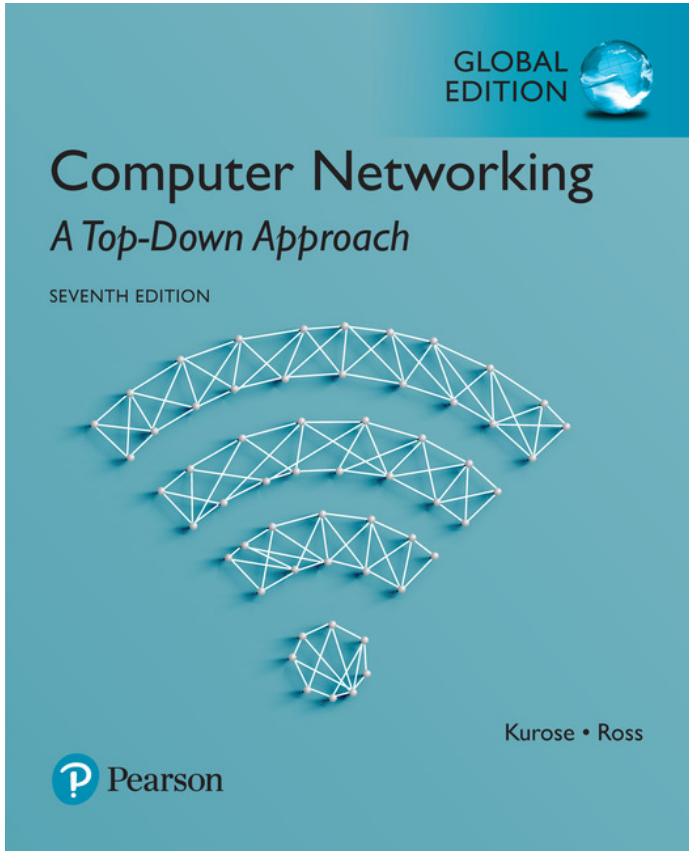
Chapter 2
Application Layer



A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and 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) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides 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

7th Edition, Global Edition
Jim Kurose, Keith Ross
Pearson
April 2016

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

Chapter 2: application layer

our goals:

- conceptual and 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 applications

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (such as YouTube, Hulu, Netflix, SVT Play, …)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- Cloud computing and storage
- …

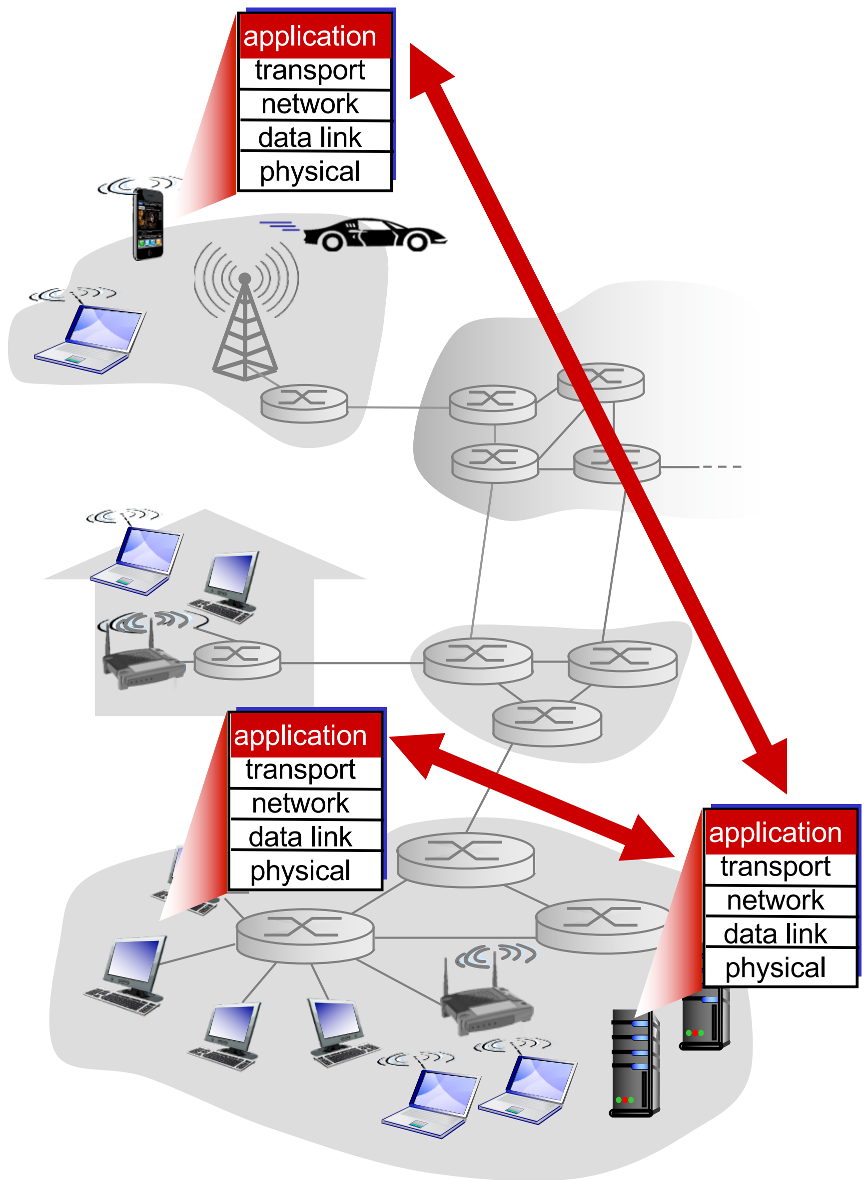
Creating applications

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 application development and propagation

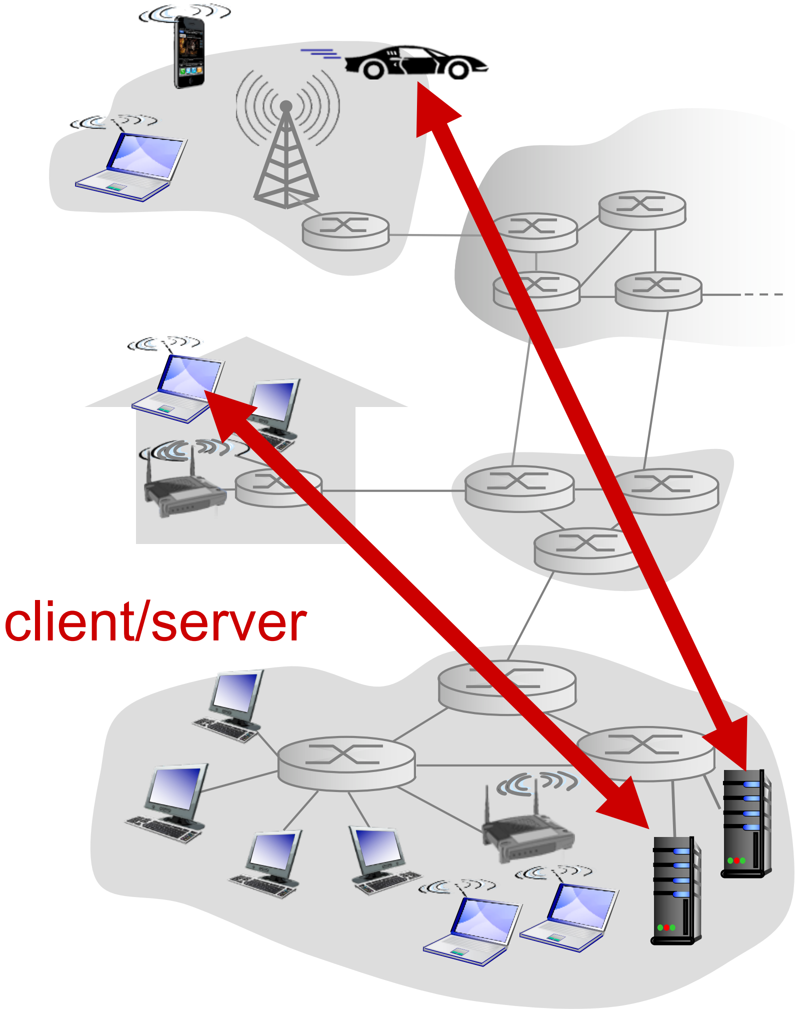


Application architectures

possible structure of applications:

- client-server
- peer-to-peer (P2P)

Client-server architecture



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

Client-server examples

Web:

- Use a web browser (client) to fetch a web page from a web server (server)

Mail:

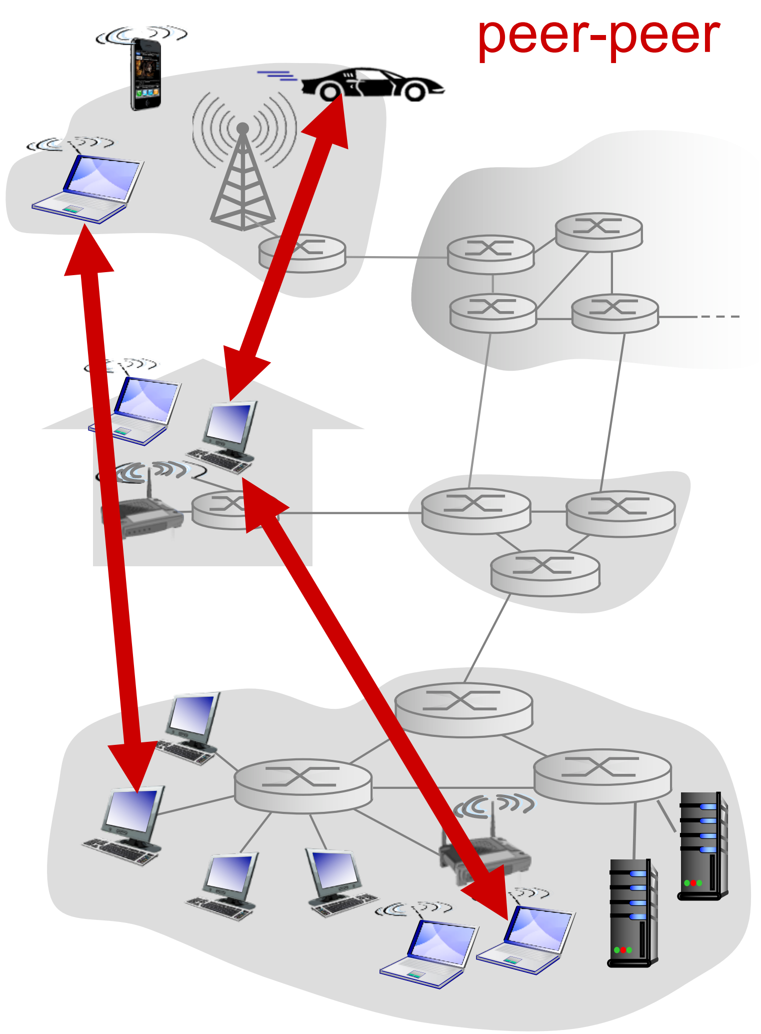
- Use a mail program (client) to connect to your mailbox on a mail server (server)

