Taibah University
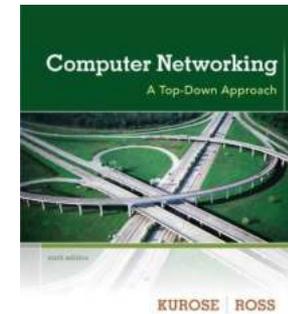**College of Computer Science and Engineering**
Computer Engineering Department

Computer Networking: A Top Down Approach
6th edition
Addison-Wesley
All material copyright 1996-2012
J.F Kurose and K.W. Ross, All Rights Reserved

Computer Networks

Ch2: Application Layer

Chapter 2: outline
Application Layer

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

Chapter 2: application layer

our goals:

❖ conceptual, implementation aspects of network application protocols

- transport-layer service models
- client-server paradigm
- peer-to-peer paradigm

❖ learn about protocols by examining popular application-level protocols

- HTTP
- FTP
- SMTP / POP3 / IMAP
- DNS

❖ creating network applications

- socket API

Some network apps

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)

- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- …
- …

Creating a network app

write programs that:

❖ run on (different) *end systems*

❖ communicate over network

❖ e.g., web server software communicates with browser software

no need to write software for network-core devices

❖ network-core devices do not run user applications

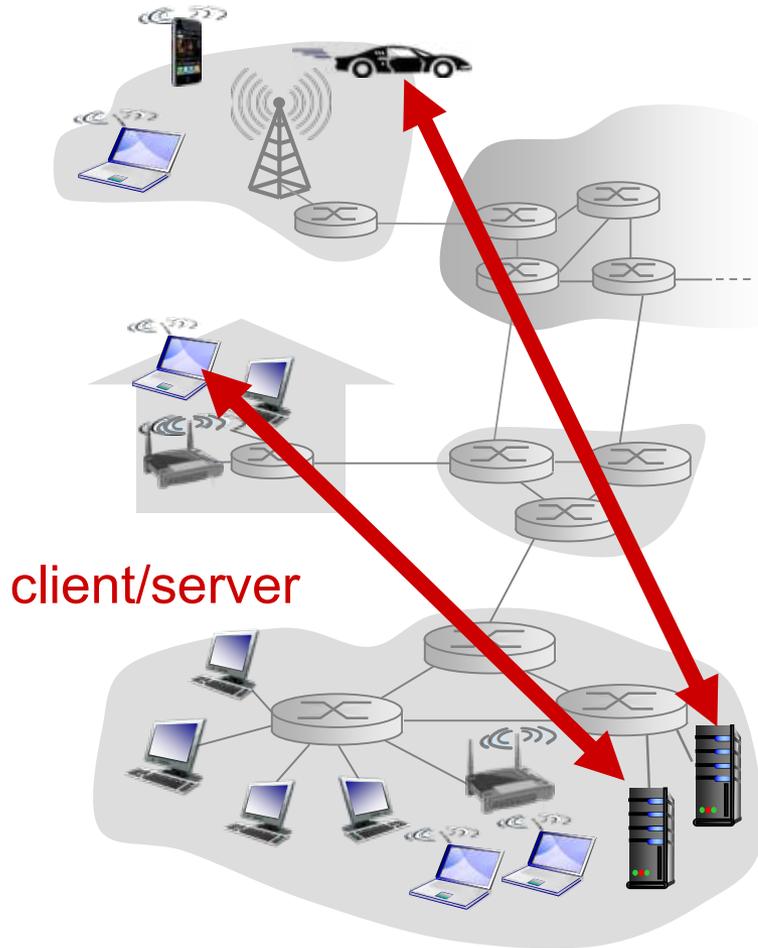❖ applications on end systems allows for rapid app development, propagation
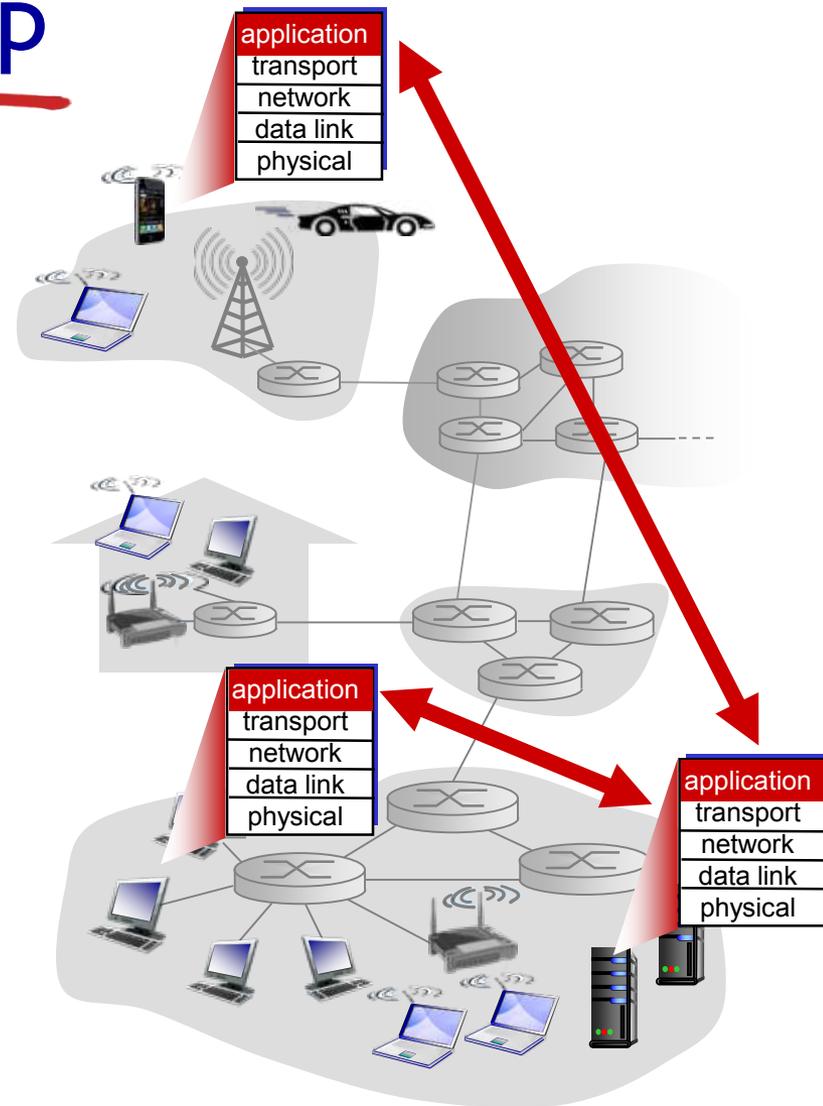
Application architectures

possible structure of applications:

❖ client-server

❖ peer-to-peer (P2P)

# Client-server architecture

client/server

server:
- ❖ always-on host
- ❖ permanent IP address
- ❖ data centers for scaling

clients:
- ❖ communicate with server
- ❖ may be intermittently connected
- ❖ may have dynamic IP addresses
- ❖ do not communicate directly with each other

*Peer-to-Peer (P2P) architecture

❖ *no* always-on server

❖ arbitrary end systems directly communicate

❖ peers request service from other peers, provide service in return to other peers

▪ *self scalability – new peers bring new service capacity, as well as new service demands*

