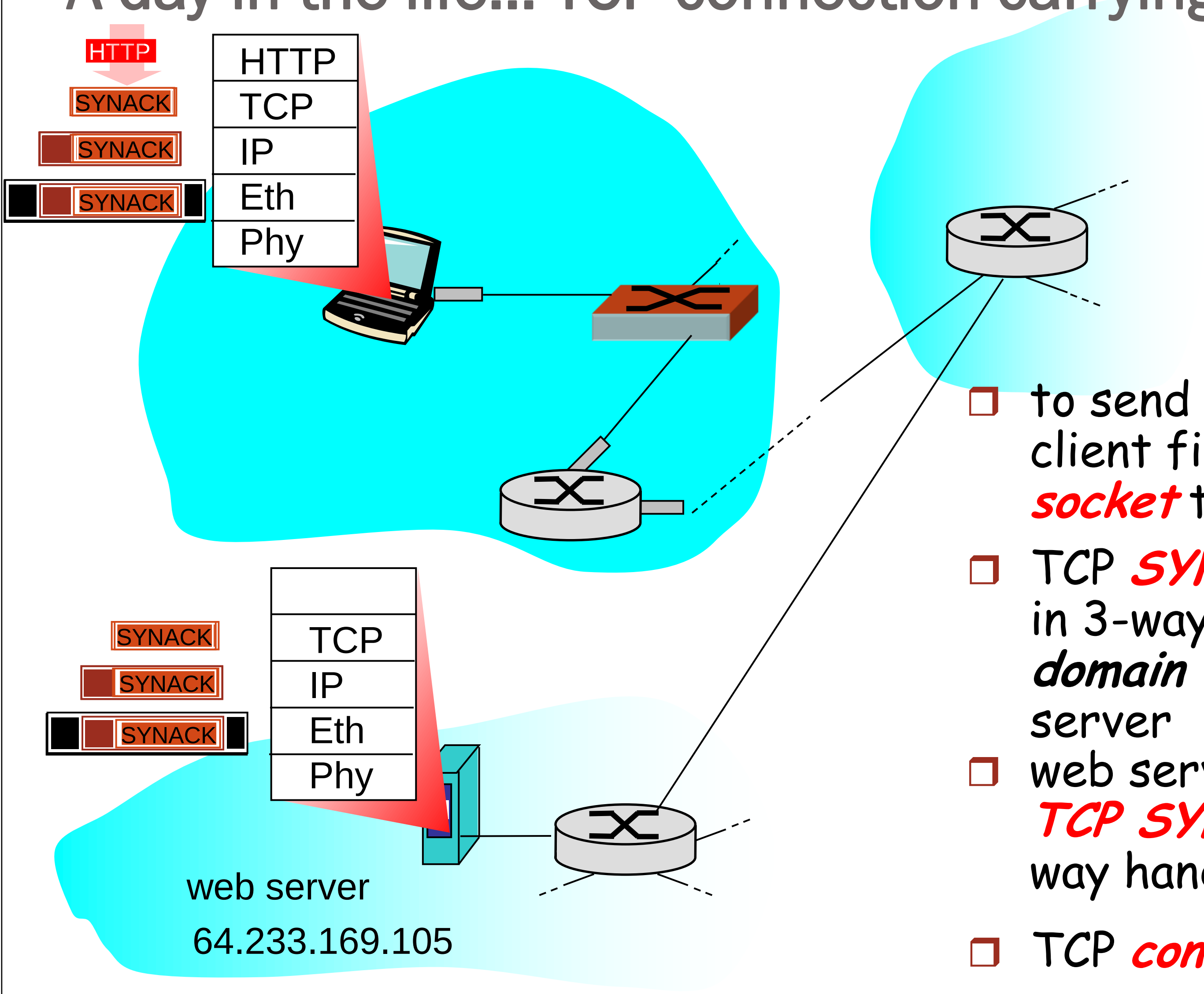
- before sending ***HTTP*** request, need IP address of www.google.com: ***DNS***
- DNS query created, encapsulated in UDP, encapsulated in IP, encasulated in Eth. In order to send frame to router, need MAC address of router interface: ***ARP***
- ***ARP query*** broadcast, received by router, which 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