Webmail:

- Use a web browser (client) to connect to you webmail server (server)
- The webmail server has a mail client built-in (client) that connects to the mail server with the mailbox (server)

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



Processes communicating

process: program running within a host

- within same host, two processes communicate using inter-process communication (defined by operating system)
- processes in different hosts communicate by exchanging messages

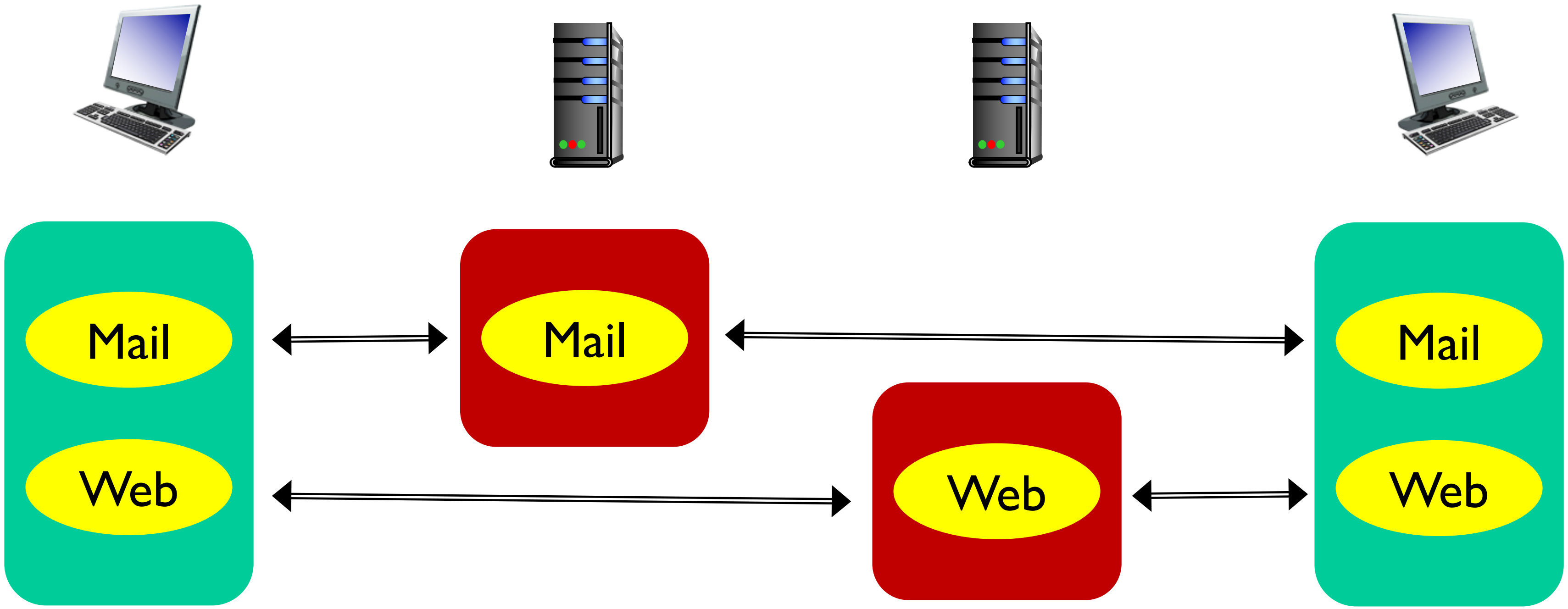
clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

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

Many-to-many Communication



- Many client processes on same host, communicating with different server processes
- A server process communicates with many client processes

Addressing processes

- to receive messages, process must have *identifier*
- each host device has unique IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* no, *many* processes can be running on same host

- *identifier* includes both IP address and port number 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…

Application 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 application need?

data integrity

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

timing

- some applications (e.g., Internet telephony, interactive games) require low delay

throughput

- some applications (e.g., multimedia) require a certain minimum amount of throughput
- other applications ("elastic") 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

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, orconnection setup,

Q: why bother? Why is there a UDP?

Internet application and 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

Securing TCP

TCP & UDP

- no encryption
- passwords traverse Internet in cleartext

SSL (Secure Socket Layer)

- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at application layer

- Applications use SSL libraries, which "talk" to TCP

SSL socket API

- passwords traverse Internet encrypted
- See Chapter 7

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

Web and HTTP

First, a review…

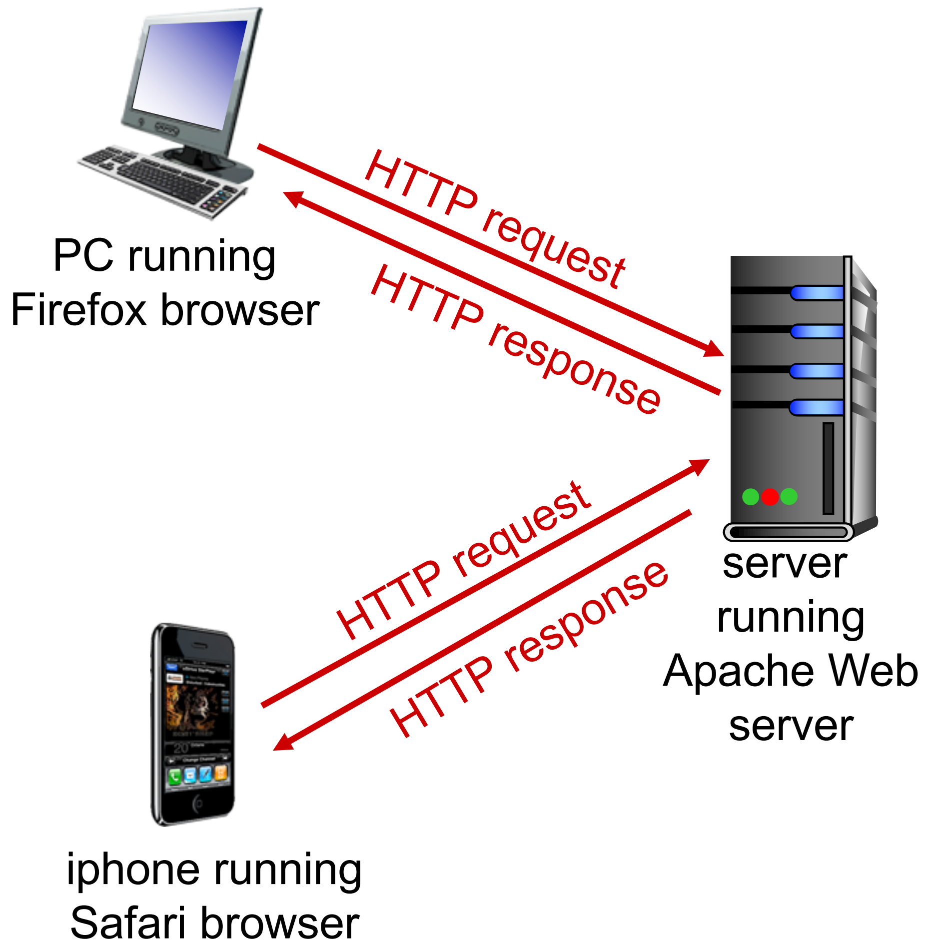
- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL, Uniform Resource Locator,* e.g.,

$$\underbrace{\texttt{http:}}_{\text{protocol}}\texttt{//}\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

HTTP overview

HTTP: hypertext transfer protocol

- Web 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



HTTP request message

- ASCII (human-readable format)
  - HTTP/1.1

request line (GET, POST, HEAD commands)

header lines

Empty line indicates end of header lines

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

```

Request Method types

HTTP/1.0:

- GET
  - Query – can include input in URL
  - **`www.somesite.com/animalsearch?monkeys&banana`**
- POST
  - Input in body – data after header
- HEAD
  - Header only – server leaves requested object out of response

In HTTP/1.1, also:

- PUT
  - Uploads file in entity body to path specified in URL field
- DELETE
  - Deletes file specified in the URL field

HTTP response message

status line (protocol status code/ status phrase)

header lines

data, e.g., requested HTML file

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

data data data data data ...
```

HTTP response status codes

- status code appears in first 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 message (Location:)

`400 Bad Request`
  - request message not understood by server

`404 Not Found`
  - requested document not found on this server

`505 HTTP Version Not Supported`

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 in sent to port 80 at cis.poly.edu

2. type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

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 is stateless

uses TCP, port 80:

- Stateless protocol
  - Client sends request
  - Server responds
  - That's it!
- server maintains no information about past client requests

aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

User-server state: cookies

many Web sites use cookies

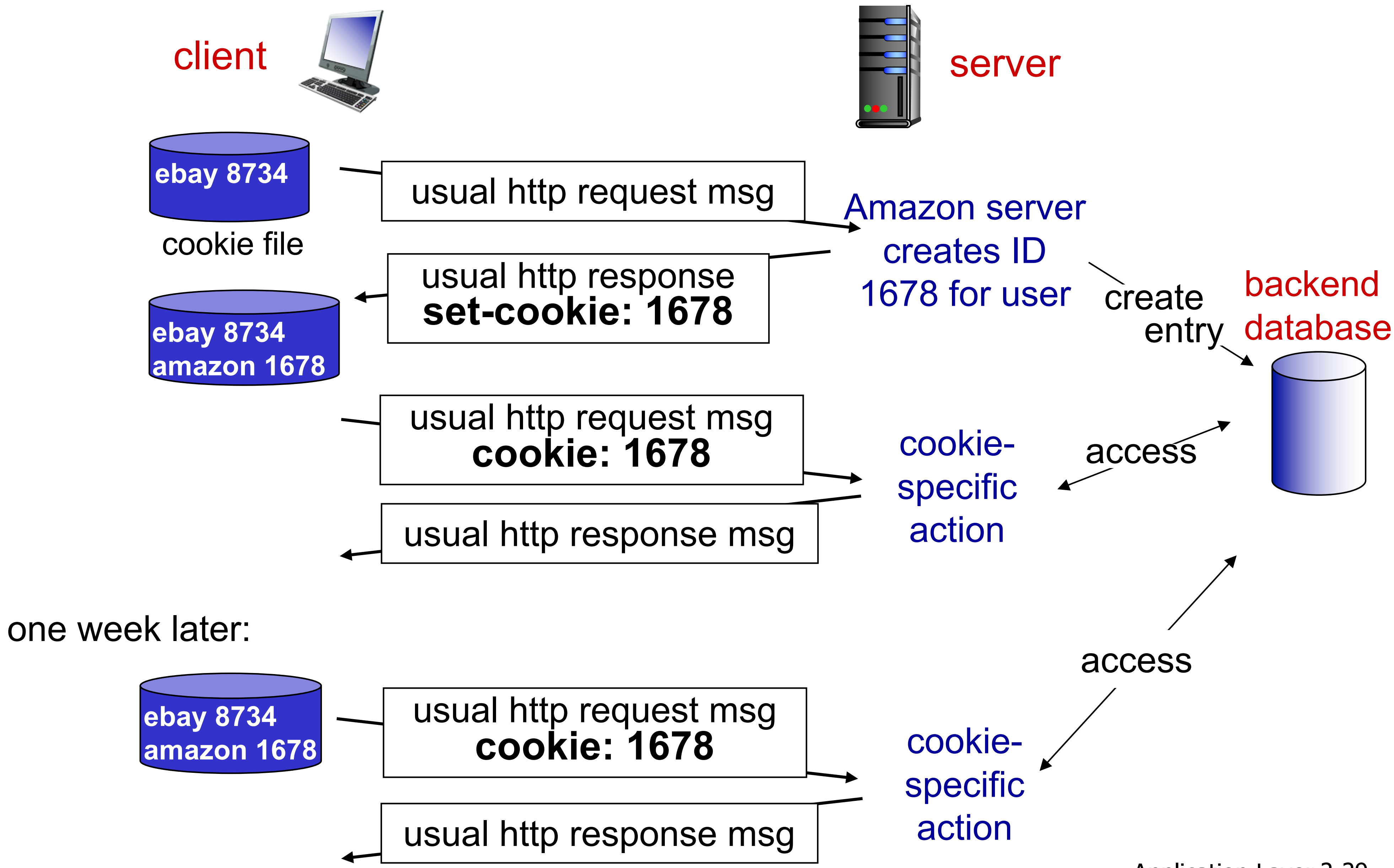
four components:

1) cookie header line of HTTP *response* message
2) cookie header line in next HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

example:

- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

what cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

how to keep "state":

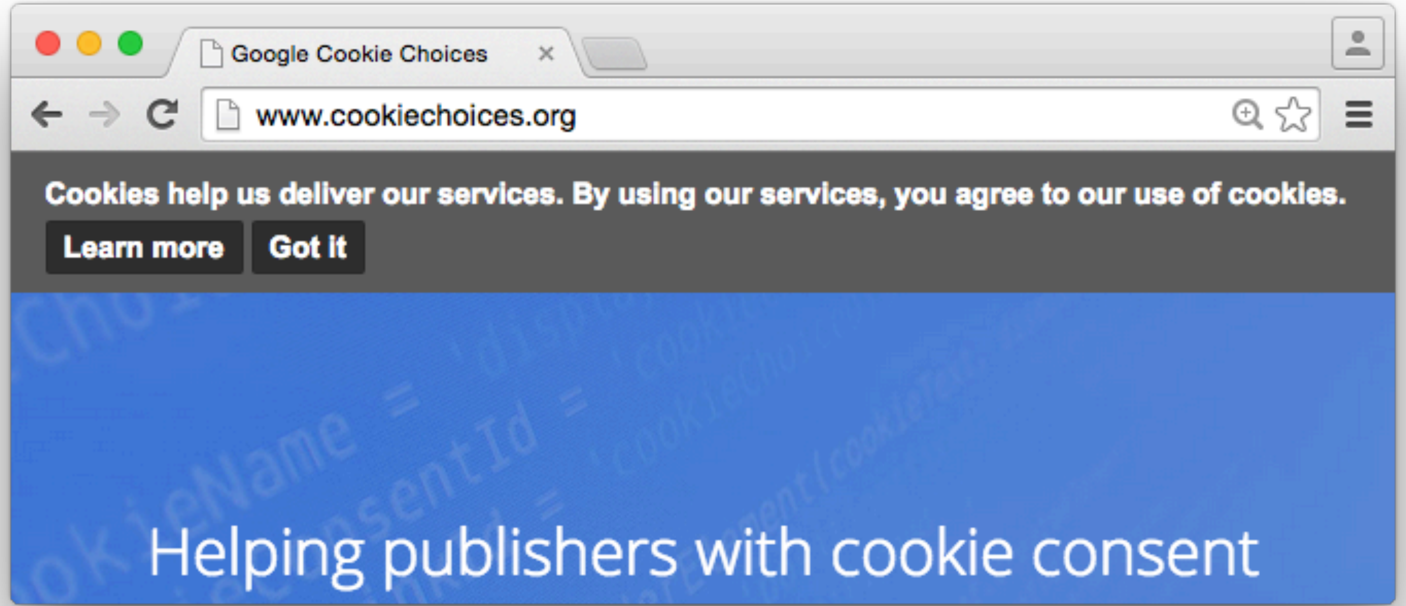
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

Laws and Regulations

In EU, placing a cookie on a user's computer requires that the user is informed, and that the user consents to it. (This applies not only to cookies, but to any similar technology that stores and accesses information on the user's device.)

Lagen om elektronisk kommunikation http://www.pts.se/sv/Bransch/Regler/Lagar/Lag-om-elektronisk-kommunikation/Cookies-kakor 2015-09-21

EU Cookie Law (ePrivacy directive, 2002/58/EC) http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0058:EN:HTML 2015-09-21

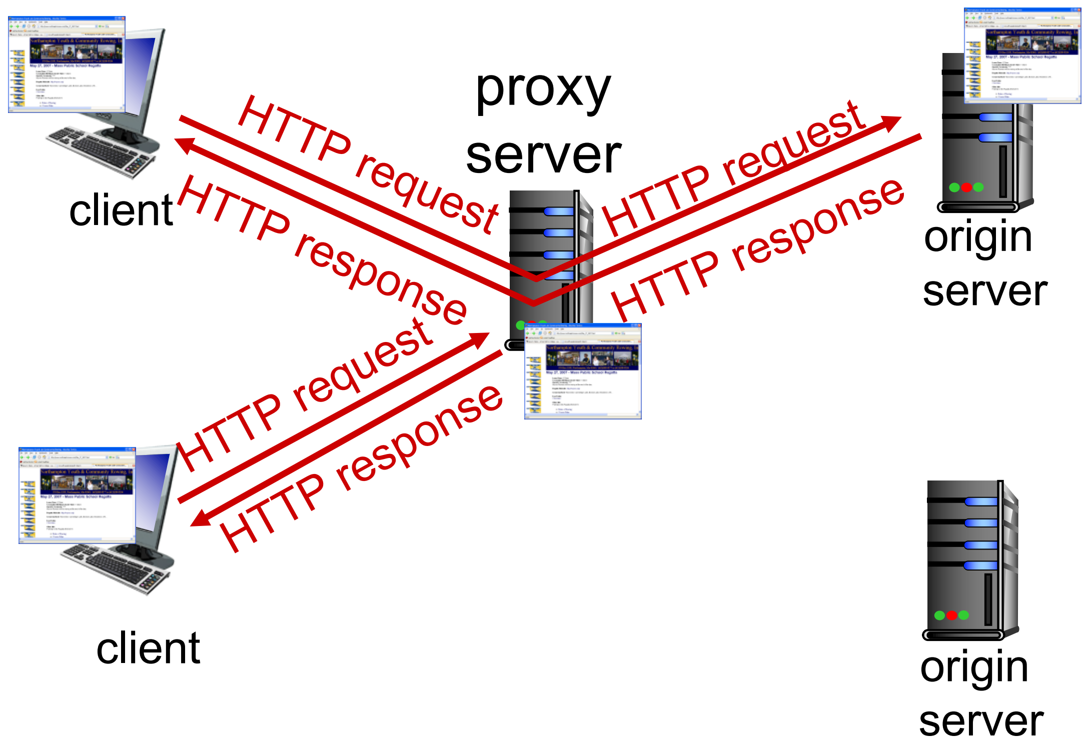


http://www.cookiechoices.org 2015-09-21

Web caches (proxy server)

goal: satisfy client request without involving origin server

- user configures browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - if 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 (so too does P2P file sharing)

HTTP/2 – Why a New Version?

- HTTP performance trends
  - More HTTP transfers per page
  - More data per transfer
  - Request/response stop-and-wait
- TCP efficiency
  - Congestion control has little effect with many short connections
  - Redundancy with same information sent many times
  - Stop-and-wait nature of TCP handshakes
- User experience suffers as page load time increases
- Negative influence on server load and performance

HTTP/2 – Main Features

- *Multiplexing* to support loading of multiple objects at the same time over single connection
- More compact header format
  - *Binary format* (not text)
  - Compression to remove redundancy
- Advanced features, such as *server push*
  - Server knows which objects the browser will request next – send them in advance
- Backward compatibility – version negotiation
- Originates from SPDY research project initiative by Google

- HTTP/2 home page https://http2.github.io/
- RFC 7540 – Hypertext Transfer Protocol version 2 (HTTP/2)
- RFC 7541 – HPACK: Header Compression for HTTP/2

Chapter 2: outline

2.1 principles of network applications
- application architectures
- application requirements

2.2 Web and HTTP

2.3 ~~FTP~~

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

Chapter 2: outline

2.1 principles of network applications
- application architectures
- application requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

