CSB051 – Computer Networks

電腦網路

Chapter 2
Application Layer

吳俊興
國立高雄大學 資訊工程學系

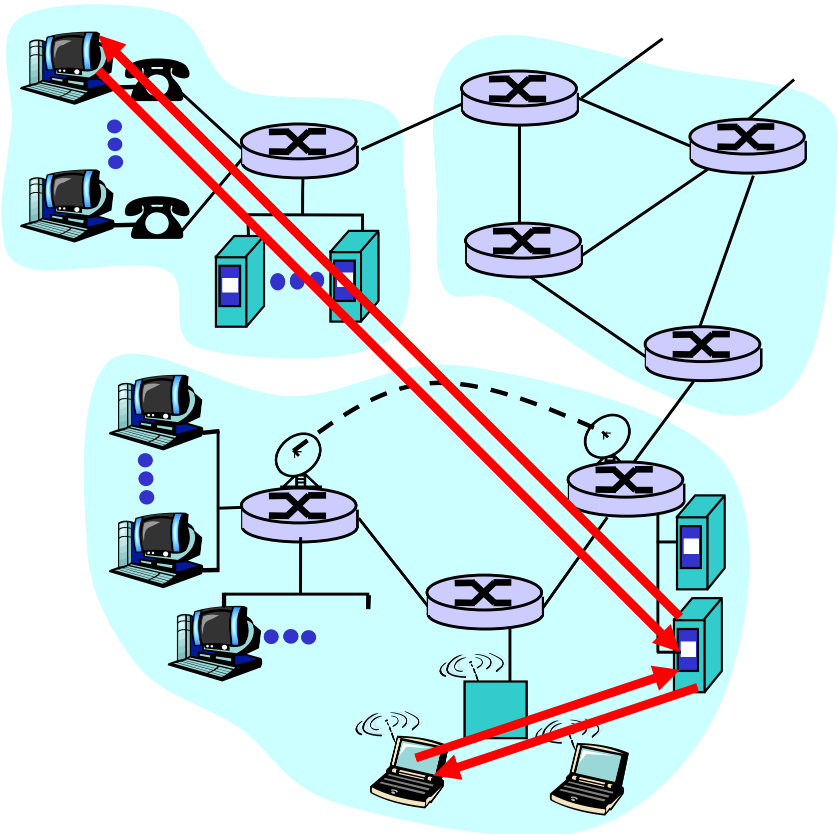
Chapter 2: Outline

- 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 TCP
- 2.8 Socket programming with UDP

Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-server architecture



server:

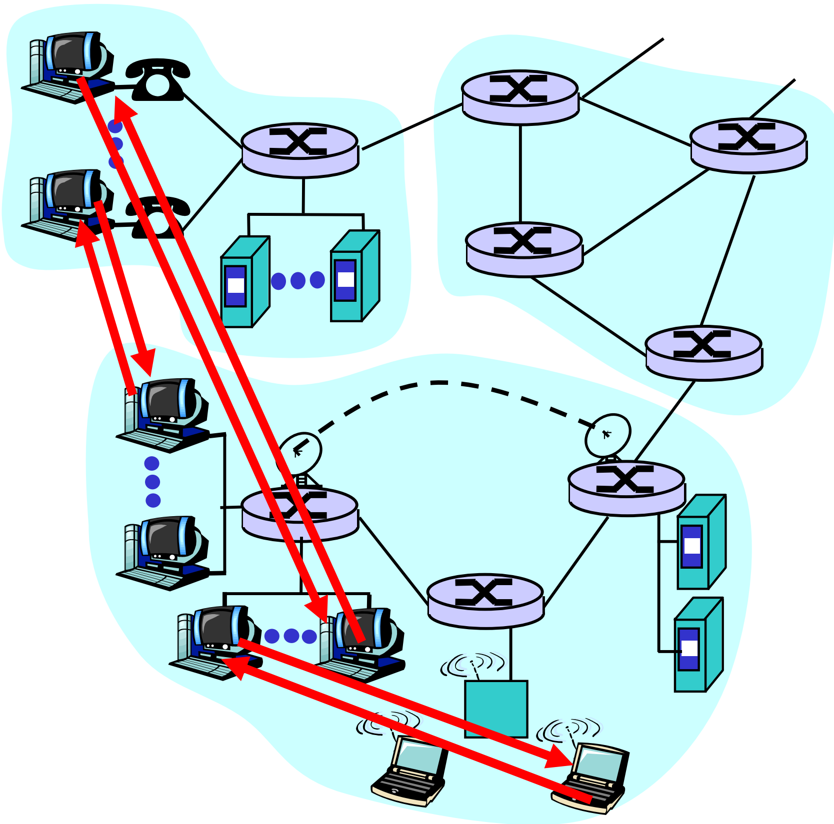
- always-on host
- permanent IP address
- server farms for scaling

clients:

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

Pure P2P architecture

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



Highly scalable

But difficult to manage

Hybrid of client-server and P2P

Napster

- File transfer P2P
- File search centralized:
  - Peers register content at central server
  - Peers query same central server to locate content

Instant messaging

- Chatting between two users is P2P
- Presence detection/location centralized:
  - User registers its IP address with central server when it comes online
  - User contacts central server to find IP addresses of buddies

Processes communicating

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

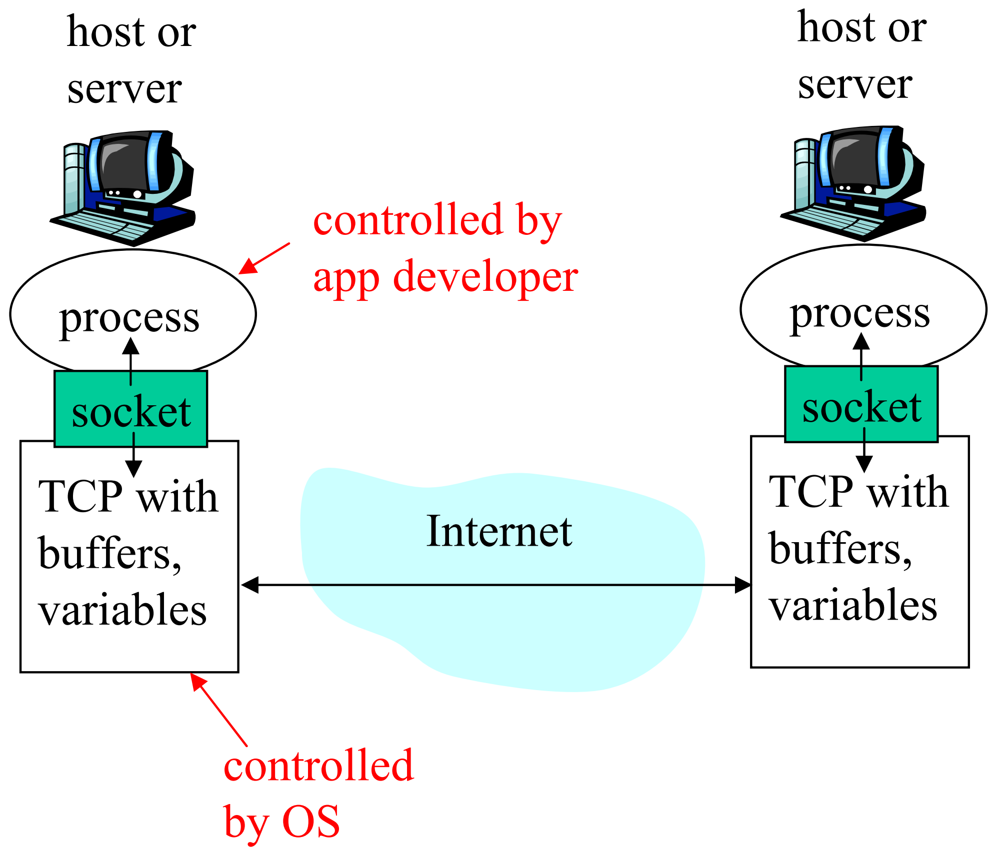
Client process: process that initiates communication

Server process: process that waits to be contacted

- Note: 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 message out door
  - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process



- API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)

Internet transport protocols services

TCP service:

- *connection-oriented:* setup required between client and server processes
- *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 guarantees, security

UDP service:

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

Q: why bother? Why is there a UDP?

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 TCP
- 2.8 Socket programming with UDP

Web and HTTP

First some jargon

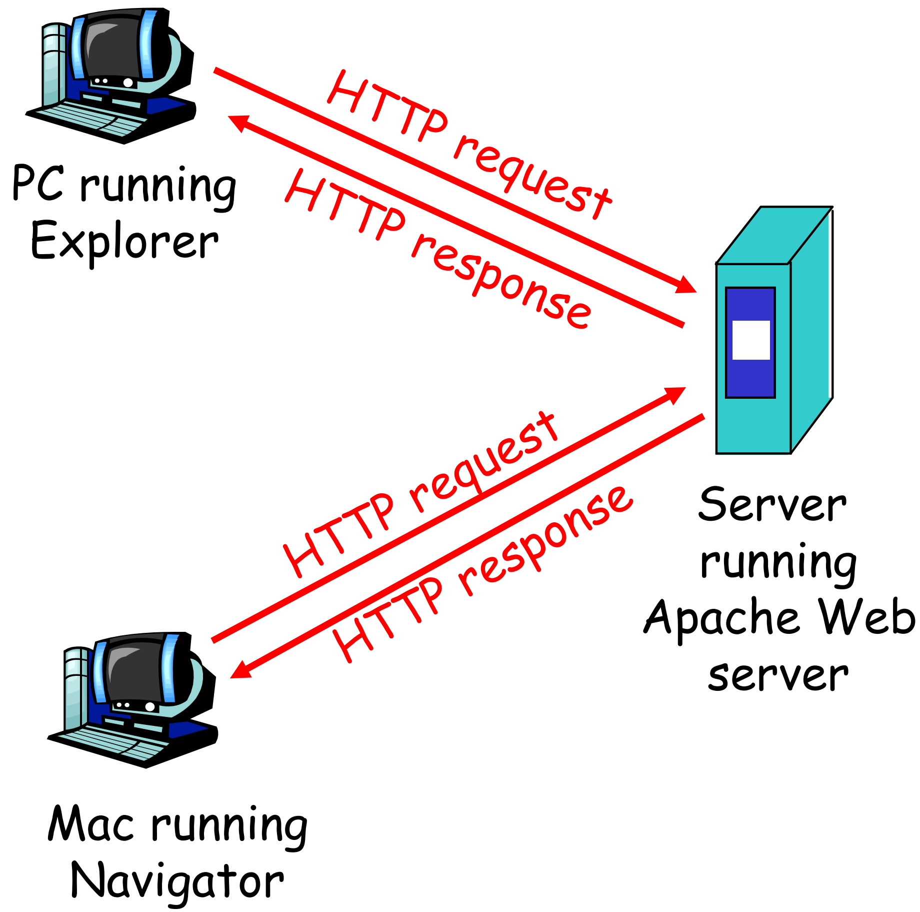
- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL
- Example URL:

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

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2616



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" are complex!

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

HTTP connections

Nonpersistent HTTP

- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent HTTP

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

Nonpersistent HTTP

Suppose user enters URL

`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)

time

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

time

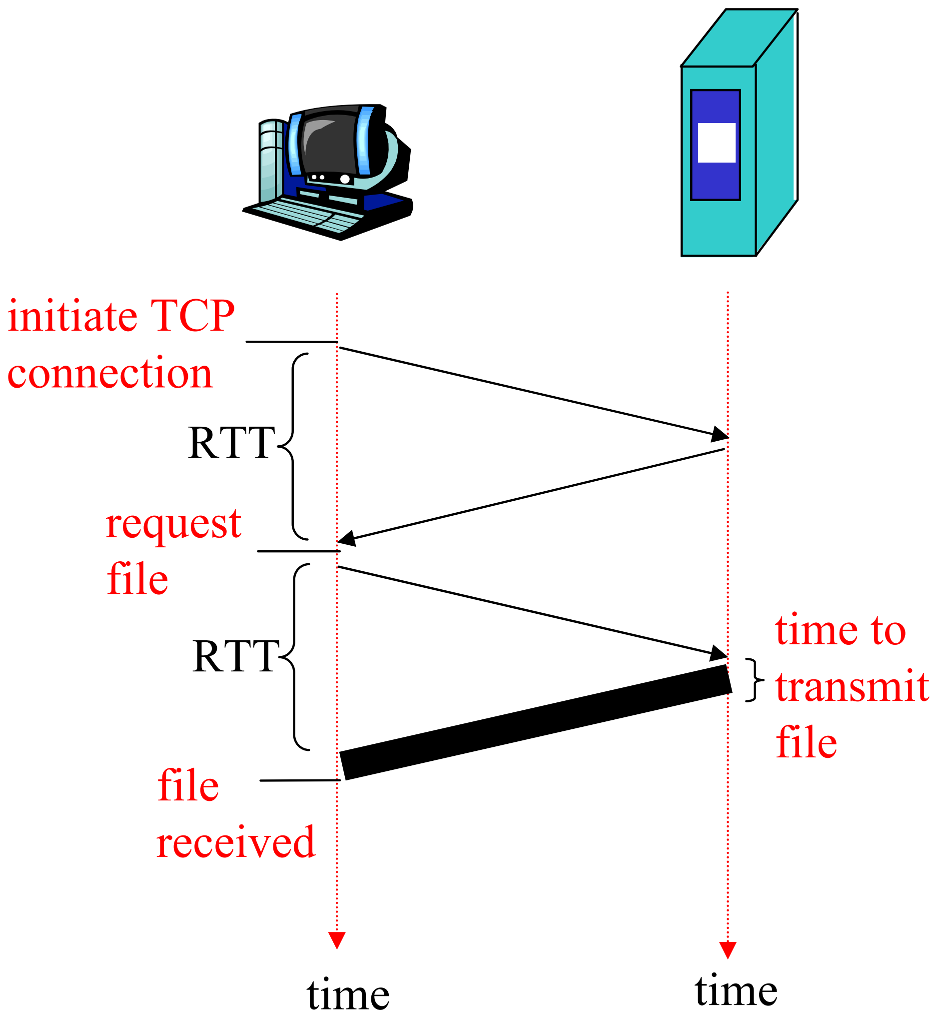
Response time modeling

Definition of RRT: time to send a small packet to travel from client to server and back.

