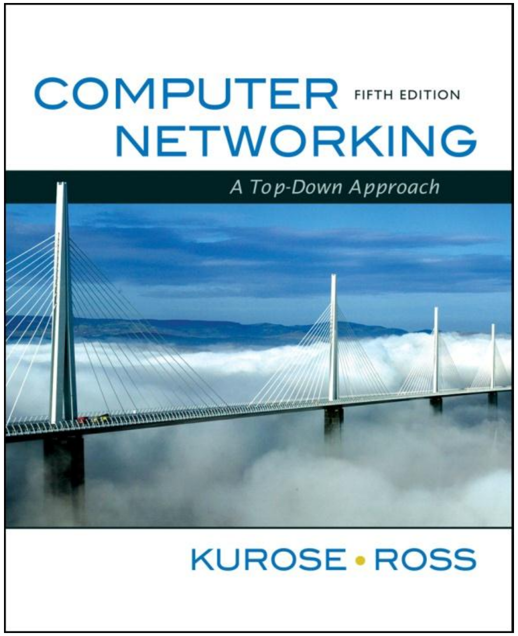
Chapter 2
Application Layer



A note on the use of these ppt slides:

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

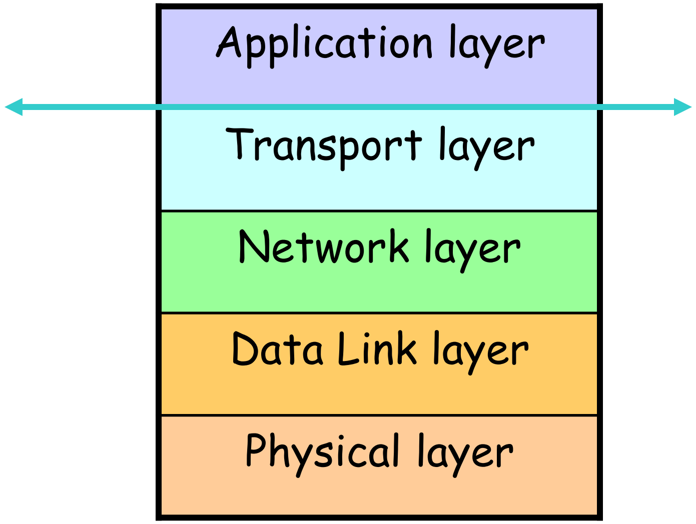
- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides,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

Thanks and enjoy! JFK/KWR

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

Chapter 2: 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
- 2.6 P2P Applications
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP



Chapter 2: Application Layer

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
- programming network applications
  - socket API

Some network apps

- e-mail
- web
- instant messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video clips
- voice over IP
- real-time video conferencing
- grid computing
- cloud computing

Network applications: some terminologies

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 governed by application-layer protocol

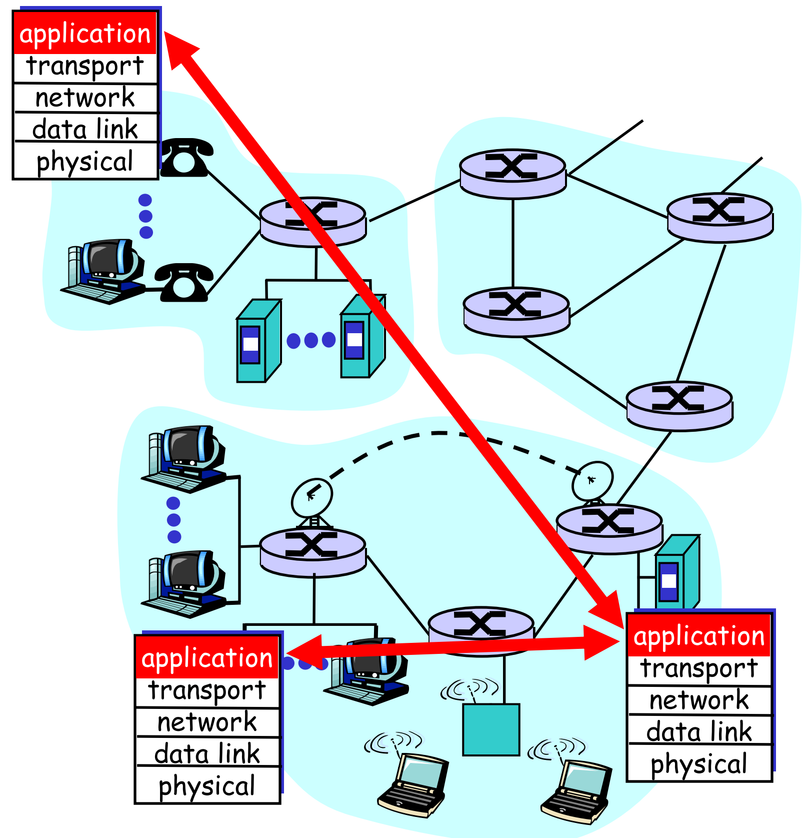
Applications and application-layer protocols

Application: communicating, distributed processes

- e.g., e-mail, Web, P2P file sharing, instant messaging
- running in end systems (hosts)
- exchange messages to implement application

Application-layer protocols

- one "piece" of an app
- define messages exchanged by apps and actions taken
- use communication services provided by lower layer protocols (TCP, UDP)



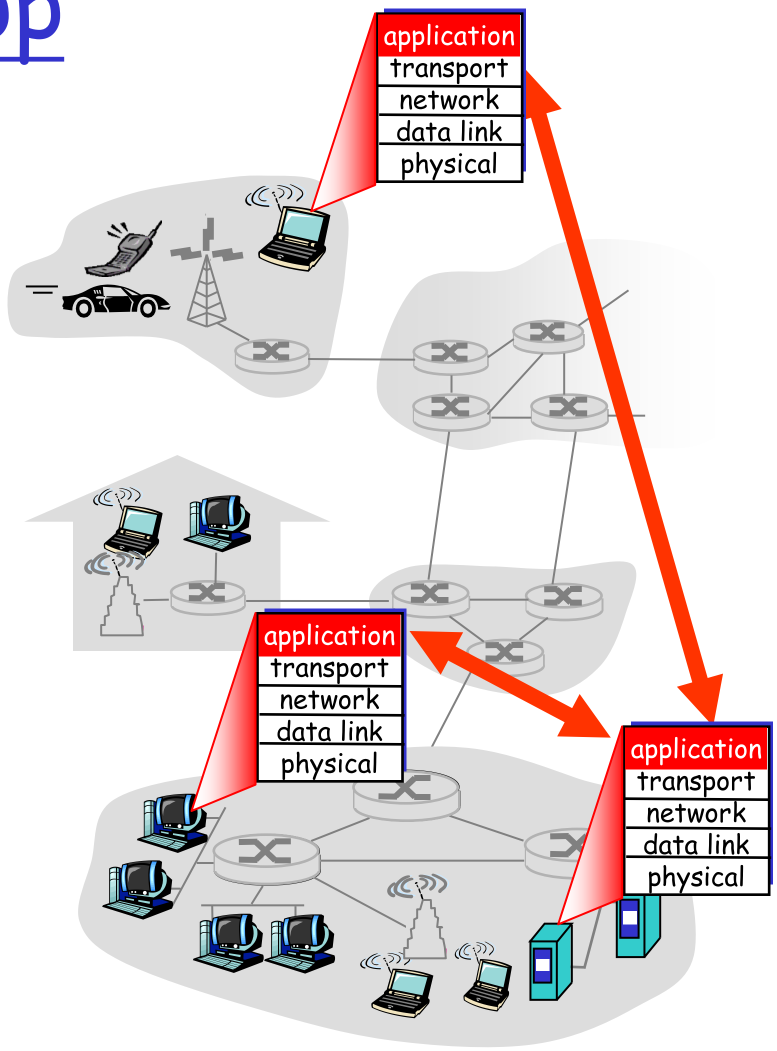
Creating a network app

write programs that

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

little software written for devices in network core

- network core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



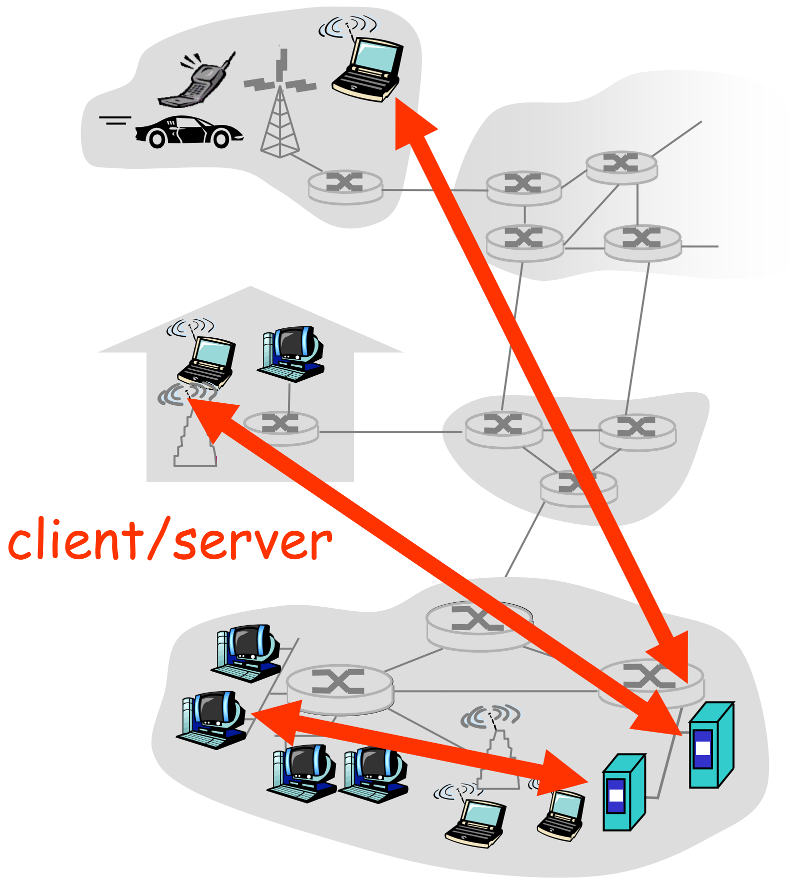
Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 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:

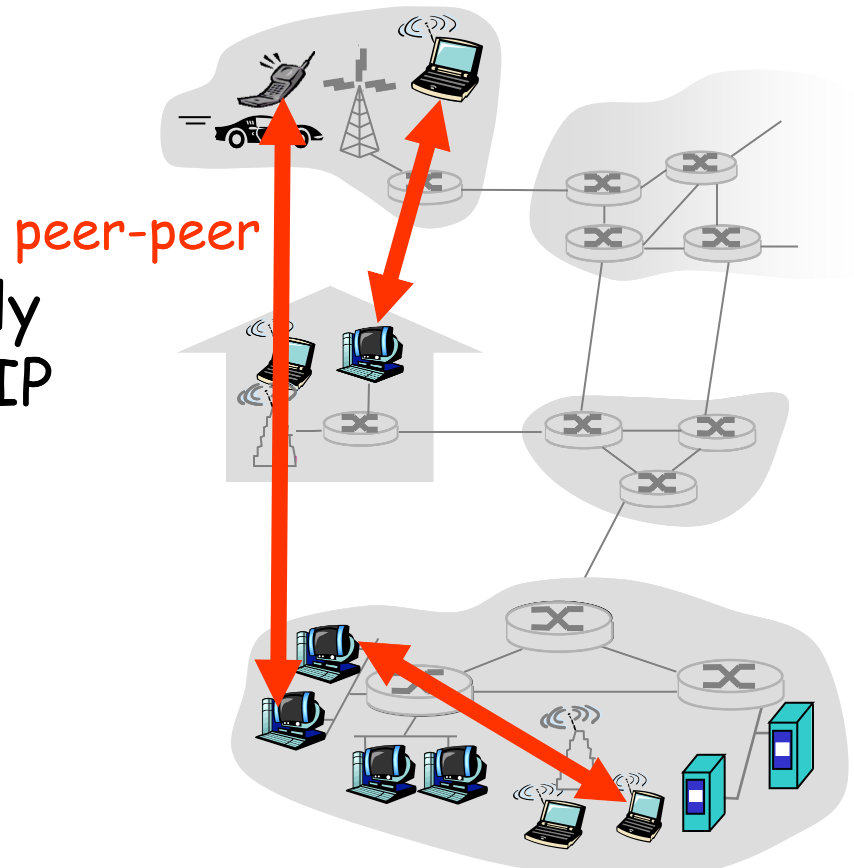
- always-on host (typically)
- permanent IP address
- server farms for scaling

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

Skype
- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)

Instant messaging
- chatting between two users is P2P
- centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

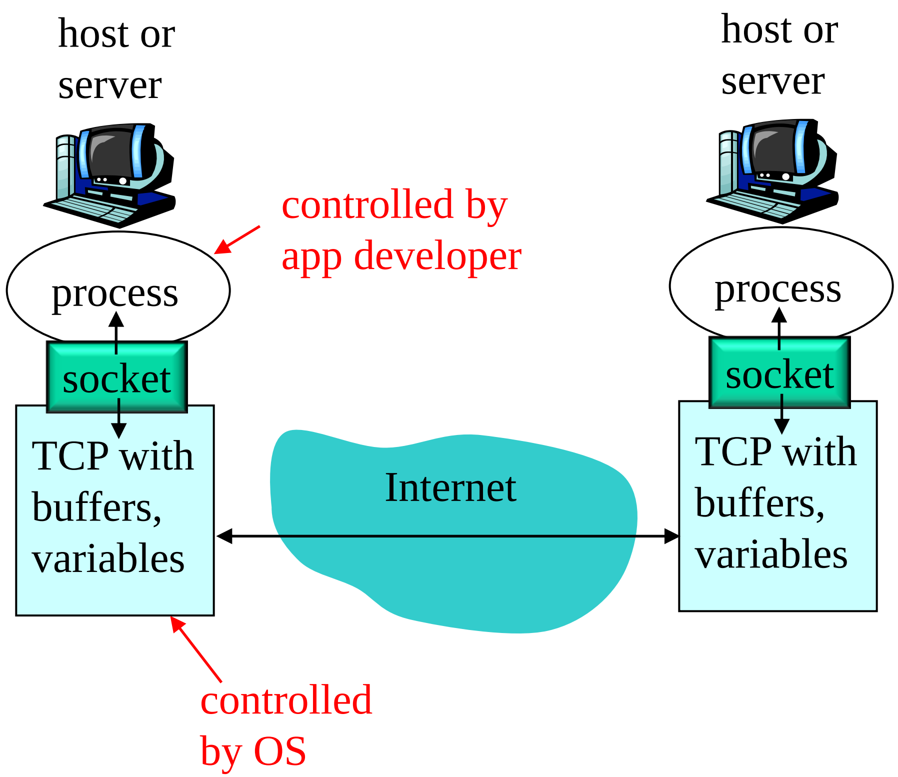
Processes communicating

- Client process: process that initiates communication
- Server process: process that waits to be contacted

- 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)

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 suffice 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

Public-domain protocols:

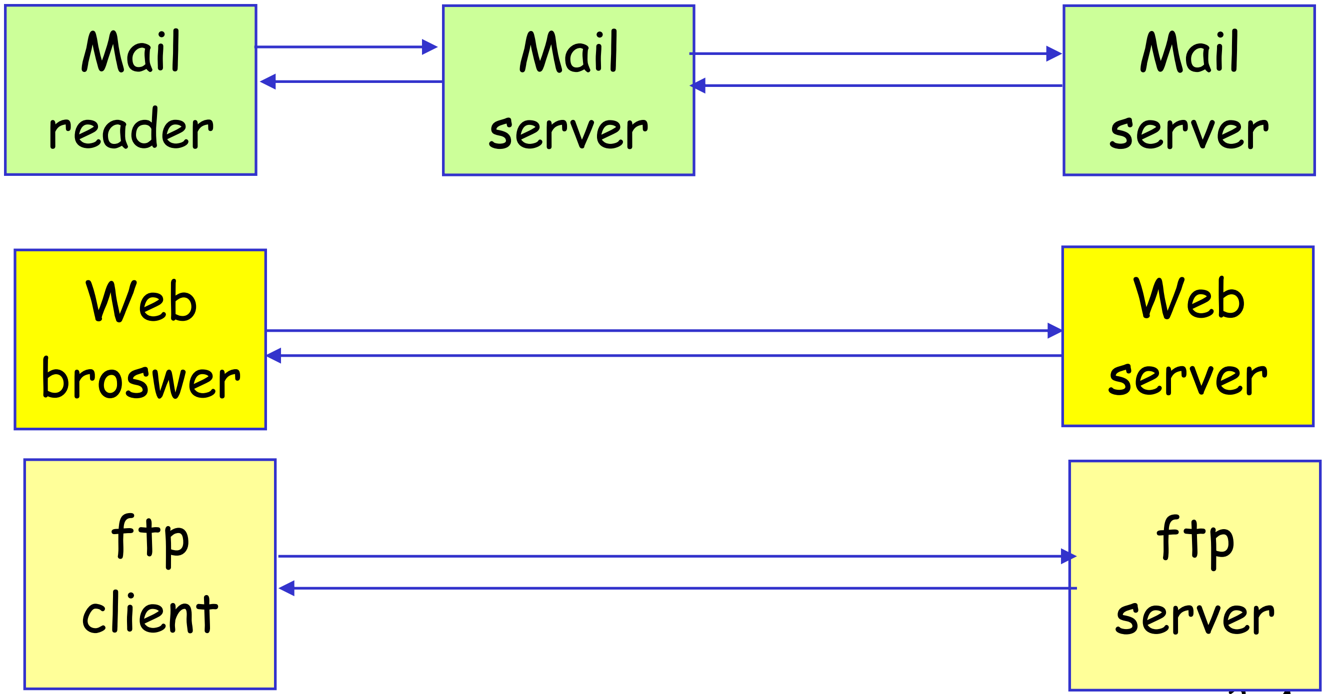
- defined in **RFCs**
- allows for interoperability
- eg, HTTP, SMTP

Proprietary protocols:

- e.g., skype

App-layer protocol defines

- Types of messages exchanged, eg, request & response messages
- Syntax of message types: what fields in messages & how fields are delineated
- Semantics of the fields, ie, meaning of information in fields
- Rules for when and how processes send & respond to messages



What transport service does an app need?

Data loss

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

Timing (e2e delay)

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

Bandwidth

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

Transport service requirements of common apps

Application	Data loss	Bandwidth	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
instant messaging	no loss	elastic	yes and no

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 bandwidth guarantees

UDP service:

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

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	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Vonage,Dialpad)	typically UDP

Chapter 2: Application layer

- 2.1 Principles of network applications
  - app architectures
  - app requirements
- 2.2 Web and HTTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 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:

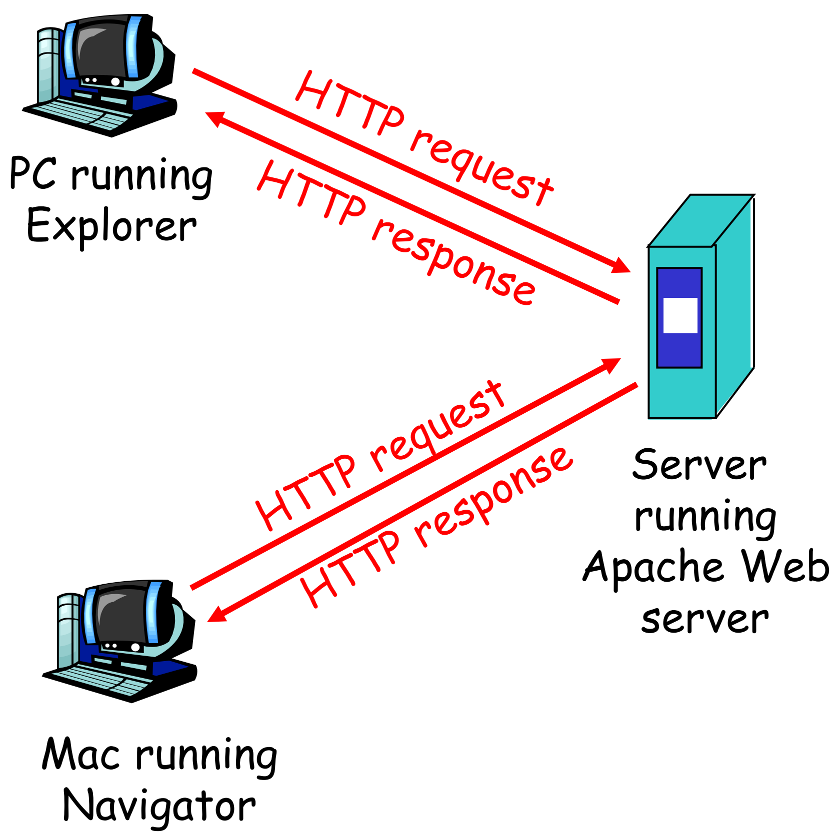
```
www.someschool.edu/someDept/pic.gif
```

host name — path name

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 2068



HTTP

Uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP 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.

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.