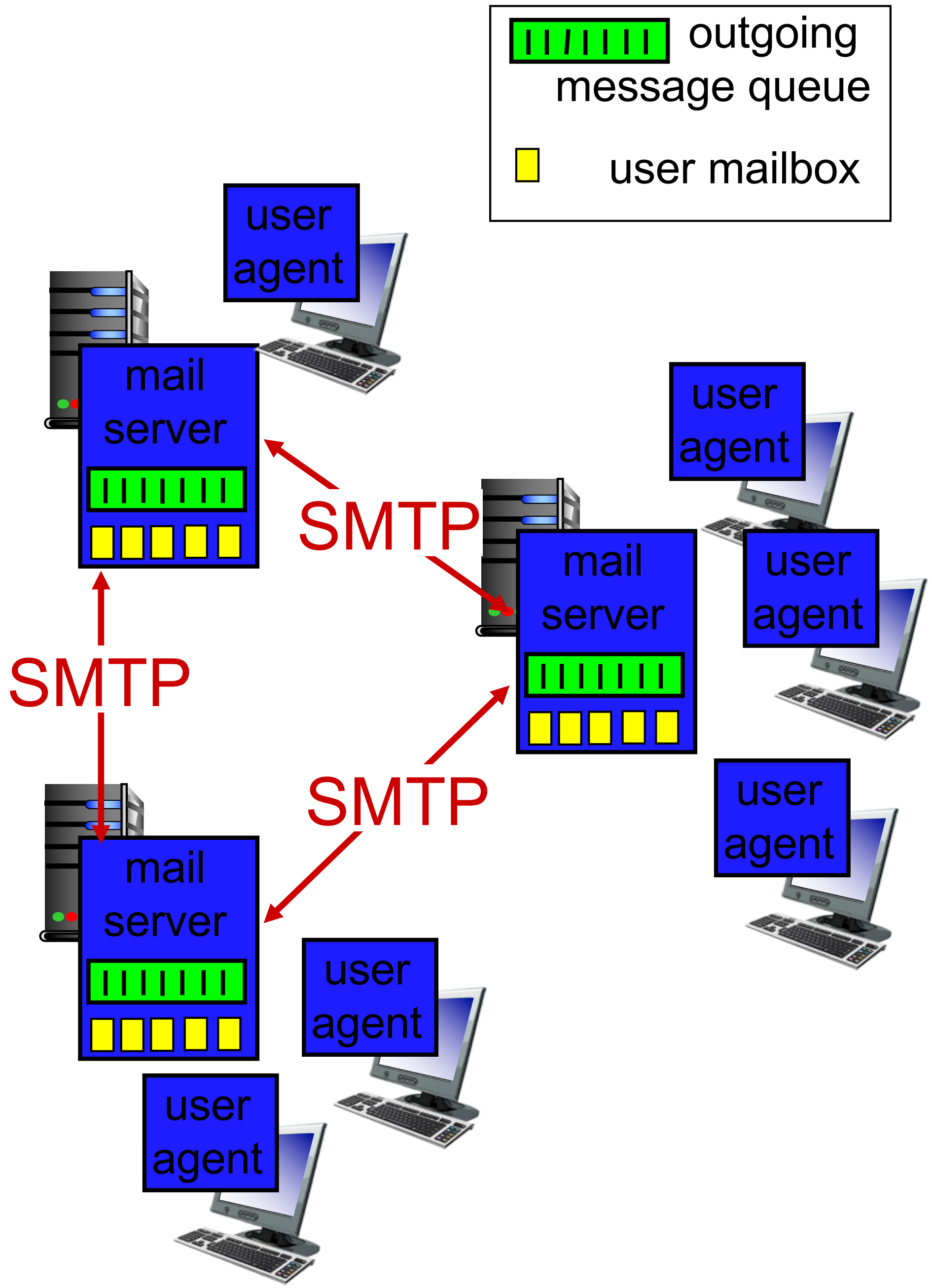
Electronic mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

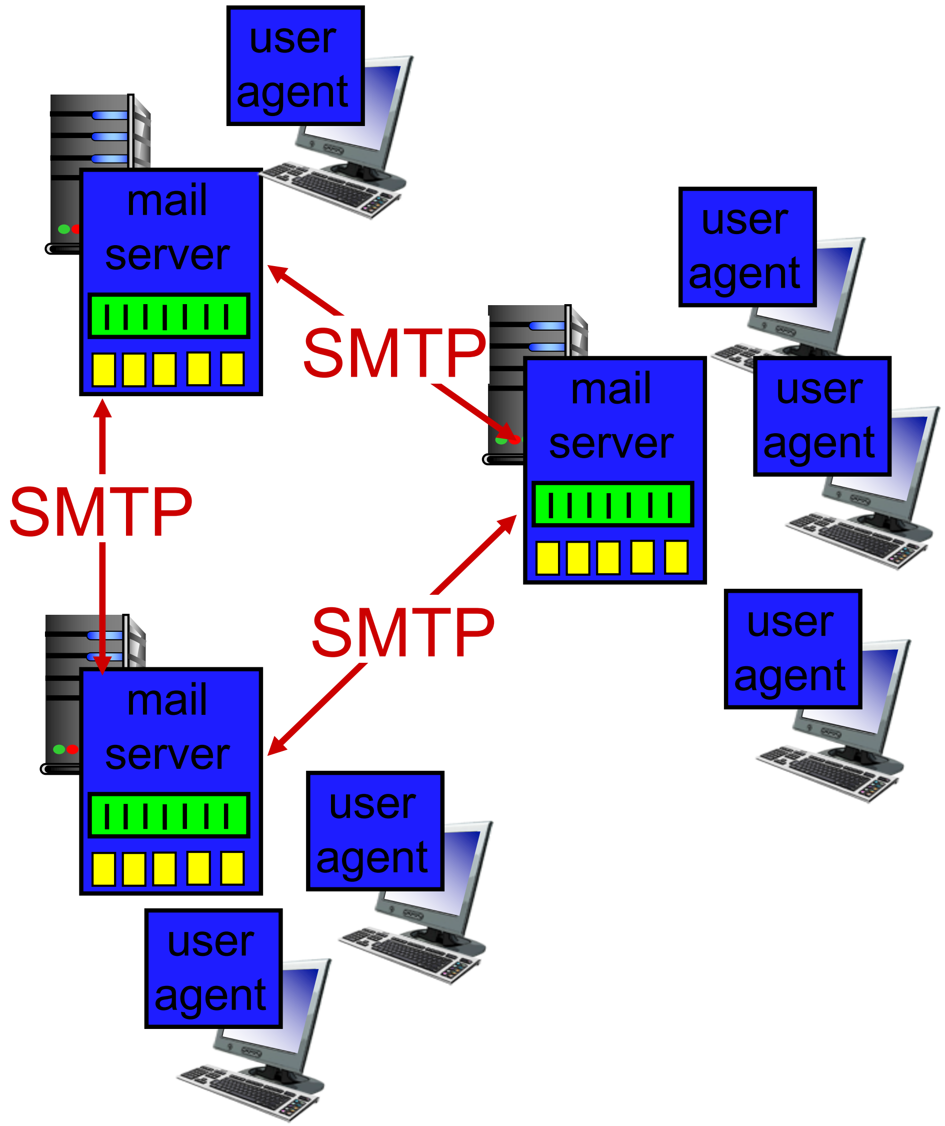
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



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
  - Note: mail server has client side and server side
    - client: sending mail server
    - server: receiving mail server

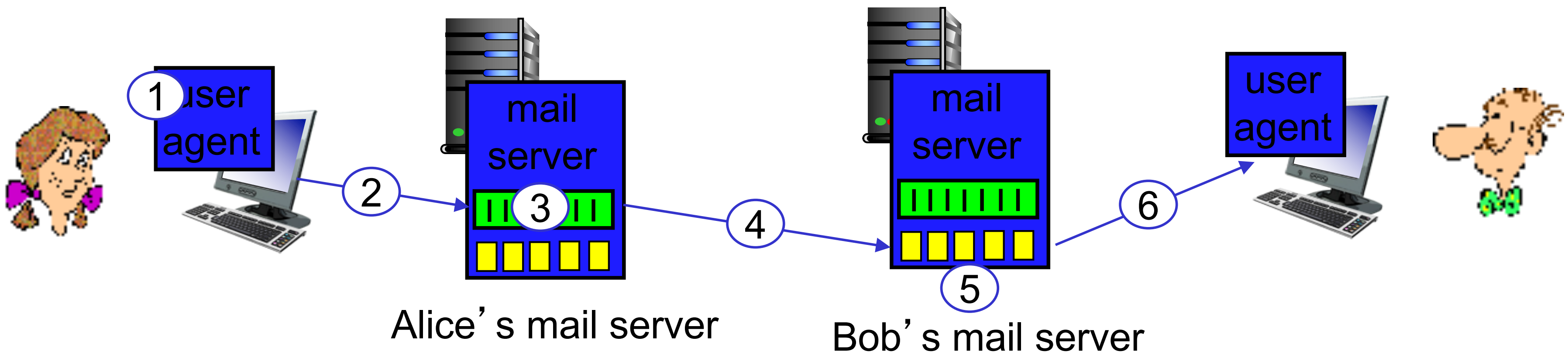


Electronic Mail: SMTP [RFC 2821]

- 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)
  - 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 at `bob@someschool.edu`
2) Alice's UA sends message to her **outgoing** mail server; message placed in message queue
3) client side of SMTP opens TCP connection with Bob's **incoming** mail server
4) SMTP client sends Alice's message over the TCP connection
5) Bob's incoming mail server places the message in Bob's mailbox
6) Bob invokes his user agent to read message