Response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

total = 2RTT+transmit time



Persistent HTTP

Nonpersistent HTTP issues:

- requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- but browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server are sent over connection

Persistent without pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent with pipelining:

- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

request line (GET, POST, HEAD commands)

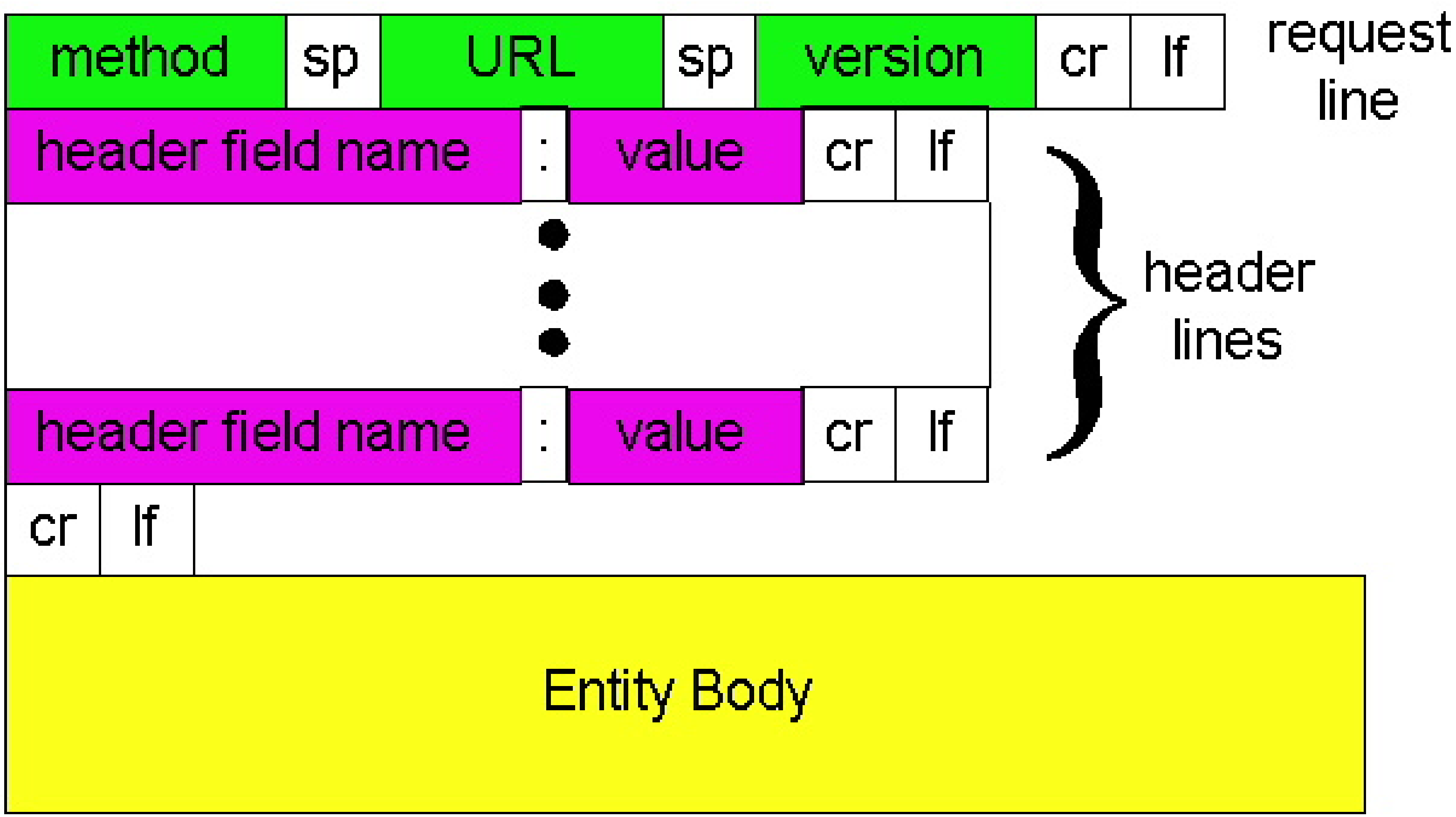
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header lines

(extra carriage return, line feed)

Carriage return, line feed indicates end of message

HTTP request message: general format



Uploading form input

Post method:

- Web page often includes form input
- Input is uploaded to server in entity body

URL method:

- Uses GET method
- Input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```

Method types

HTTP/1.0

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1

- GET, POST, HEAD
- 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)

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

header
lines

data, e.g.,
requested
HTML file

HTTP response status codes

In first line in server->client response message.

A few sample codes:

200 OK

- request succeeded, requested object later in this message

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!

User-server state: cookies

Many major Web sites use cookies

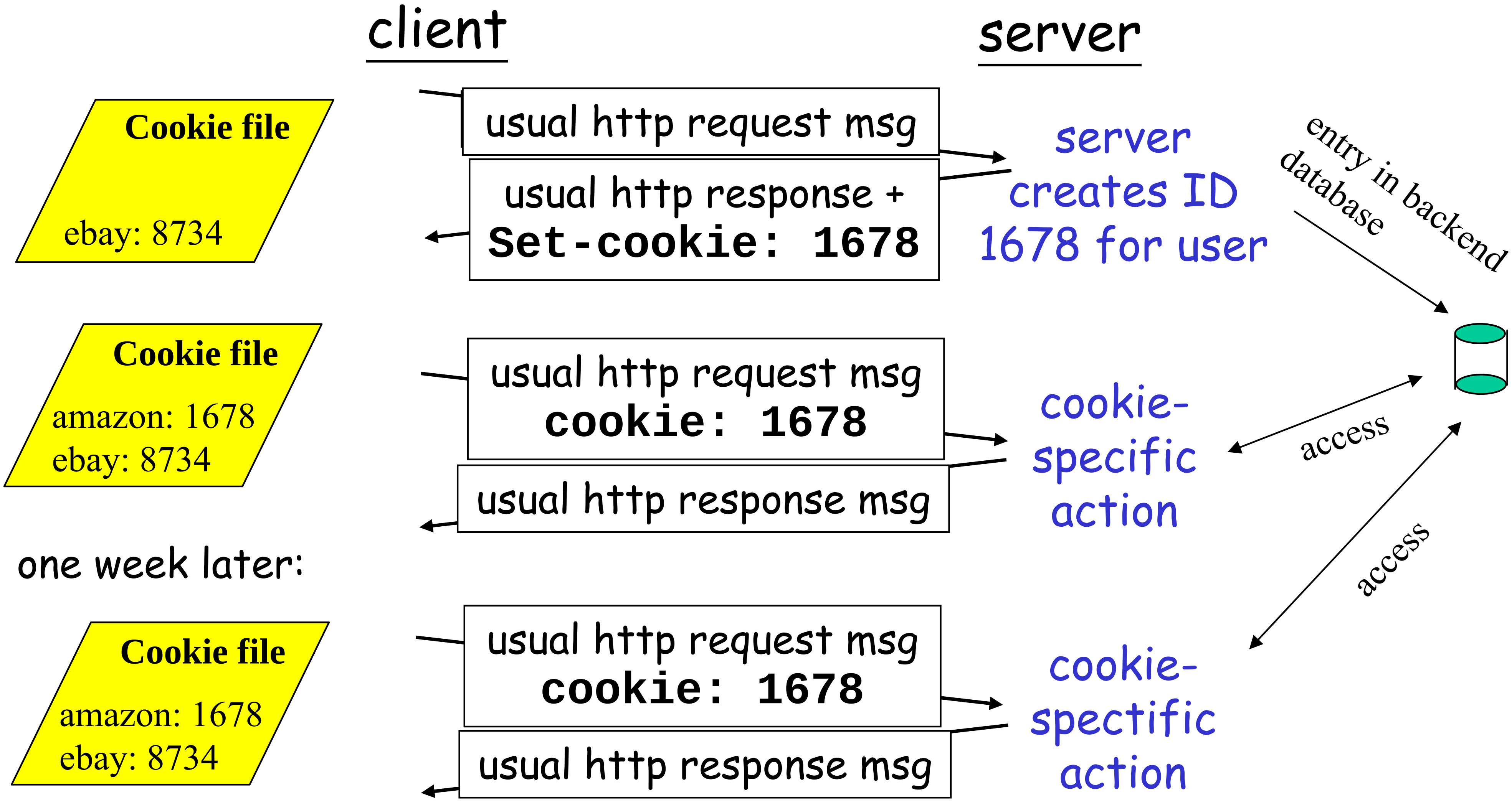
Four components:

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

Example:

- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

What cookies can bring:

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

aside

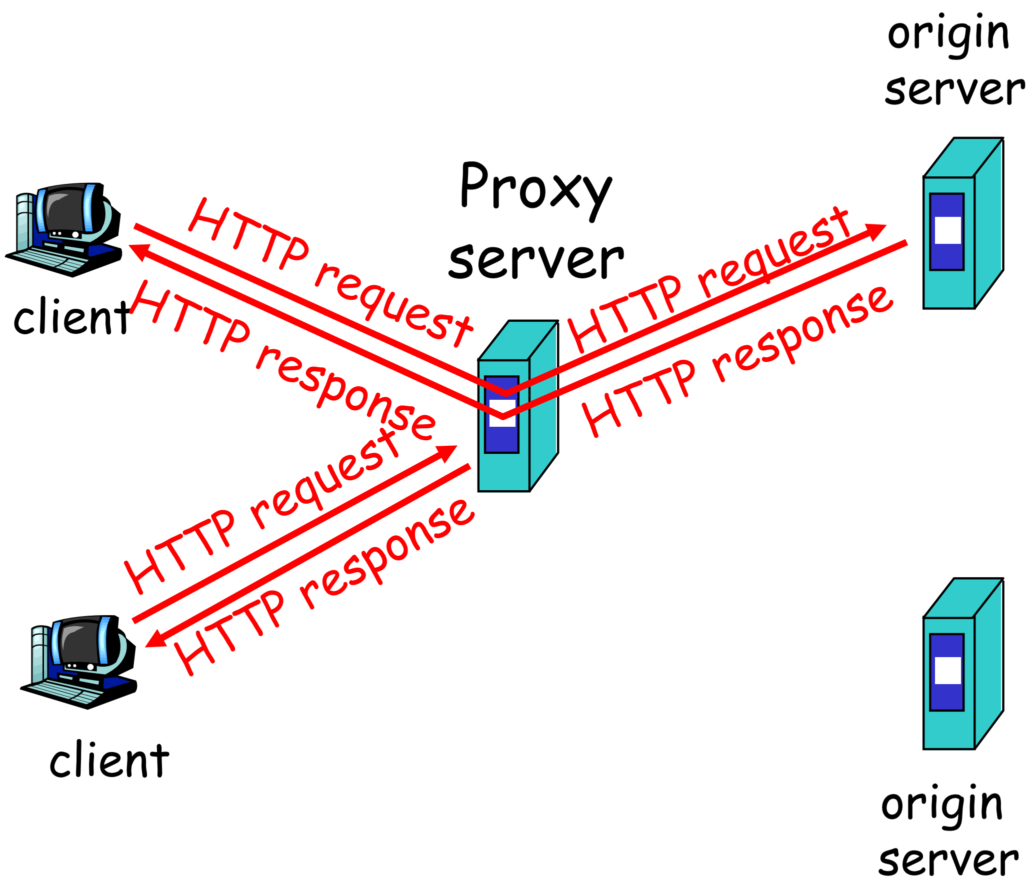
Cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use redirection & cookies to learn yet more
- advertising companies obtain info across sites