Nonpersistent HTTP

Suppose user enters URL
www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)

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

time

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

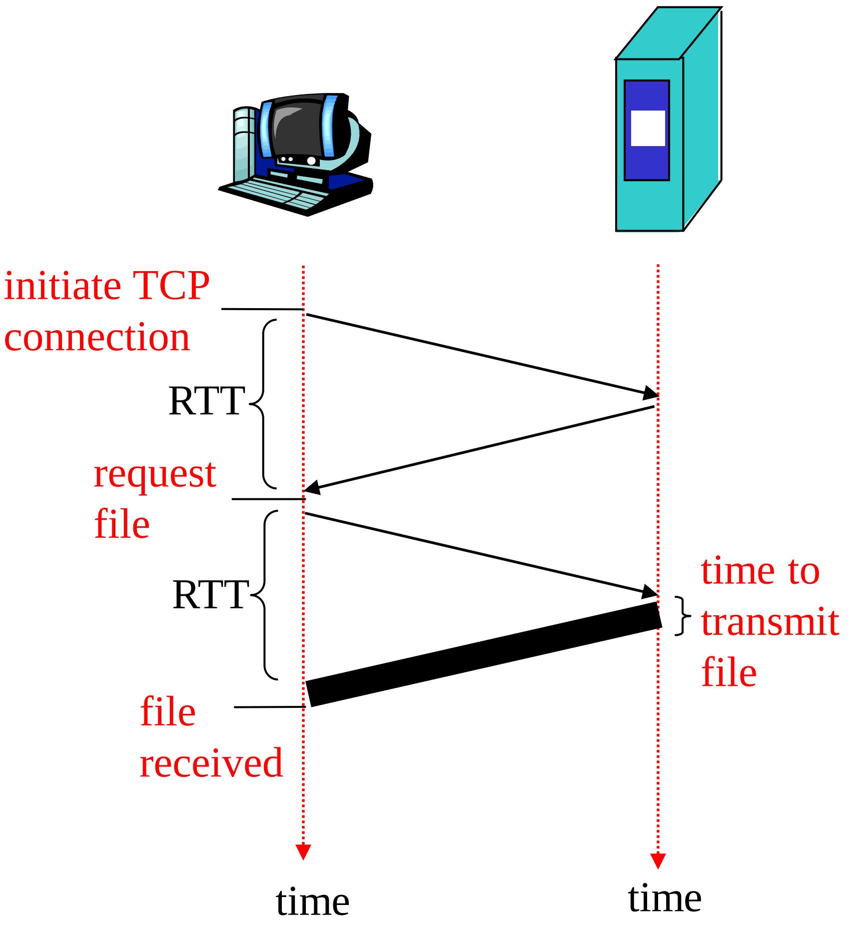
Response time modeling

Definition of RTT: 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 overhead for *each* TCP connection
- 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 sent over open connection
- 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)

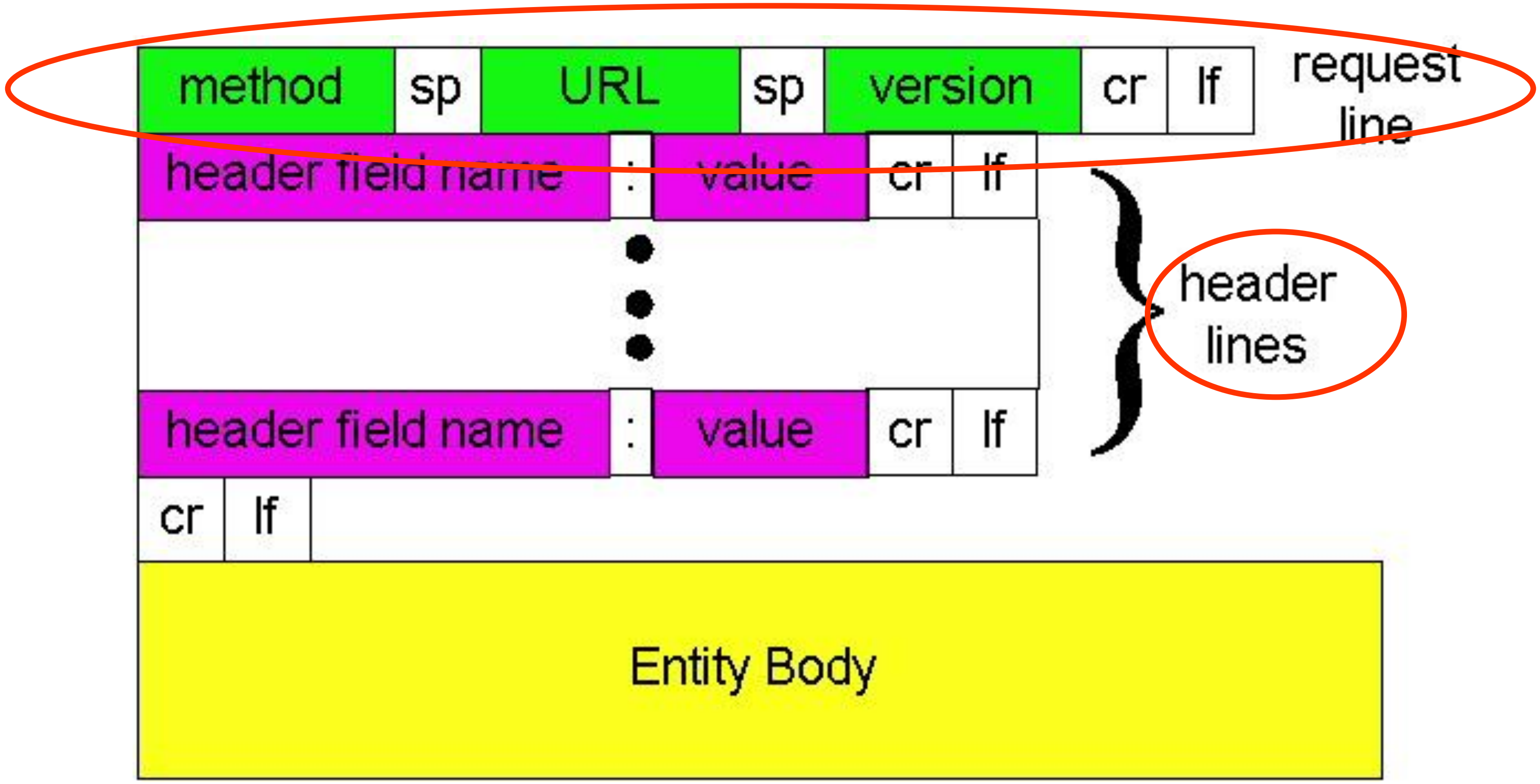
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header lines

(extra carriage return, line feed)

Carriage return, line feed indicates the 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
  - to retrieve any type of information identified by the Request-URI.
- 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!

Cookies: keeping "state"

Many major Web sites use cookies

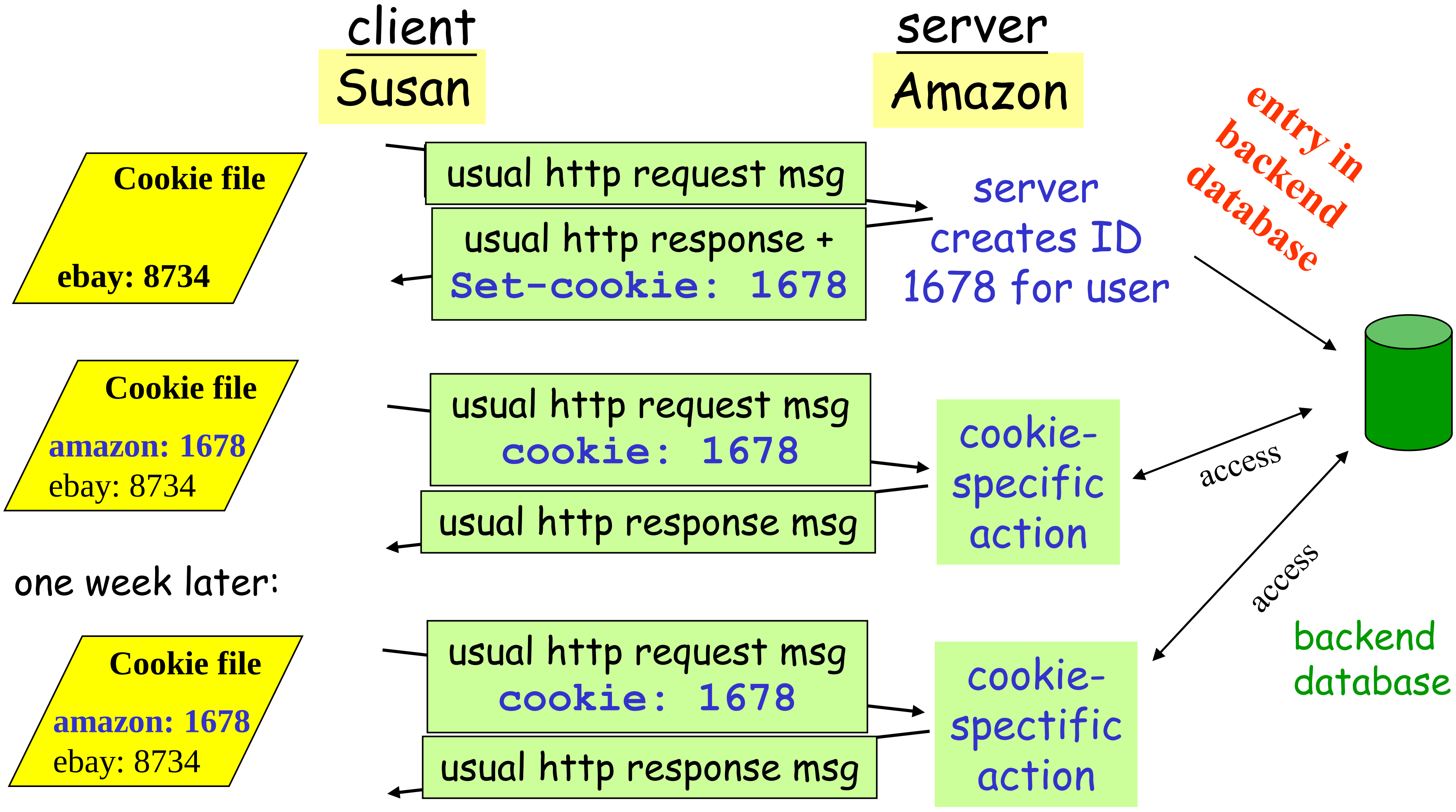
Four components:

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
  - an entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

What cookies can bring:

- authorization
- shopping carts
- Recommendations (personalization)
- 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

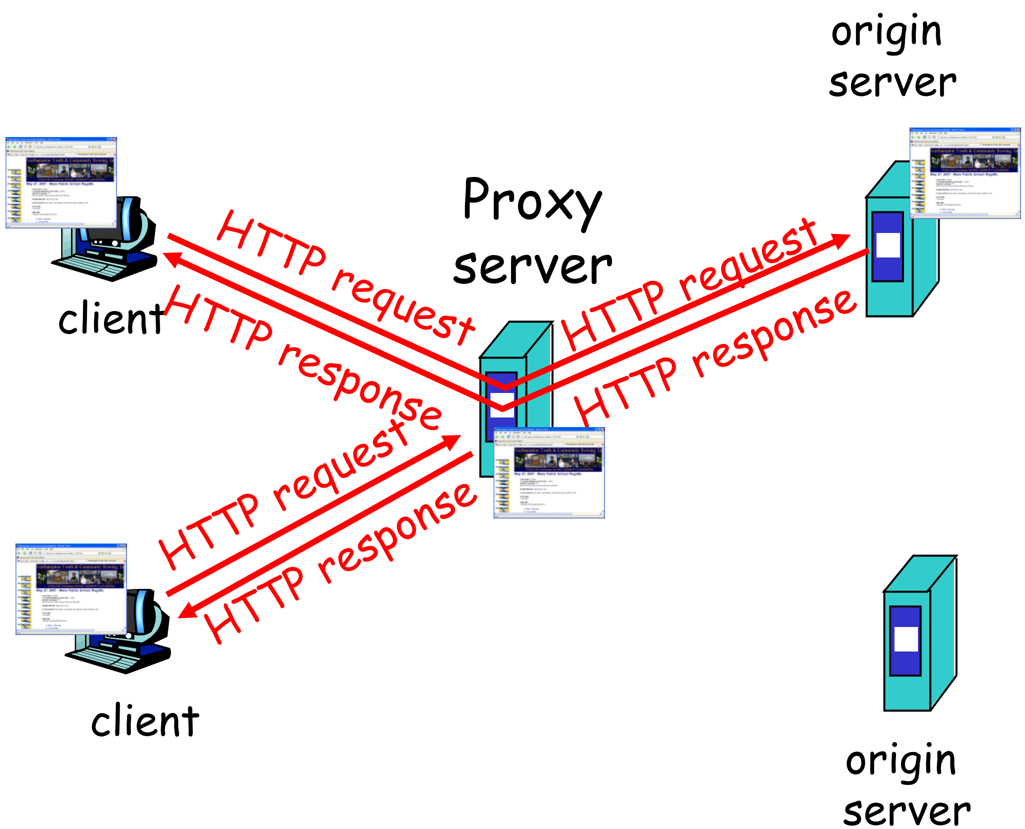
How to keep "state":