❖ peers are intermittently connected and change IP addresses

▪ complex management

peer-peer
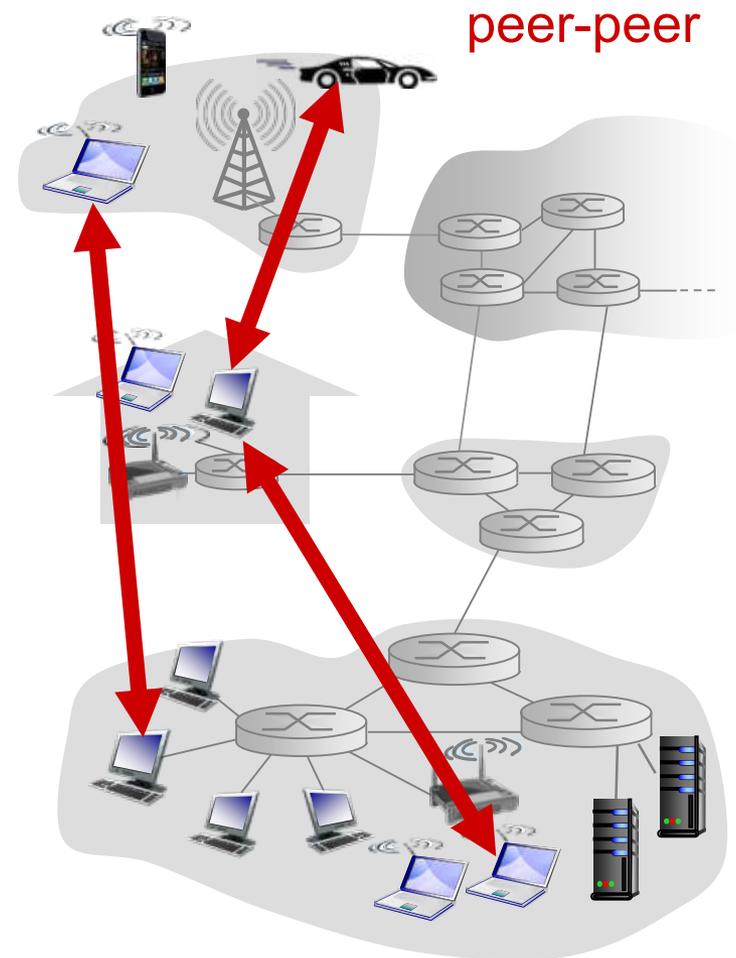
Processes communicating
~~///EA M21sep, C8 Tu 22sep~~

process: program running within a host

❖ within same host, two processes communicate using inter-process communication (defined by OS)

❖ processes in different hosts communicate by exchanging messages

clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

❖ aside: applications with P2P architectures have client processes & server processes

Sockets

❖ process sends/receives messages to/from its socket

❖ socket analogous to door

■ sending process shoves (pushes or pulls) message out door

■ sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
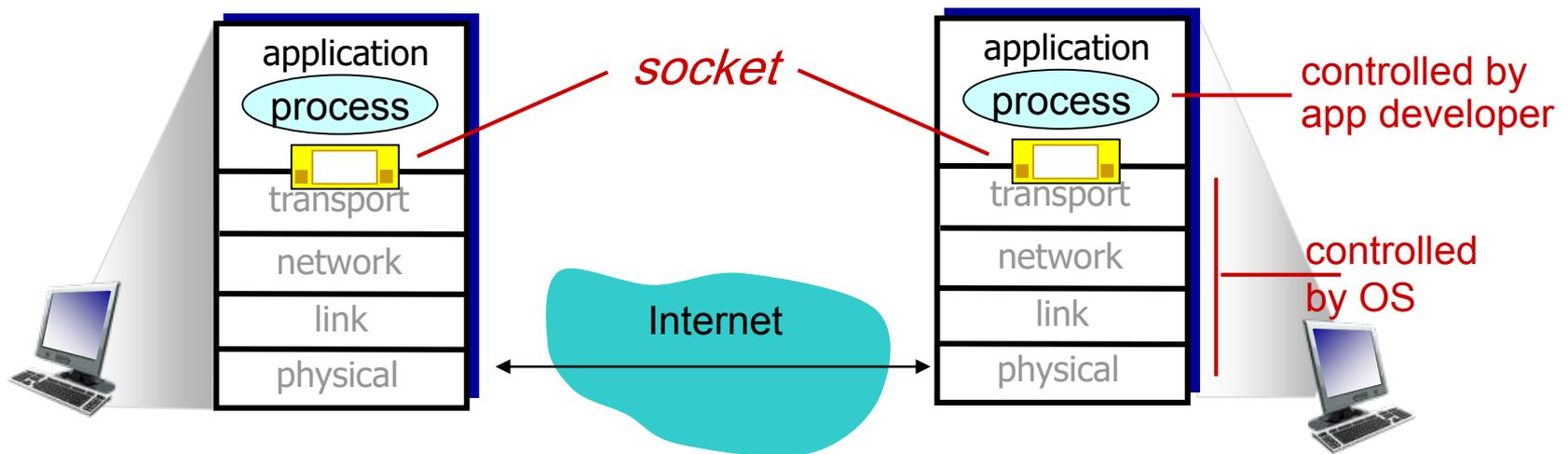
Addressing processes

❖ to receive messages, process must have *identifier*

❖ host device has unique 32-bit IP address

❖ *Q:* does IP address of host on which process runs be enough for identifying the process?

  ▪ *A:* no, *many* processes can be running on same host

❖ *identifier* includes both IP address and **port numbers** associated with process on host.

❖ example port numbers:

  ▪ HTTP server: 80
  ▪ mail server: 25

❖ to send HTTP message to gaia.cs.umass.edu web server:

  ▪ IP address: 128.119.245.12
  ▪ port number: 80

❖ more shortly…

App-layer protocol defines

- ❖ types of messages exchanged,
  - ▪ e.g., request, response
- ❖ message syntax:
  - ▪ what fields in messages & how fields are delineated
- ❖ message semantics
  - ▪ meaning of information in fields
- ❖ rules for when and how processes send & respond to messages

open protocols:
- ❖ defined in RFCs
- ❖ allows for interoperability
- ❖ e.g., HTTP, SMTP

proprietary protocols:
- ❖ e.g., Skype

What transport service does an app need?

data integrity

- ❖ some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- ❖ other apps (e.g., audio) can tolerate some loss

timing

- ❖ some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

throughput

- ❖ some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- ❖ other apps ("elastic apps") make use of whatever throughput they get

security

- ❖ encryption, data integrity, …

Transport service requirements: common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
text messaging	no loss	elastic	yes and no

Internet transport protocols services

~~///CA14,M M21sep~~

TCP service:

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security
- *connection-oriented:* setup required between client and server processes

UDP service:

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,

Q: why bother? Why is there a UDP?

Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Web and HTTP: hypertext transfer protocol

///C8 Sun 27Sep, EA M 28Sep

First, a review…

❖ *web page* consists of *objects*

❖ object can be HTML file, JPEG image, Java applet, audio file,…

❖ web page consists of *base HTML-file* which includes *several referenced objects*

❖ each object is addressable by a *URL,* e.g.,

```
www.someschool.edu/someDept/pic.gif
```

host name   path name

HTTP overview

HTTP: hypertext transfer protocol

❖ Web's application layer protocol

❖ client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests

PC running
Firefox browser

HTTP request

HTTP response

server running Apache Web server

HTTP request

HTTP response

iphone running
Safari browser

*HTTP overview (continued)

uses TCP:

❖ client initiates TCP connection (creates socket) to server, port 80

❖ server accepts TCP connection from client

❖ HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)