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
```

Try SMTP interaction for yourself:

- **`telnet servername 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

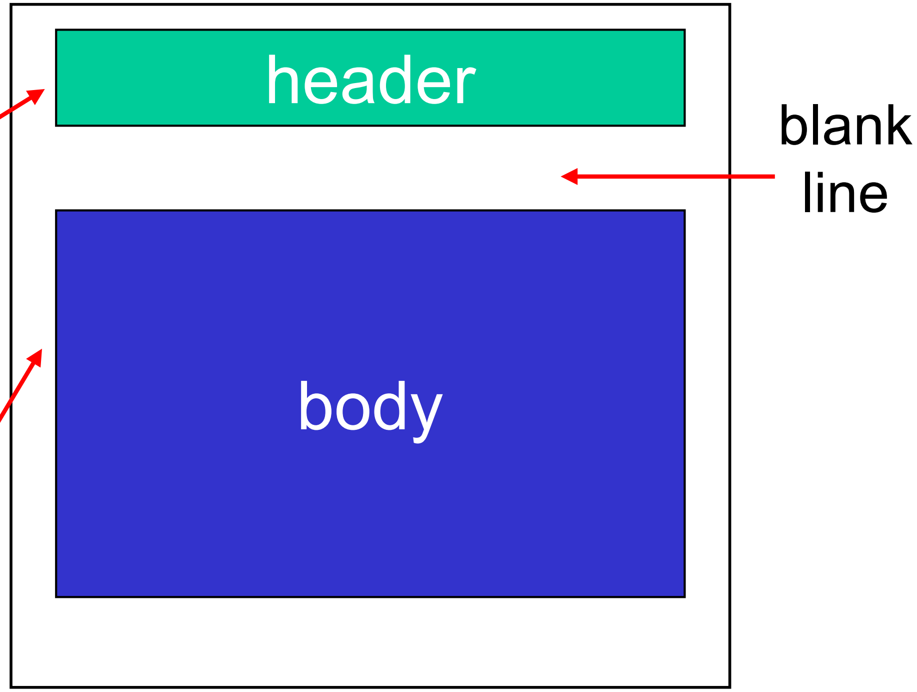
above lets you send email without using email client (reader)

Mail message format

RFC 5322 (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



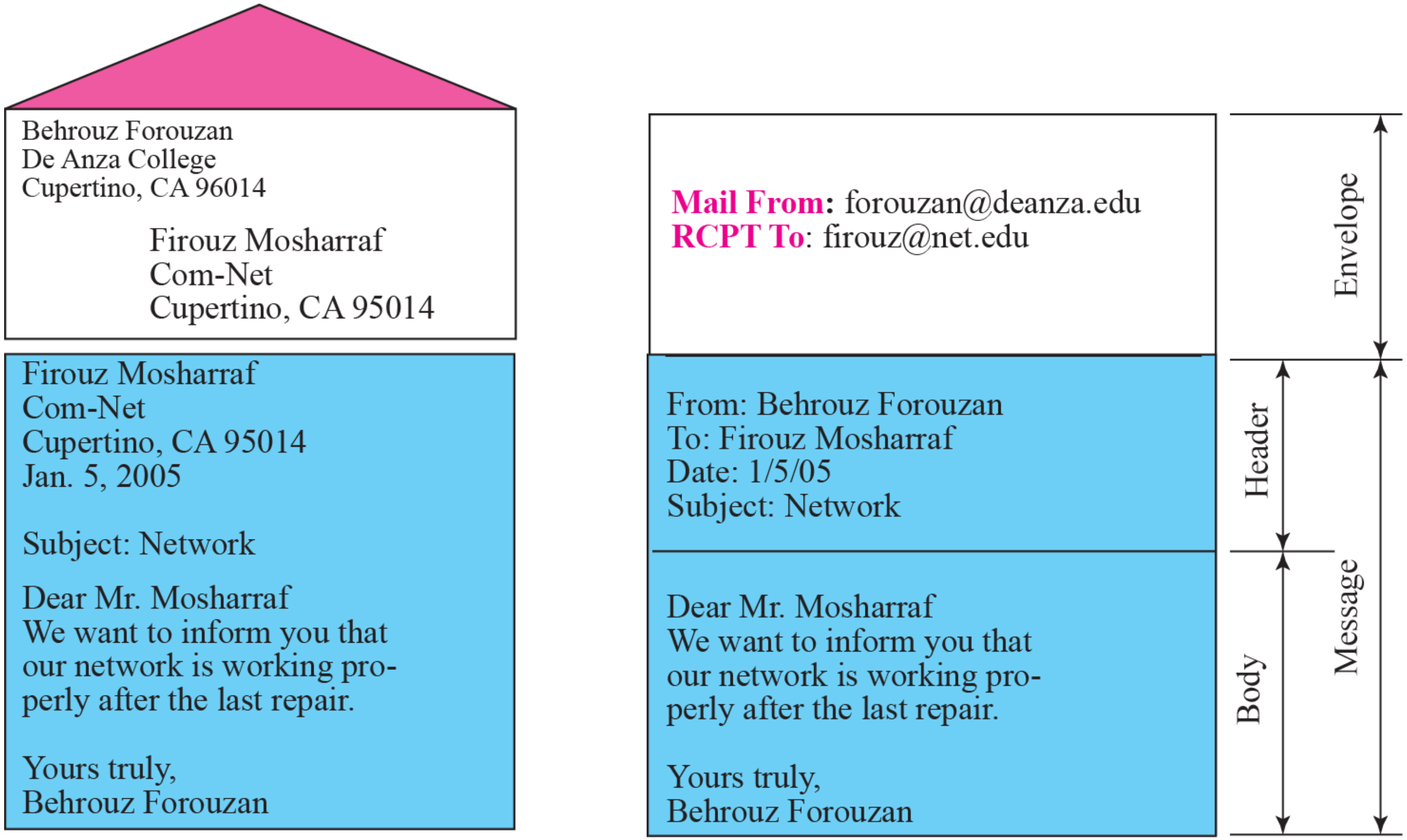
MIME

- Multipurpose Internet Mail Extensions, RFC 2045 and more
- Content formats and encodings for SMTP (7-bit ASCII)
  - Binary (non-text) objects (binary files)
  - Non-ASCII text ("Å", "Ä", "Ö" for instance)
  - Multi-part message bodies
- Extensions for secure email – S/MIME, PGP, …

```
MIME-version: 1.0
Content-type: multipart/mixed; boundary="frontier"