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

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



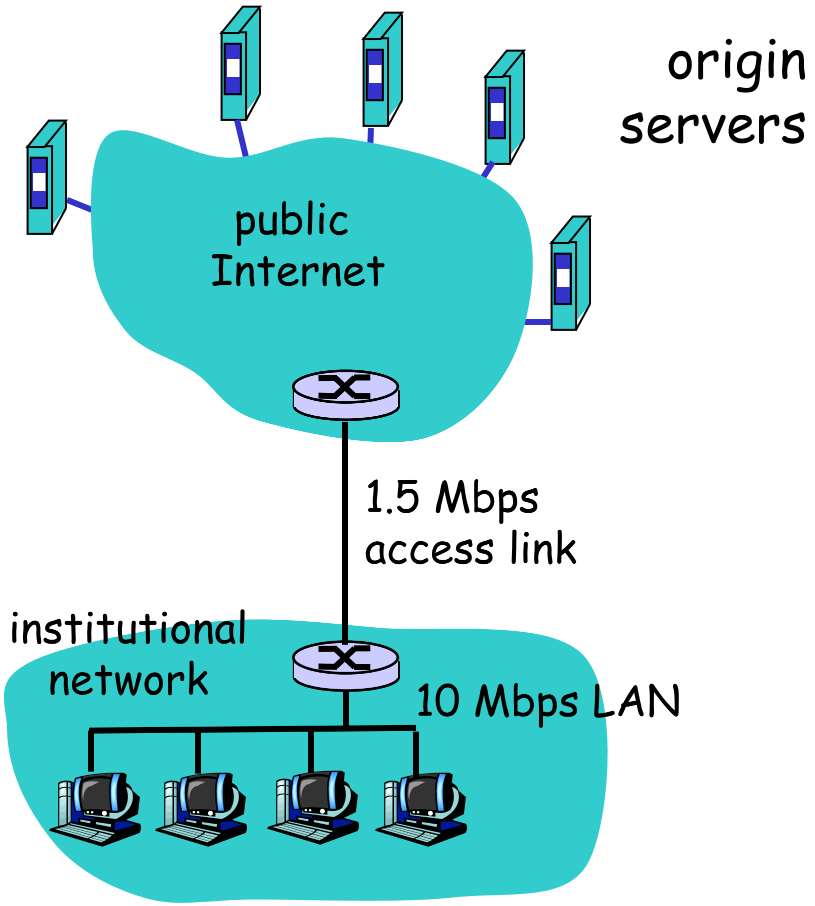
More about Web caching

- 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 (tightly packed) with caches: enables "poor" content providers to effectively deliver content (but so does P2P file sharing)

Caching example (1)



Assumptions

- average object size = 100,000 bits
- avg. request rate from institution's browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

Consequences

- utilization on LAN = 15%
- utilization on access link = 100% !?
- total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds

Caching example (2)

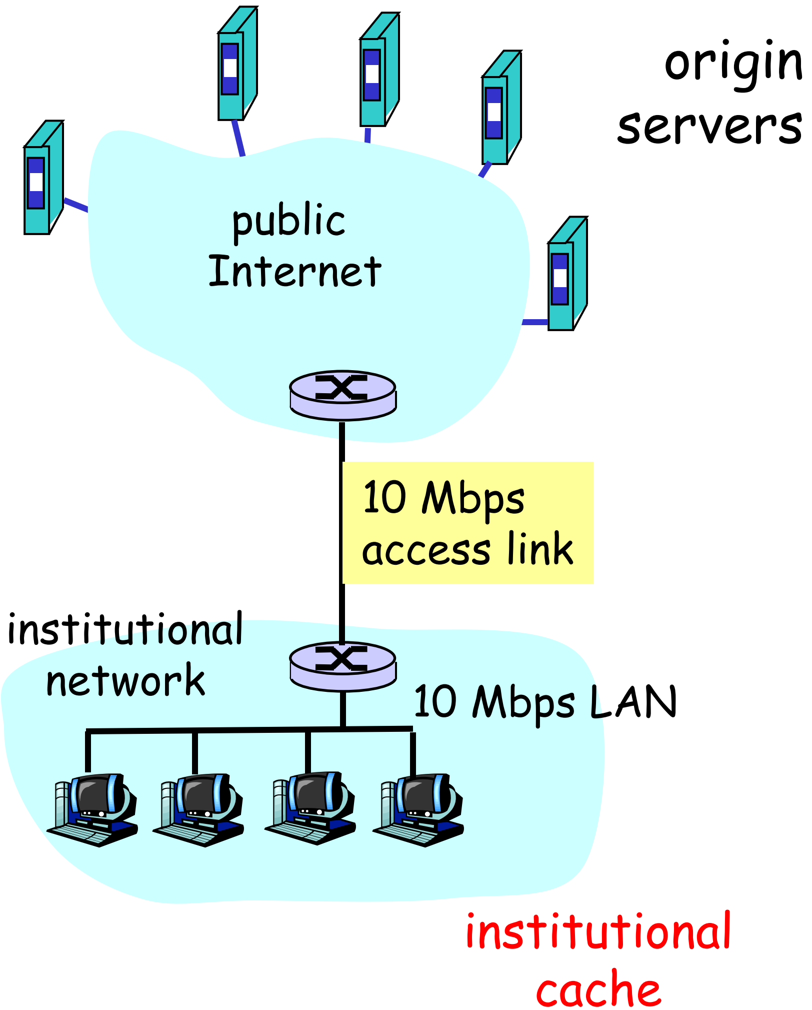
Possible solution

- increase bandwidth of access link to, say, 10 Mbps

Consequences

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

  = 2 sec + msecs + msecs
- often a costly upgrade



Caching example (3)

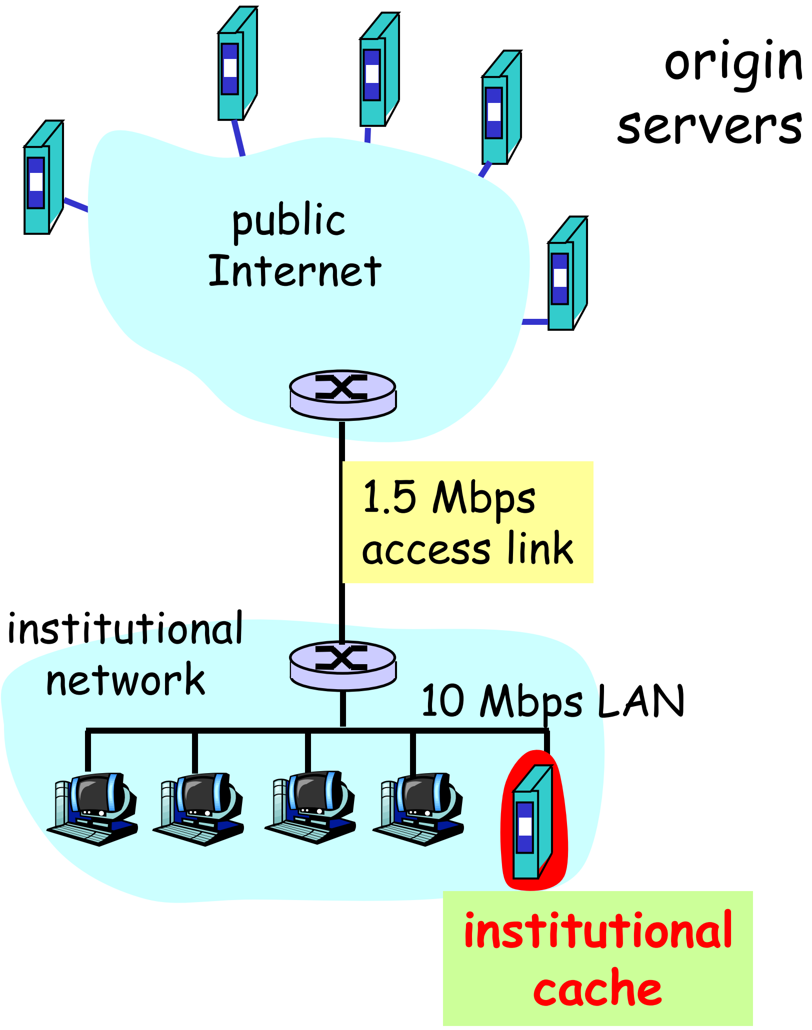
Install cache

- suppose hit rate is .4

Consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total delay = Internet delay + access delay + LAN delay