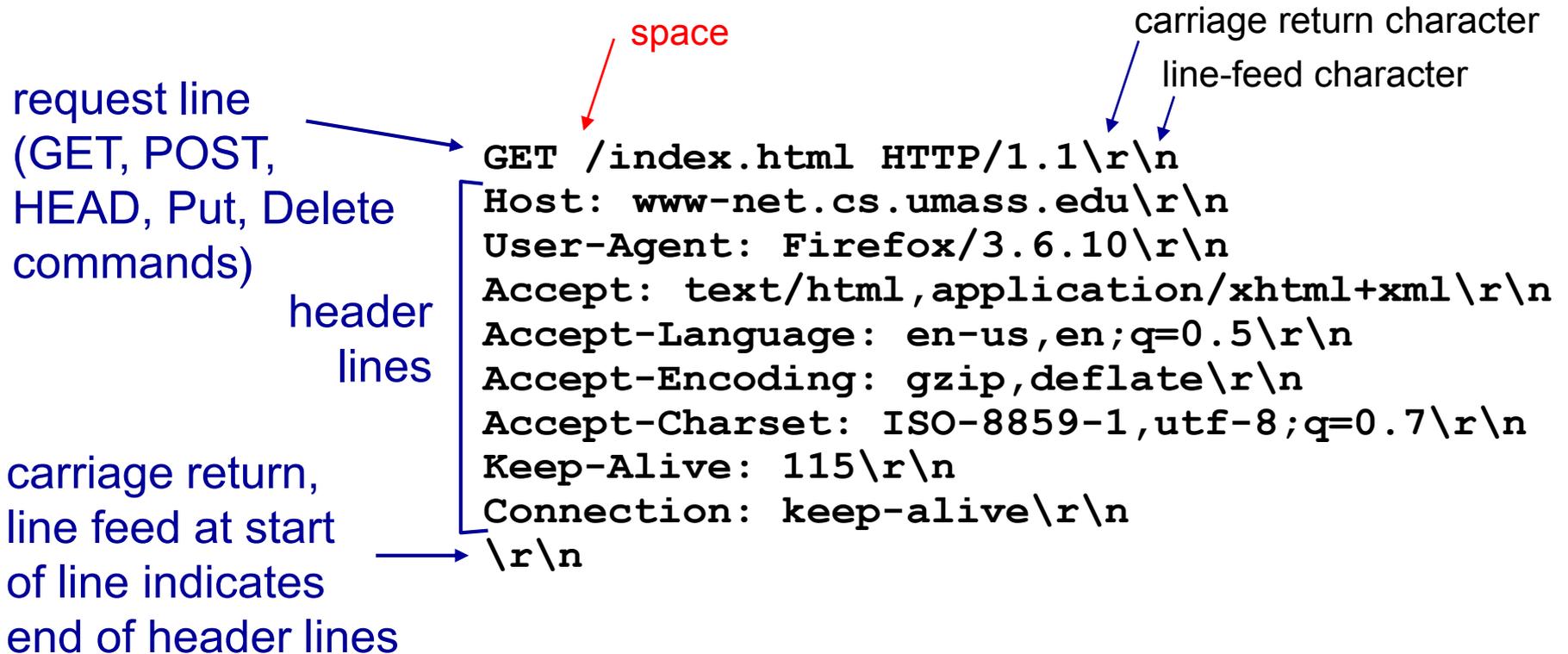❖ TCP connection closed

HTTP is "stateless"

❖ server maintains no information about past client requests

aside

protocols that maintain "state" (State-full) are complex!

❖ past history (state) must be maintained

❖ if server/client crashes, their views of "state" may be inconsistent, must be reconciled

*HTTP request message

❖ two types of HTTP messages: *request, response*

❖ HTTP request message:
   ▪ ASCII (human-readable format)

space

carriage return character

line-feed character

request line
(GET, POST,
HEAD, Put, Delete
commands)

header
lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

carriage return,
line feed at start
of line indicates
end of header lines

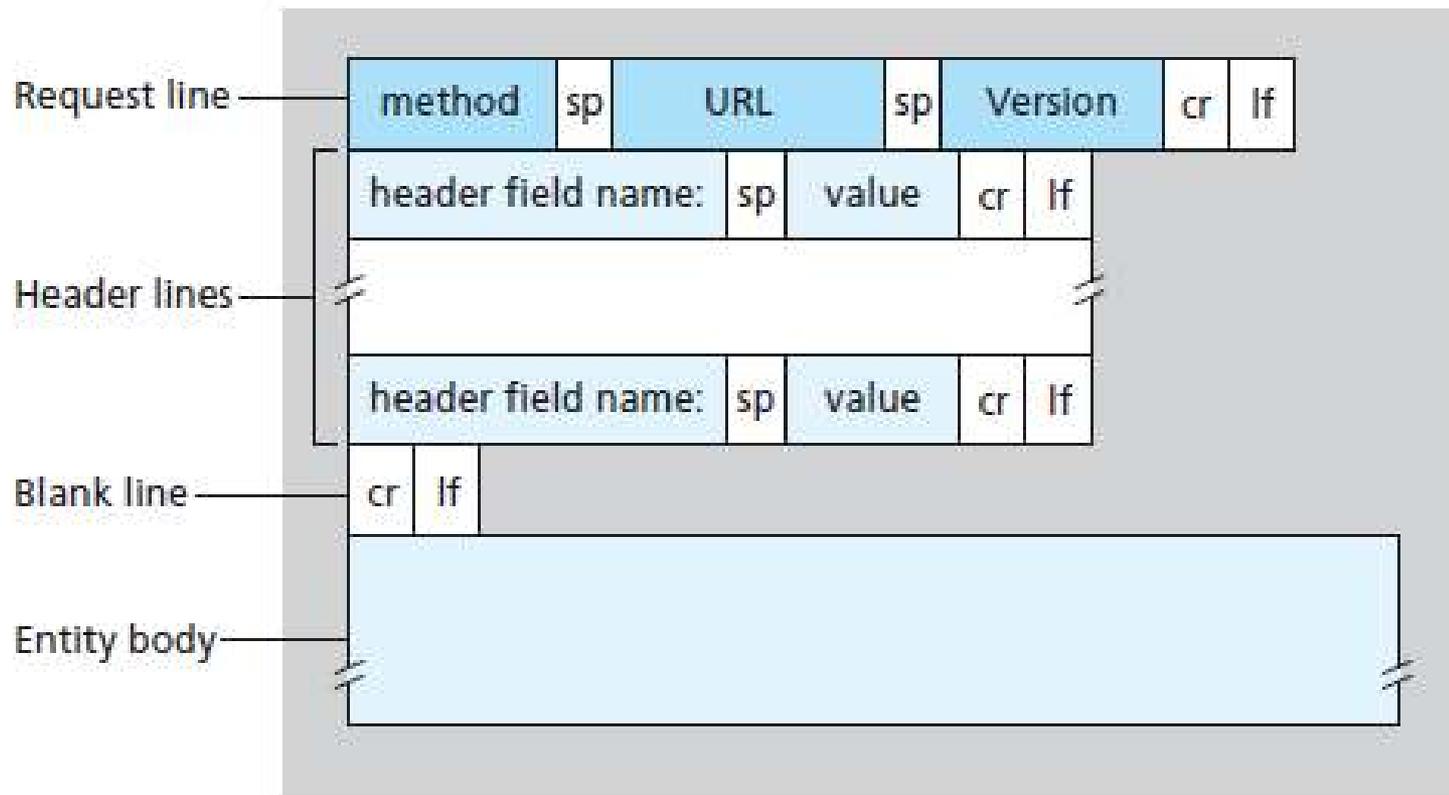# Format of an HTTP request message

Figure 2.8 ◆ General format of an HTTP request message

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

telnet cis.poly.edu 80   opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. anything typed is sent to port 80 at cis.poly.edu

2. type in a GET HTTP request:

GET /~ross/ HTTP/1.1
Host: cis.poly.edu
Connection: Close
User-agent: Mozilla/5.0
Accept-language: fr

   by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

HTTP response message

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

header
lines

data, e.g.,
requested
HTML file

HTTP response status codes

❖ status code appears in 1st line in server-to-client response message.

❖ some sample codes:

200 OK
  ▪ request succeeded, requested object later in this msg

301 Moved Permanently
  ▪ requested object moved, new location specified later in this msg (Location:)

400 Bad Request
  ▪ request msg not understood by server

404 Not Found
  ▪ requested document not found on this server

505 HTTP Version Not Supported

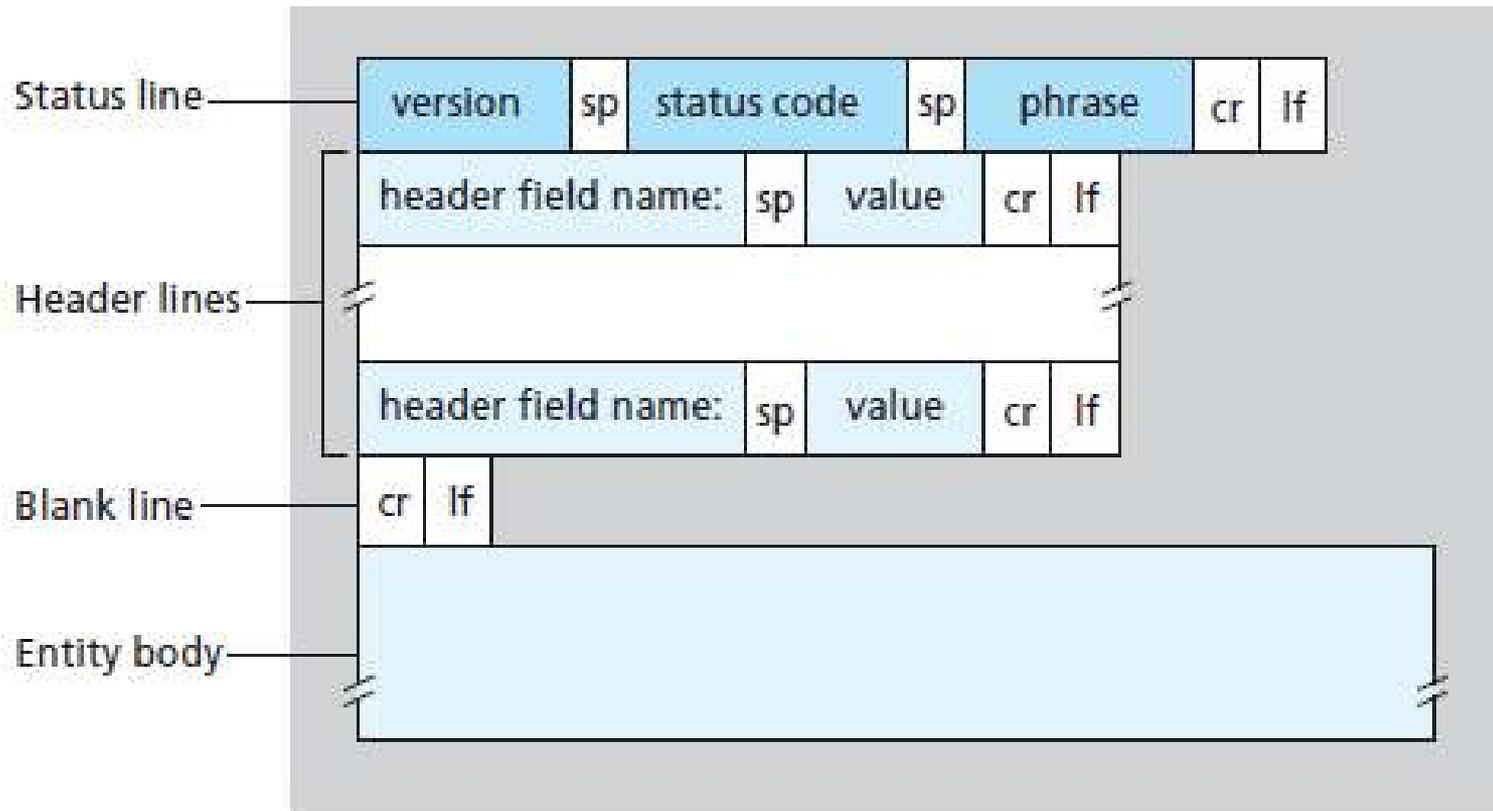# Format of an HTTP response message

Figure 2.9 ◆ General format of an HTTP response message

Web caches (proxy server)

goal: satisfy client request without involving origin server

❖ user sets browser: Web accesses via cache
❖ browser sends all HTTP requests to cache
  ▪ object in cache: cache returns object
  ▪ else cache requests object from origin server, then returns object to client
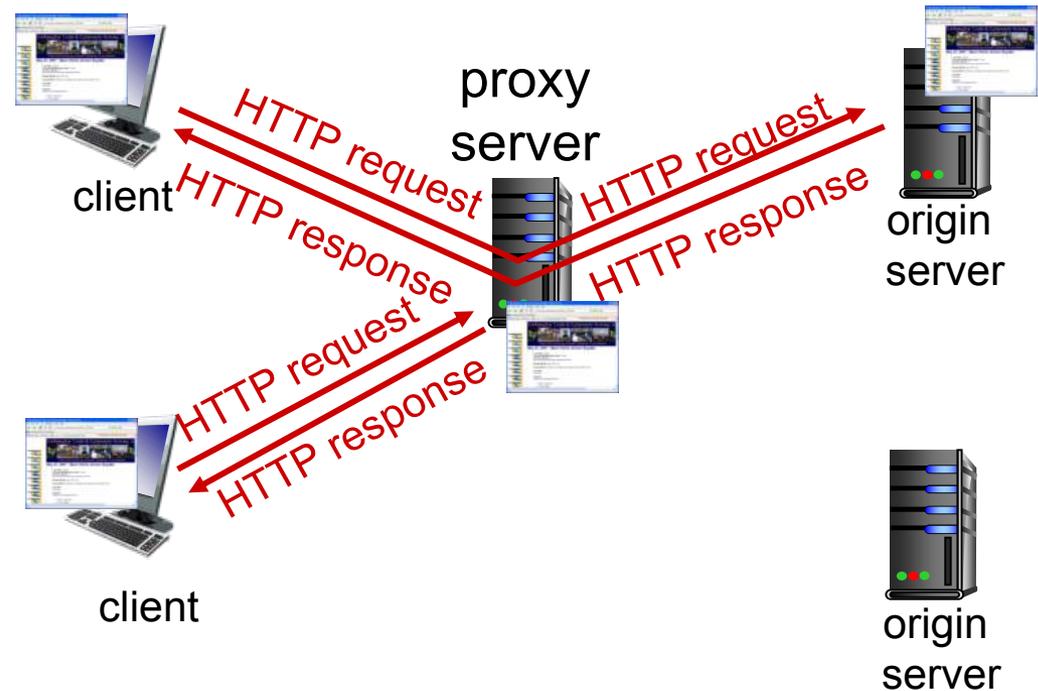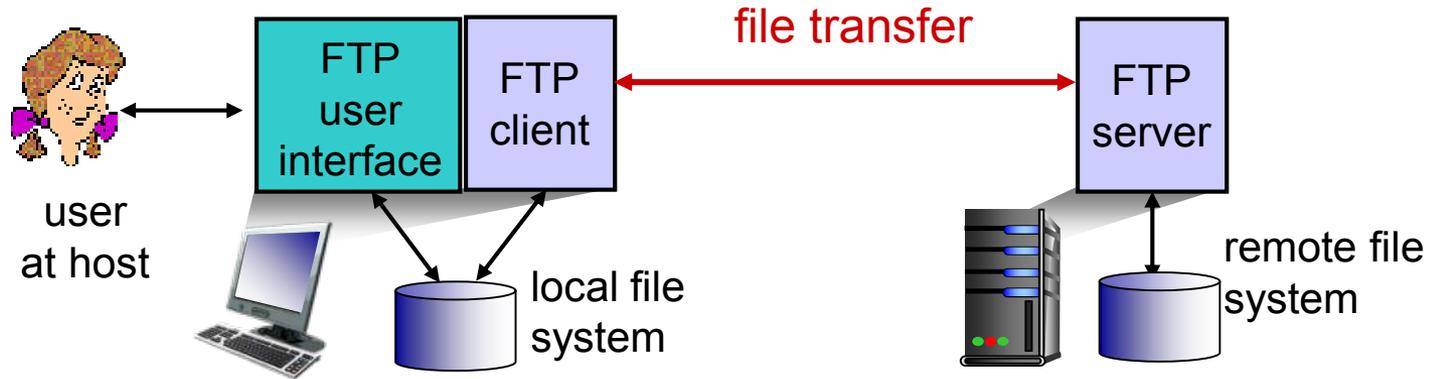
More about Web caching

* cache acts as both client and server
  * server for original requesting client
  * client to origin server
* typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

* reduce response time for client request
* reduce traffic on an institution's access link
* Internet dense with caches: enables "poor" content providers to effectively deliver content.