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

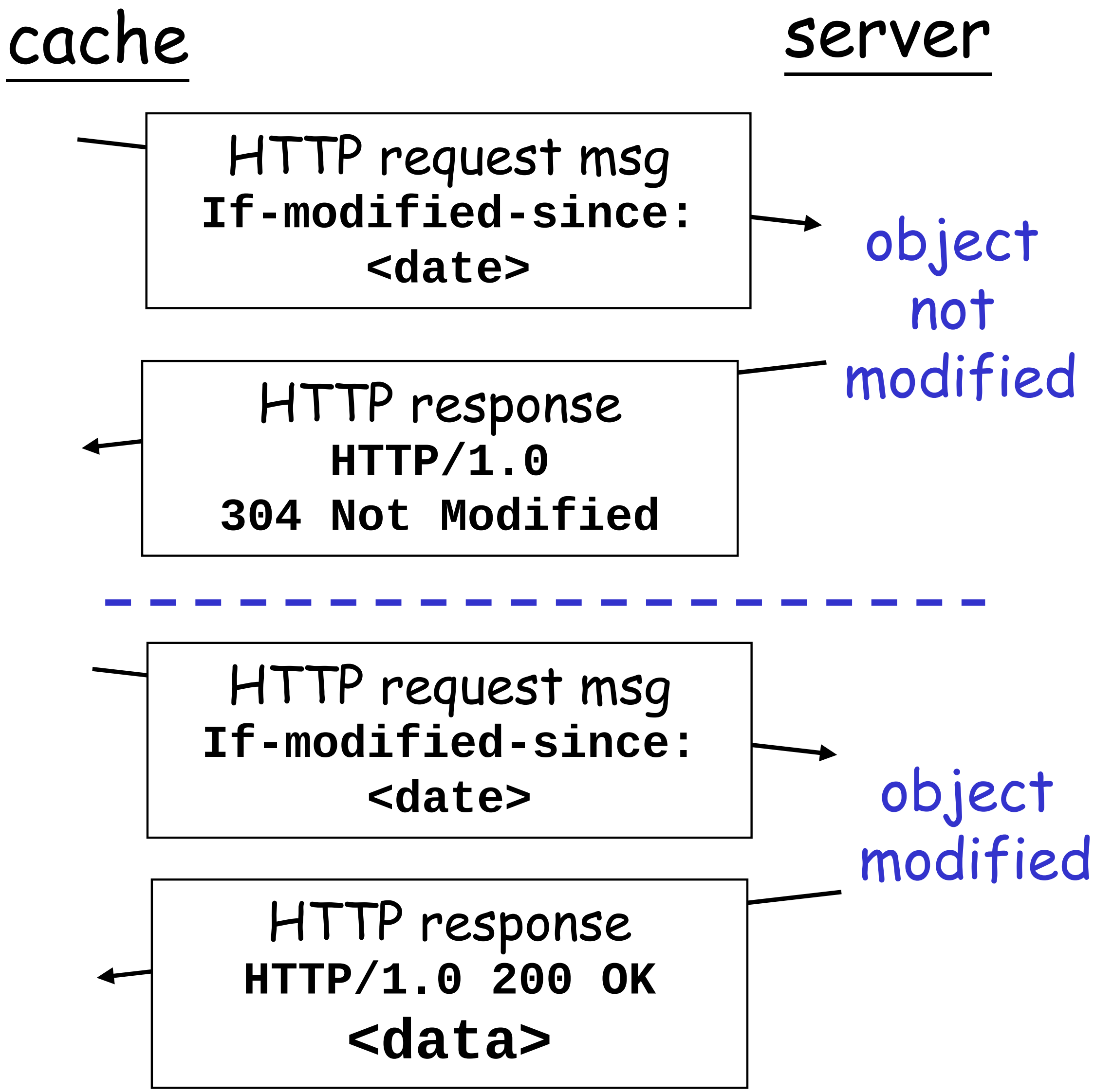
- Cache acts as both client and server
- Typically cache is installed by ISP (university, company, residential ISP)

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 (but so does P2P file sharing)

Conditional GET

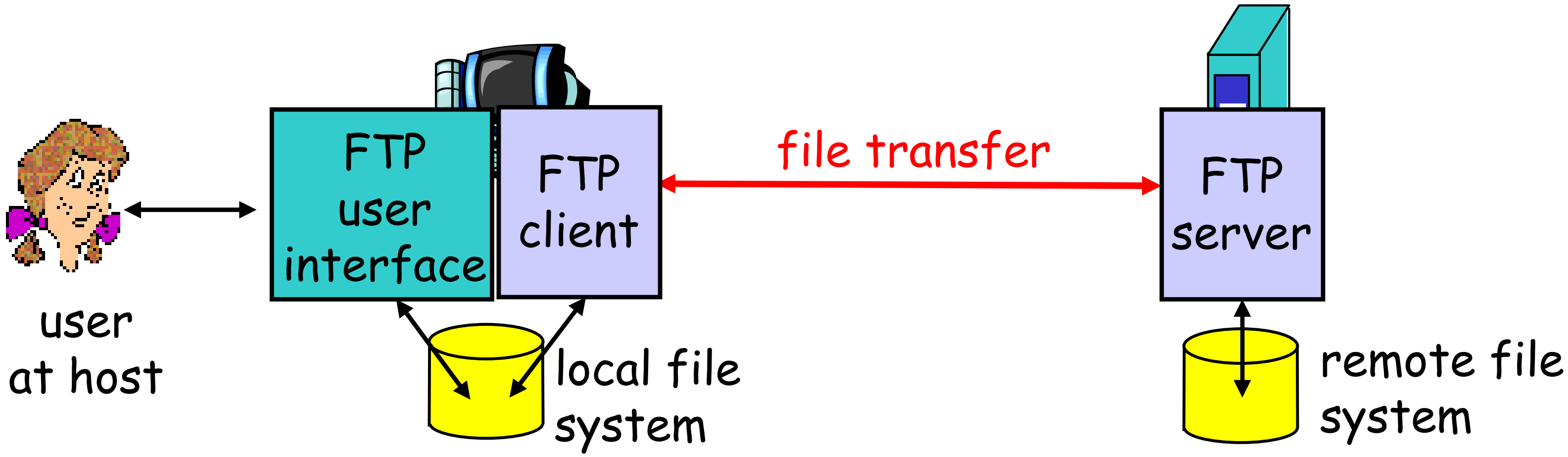
- Goal: don't send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request
  If-modified-since: <date>
- server: response contains no object if cached copy is up-to-date:
  HTTP/1.0 304 Not Modified



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 TCP
- 2.8 Socket programming with UDP

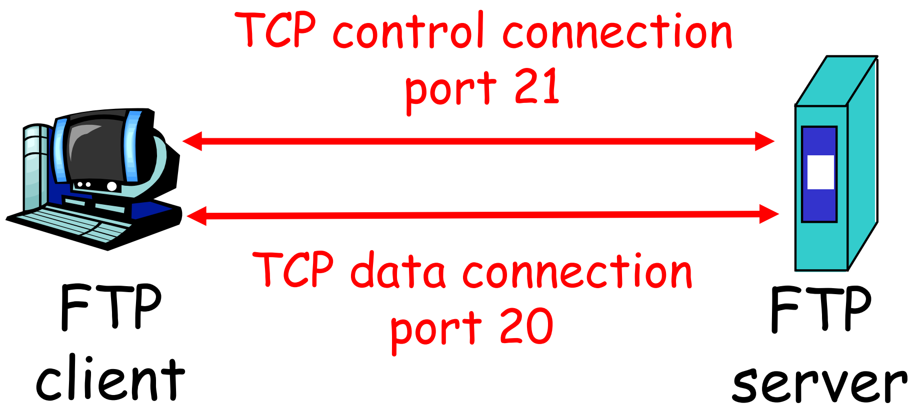
FTP: the file transfer protocol



- 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: port 21

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, specifying TCP as transport protocol
- Client obtains authorization over control connection
- Client browses remote directory by sending commands over control connection.
- When server receives a command for a file transfer, the server opens a TCP data connection to client
- After transferring one file, server closes connection.



- Server opens a second 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

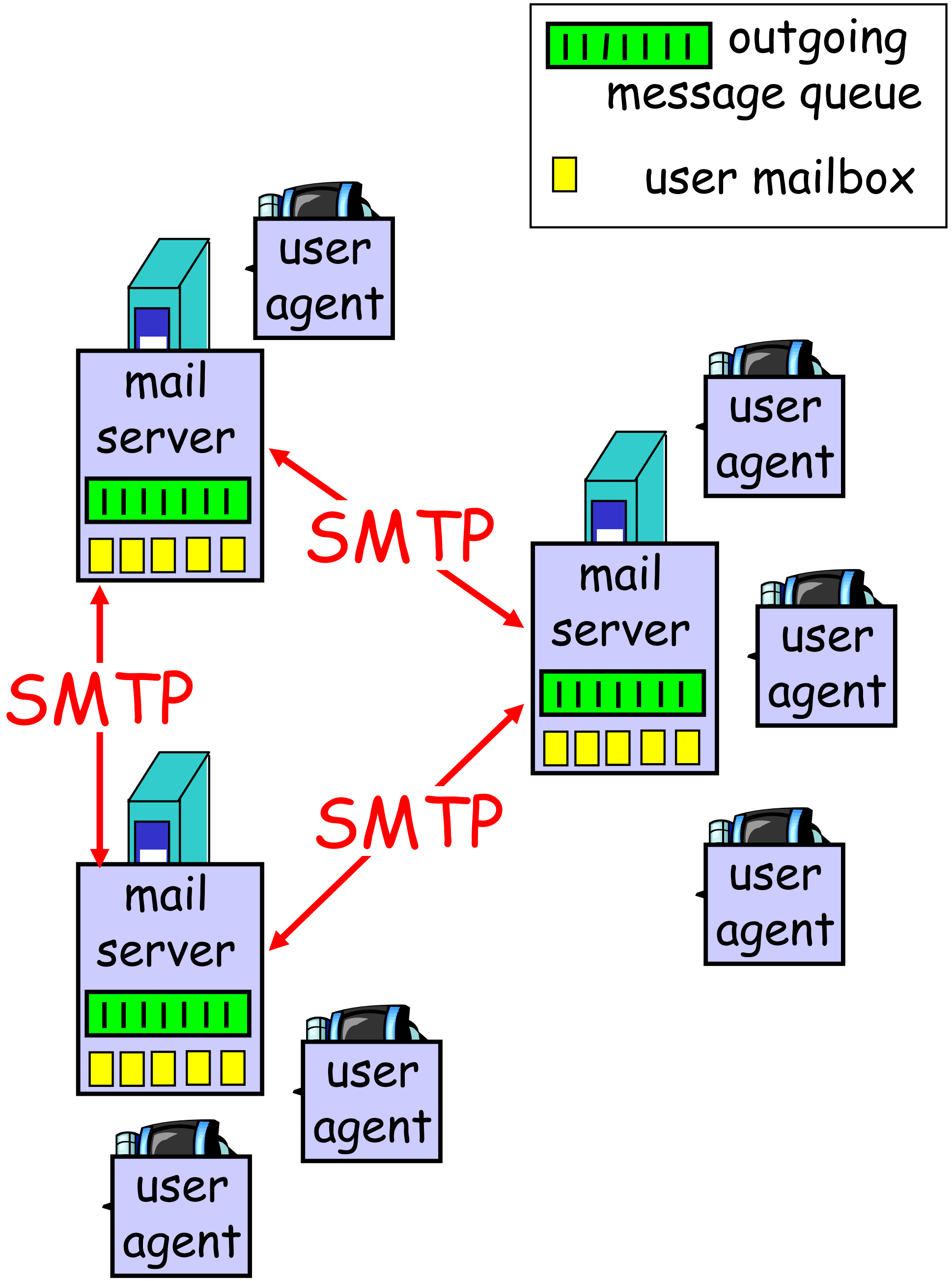
Sample return codes

- 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**

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 TCP
- 2.8 Socket programming with UDP

Electronic Mail



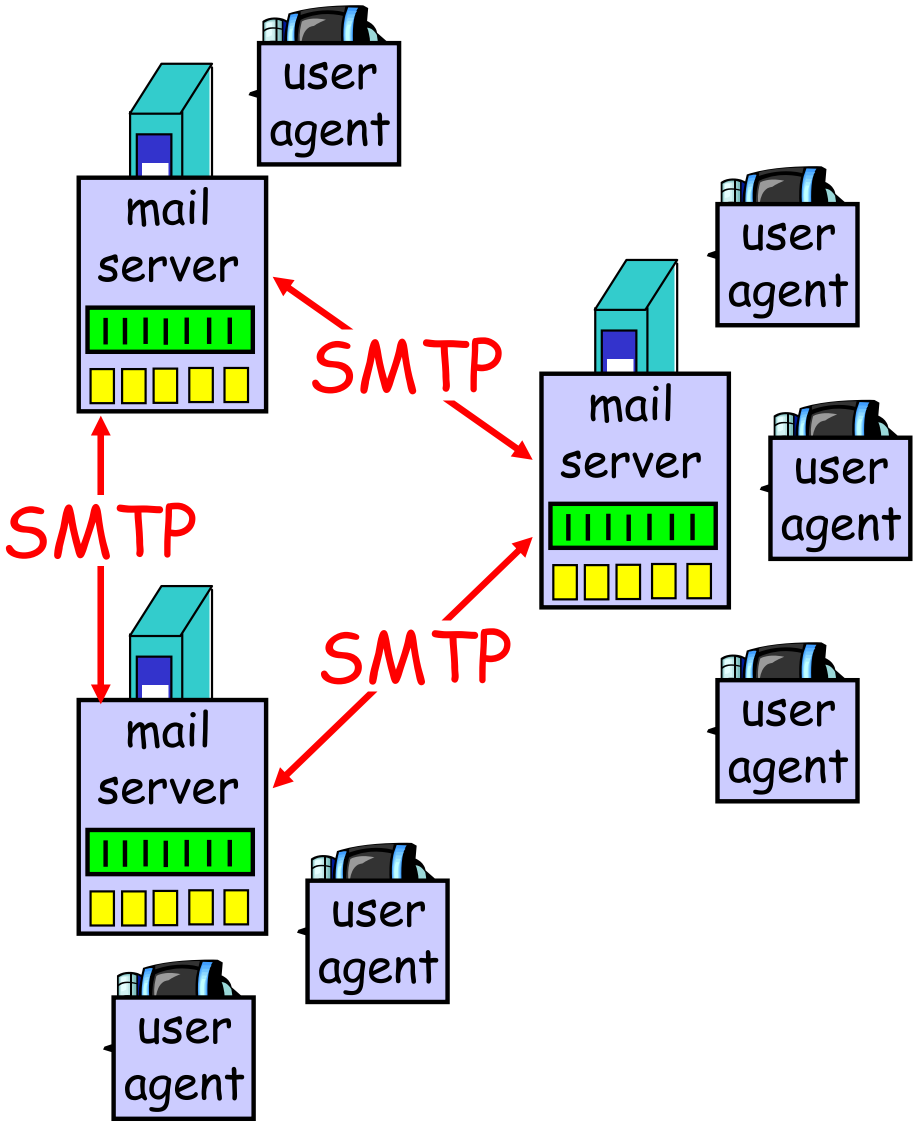
Three major components:
- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Netscape Messenger
- 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
  - client: sending mail server
  - "server": receiving mail server