= .6*2 sec + .6*.01 secs + milliseconds < 1.3 secs

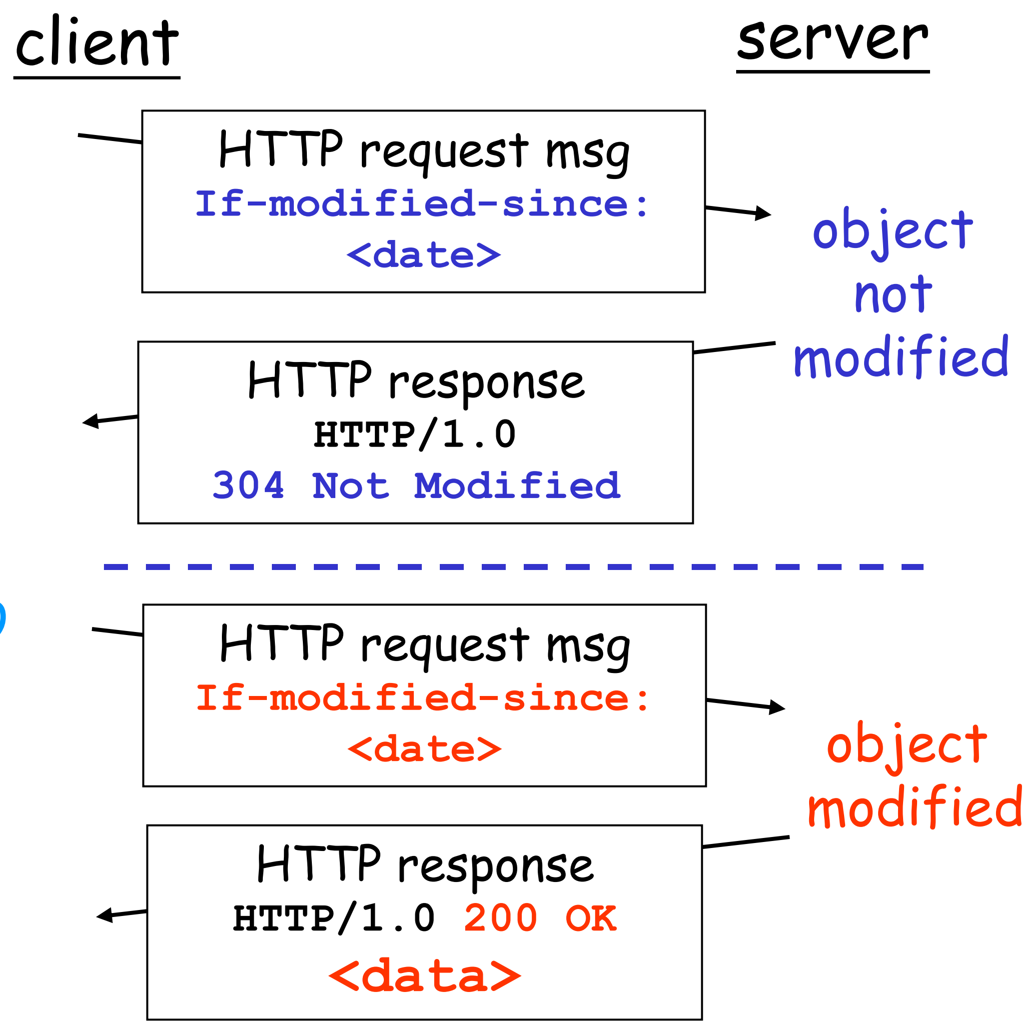


Conditional GET: client-side caching

- Goal: DON'T send object if client has up-to-date cached version
- client: specify DATE of cached copy in HTTP request

  If-modified-since: <date>
- server: response contains NO object if cached copy is up-to-date:

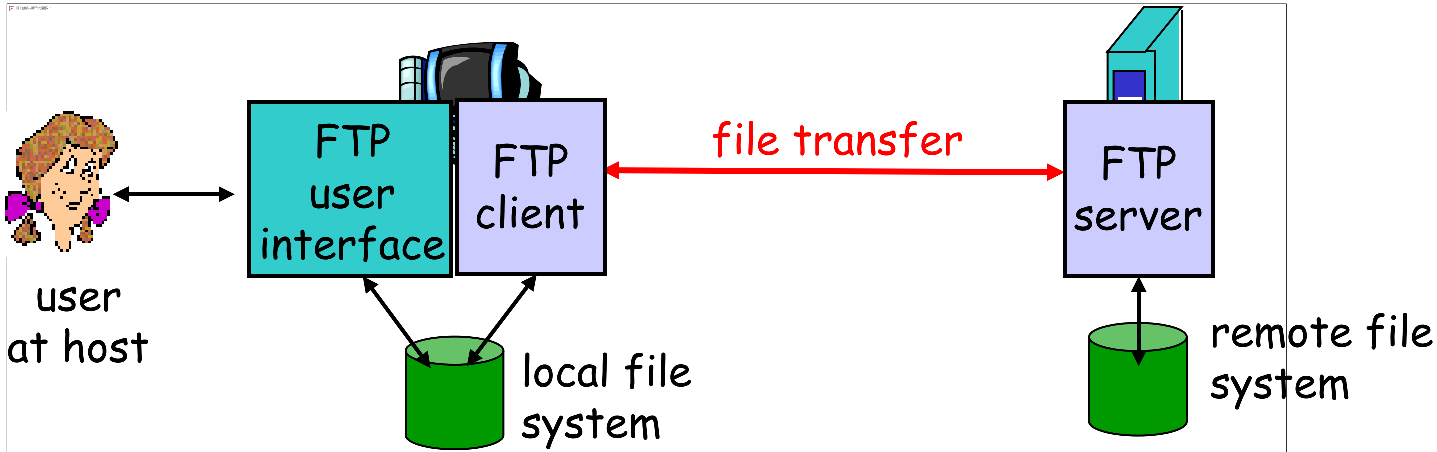
  HTTP/1.0 304 Not Modified



Chapter 2: 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
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP

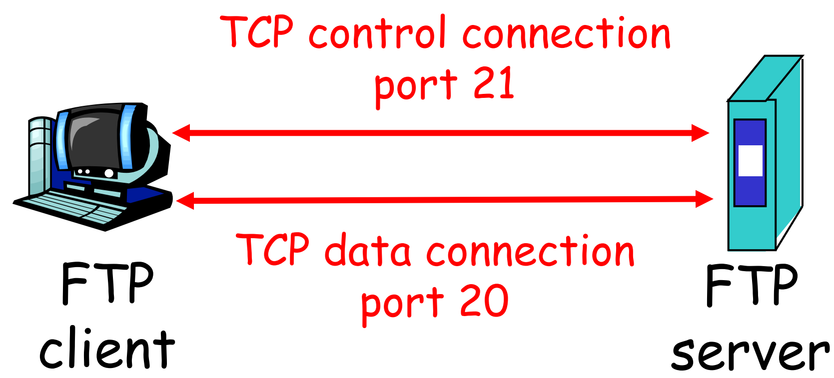
FTP: the file transfer protocol



- 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 (e.g., list, dir) by sending commands over control connection.
- When server receives a command for a file transfer (e.g., get, put), the server opens *2nd* TCP data connection to client
- After transferring one file, server closes data 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

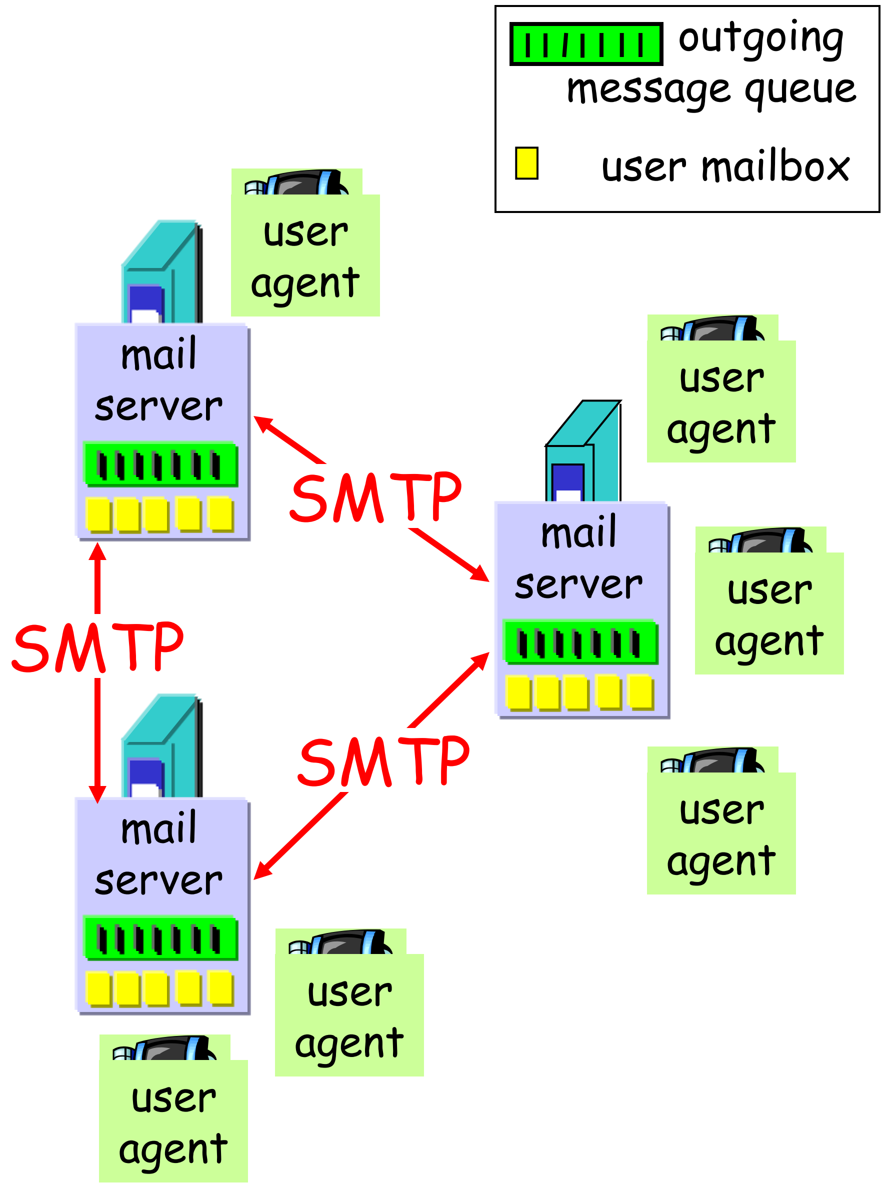
Sample return codes

- 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: 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
- 2.6 P2P file sharing
- 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