*FTP: the file transfer protocol
///CA14, M M 28Sep, C8 Tu 29 sep

file transfer

FTP user interface | FTP client

FTP server

user at host

local file system

remote file system

- ❖ transfer file to/from remote host
- ❖ client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ❖ ftp: RFC 959
- ❖ ftp server: ports 20, 21

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, using TCP
- client authorized over control connection
- client browses remote directory, sends commands over control connection
- when server receives file transfer command, *server* opens 2*nd* TCP data connection (for file) *to* client
- after transferring one file, server closes data connection

TCP control connection, server port 21

TCP data connection, server port 20

FTP client   FTP server
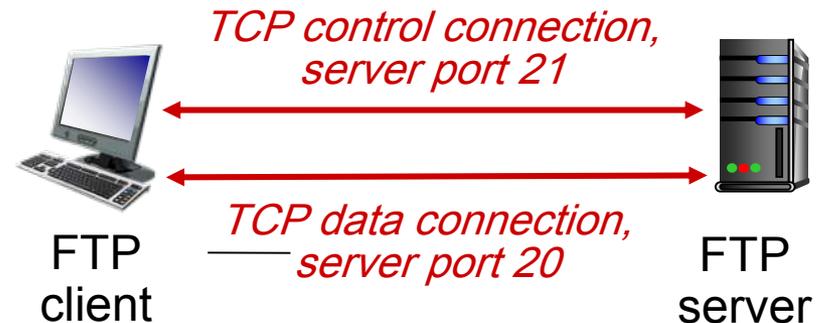
- server opens another TCP data connection to transfer another file
- control connection: *"out of band"*
- FTP server maintains "state": current directory, earlier authentication

/FTP commands, responses

sample commands:

- ❖ sent as ASCII text over control channel
- ❖ **USER *username***
- ❖ **PASS *password***
- ❖ **LIST** return list of file in current directory
- ❖ **RETR filename** retrieves (gets) file
- ❖ **STOR filename** stores (puts) file onto remote host

sample return codes

- ❖ status code and phrase (as in HTTP)
- ❖ **331 Username OK, password required**
- ❖ **125 data connection already open; transfer starting**
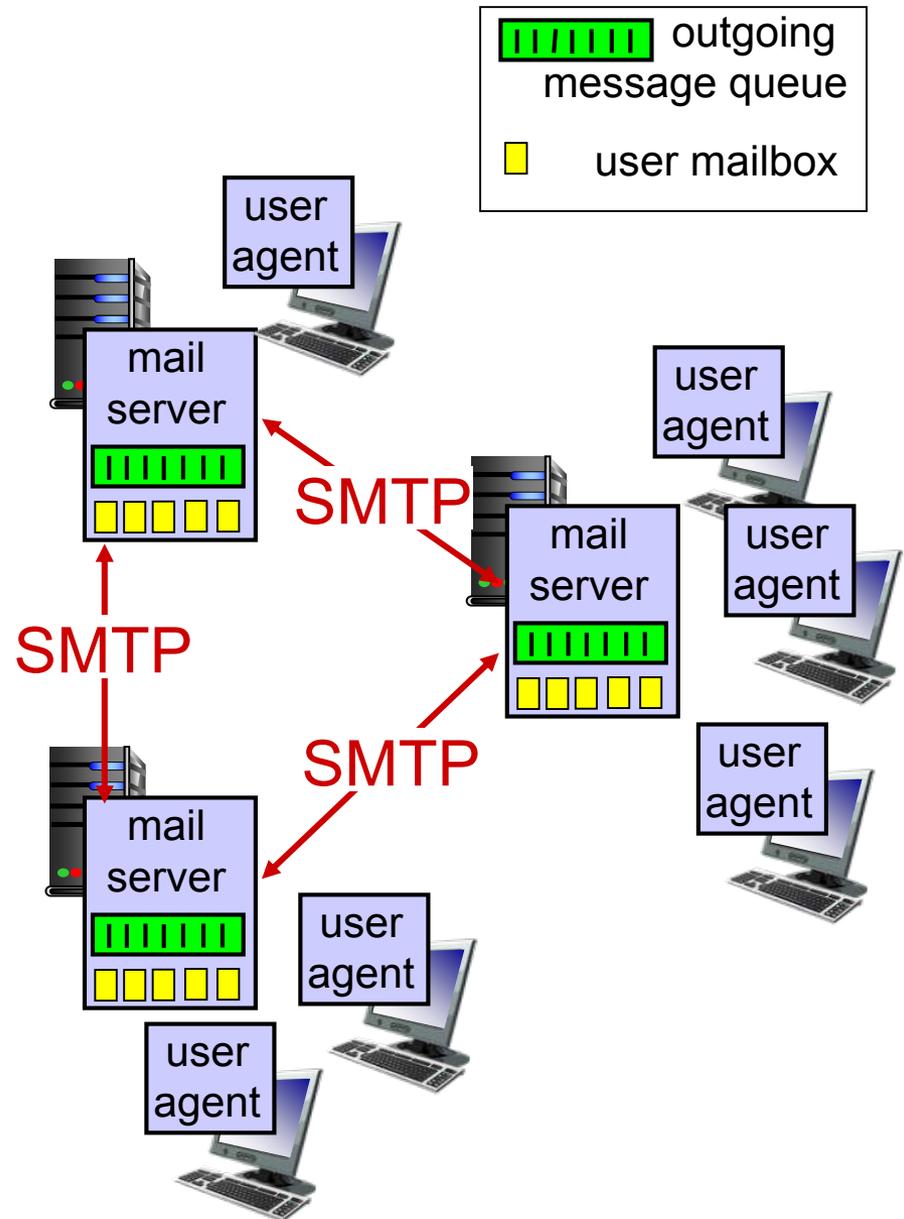- ❖ **425 Can't open data connection**
- ❖ **452 Error writing file**

*Electronic mail

Three major components:

❖ user agents

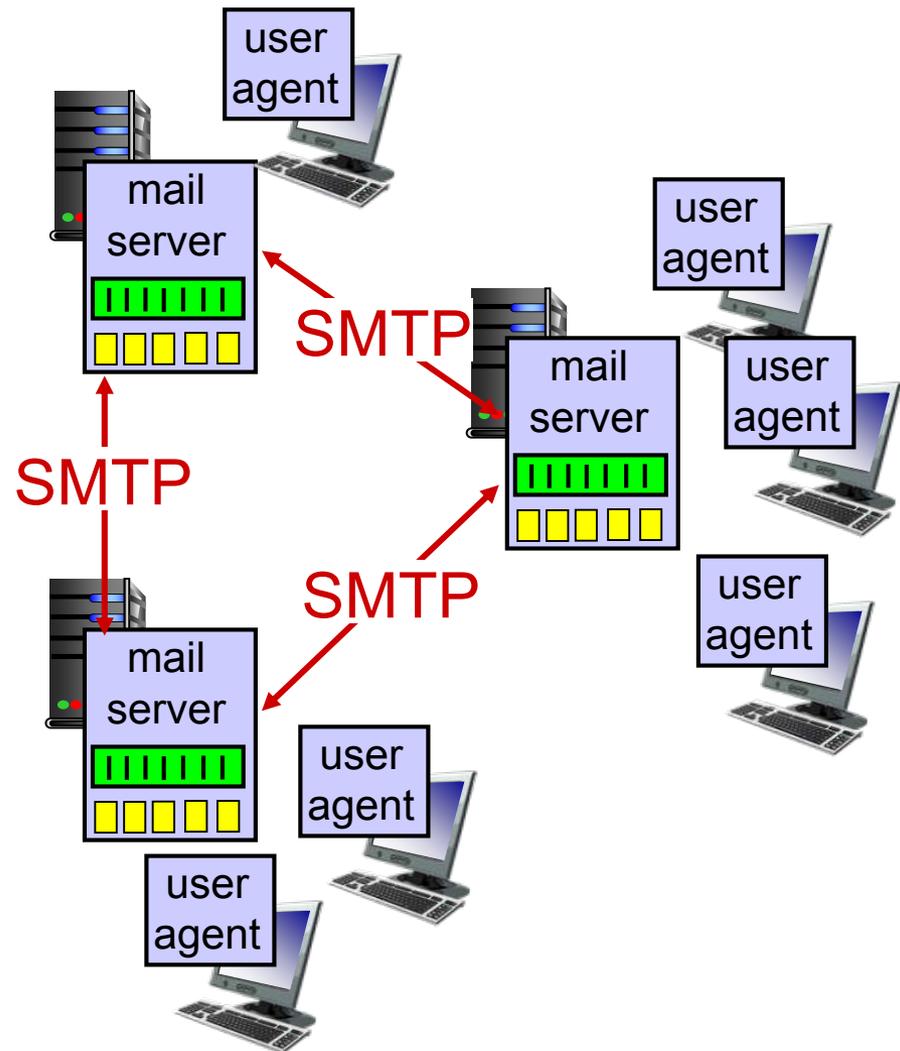❖ mail servers

❖ simple mail transfer protocol: SMTP

User Agent

❖ ~~a.k.a. "mail reader"~~

❖ composing, editing, reading mail messages

❖ e.g., Outlook, Thunderbird, iPhone mail client

❖ outgoing, incoming messages stored on server

outgoing message queue

user mailbox

SMTP

SMTP

SMTP

Electronic mail: mail servers

mail servers:

❖ *mailbox* contains incoming messages for user

❖ *message queue* of outgoing (to be sent) mail messages

❖ *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server

*Electronic Mail: SMTP [RFC 2821]

❖ uses TCP to reliably transfer email message from client to server, port 25

❖ direct transfer: sending server to receiving server

❖ three phases of transfer
  ▪ handshaking (greeting)   (Connection Setup)
  ▪ transfer of messages
  ▪ closure

❖ command/response interaction (like HTTP, FTP)
  ▪ commands: ASCII text
  ▪ response: status code and phrase

❖ messages must be in 7-bit ASCII

*Scenario: Alice sends message to Bob

1) Alice uses UA to compose message "to" `bob@someschool.edu`

2) Alice's UA sends message to her mail server; message placed in message queue

3) client side of SMTP opens TCP connection with Bob's mail server

4) SMTP client sends Alice's message over the TCP connection

5) Bob's mail server places the message in Bob's mailbox

6) Bob invokes his user agent to read message

Mail Access Protocol: POP or IMAP

Alice

Alice's mail server

Bob's mail server

Bob
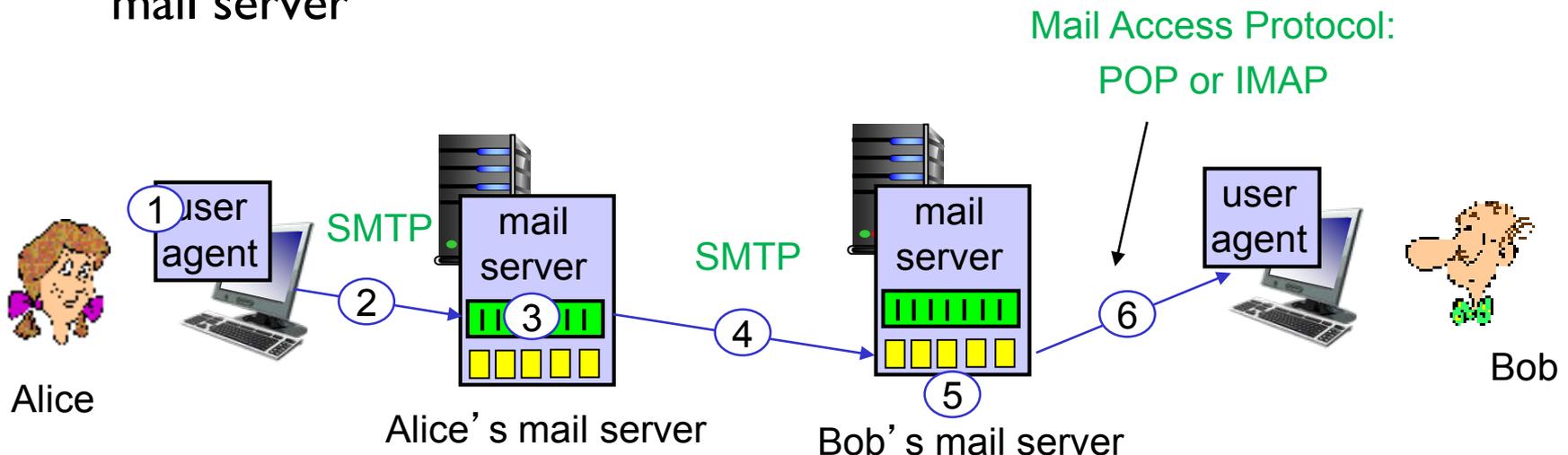
/Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

*SMTP: final words

comparison with HTTP: (page 124)

- ❖ SMTP uses persistent connections
- ❖ SMTP requires message (header & body) to be in 7-bit ASCII
- ❖ SMTP server uses `CRLF.CRLF` to determine end of message

- ❖ HTTP: pull
  - A websites loaded on a Web server and users use HTTP to pull the information from the server at their convenience.
- ❖ SMTP: push
  - the sending mail server pushes the file to the receiving mail server

- ❖ Both have ASCII command/response interaction, status codes

- ❖ HTTP: if a page has multiple objects*, each object encapsulated in its own response msg
- ❖ SMTP: if a message has multiple objects*, they are sent in ONE msg

*multiple objects: Text, Image, other media types

Mail message format

SMTP: protocol for exchanging email msgs
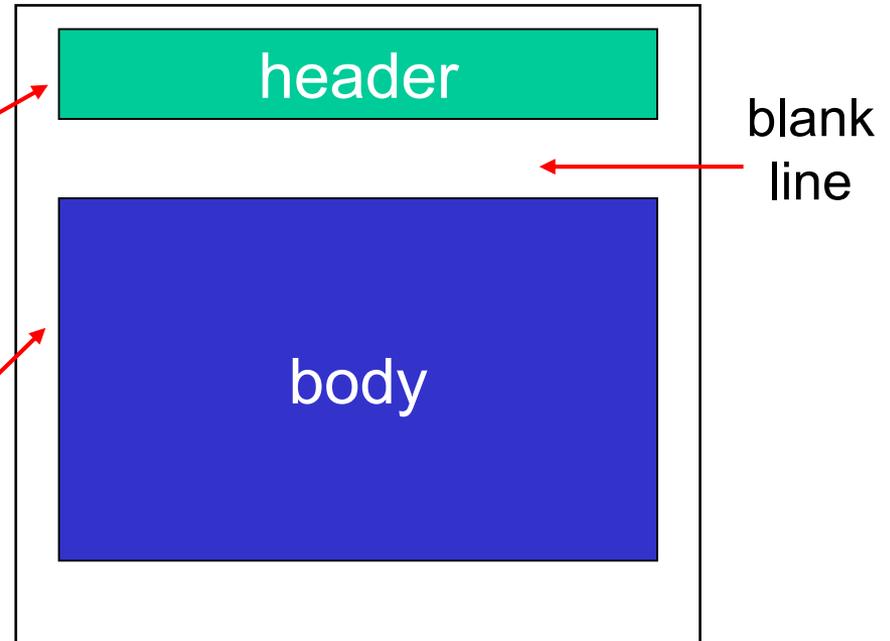
RFC 822: standard for text message format:

❖ header lines, e.g.,
- To:
- From:
- Subject:

different from SMTP MAIL FROM, RCPT TO: commands!

❖ Body: the "message"
- ASCII characters only

header

blank line

body

*Mail access protocols
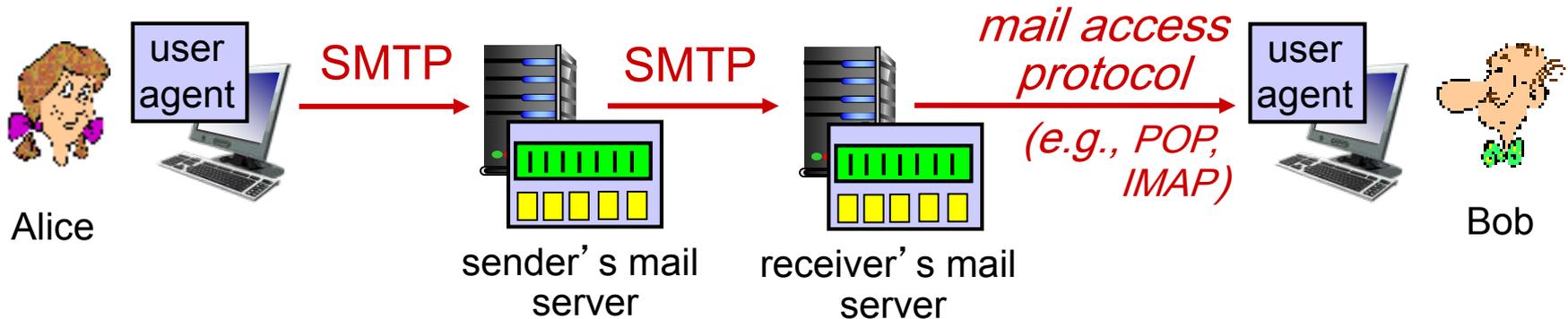
❖ **SMTP:**
- push sender's e-mail message into his/her mail server
- delivery/storage to receiver's mail server

❖ mail access protocol: retrieval from receiver's mail server
- **POP:** Post Office Protocol [RFC 1939]: authorization, download
- **IMAP:** Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
- **HTTP:** gmail, Hotmail, Yahoo! Mail, etc.