This is a multi-part message in MIME format.
--frontier
Content-type: text/plain

This is the body of the message.
--frontier
Content-type: application/octet-stream
Content-transfer-encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+VGhpcyBpcyB0aGUg
Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg==
--frontier--
```

From Wikipedia

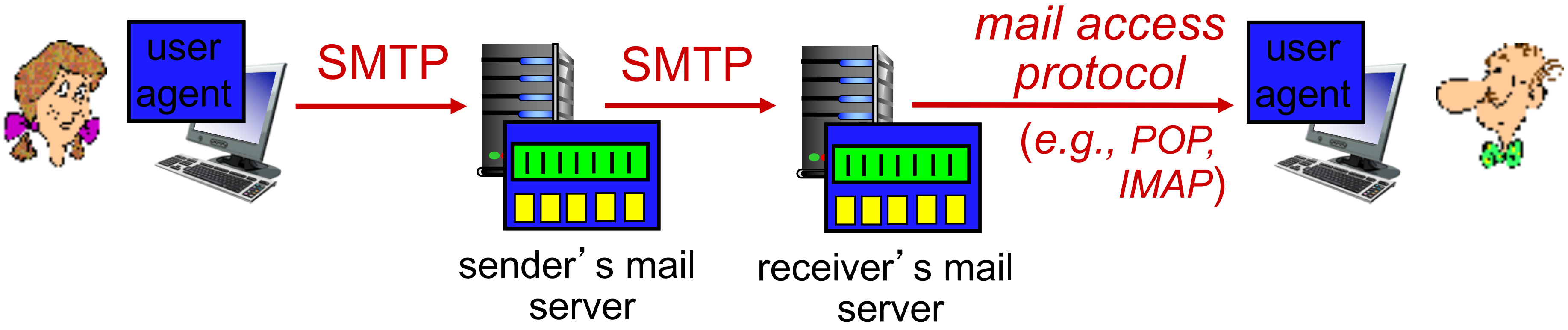
Format of an Email



SMTP

RFC 5322,
MIME, etc

Mail access protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from 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 1 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

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

DNS: domain name system

people: many identifiers:

- Social security number, name, passport number

Internet hosts, routers:

- IP address - 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

Internet Domains

- Country domains (country code top-level domains, ccTLDs)
  - Two-letter country domains (per ISO 3166)
  - “.cn”, “.fi”, “.nu”, “.se”, “.uk”, “.us”, …
- Generic domains (generic top-level domains, gTLDs)
  - Three letters or more
  - “.com”, “.net”, “.org”, …

Generic Domain Labels

Domain	Intended use
aero	the air transport industry.
asia	companies, organizations and individuals in the Asia-Pacific region
biz	business use
cat	Catalan language/culture
com	commercial organizations, but unrestricted
coop	cooperatives
edu	U.S. post-secondary educational establishments
gov	U.S. government entities at the federal, state, and local levels
info	informational sites, but unrestricted
int	international organizations established by treaty
jobs	employment-related sites

Domain	Intended use
mil	the U.S. military
mobi	sites catering to mobile devices
museum	museums
name	families and individuals
net	originally for network infrastructures, now unrestricted
org	originally for organizations not clearly falling within the other gTLDs, now unrestricted
pro	certain professions
tel	services involving connections between the telephone network and the Internet
travel	travel agents, airlines, hoteliers, tourism bureaus, etc.
xxx	pornography

From: Wikipedia, 2013-09-30

Generic Domain Labels

Domain	Intended use
aero	the air transport industry.
asia	companies, organizations and individuals in the Asia-Pacific region
biz	business use
cat	Catalan language/culture
com	commercial organizations, but unrestricted
coop	cooperatives
edu	U.S. post-secondary educational establishments
gov	U.S. government entities at the federal, state, and local levels
info	informational sites, but unrestricted
int	international organizations established by treaty
jobs	employment-related sites

Domain	Intended use
mil	the U.S. military
mobi	sites catering to mobile devices
museum	museums
name	families and individuals
net	originally for network infrastructures, now unrestricted
org	originally for organizations not clearly falling within the other gTLDs, now unrestricted
post	postal services
pro	certain professions
tel	services involving connections between the telephone network and the Internet
travel	travel agents, airlines, hoteliers, tourism bureaus, etc.
xxx	pornography

From: Wikipedia, 2013-09-30

ICANN New gTLD Program

- Internet Corporation for Assigned Names and Numbers
  - http://newgtlds.icann.org
  - "Largest-ever expansion of the Domain Name System"
  - ICANN accepting applications for new gTLDs since 2012