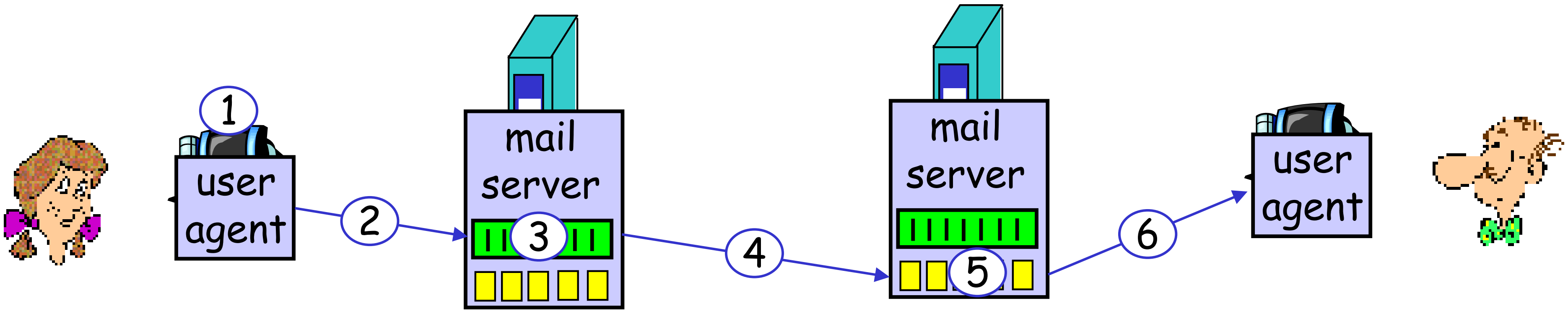


Electronic Mail: SMTP [RFC 2821]

- 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
  - 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 and "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



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)

SMTP: final words

- 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

Comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

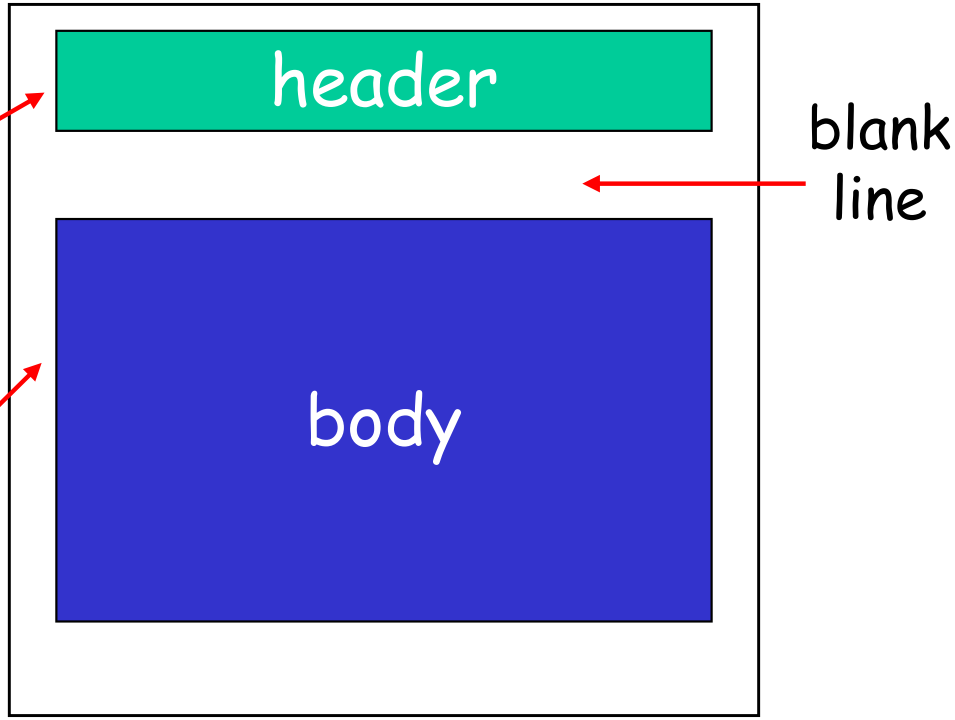
Mail message format

SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

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

  different from SMTP commands!
- body
  - the "message", ASCII characters only



Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type

MIME version

method used to encode data

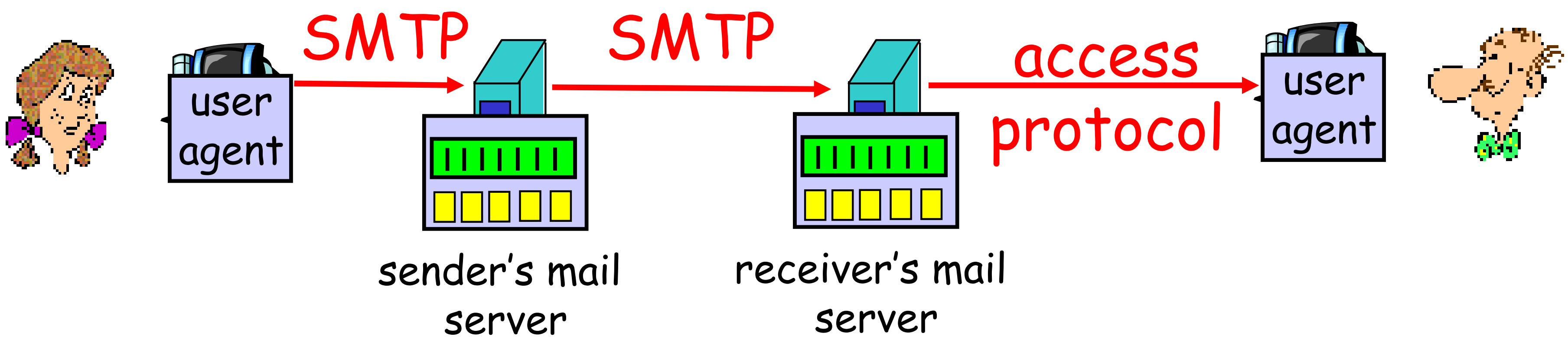
multimedia data type, subtype, parameter declaration

encoded data

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
```

Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server
  - HTTP: 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
```

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 TCP
- 2.8 Socket programming with UDP

DNS: Domain Name System

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

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

Q: map between IP addresses and name ?

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

DNS

DNS services

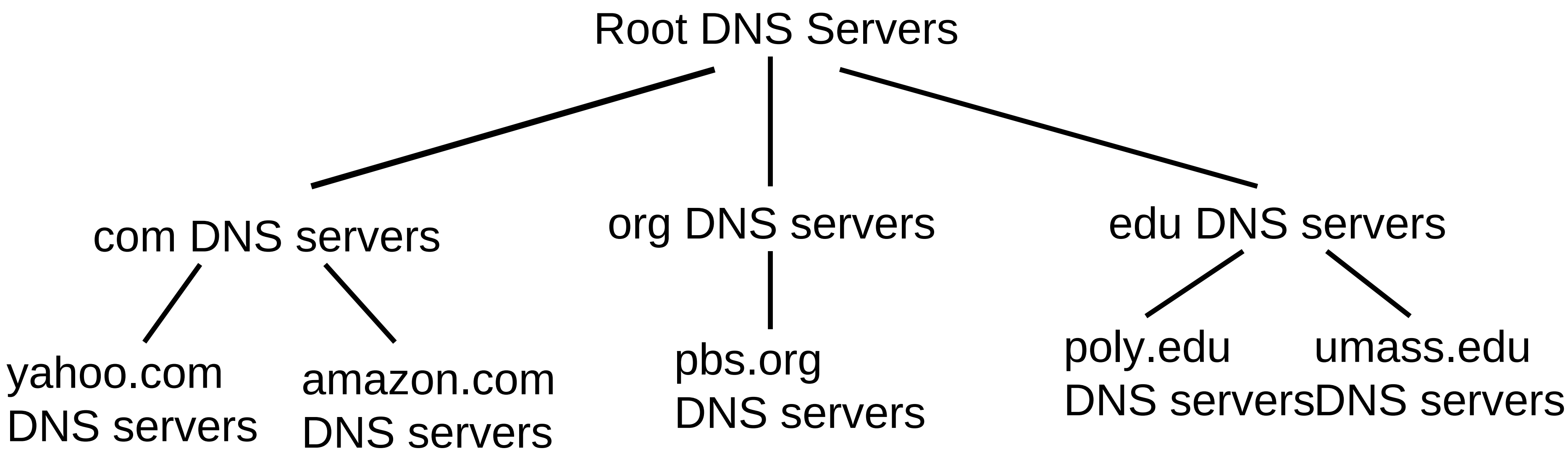
- Hostname to IP address translation
- Host aliasing
  - Canonical and alias names
- Mail server aliasing
- Load distribution
  - Replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

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

doesn't *scale!*

Distributed, Hierarchical Database

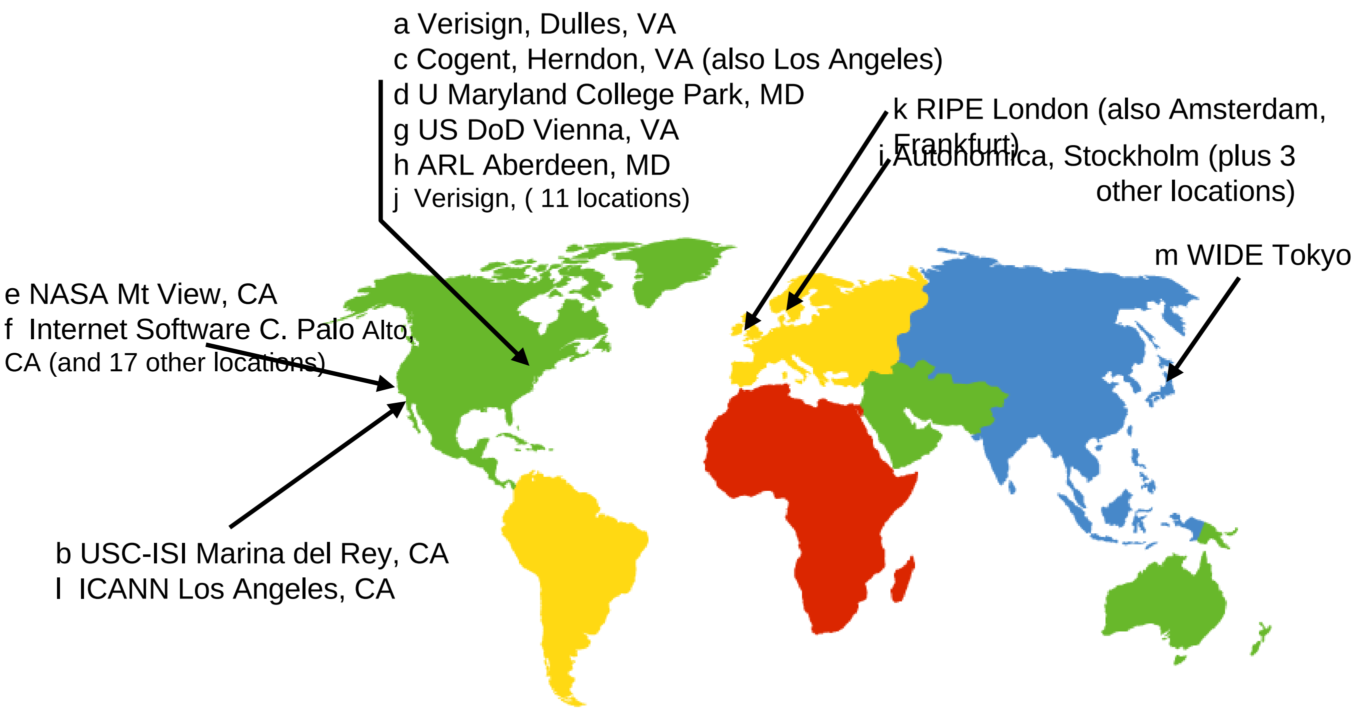


Client wants IP for www.amazon.com; 1st approx:

- 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: 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