/POP3 protocol

authorization phase

- ❖ client commands:
    - ▪ **user**: declare username
    - ▪ **pass**: password
- ❖ server responses
    - ▪ **+OK**
    - ▪ **-ERR**

transaction phase, client:

- ❖ **list**: list message numbers
- ❖ **retr**: retrieve message by number
- ❖ **dele**: delete
- ❖ **quit**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 2 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (more) and IMAP

more about POP3

❖ previous example uses POP3 "download and delete" mode
  ▪ Bob cannot re-read e-mail if he changes client
❖ POP3 "download-and-keep": copies of messages on different clients
❖ POP3 is stateless across sessions

IMAP

❖ keeps all messages in one place: at server
❖ allows user to organize messages in folders
❖ keeps user state across sessions:
  ▪ names of folders and mappings between message IDs and folder name

DNS: domain name system
~~///EA, CA14, M Wed 30Sep~~

people: many identifiers:
- SSN, name, passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System:

- ❖ *distributed database* implemented in hierarchy of many *name servers*
- ❖ *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

DNS: services, structure

DNS services

❖ hostname to IP address translation
❖ host aliasing
  ▪ canonical, alias names
❖ mail server aliasing
❖ load distribution
  ▪ replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

❖ single point of failure
❖ traffic volume
❖ distant centralized database
❖ maintenance

A: *doesn't scale!*

*DNS: a distributed, hierarchical database

Root DNS

Root DNS Servers

… | …

TLD

com DNS servers   org DNS servers   edu DNS servers

yahoo.com   amazon.com   pbs.org   poly.edu   umass.edu
DNS servers   DNS servers   DNS servers   DNS serversDNS servers

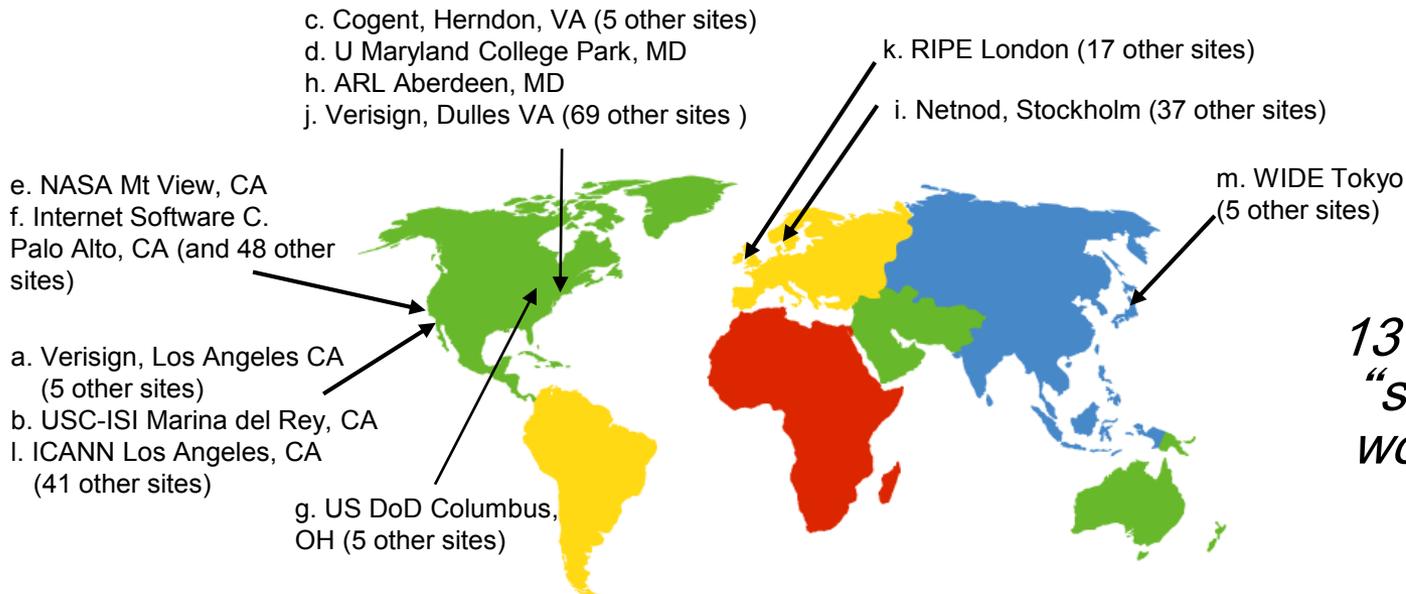**LNS or ANS**

client wants IP for www.amazon.com; 1st approx:

❖ client queries 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

TLD: Top Level Domain Server
LNS: Local Name Server = ANS: Authoritative Name Server

*DNS: **root name servers**

❖ contacted by **local name server** that can not resolve name

❖ root name server:

- contacts **authoritative name server** if name mapping not known

- gets mapping

- returns mapping to local name server

c. Cogent, Herndon, VA (5 other sites)
d. U Maryland College Park, MD
h. ARL Aberdeen, MD
j. Verisign, Dulles VA (69 other sites)

k. RIPE London (17 other sites)

i. Netnod, Stockholm (37 other sites)

e. NASA Mt View, CA
f. Internet Software C.
Palo Alto, CA (and 48 other sites)

m. WIDE Tokyo
(5 other sites)

a. Verisign, Los Angeles CA
   (5 other sites)
b. USC-ISI Marina del Rey, CA
l. ICANN Los Angeles, CA
   (41 other sites)

g. US DoD Columbus,
OH (5 other sites)