  - 1192 "Registry Agreements" signed for new gTLDs as of Sept 25, 2015
    - Still more in process
- Examples
  - Commonly used words – .CULTURE, .MUSICAL, .TRUSTED, .PIZZA
  - Geographic – .WALES, .BUDAPEST
  - Community – .CLEANWATER, .LITERACY
  - Brand – .BMW, .YOUTUBE
  - Internationalized Domain Names – онлайн, 游戏

DNS: services, structure

DNS services

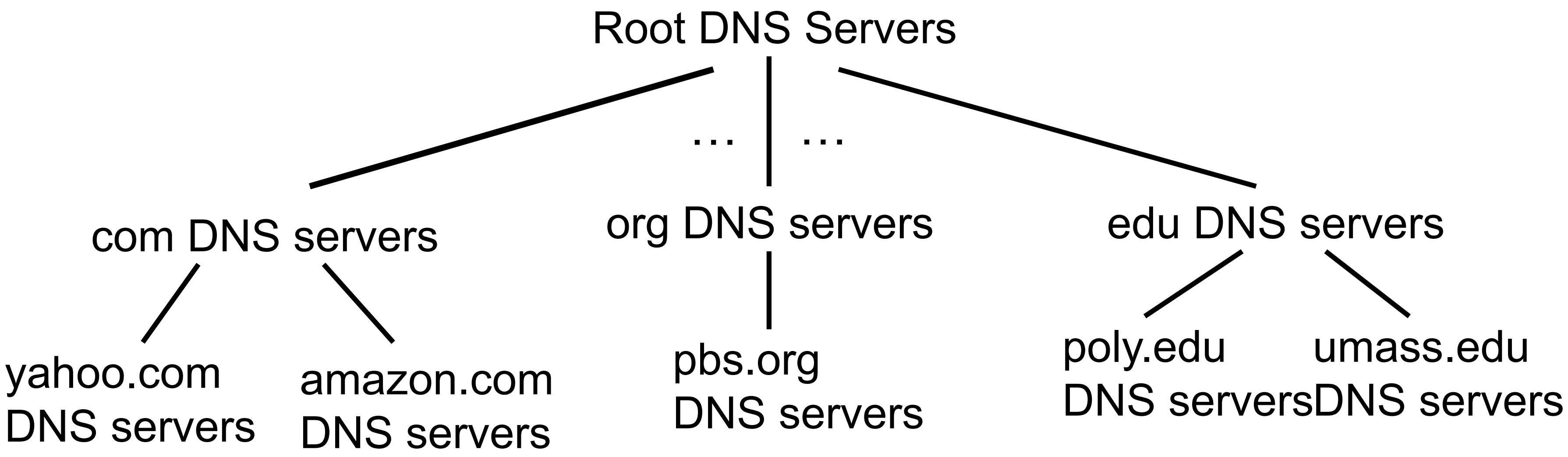
- hostname to IP address translation
  - "resolving"
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

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

A: doesn't scale!

DNS: a distributed, hierarchical database

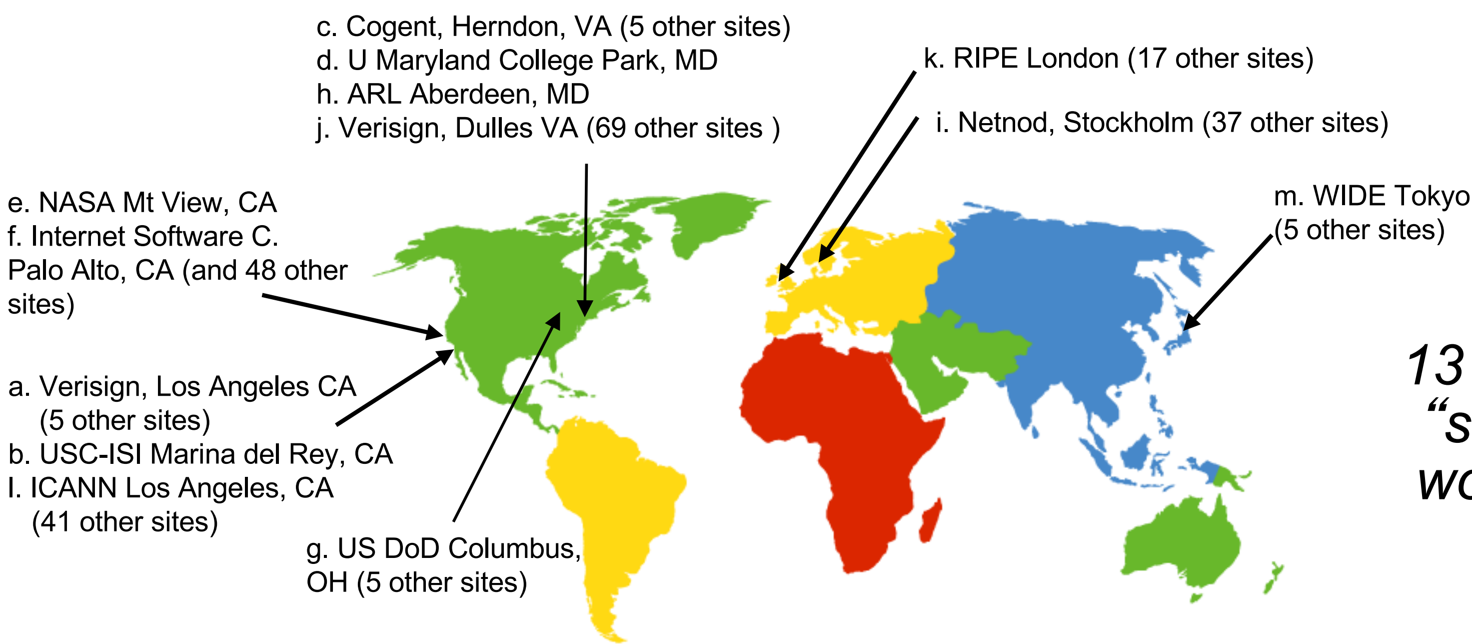


client wants IP address for www.amazon.com; 1st approx:

- 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"

DNS: root name servers

- contacted by local name server when it cannot resolve name
- root name server:
  - Maintains database of TLD (top-level domain) servers
  - When contacted, returns list of DNS servers for TLD in question



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
- When contacted, returns list of authoritative DNS servers for organization in question

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
- When contacted, returns host's IP address(es)
  - Or, possibly, refers to other servers in the organization

Local DNS name server

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
    - Part of a host's network configuration
- 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

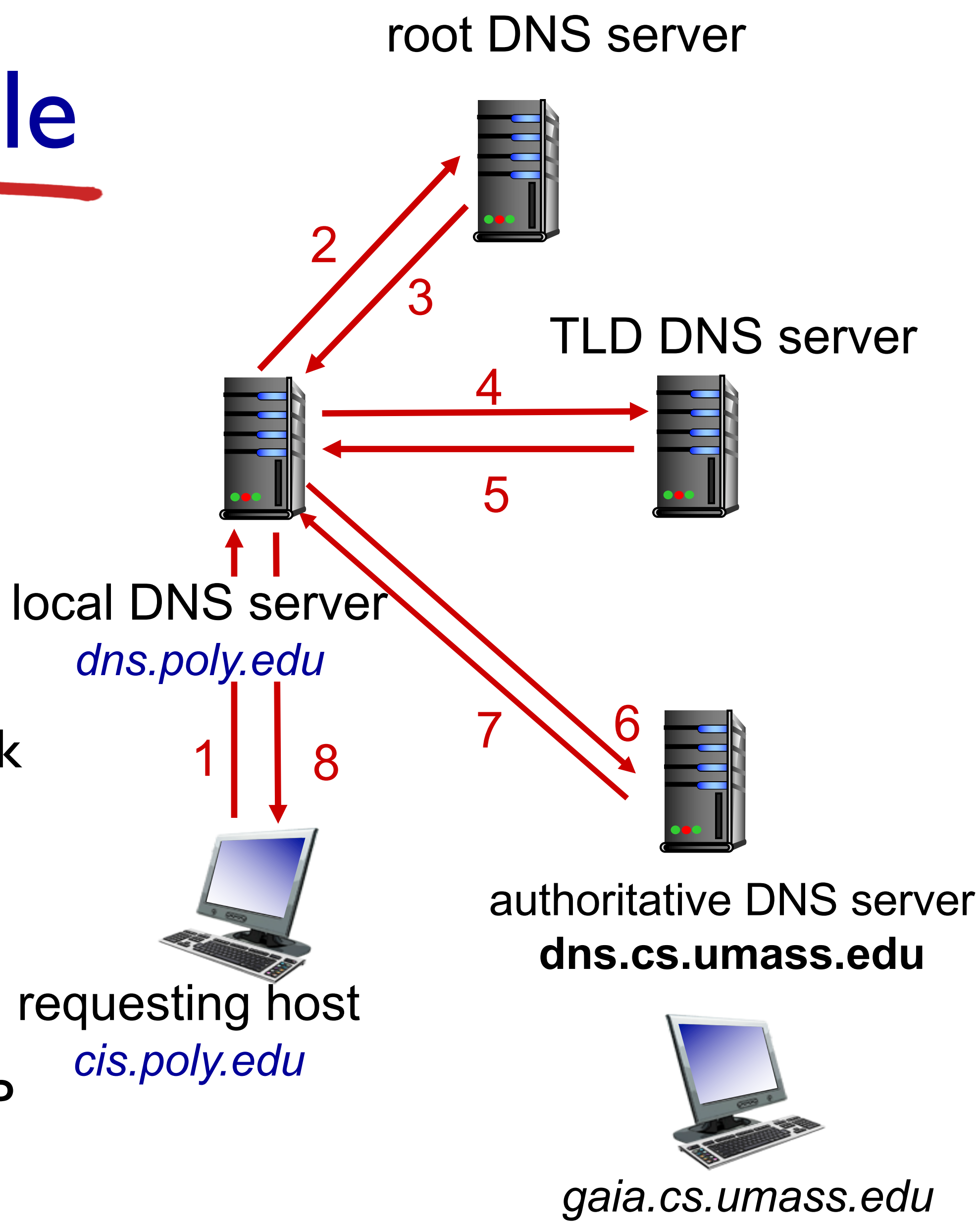
- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"
- This is what root, TLD, and authoritative DNS server do

recursive query:

- contacted server replies with IP address
  - Resolves name
- This is what local DNS server does



DNS: caching, updating records

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

Inserting records into DNS

- example: new startup "Network Utopia"
- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
  - provide names and IP addresses of authoritative name servers (primary and secondary)
  - registrar inserts two RRs into .com TLD server:
    `(networkutopia.com, dns1.networkutopia.com, NS)`
    `(dns1.networkutopia.com, 212.212.212.1, A)`
- create authoritative server type A record for www.networkuptopia.com; type MX record for networkutopia.com

Attacking DNS

DDoS attacks

- Bombard root servers with traffic
  - Not successful to date
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - Potentially more dangerous

Redirect attacks

- Man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus entries to DNS server, which caches

Exploit DNS for DDoS

- Send queries with spoofed source address: target IP
- Requires amplification

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

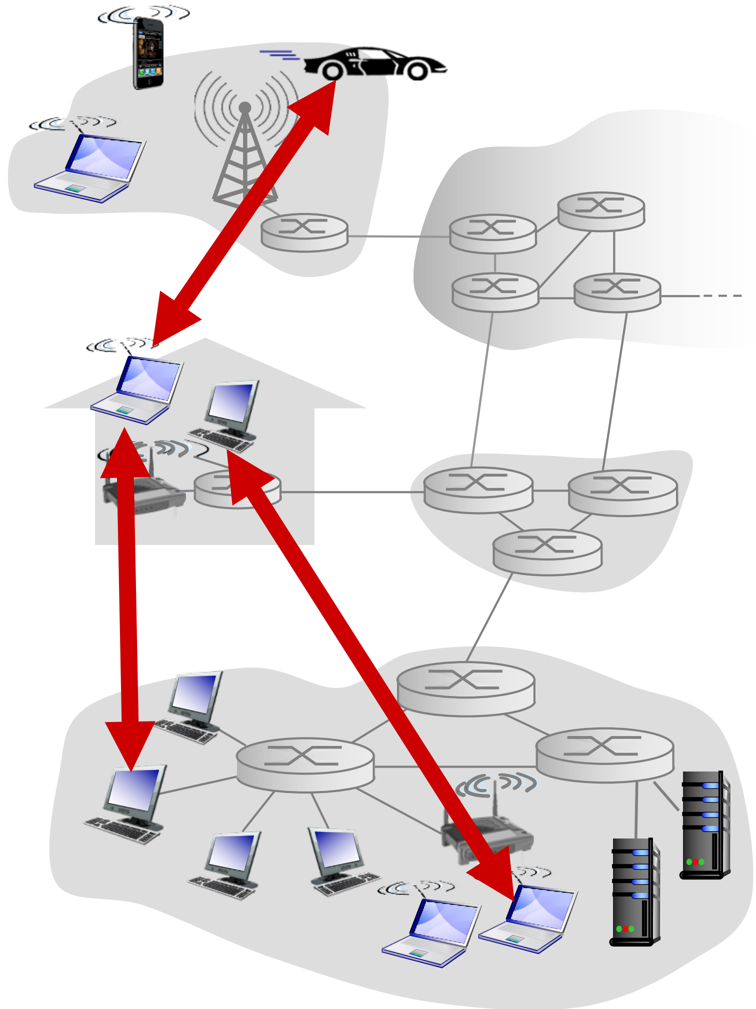
2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses



examples:

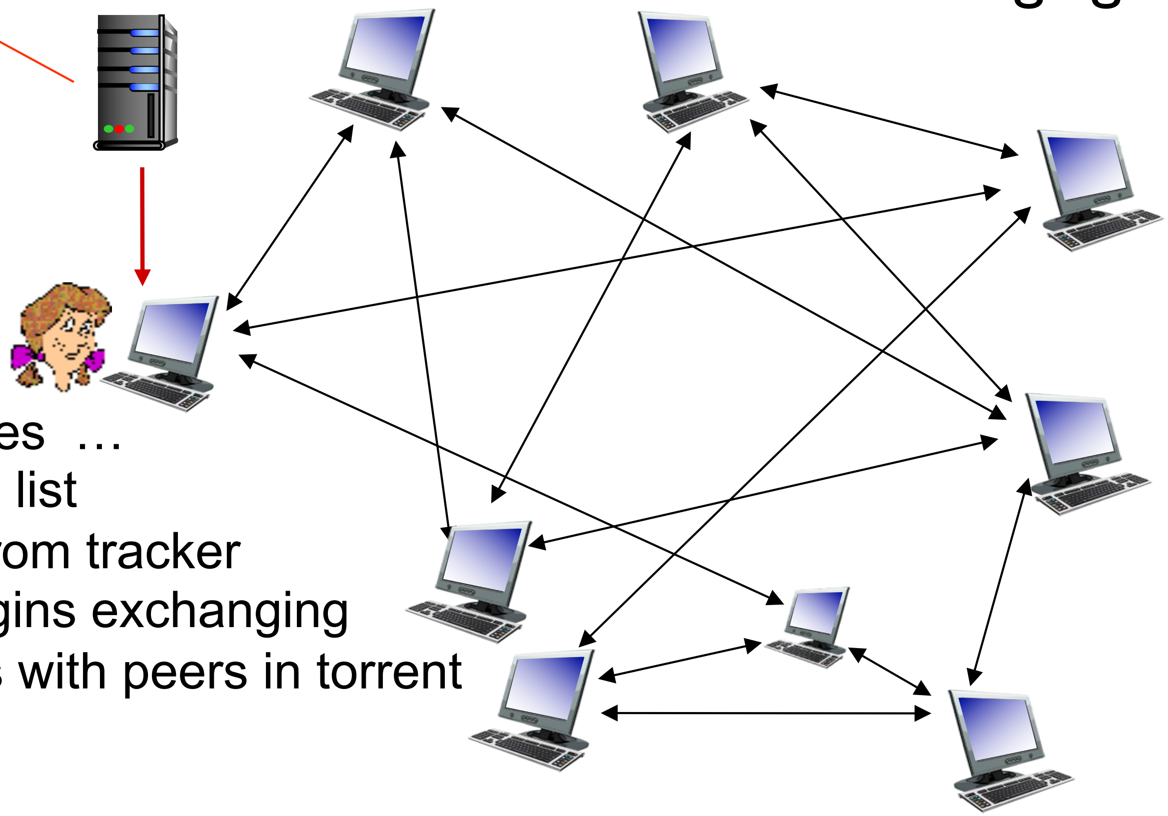
- file distribution (BitTorrent)
- Streaming (Kankan)
- VoIP (Skype)

File sharing – In Five Steps

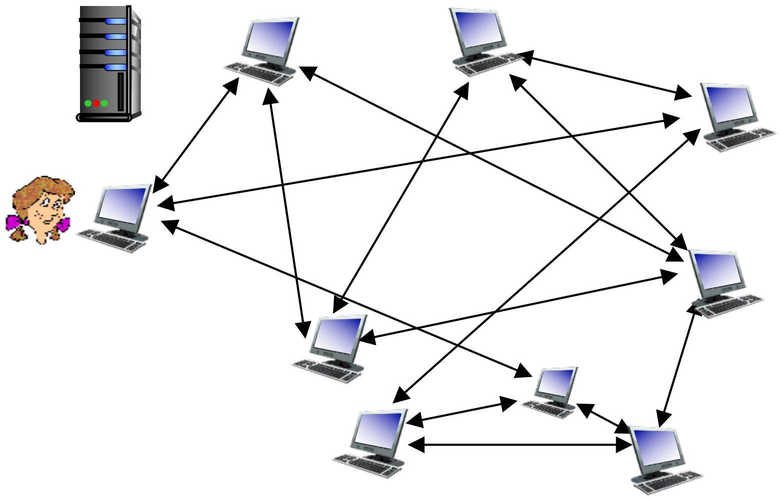
- Figuring out what content you want to download
  - Reading newspapers etc to learn of, e.g., "P2P – The Movie"
- Mapping content names to metadata objects
  - Metadata objects contain information about data objects with the desired content.
    - ".torrent" files
  - Searching metadata directories, e.g., The Pirate Bay
- Finding and downloading metadata objects