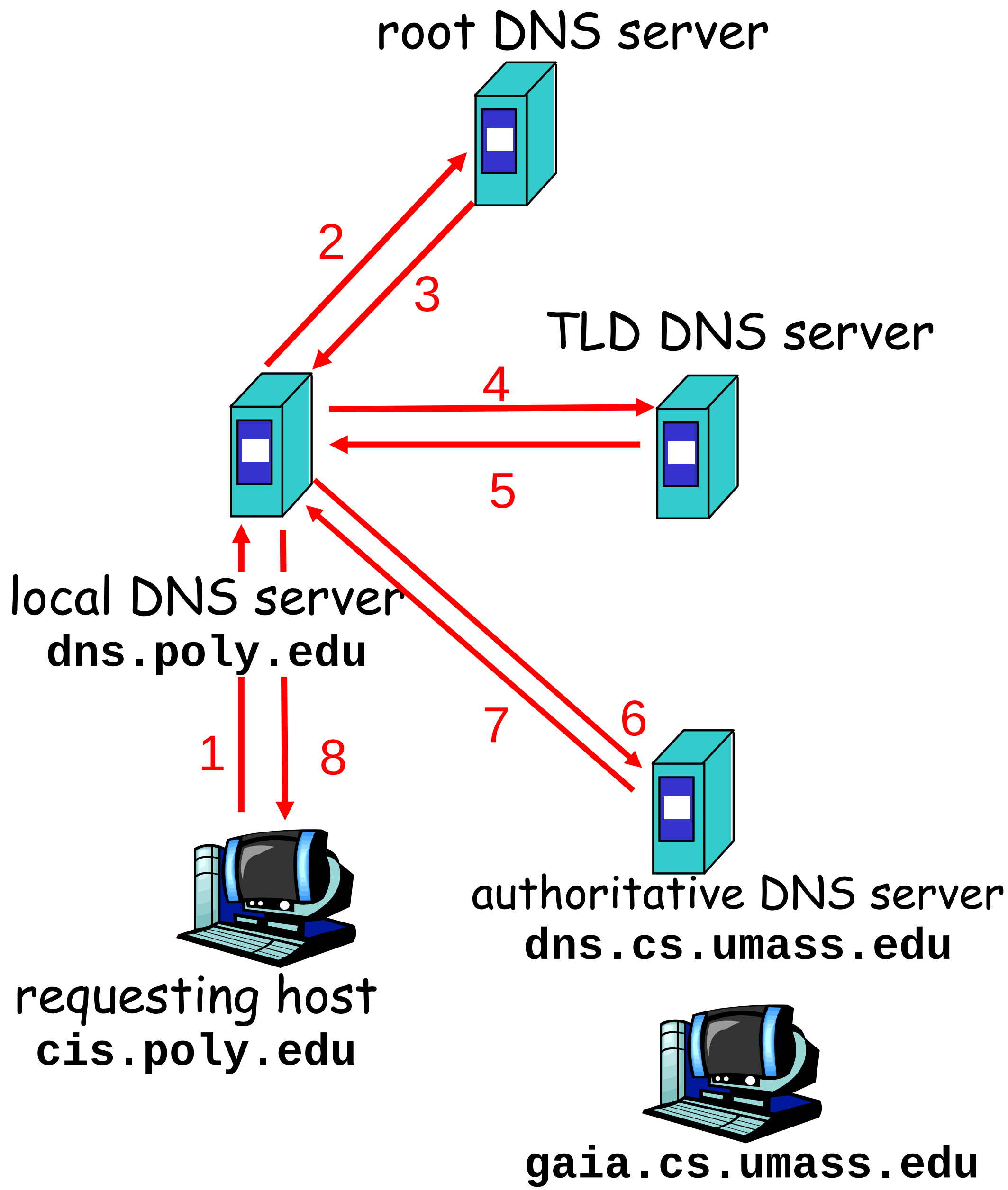
13 root name servers worldwide

Name Servers

- Top-level domain (TLD) servers: responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  - Network solutions maintains servers for com TLD
  - Educause for edu TLD
- Authoritative DNS servers: organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web and mail).
  - Can be maintained by organization or service provider
- Local Name Server: also called "default name server"
  - When a host makes a DNS query, query is sent to its local DNS server. Acts as a proxy, forwards query into hierarchy.
  - Does not strictly belong to hierarchy
    - Each ISP (residential ISP, company, university) has one.

Example

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



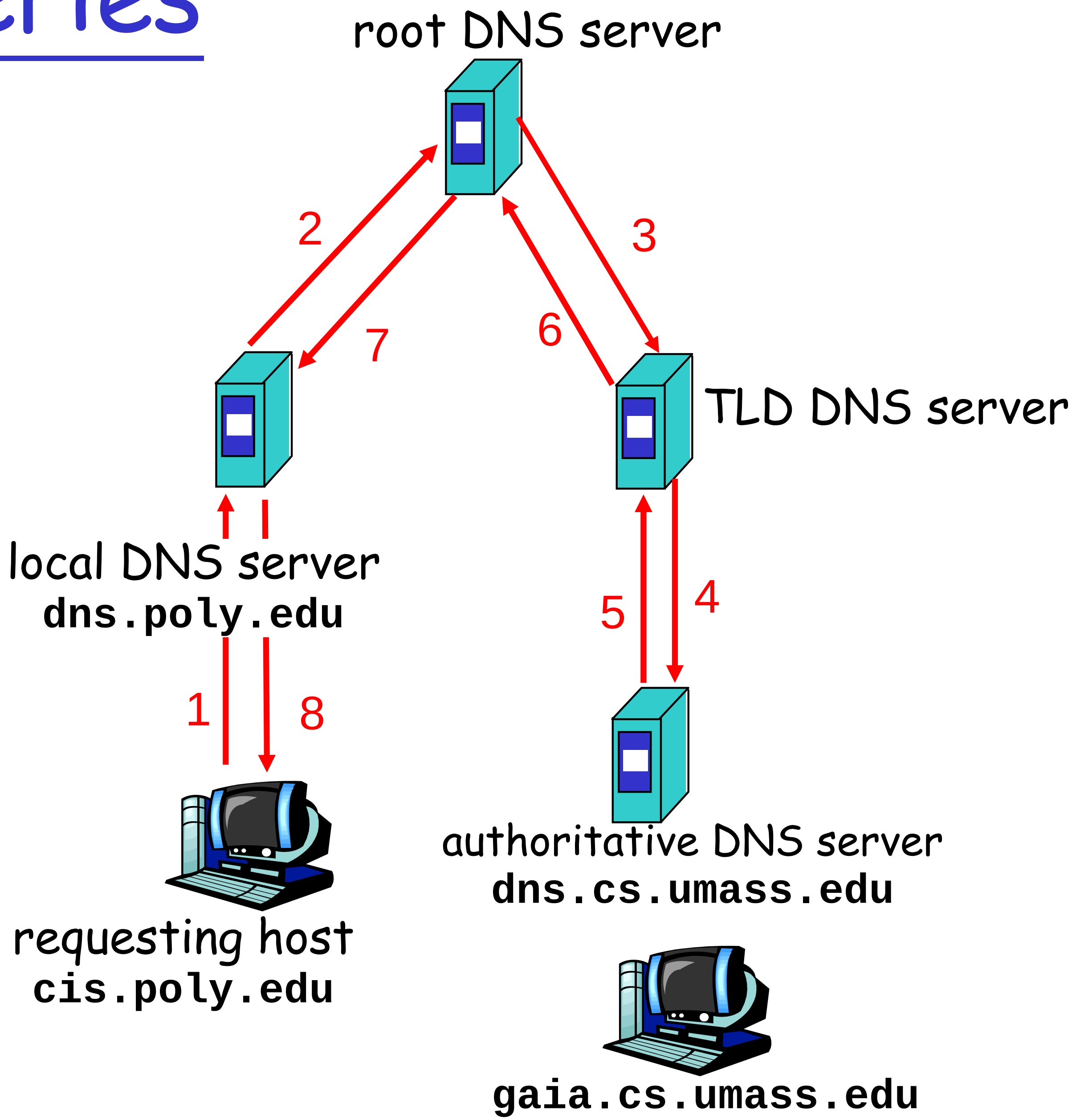
Recursive queries

recursive query:

- puts burden of name resolution on contacted name server
- heavy load?

iterated query:

- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS: caching and updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited
- update/notify mechanisms under design by IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

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 IP address of authoritative name server for this domain
- Type=CNAME
  - **name** is alias name for some "cannonical" (the real) name

    control

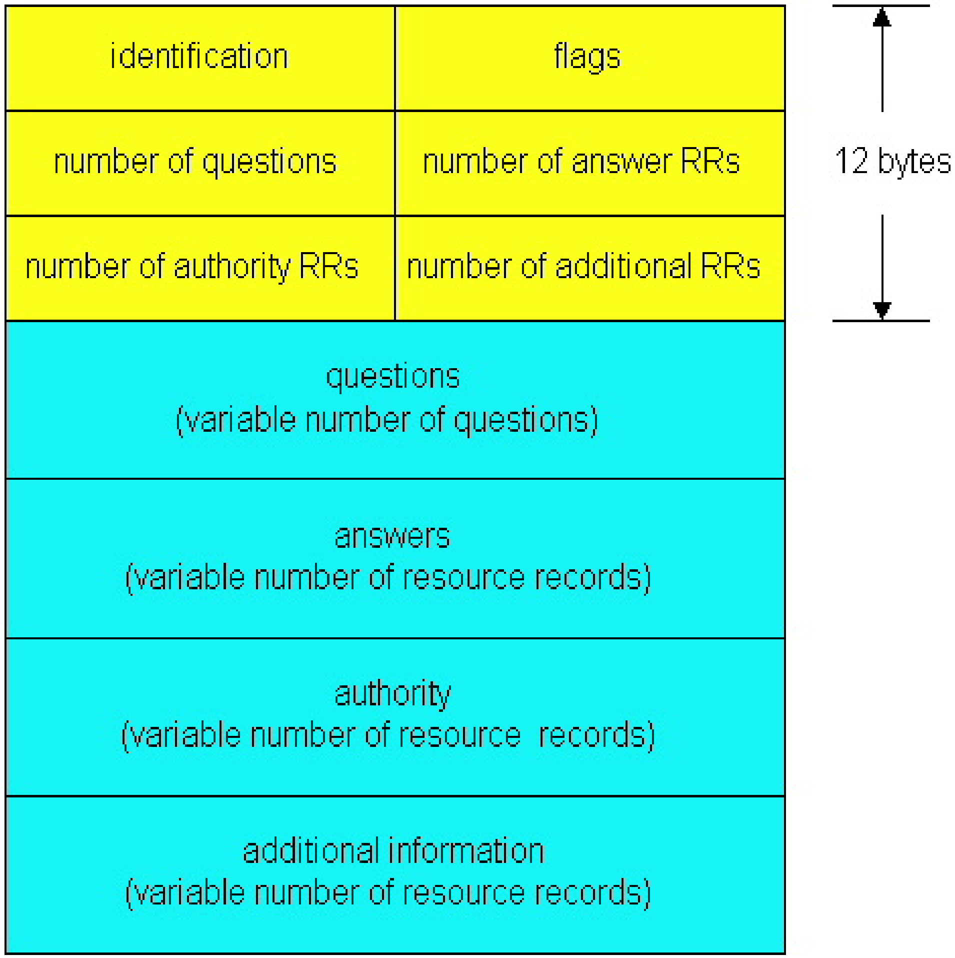
    `www.ibm.com` is really `servereast.backup2.ibm.com`
  - **value** is cannonical name
- Type=MX
  - **value** is name of mailserver associated with **name**

DNS protocol, messages

DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



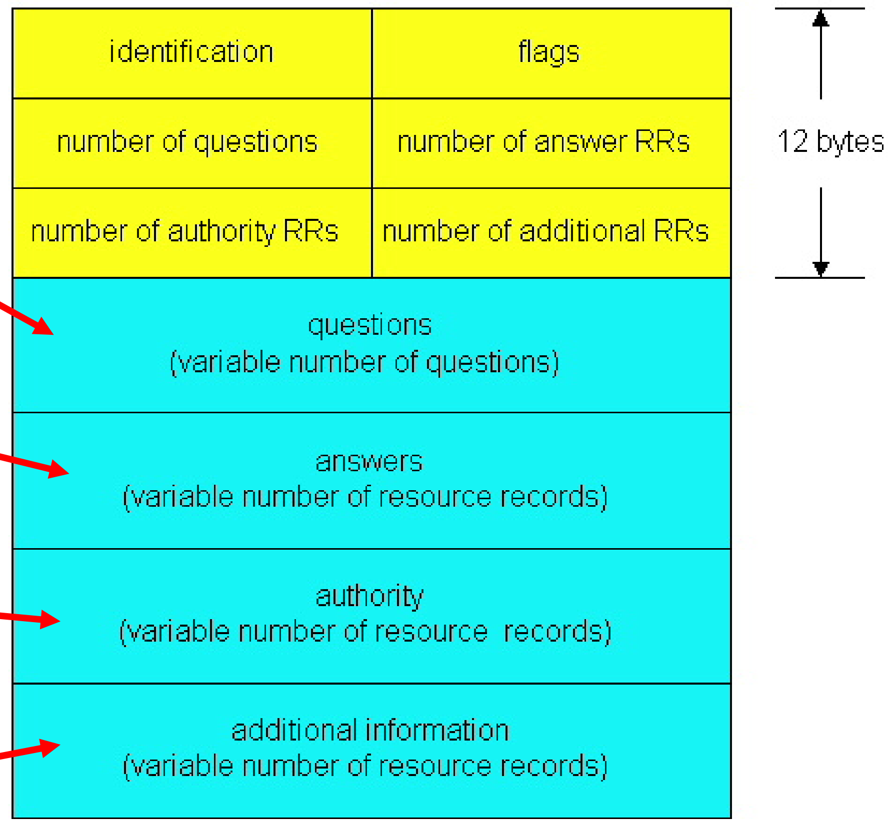
DNS protocol, messages

Name, type fields
for a query

RRs in reponse
to query

records for
authoritative servers

additional "helpful"
info that may be used



Inserting records into DNS

- Example: just created startup “Network Utopia”
- Register name networkuptopia.com at a registrar (e.g., Network Solutions)
  - Need to provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  - Registrar inserts two RRs into the com TLD server:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

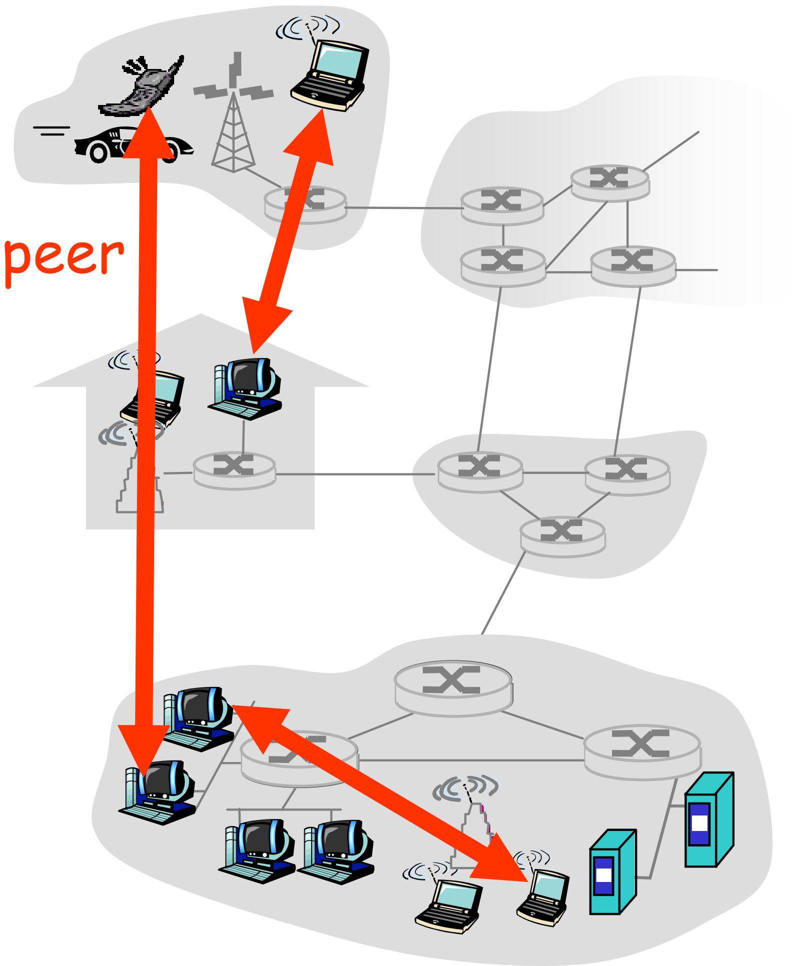
- Put in authoritative server Type A record for www.networkuptopia.com and Type MX record for networkutopia.com
- How do people get the IP address of your Web site?