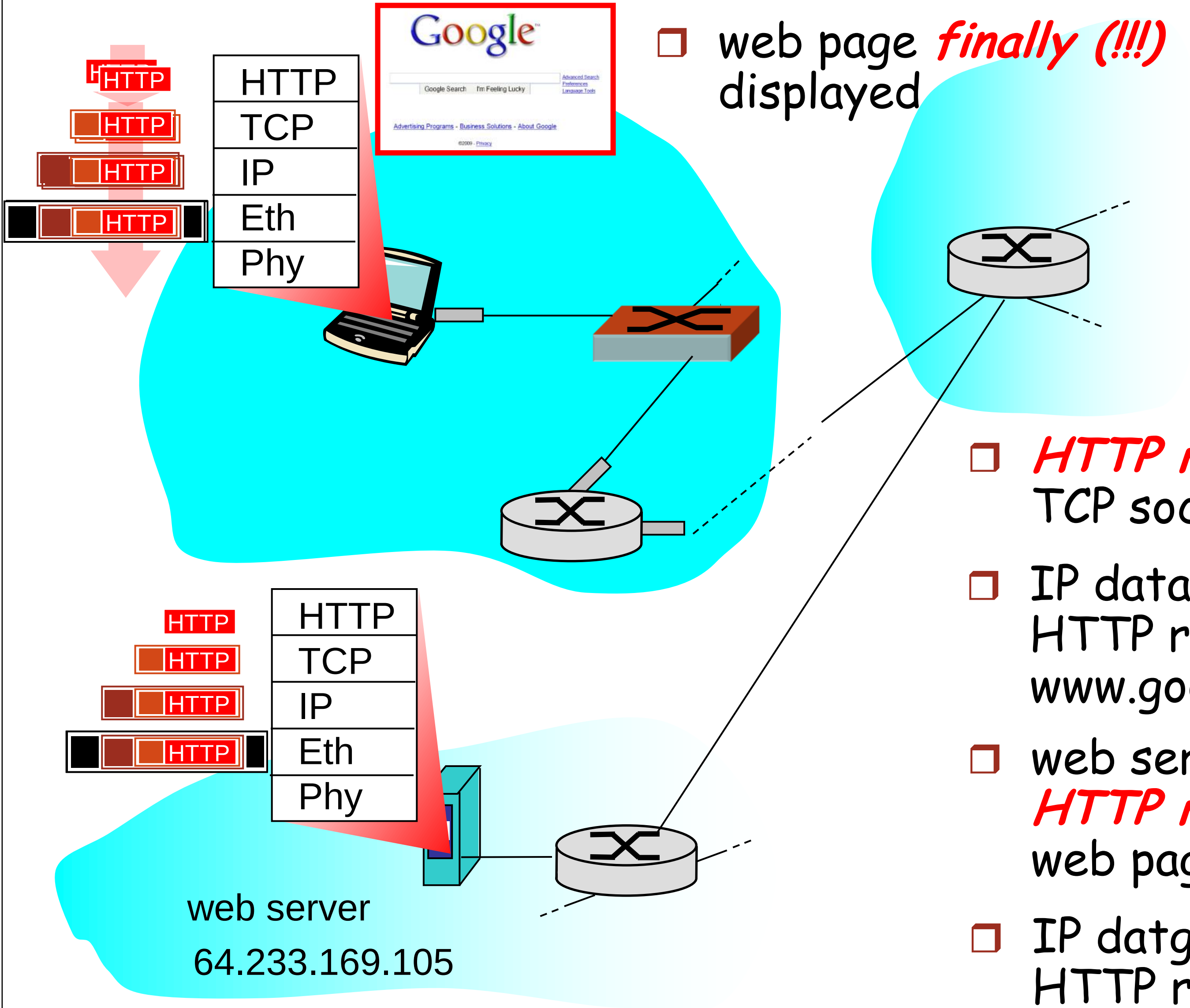
- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
- IP datagram forwarded from campus network into comcast network, routed (tables created by ***RIP, OSPF, IS-IS*** and/or ***BGP*** routing protocols) to DNS server
- demux'ed to DNS server
- DNS server replies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

Summary

- Many Principles/Laws guide computer system design and evolution
  - Helps to understand and master complexity; reduce cost and effort
  - Forgetting them leads to wasted effort, un-manageable systems etc.
- Abstraction is the approach used to design computer systems software and hardware.
- An **abstraction** consists of hierarchical levels (layers) with each lower level hiding details from the level above.
- Layering architecture for complex computer systems
  - End to end design principle to avoid redundant implementation in multiple layers
- Holistic – whole system thinking – is important to maximize performance while minimizing cost
  - Example: TCP/IP Network