  - Function no longer provided by, e.g., The Pirate Bay.
- Using metadata objects to identify desired data objects
  - Typically done internally by file sharing application
- Finding host(s) capable of providing desired data object(s)
  - Using tracker, by gossiping, by Distributed Hash Table (DHT), etc

P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks



P2P file distribution: BitTorrent



- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

BitTorrent: requesting, sending file chunks

requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

sending chunks: tit-for-tat

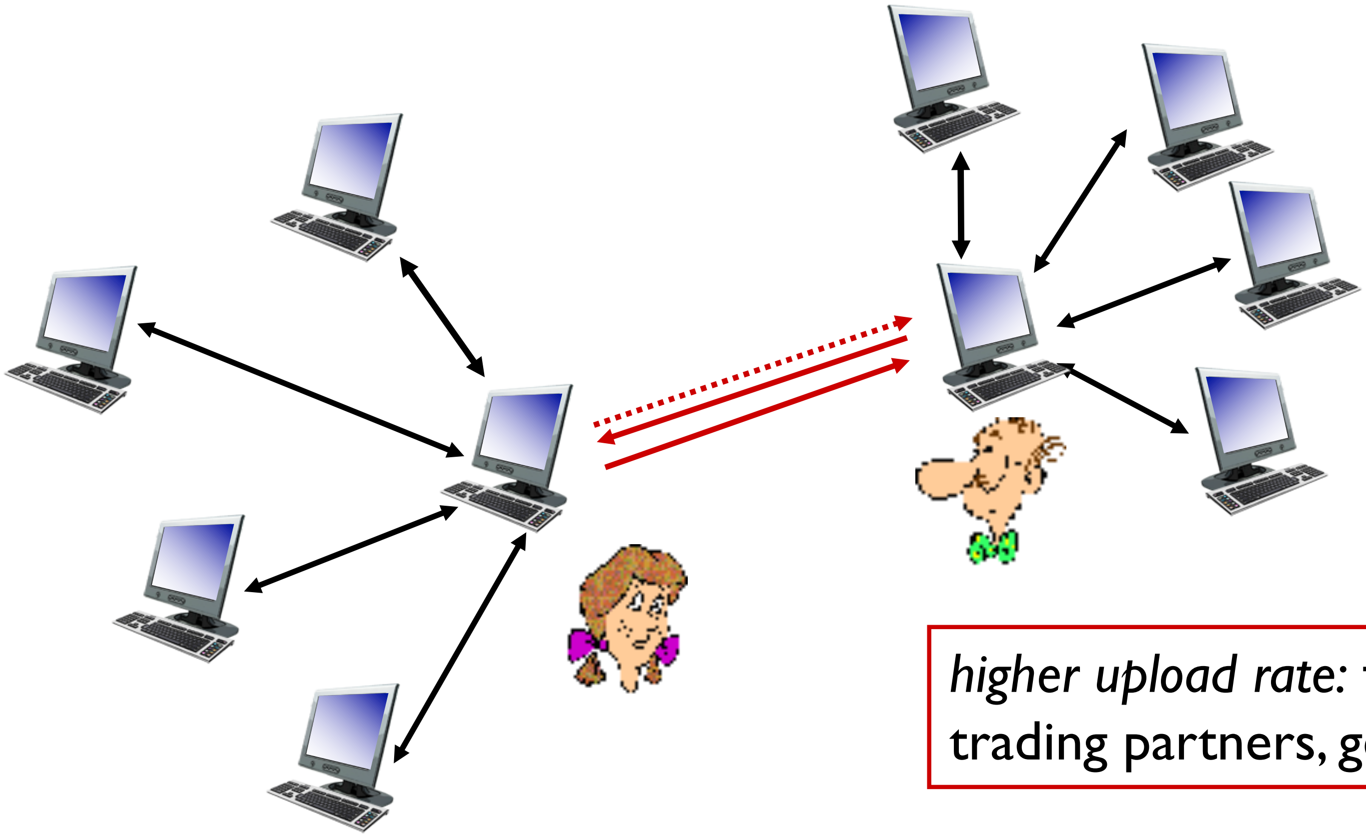
- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

BitTorrent: tit-for-tat

(1) Alice "optimistically unchokes" Bob

(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers



higher upload rate: find better trading partners, get file faster!

Distributed Hash Table (DHT)

- DHT: a *distributed P2P database*
- database has (key, value) pairs; examples:
  - key: social security number; value: human name
  - key: movie title; value: IP address
- Distribute the (key, value) pairs over the (millions of peers)
- a peer queries DHT with key
  - DHT returns values that match the key
- peers can also insert (key, value) pairs

DHTs are used, for example, in BitTorrent's distributed tracker. The key is a torrent identifier, and the value is the set of IP addresses of the hosts currently in the torrent.

Q: how to assign keys to peers?

- central issue:
  - assigning (key, value) pairs to peers.
- basic idea:
  - convert each key to an integer
  - Assign integer to each peer
  - put (key,value) pair in the peer that is closest to the key

DHT identifiers

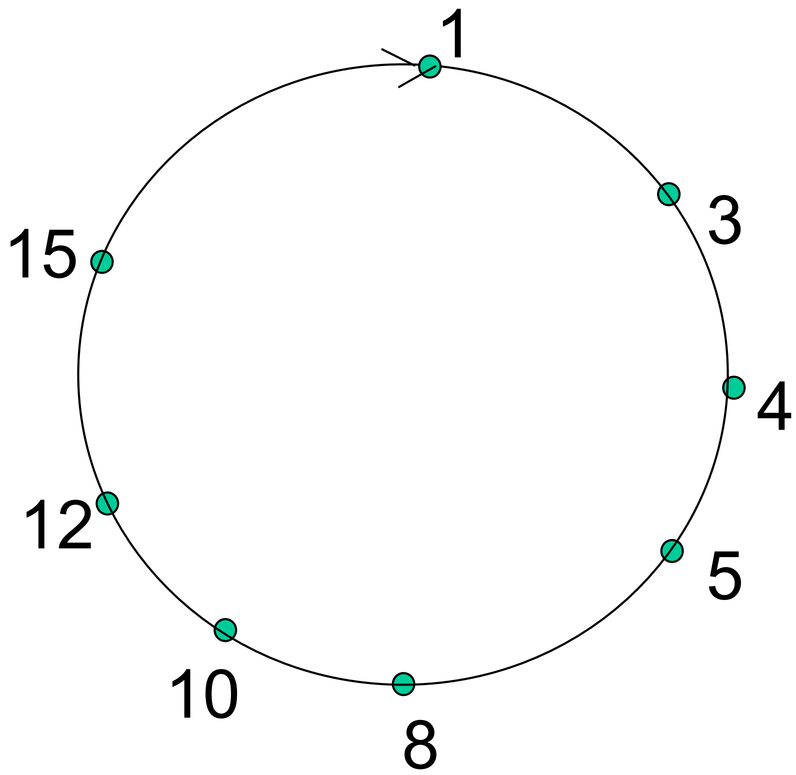
- assign integer identifier to each peer in range $[0,2^n-1]$ for some *n*.
  - each identifier represented by *n* bits.

- require each key to be an integer in same range
- to get integer key, hash original key
  - e.g., key = hash("Led Zeppelin IV")
  - this is why its is referred to as a *distributed "hash" table*

Assign keys to peers

- rule: assign key to the peer that has the *closest* ID.
- convention in lecture: closest is the *immediate successor* of the key.
- e.g., n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

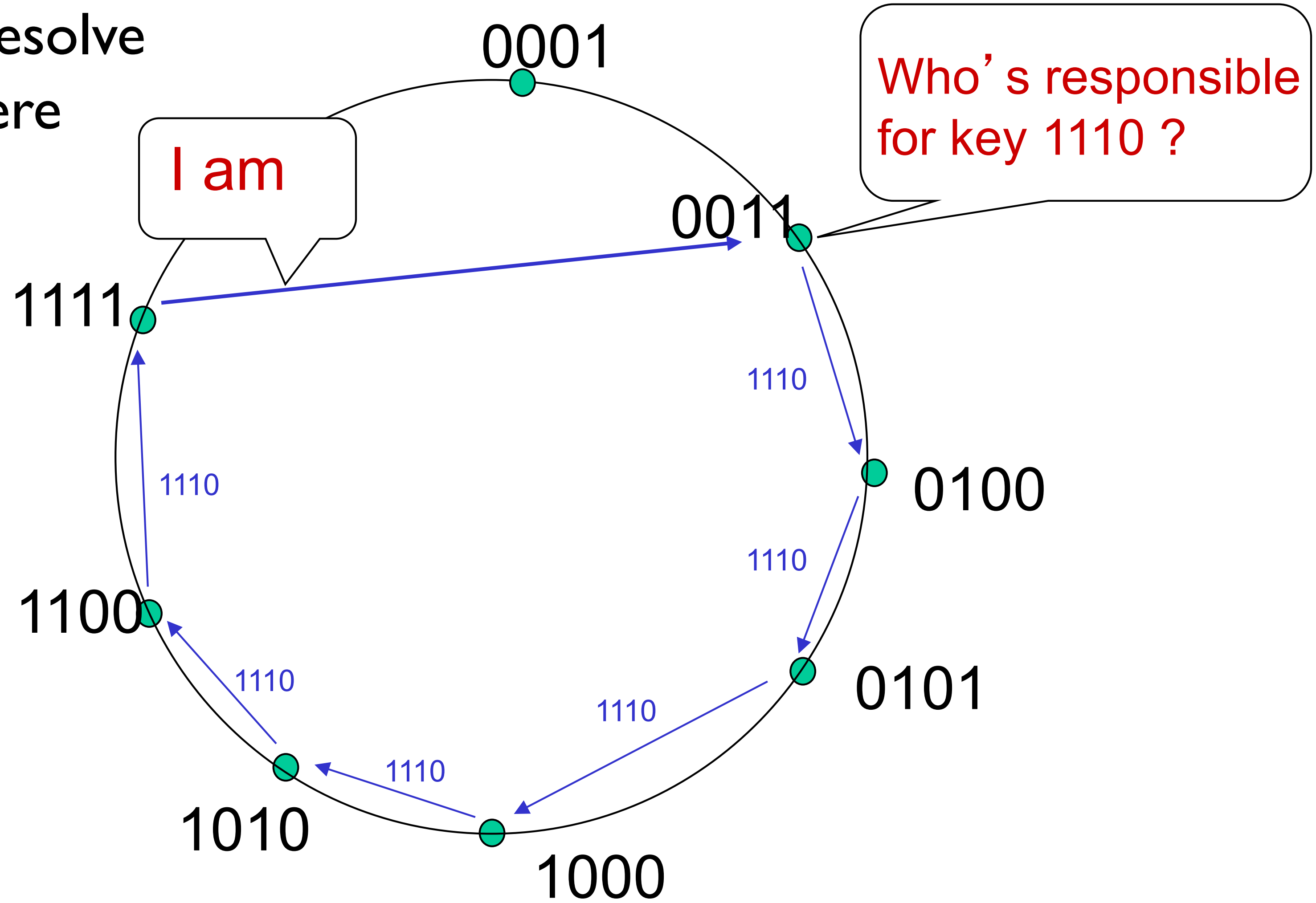
Circular DHT (1)



- each peer *only* aware of immediate successor and predecessor.
- "overlay network"

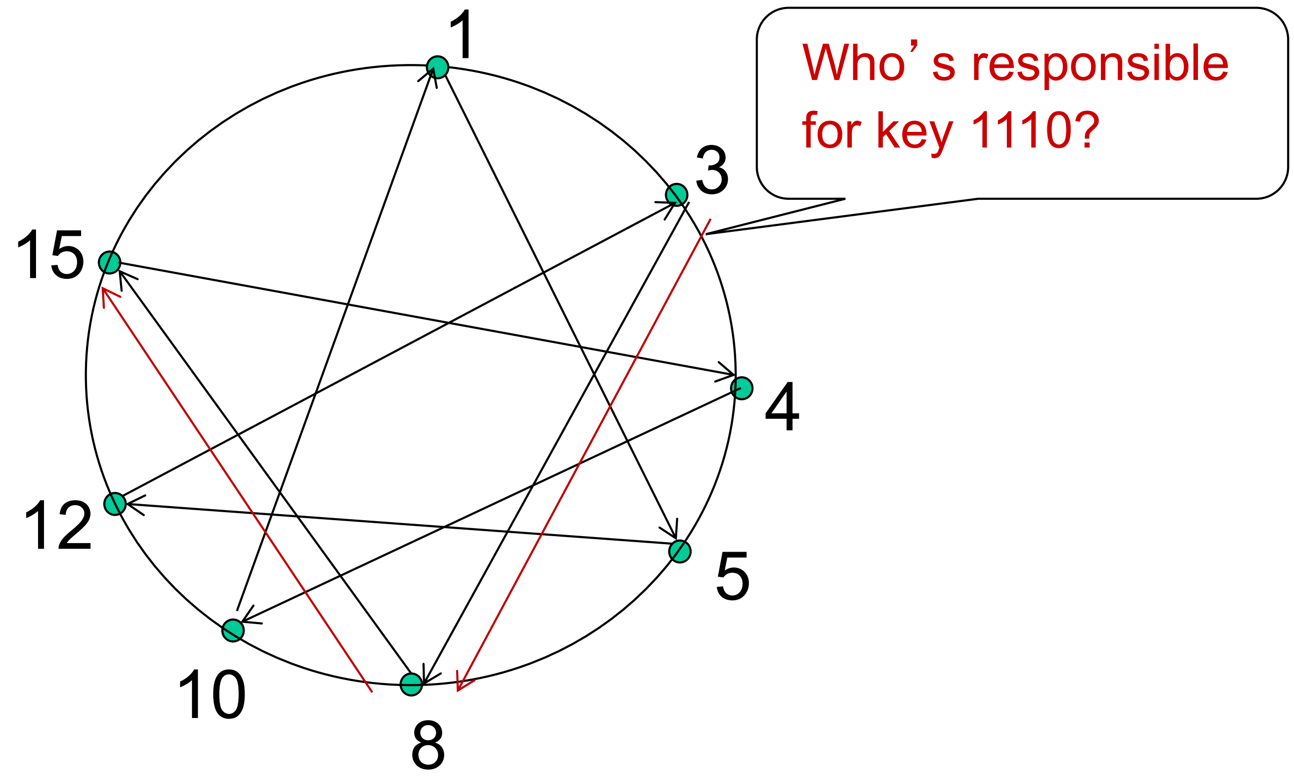
Circular DHT (I)

O(N) messages
on average to resolve
query, when there
are *N* peers



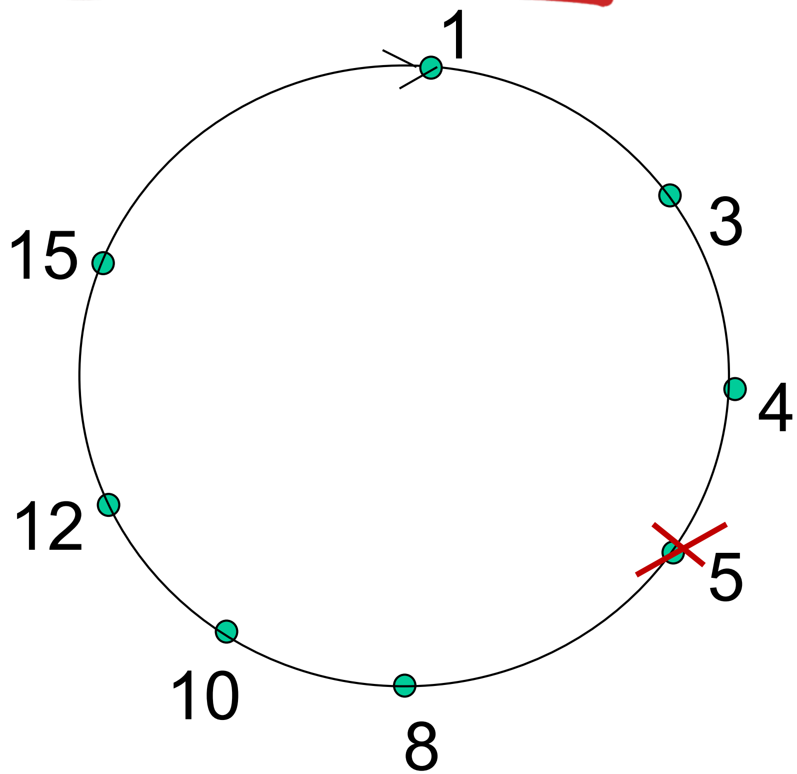
Define closest
as closest
successor

Circular DHT with shortcuts



- each peer keeps track of IP addresses of predecessor, successor, and short cuts.
- reduced from 6 to 2 messages.
- possible to design shortcuts so *O(log N)* neighbors, *O(log N)* messages in query

Peer churn



handling peer churn:

- peers may come and go (churn)
- each peer knows address of its two successors
- each peer periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

- peer 4 detects peer 5 departure; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
- what if peer 13 wants to join?

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

~~2.7 socket programming with UDP and TCP~~

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
  - P2P: BitTorrent, DHT
- ~~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"