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 TCP
- 2.8 Socket programming with UDP

Pure P2P architecture

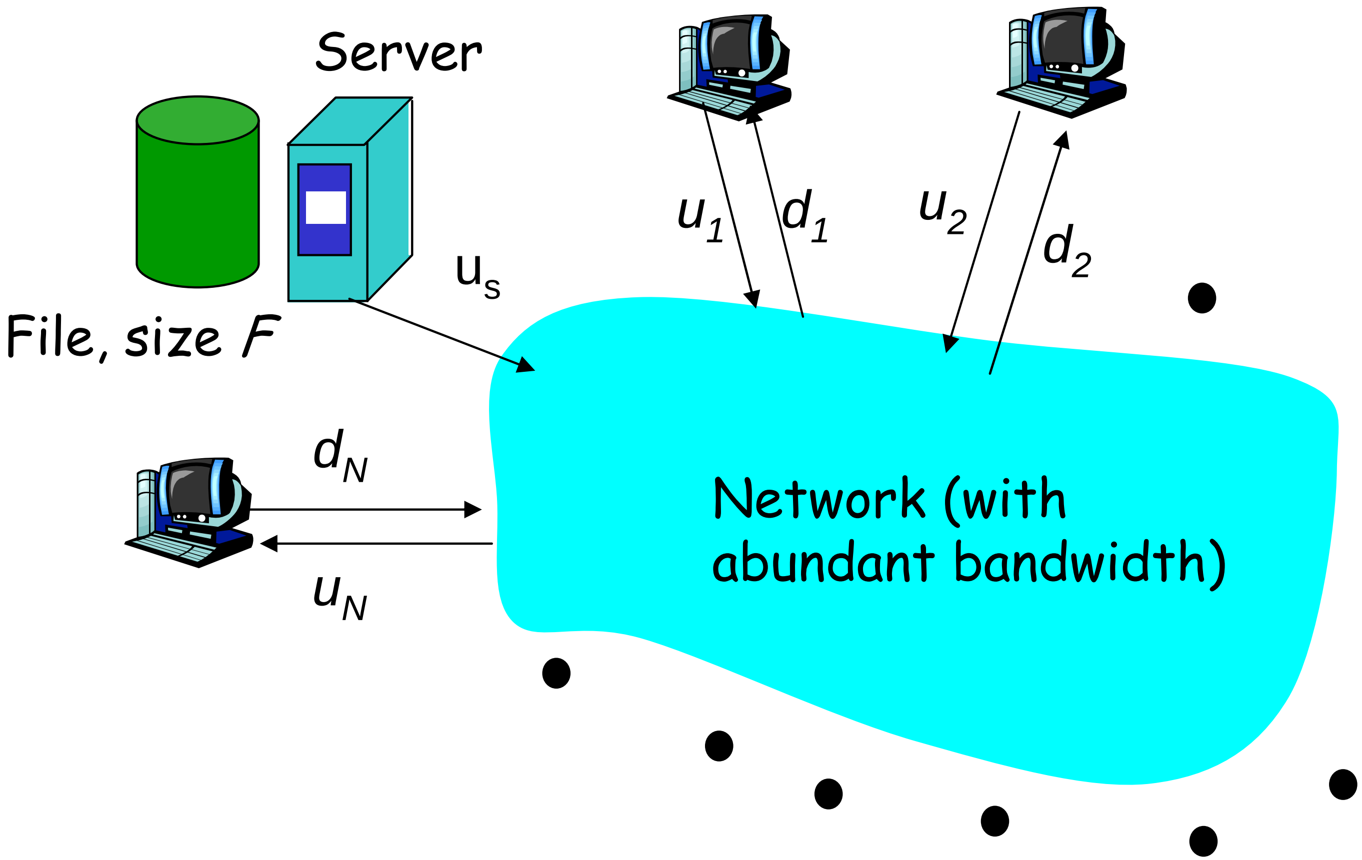
- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses



- Three topics:
  - File distribution
  - Searching for information
  - Case Study: Skype

File Distribution: Server-Client vs P2P

Question : How much time to distribute file from one server to *N peers*?



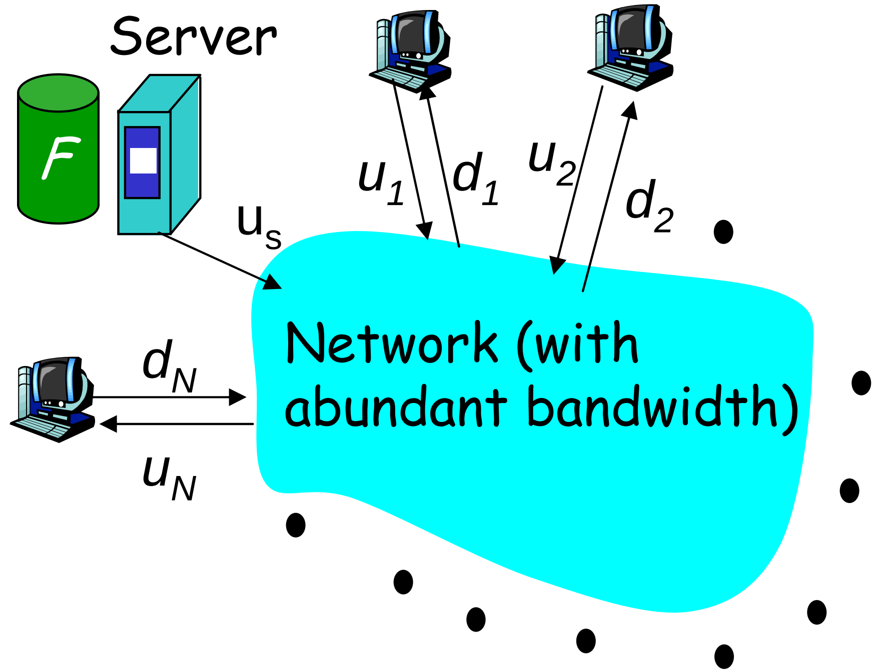
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download

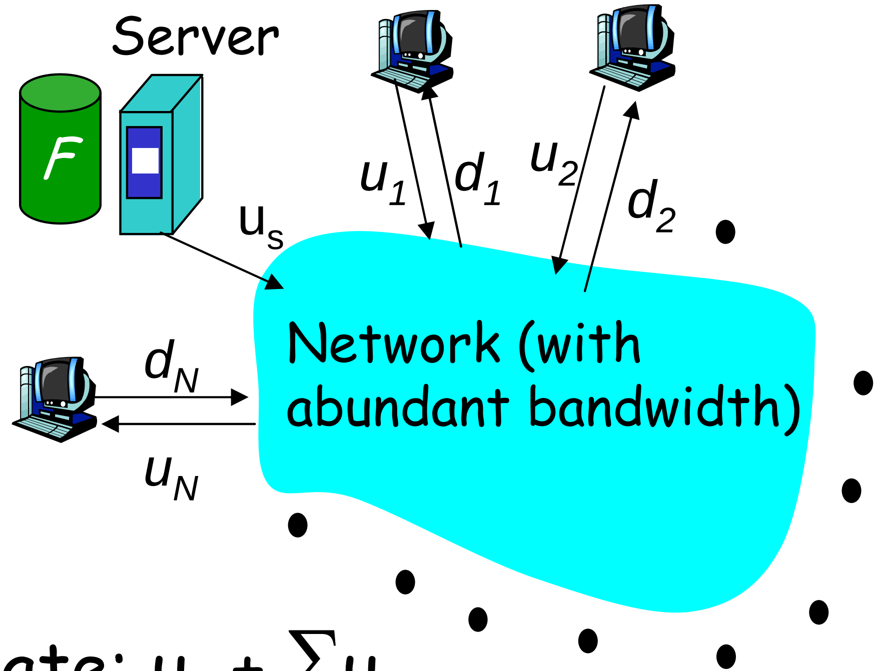


Time to distribute F to N clients using client/server approach $= d_{cs} = \max_i \{ NF/u_s, F/\min(d_i) \}$

increases linearly in N (for large N)

File distribution time: P2P

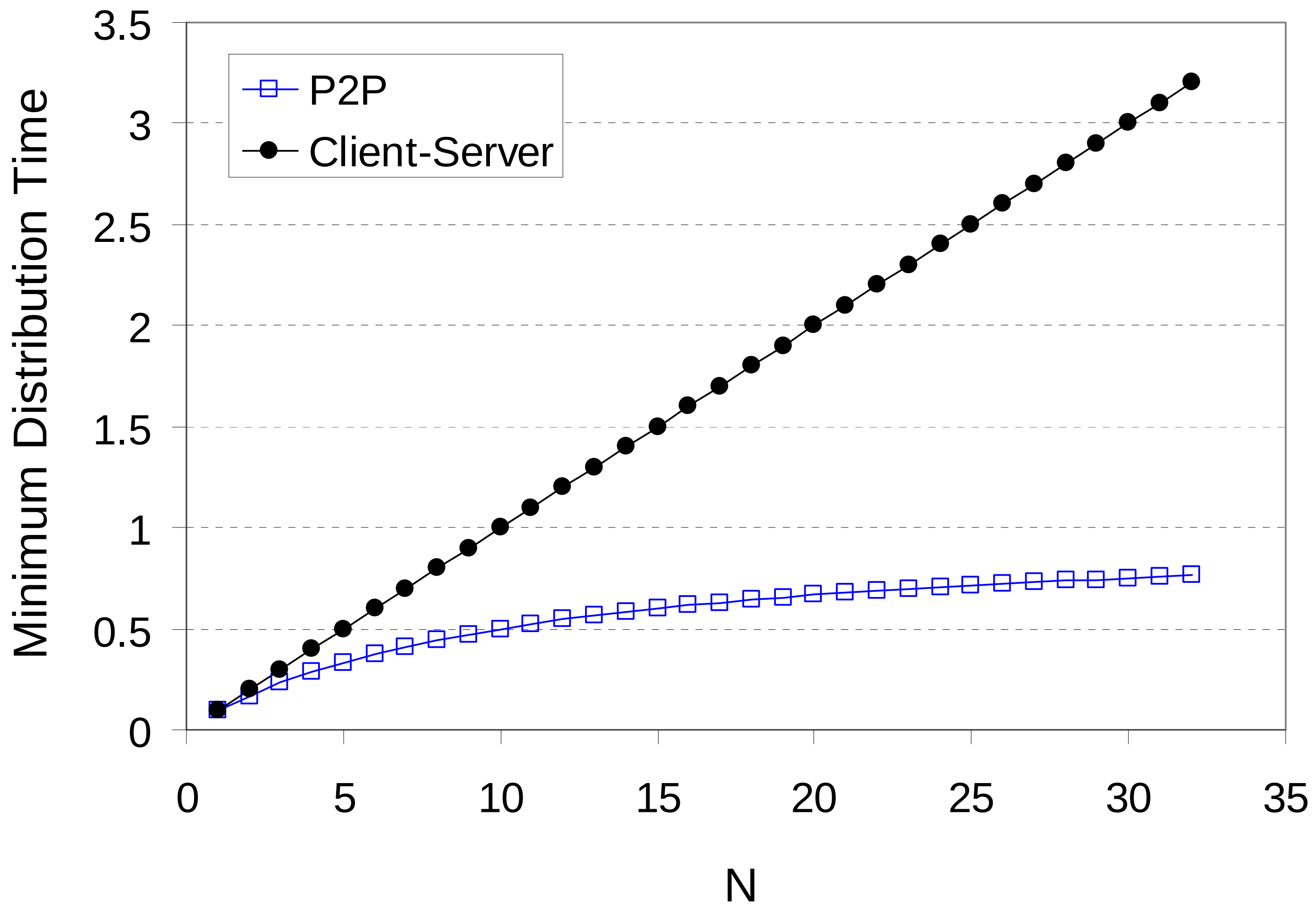
- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \Sigma u_i$