13 root name "servers" worldwide

TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider
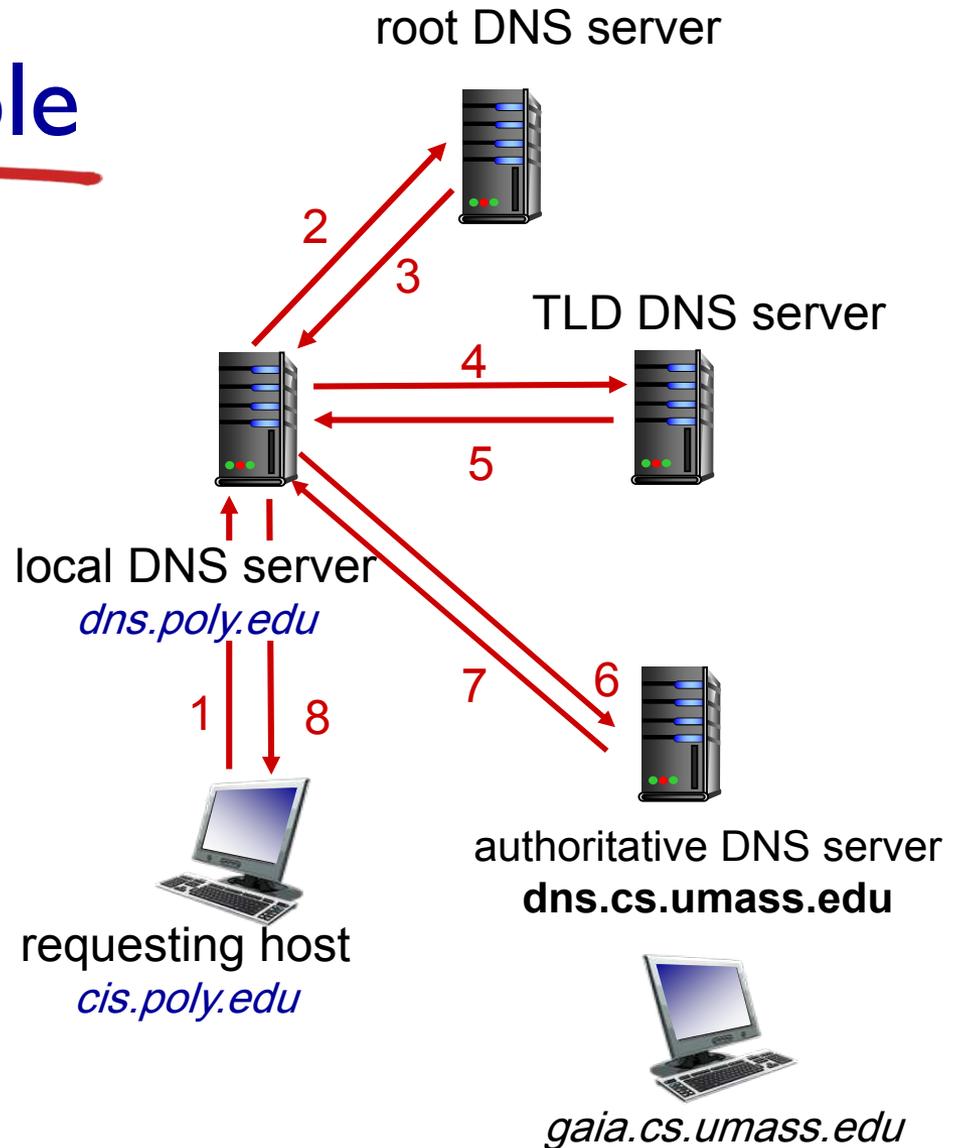
Local DNS name server

- ❖ does not strictly belong to hierarchy
- ❖ each ISP (residential ISP, company, university) has one
  - ▪ also called "default name server"
- ❖ when host makes DNS query, query is sent to its local DNS server
  - ▪ has local cache of recent name-to-address translation pairs (but may be out of date!)
  - ▪ acts as proxy, forwards query into hierarchy

DNS name resolution example

❖ host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

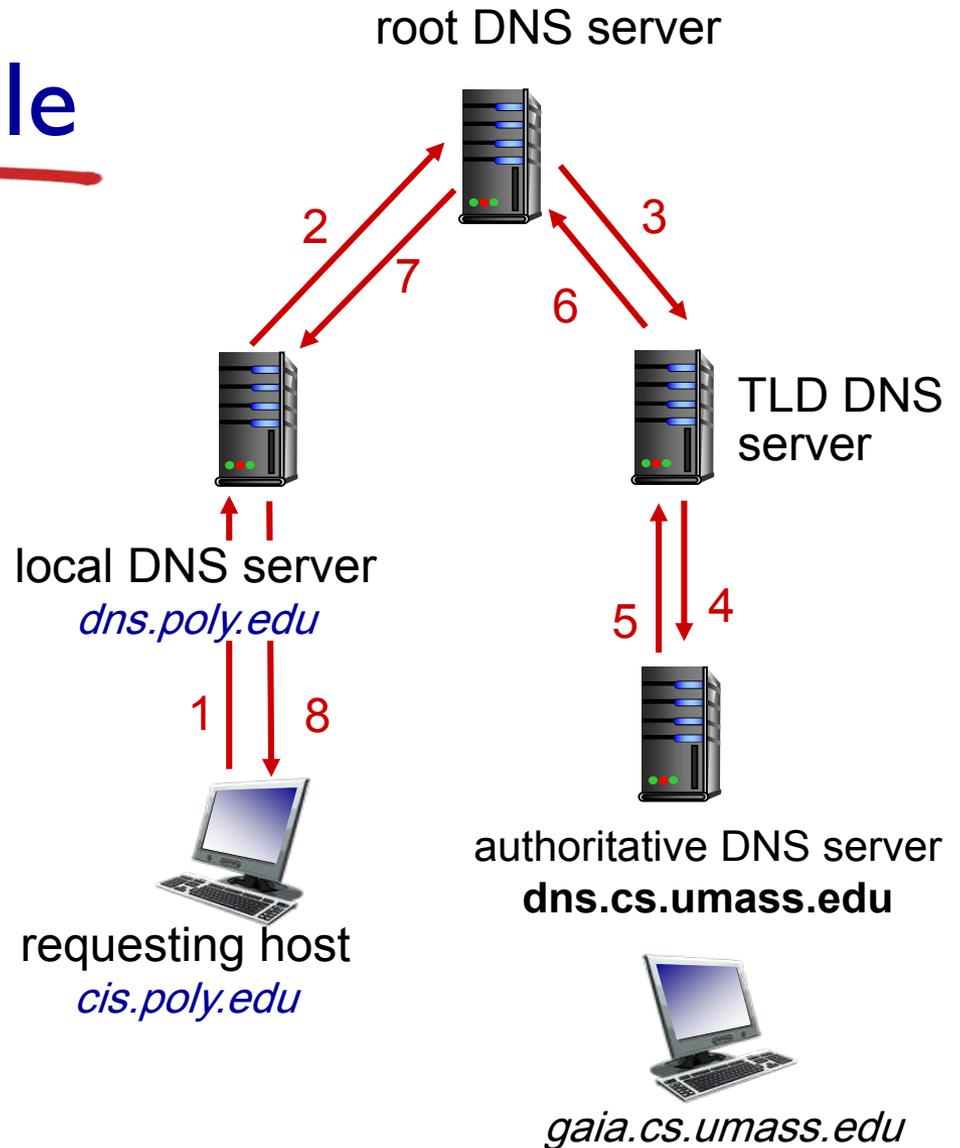❖ contacted server replies with name of server to contact

❖ "I don't know this name, but ask this server"

root DNS server

TLD DNS server

local DNS server
dns.poly.edu

authoritative DNS server
dns.cs.umass.edu

requesting host
cis.poly.edu

gaia.cs.umass.edu

DNS name resolution example

recursive query:

❖ puts burden of name resolution on contacted name server

❖ heavy load at upper levels of hierarchy?

root DNS server

local DNS server
dns.poly.edu

TLD DNS server

authoritative DNS server
dns.cs.umass.edu

requesting host
cis.poly.edu

gaia.cs.umass.edu

2
7
3
6
1
8
5
4

DNS: caching, updating records

❖ once (any) name server learns mapping, it *caches* mapping
  ▪ cache entries timeout (disappear) after some time (TTL)
  ▪ TLD servers typically cached in local name servers
    • thus root name servers not often visited

❖ cached entries may be *out-of-date* (best effort name-to-address translation!)
  ▪ if name host changes IP address, may not be known Internet-wide until all TTLs expire

❖ update/notify mechanisms proposed IETF standard
  ▪ RFC 2136

DNS records

DNS: distributed db storing resource records (RR)

> RR format: `(name, value, type, ttl)`

type=A

- `name` is hostname
- `value` is IP address

type=NS

- `name` is domain (e.g., foo.com)
- `value` is hostname of authoritative name server for this domain

type=CNAME

- `name` is alias name for some "canonical" (the real) name
- `www.ibm.com` is really `servereast.backup2.ibm.com`
- `value` is canonical name

type=MX

- `value` is name of mailserver associated with `name`

Chapter 2: summary

our study of network apps now complete!

- ❖ application architectures
  - client-server
  - P2P
- ❖ application service requirements:
  - reliability, bandwidth, delay
- ❖ Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP

- ❖ specific protocols:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
- ❖ socket programming: TCP, UDP sockets

Chapter 2: summary

most importantly: learned about protocols!

❖ typical request/reply message exchange:
- client requests info or service
- server responds with data, status code

❖ message formats:
- headers: fields giving info about data
- data: info being communicated

important themes:

❖ control vs. data msgs
- in-band, out-of-band

❖ centralized vs. decentralized

❖ stateless vs. stateful

❖ reliable vs. unreliable msg transfer

❖ "complexity at network edge"