$$d_{P2P} = \max \{ F/u_s, F/min(d_i) , NF/(u_s + \Sigma u_i) \}$$

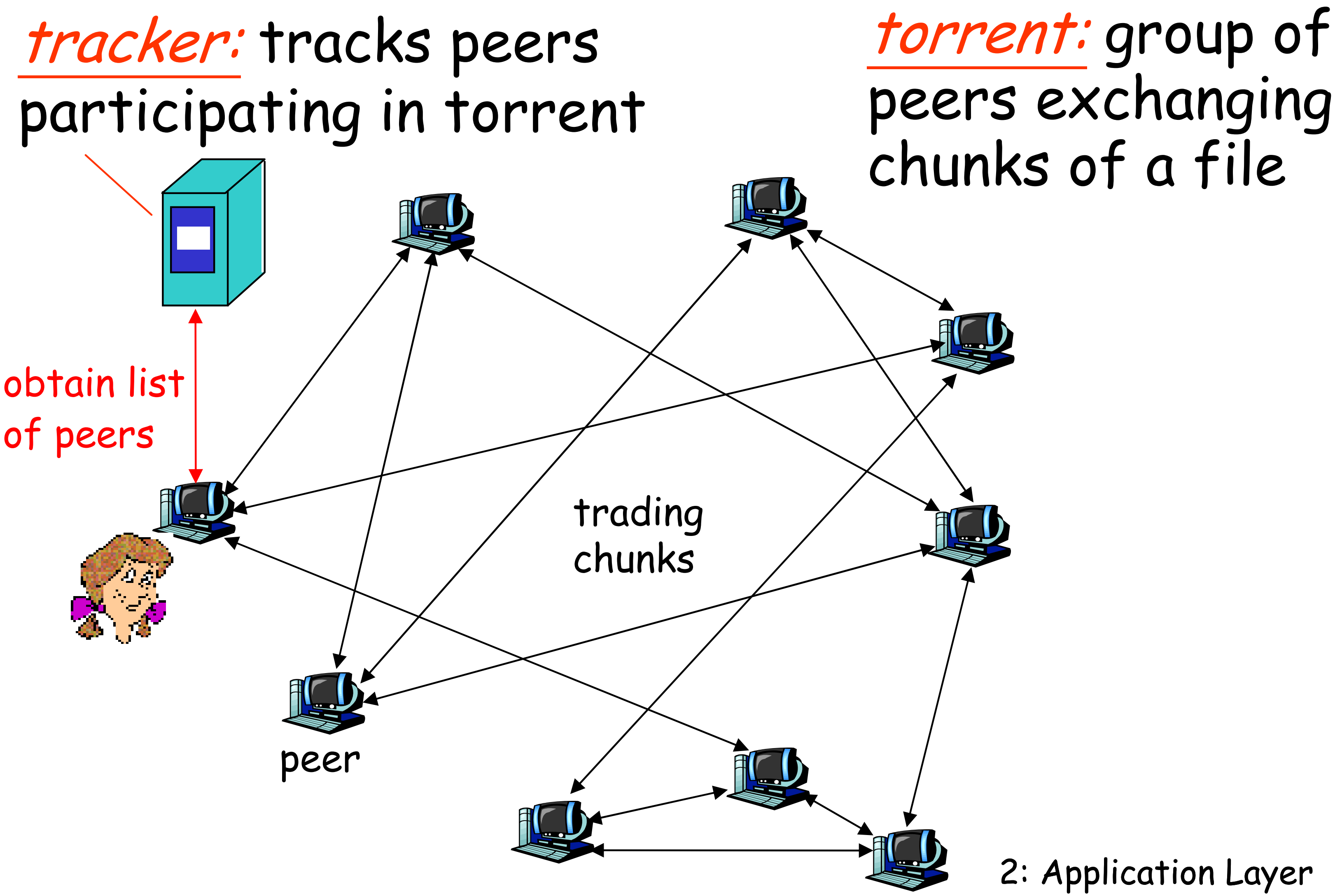
Server-client vs. P2P: example

Client upload rate = u, F/u = 1 hour, u_s = 10u, $d_{min} \geq u_s$

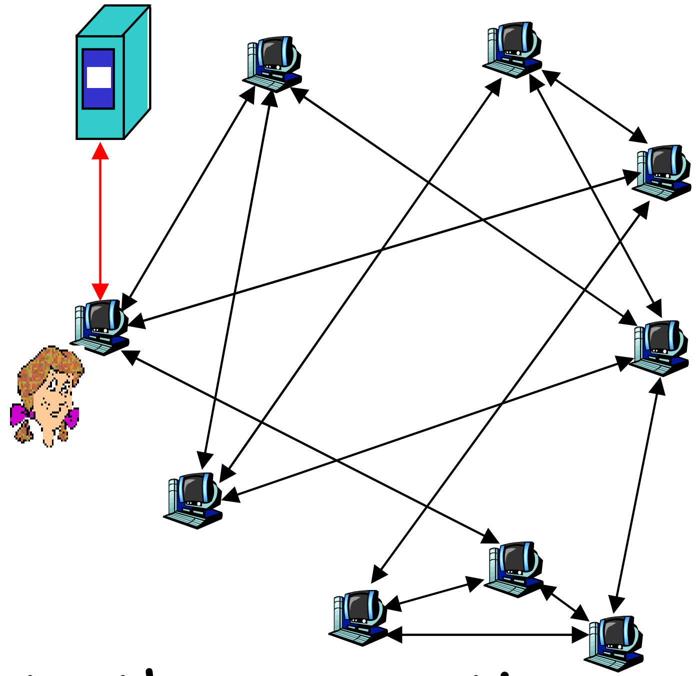


File distribution: BitTorrent

- P2P file distribution



BitTorrent (1)



- file divided into 256KB *chunks*.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain

BitTorrent (2)

Pulling Chunks

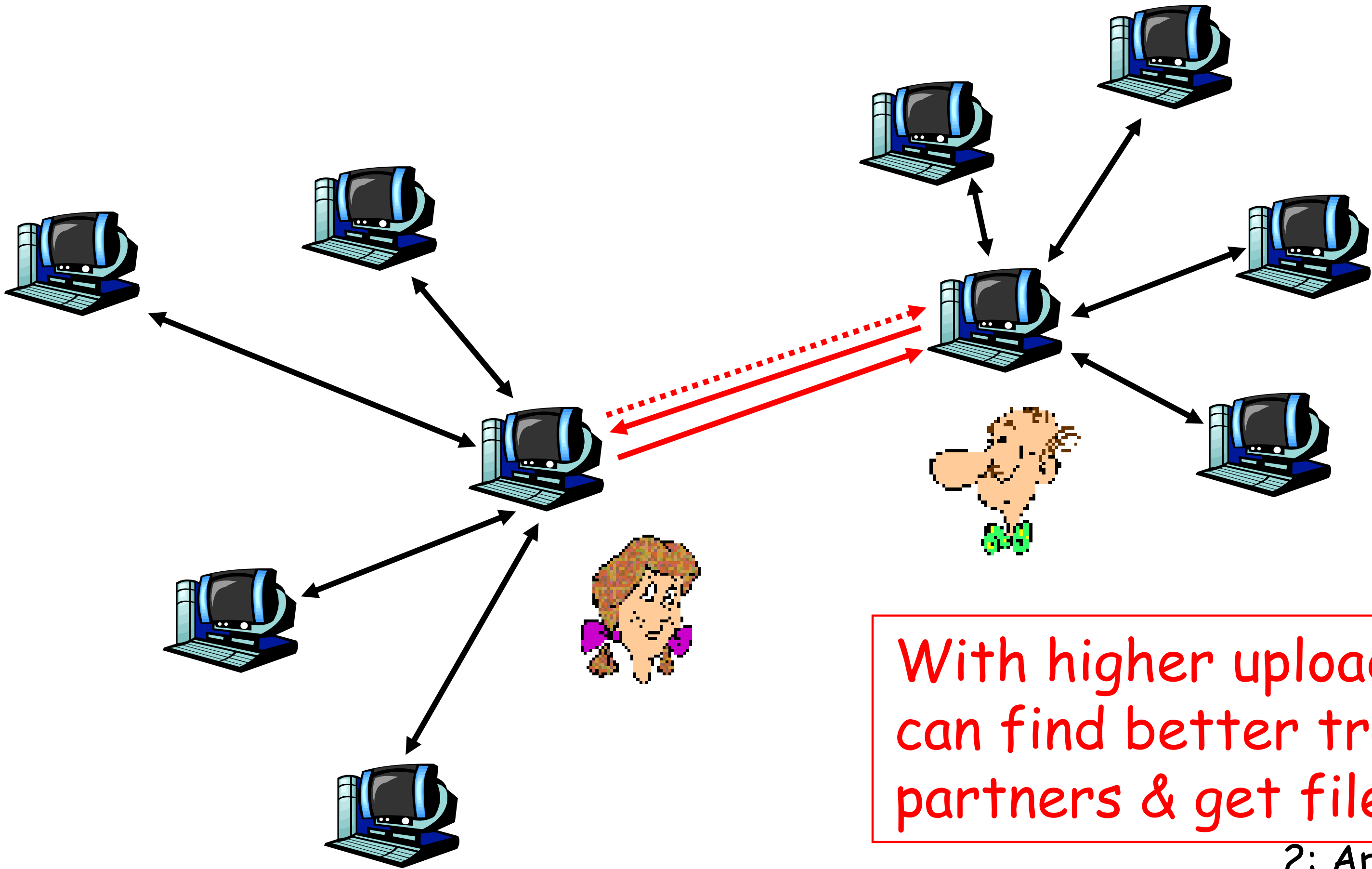
- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

- Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
  - "optimistically unchoke"

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



With higher upload rate, can find better trading partners & get file faster!

Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) peers

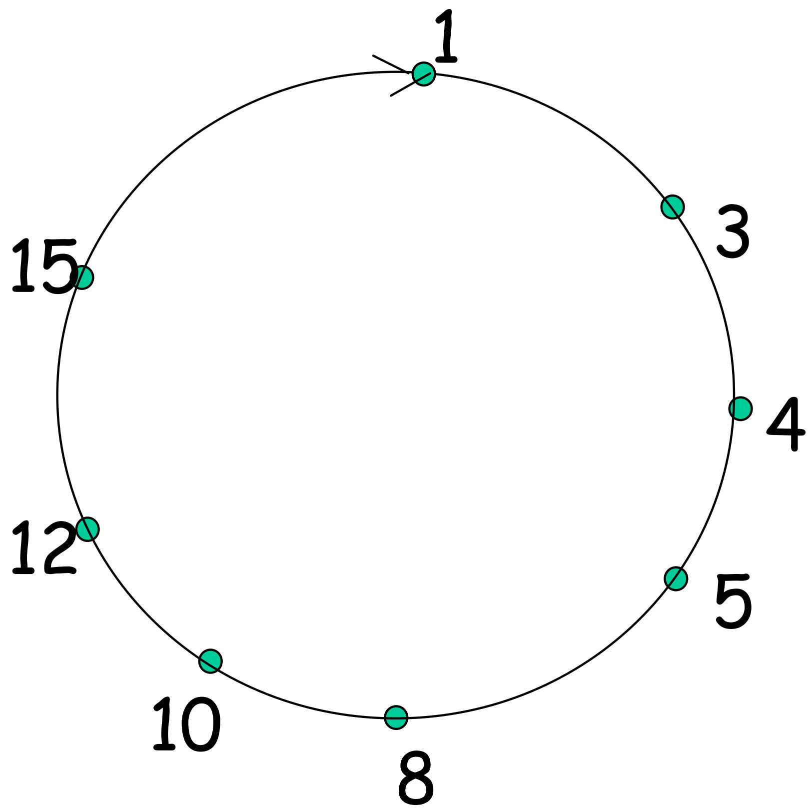
DHT Identifiers

- Assign integer identifier to each peer in range $[0,2^n-1]$.
  - Each identifier can be represented by n bits.
- Require each key to be an integer in same range.
- To get integer keys, hash original key.
  - eg, key = h("Led Zeppelin IV")
  - This is why they call it a distributed "hash" table

How to assign keys to peers?

- Central issue:
  - Assigning (key, value) pairs to peers.
- Rule: assign key to the peer that has the closest ID.
- Convention in lecture: closest is the immediate successor of the key.
- Ex: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

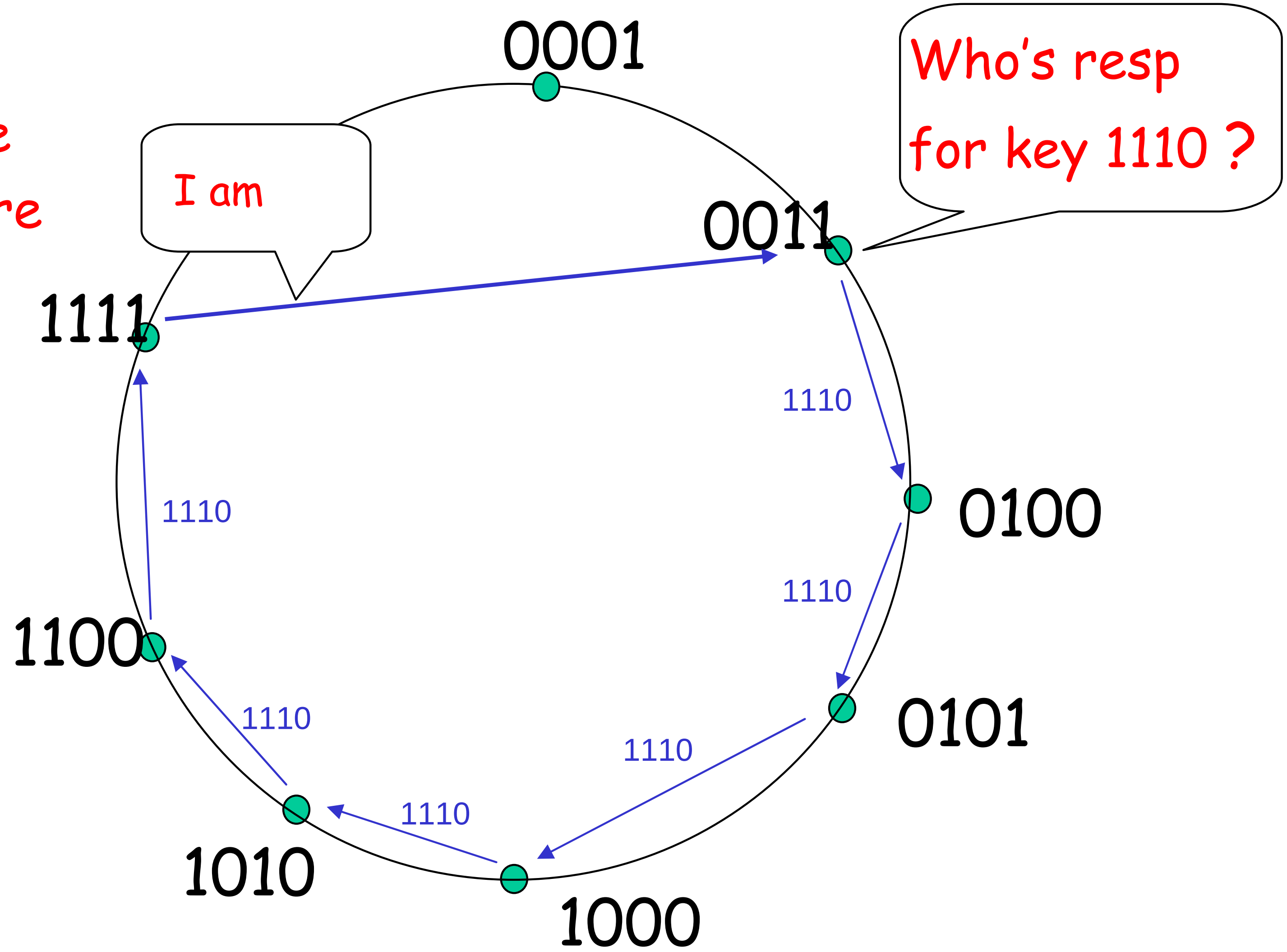
Circular DHT (1)



- Each peer *only* aware of immediate successor and predecessor.
- "Overlay network"

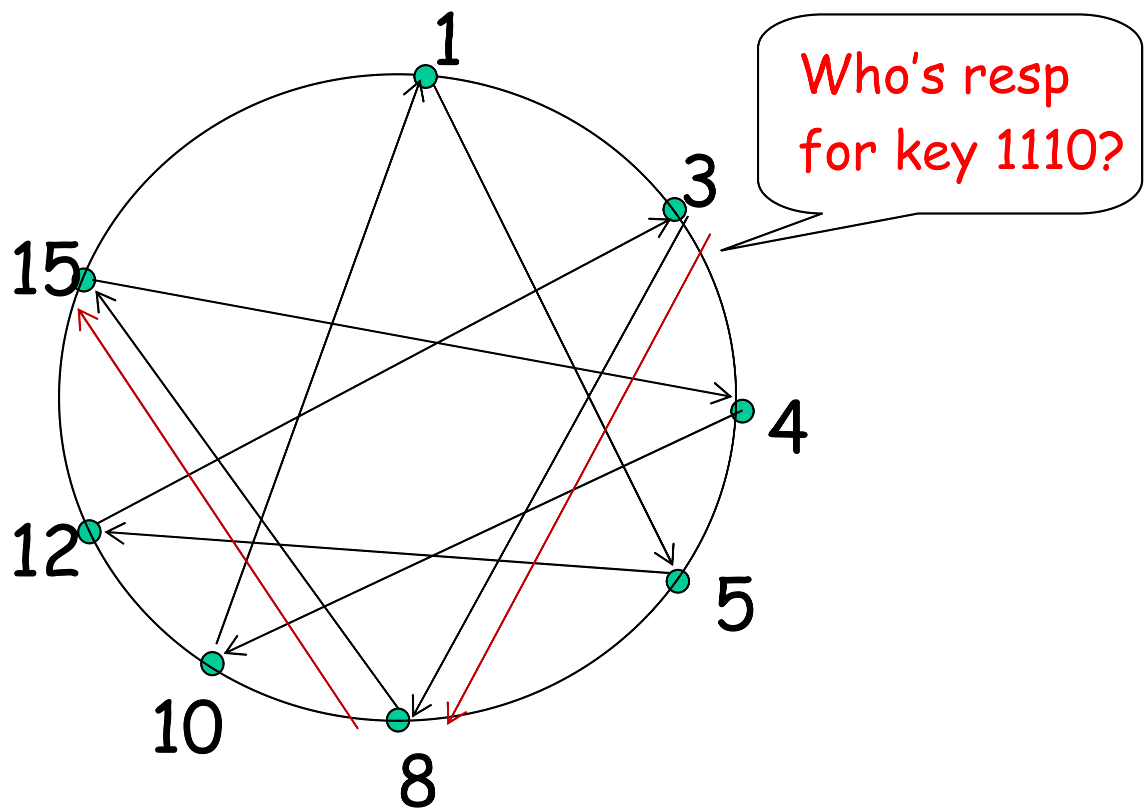
Circle DHT (2)

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



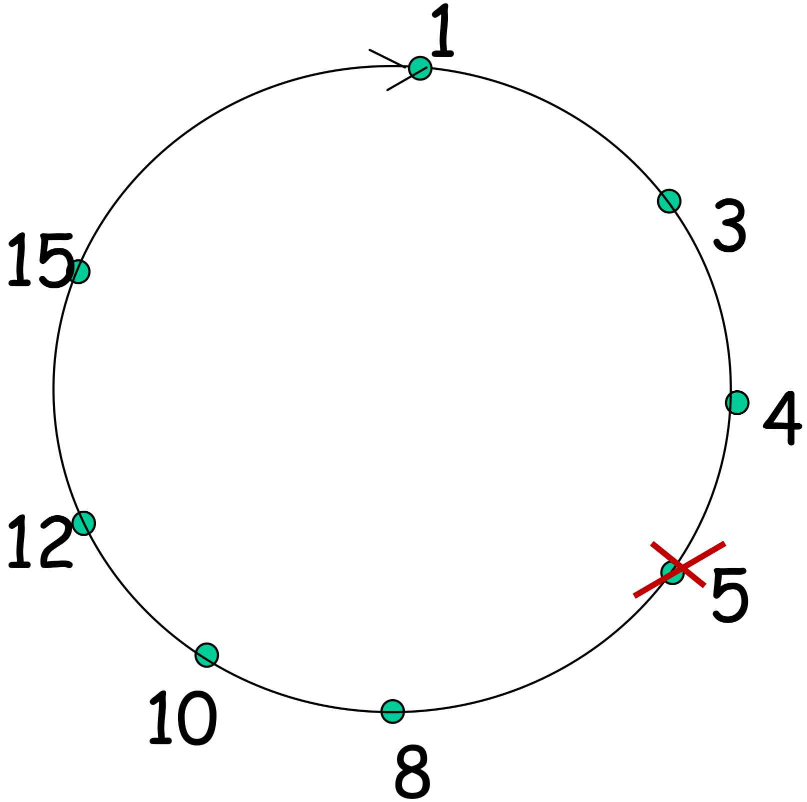
Define closest as closest successor

Circular DHT with Shortcuts



- Each peer keeps track of IP addresses of predecessor, successor, 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

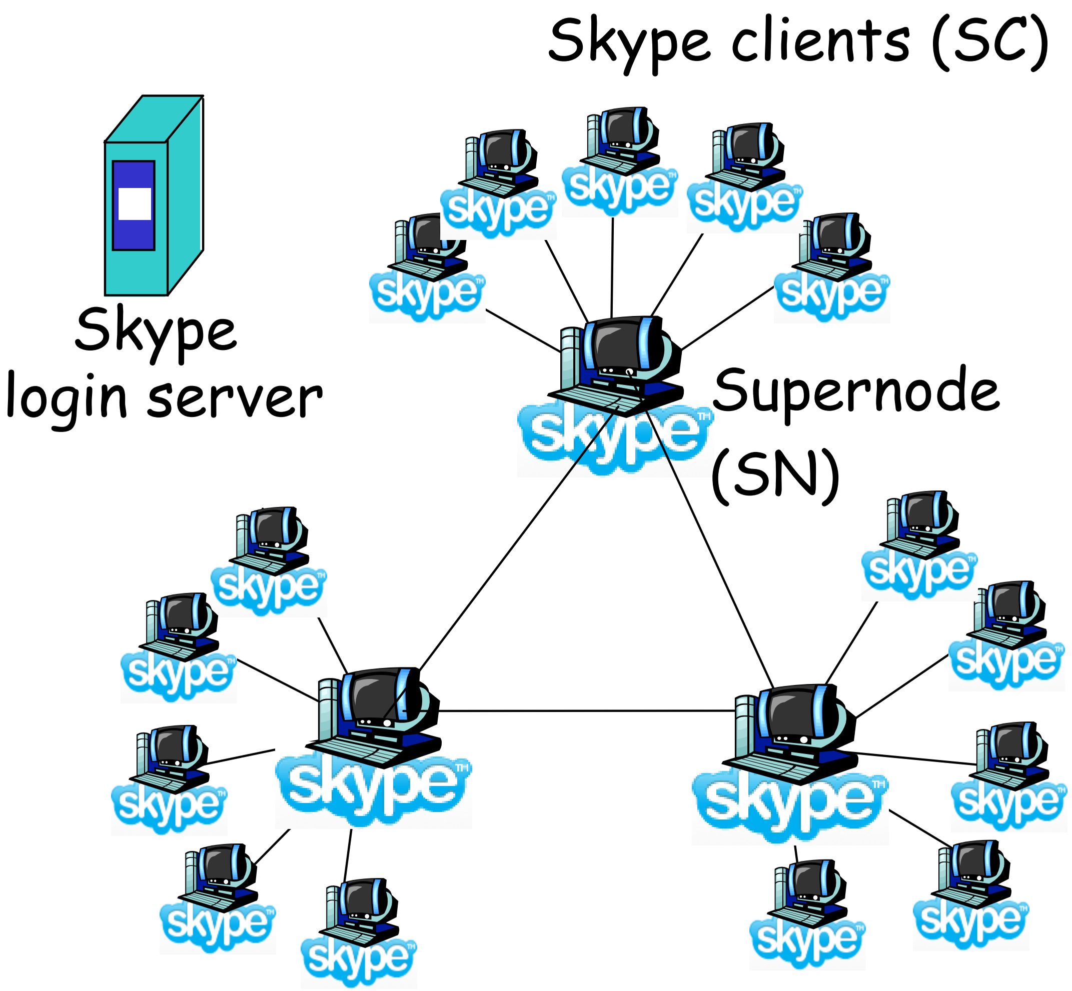


- To handle peer churn, require each peer to know the IP address of its two successors.
- Each peer periodically pings its two successors to see if they are still alive.

- Peer 5 abruptly leaves
- Peer 4 detects; 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?

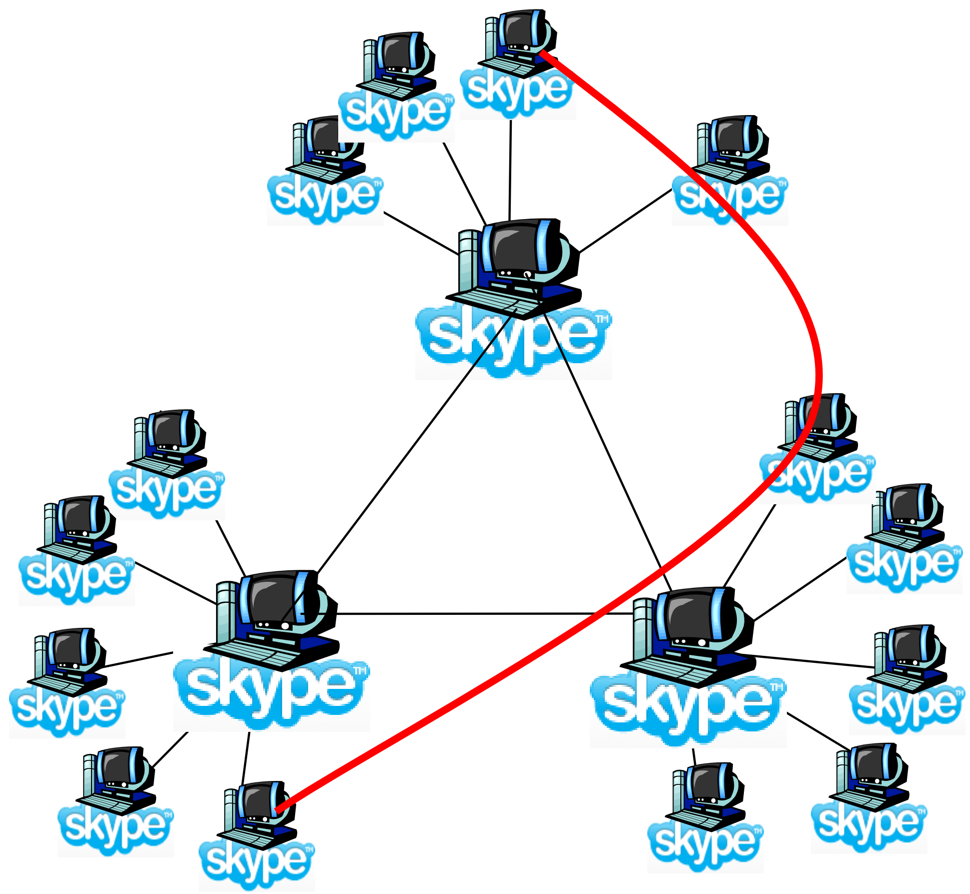
P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs



Peers as relays

- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer
- Solution:
  - Using Alice's and Bob's SNs, Relay is chosen
  - Each peer initiates session with relay.
  - Peers can now communicate through NATs via relay



Chapter 2: Summary

- Application architectures
  - client-server
  - P2P
  - hybrid
- 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