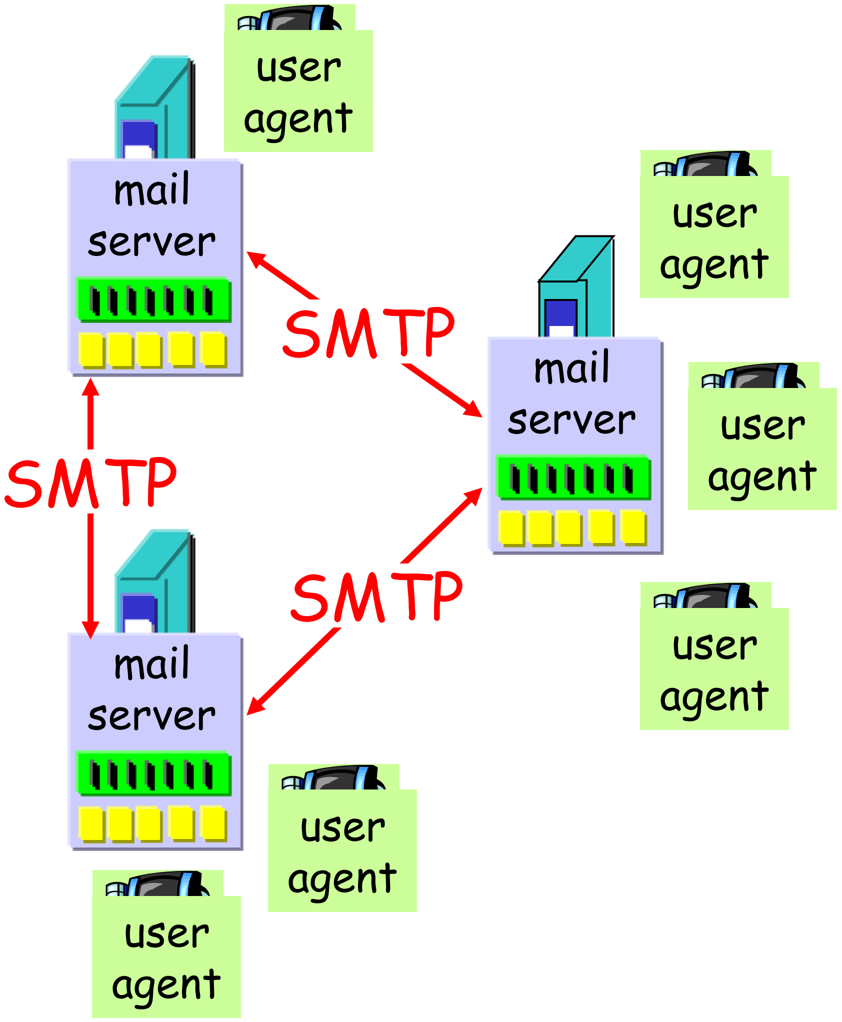
User Agent

- 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

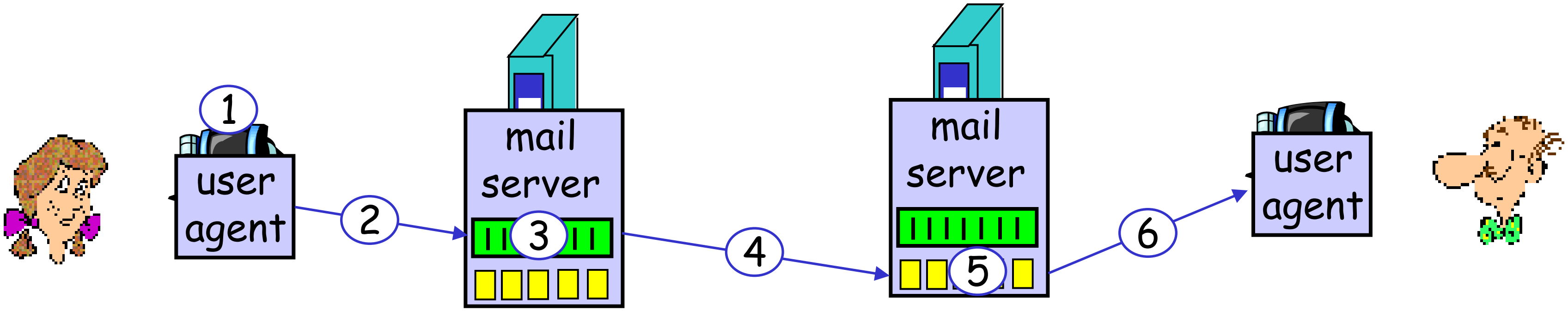
Mail Servers

- 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



Scenario: Alice sends message to Bob

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

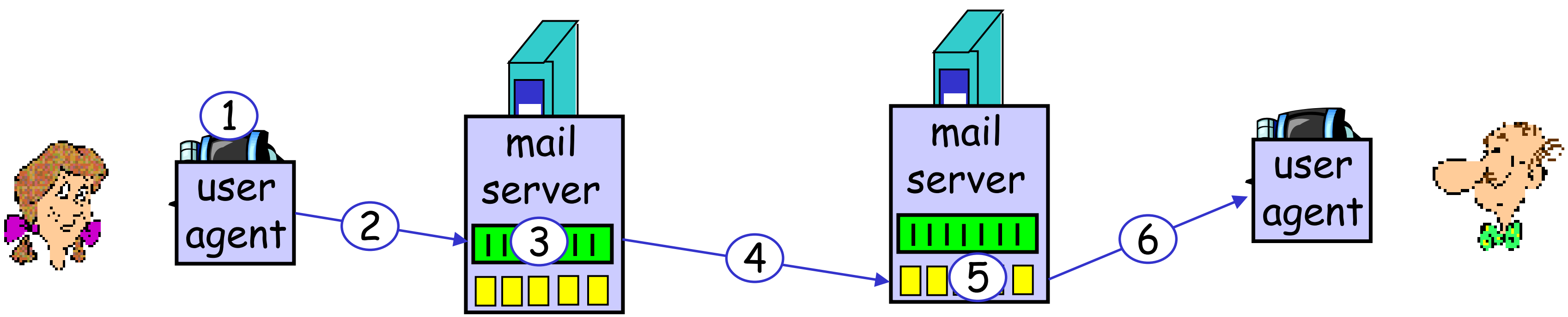
2) Alice's UA sends message to her mail server; message placed in message queue

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

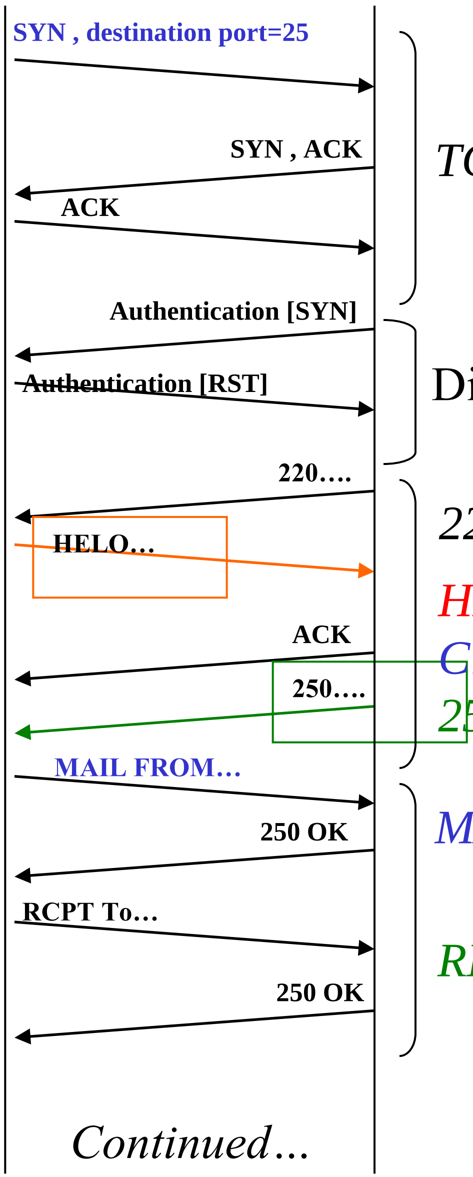
4) SMTP client sends Alice's message over the TCP connection

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

6) Bob invokes his user agent to read message



Client MTA Server MTA (Mail Transfer Agent)



TCP three-way handshaking (IP of Client MTA)

Different ports (don’t have the function of authentication now)

220: service ready

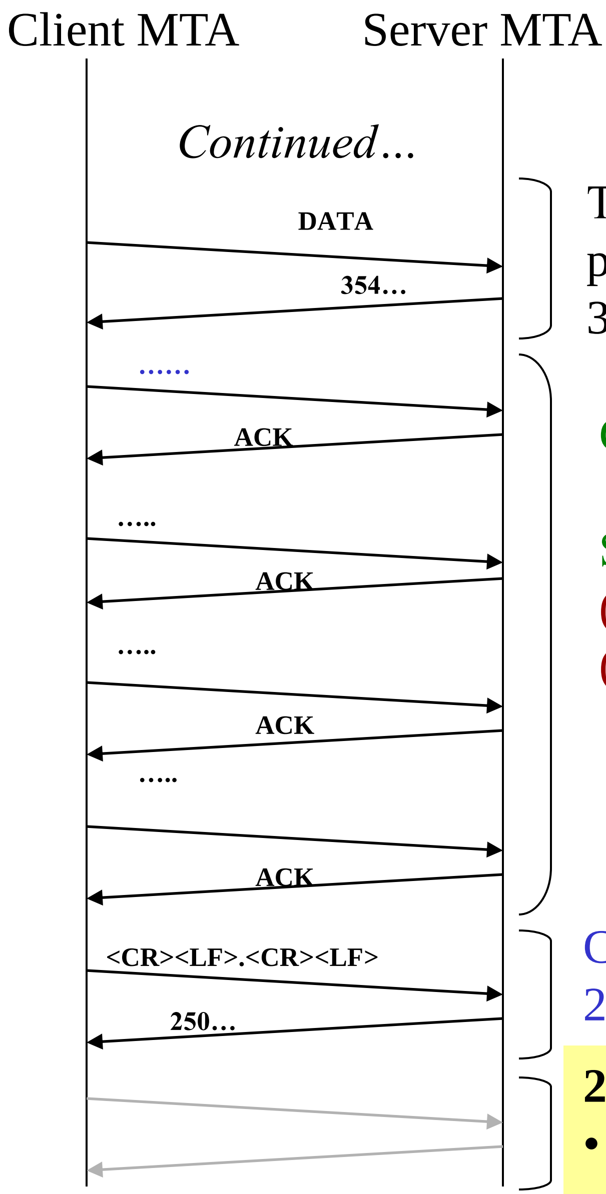
HELO <SP><domain><CRLF>

Client MTA use it to identify itself

250 <Server MTA domain>

MAIL FROM: reversing path (domain of relaying MTA, sender’s mail account)

RECP TO: forwarding path (receiver’s mail account)



The receiver treats the lines following the "DATA" packet as **mail data** from the sender.
354: Start mail input; end with .

Client MTA sends **the content of the mail object.**

Server MTA replies with "ACK" packet
(IP of relaying MTAs)
(IP of original host)

Client MTA sends the end-of-mail command (.)
250: Requested mail action okay, completed

2 cases:
- Client MTA has other mails to send, go back to "MAIL FROM"
- Client MTA has no mail to send anymore, sends "QUIT" packet
- Server MTA replies with 221 and closes the connection

Example:

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

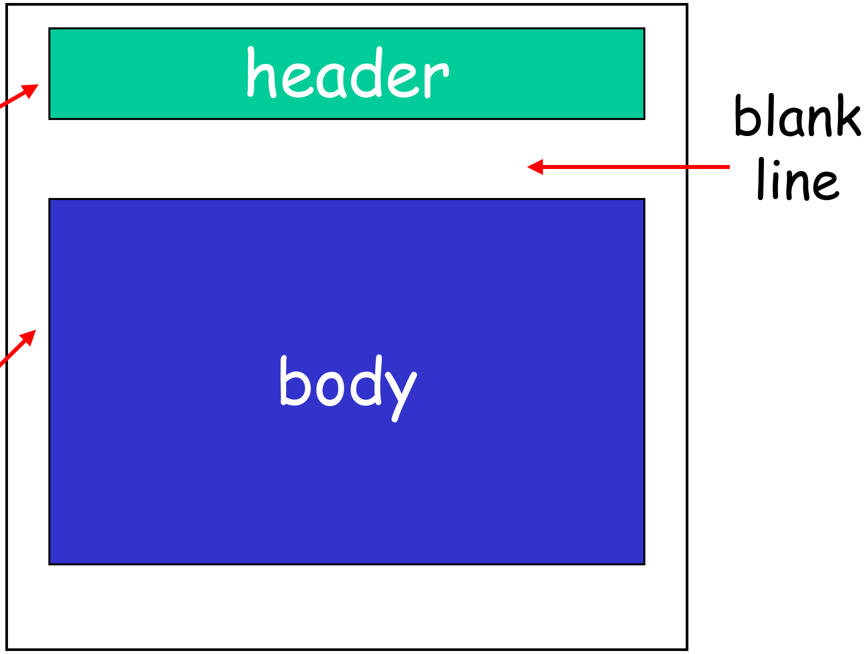
Mail message format

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

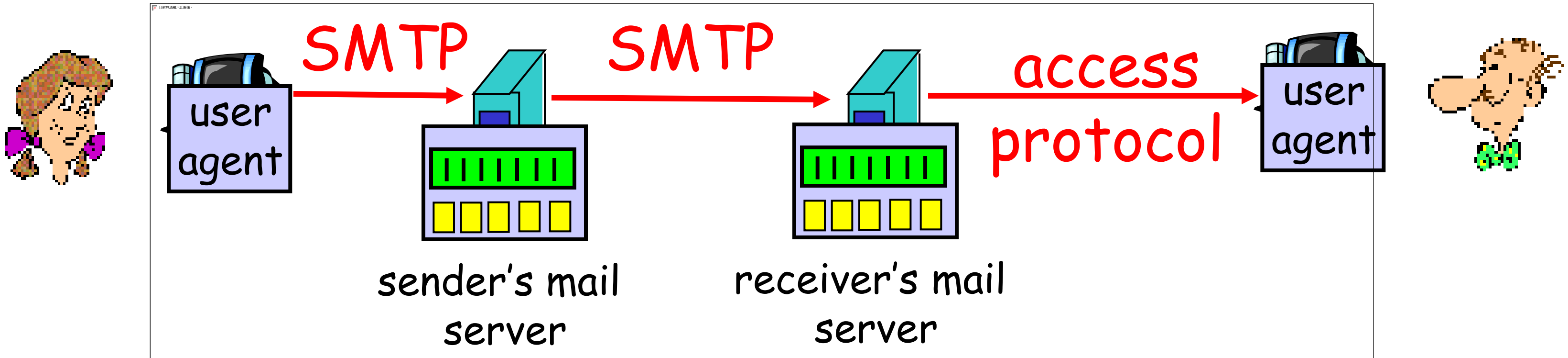
multimedia data type, subtype, parameter declaration

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

IMAP

- Keep all messages in one place: the server
- Allows user to organize messages in folders
- IMAP keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

Chapter 2: 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
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP

DNS: Domain Name System

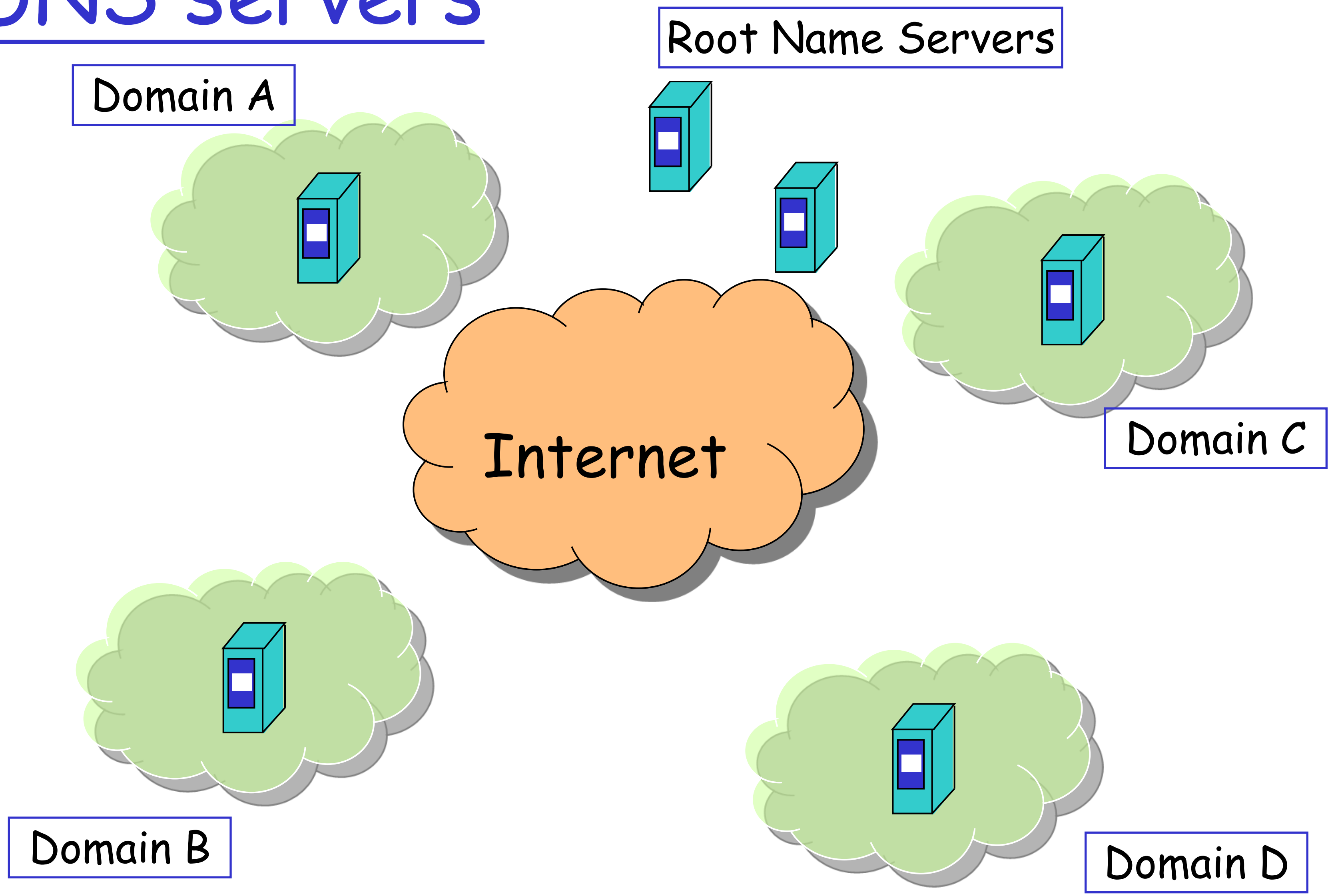
People: many identifiers:
- SSN, name, passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., gaia.cs.umass.edu - 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* - to *resolve* names (address/name translation)

DNS servers


Root Name Servers
Domain A
Internet
Domain C
Domain B
Domain D

DNS: Domain Name System

DNS services

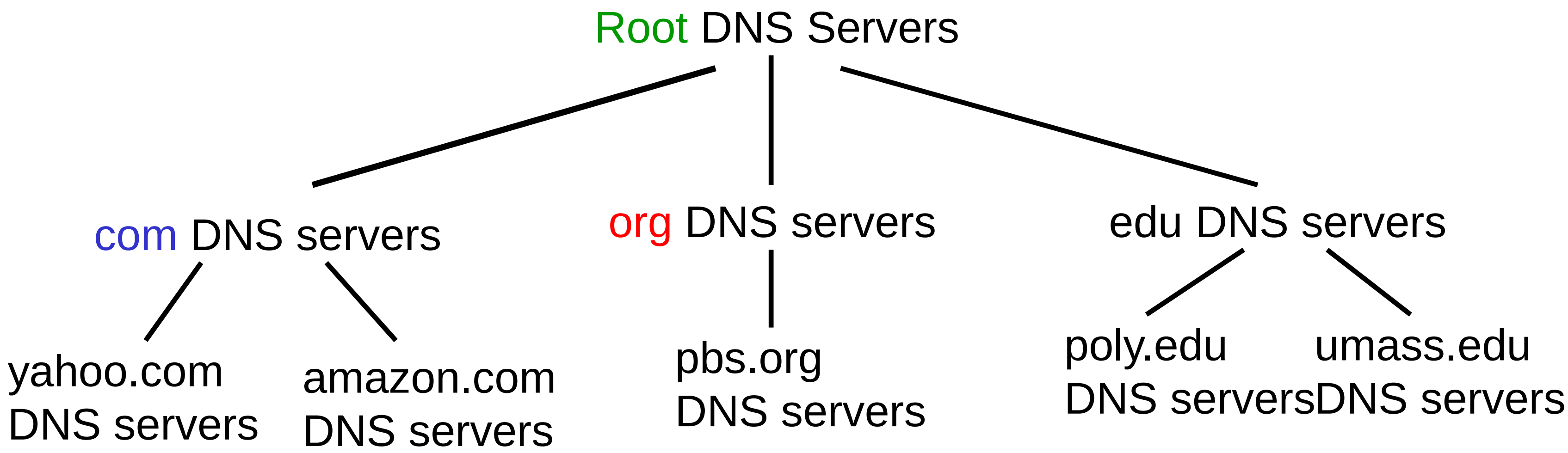
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

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

doesn't *scale!*

Distributed, Hierarchical Database

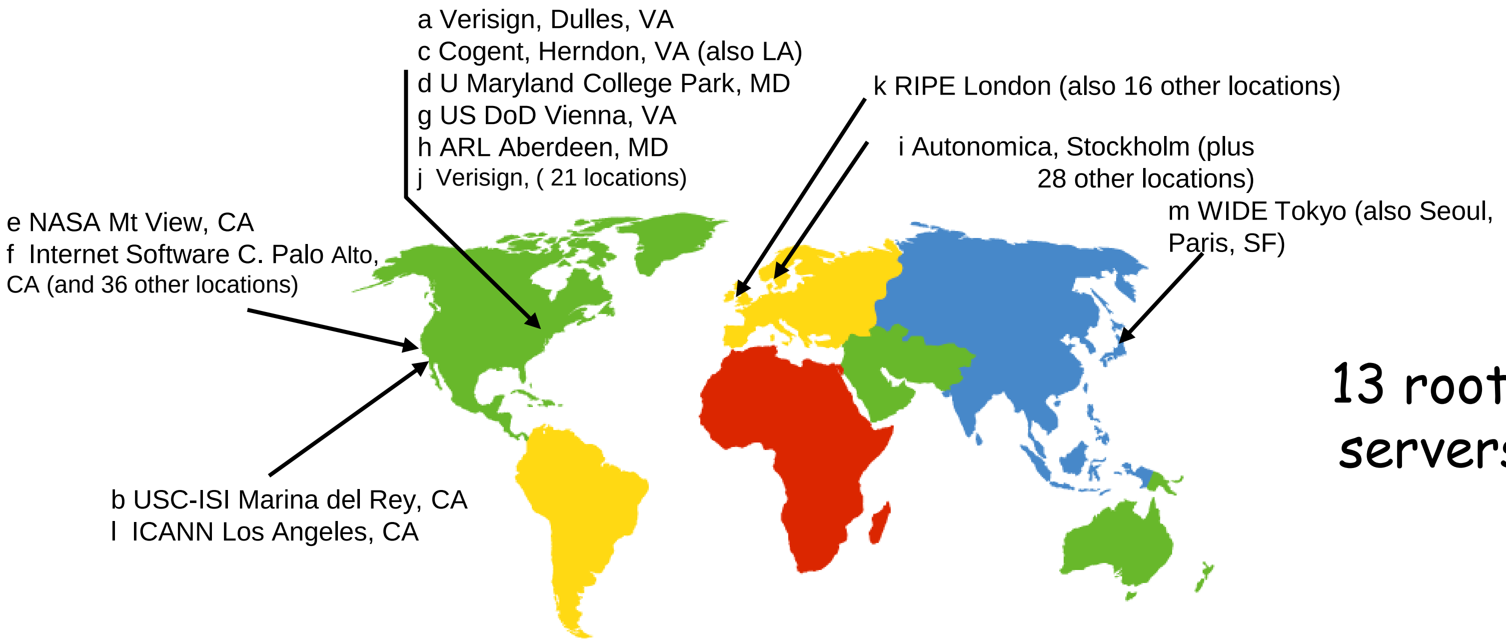


Client wants IP for www.amazon.com; 1st approx:

- 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

TLD and Authoritative 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, mail).
  - can be maintained by organization or service provider

Local Name Server

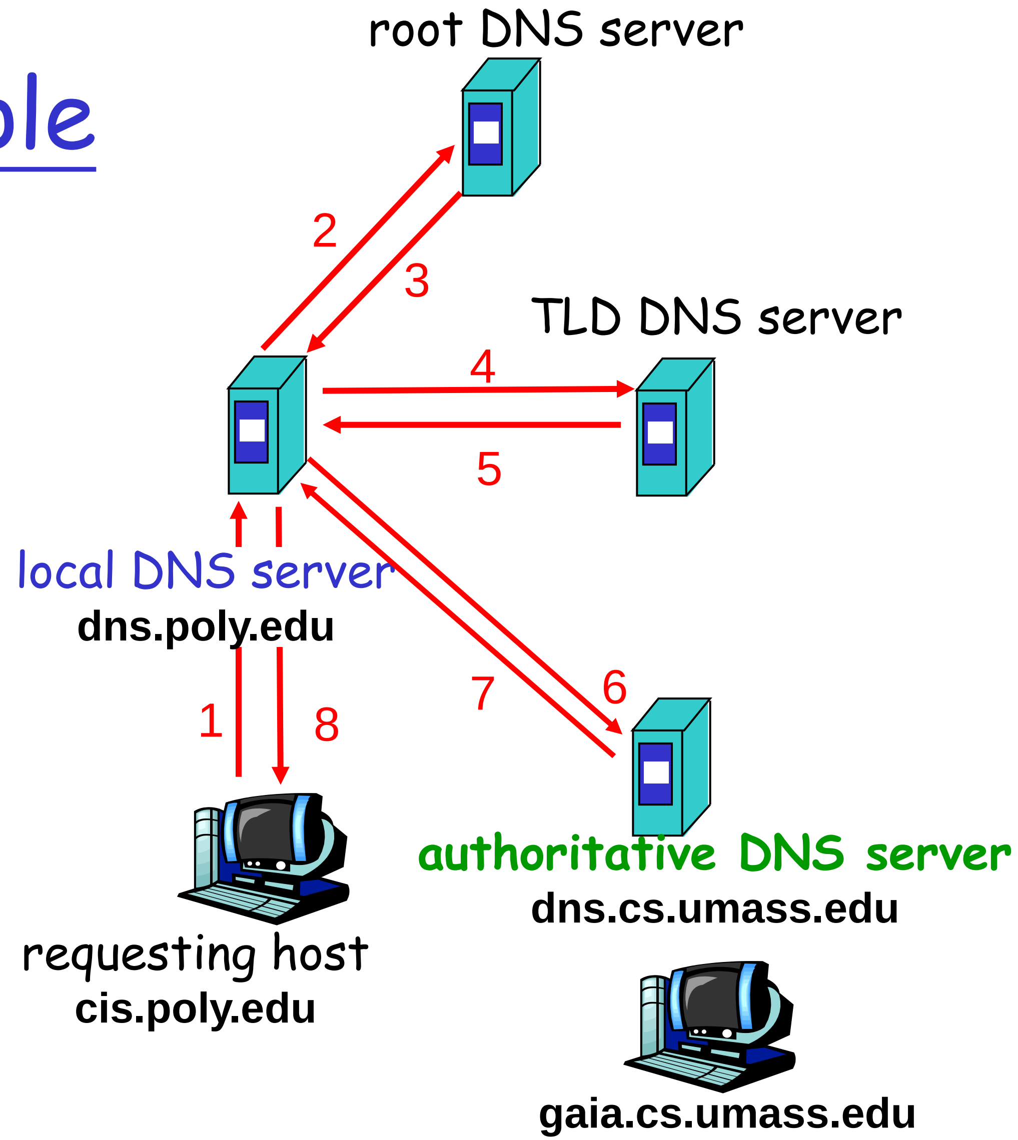
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one.
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

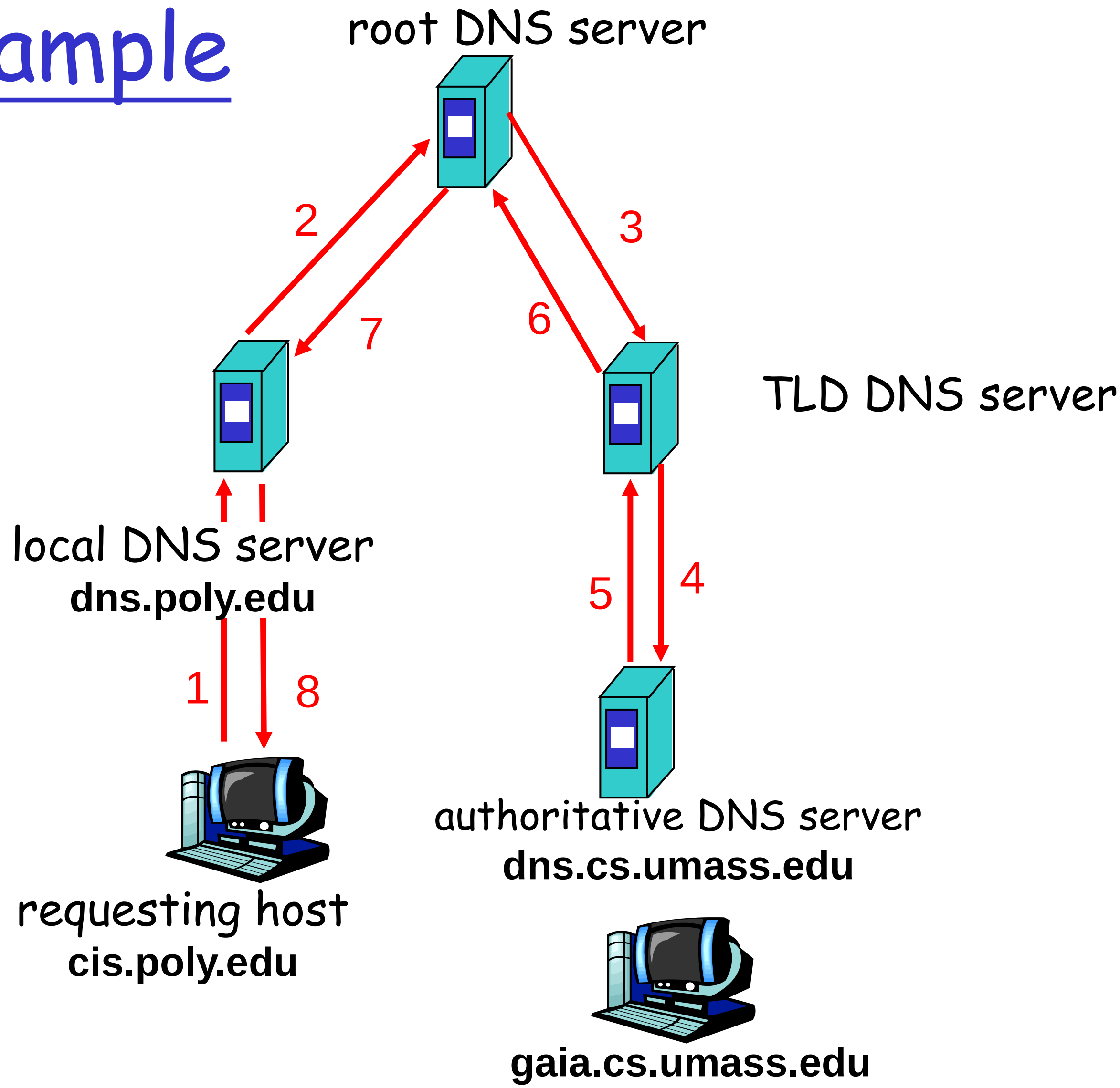
- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

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



DNS name resolution example



recursive query:

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

Domain Name Service (DNS) (cont'd)

- DNS protocol runs over UDP and uses port 53.
- Used by other application-layer protocols -- including HTTP, SMTP and FTP for name translation
- Name translation adds an *additional* delay -- sometimes substantial -- to the Internet applications that use DNS

DNS: caching and updating records

- once 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 are 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)`

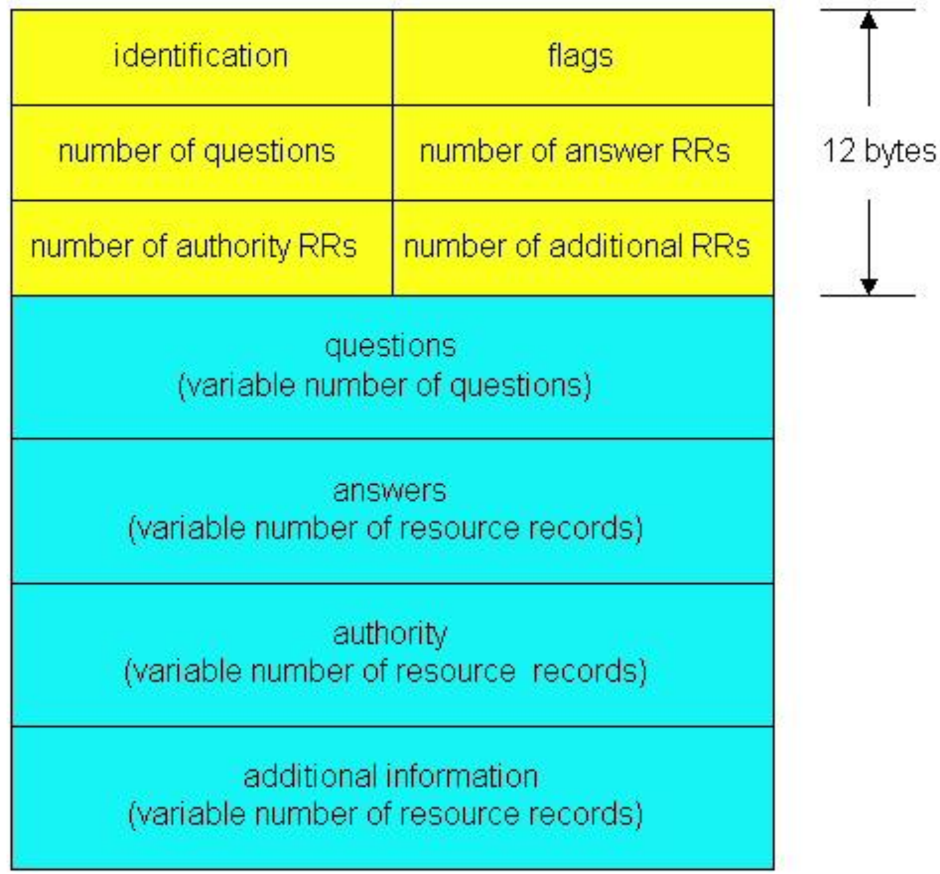
- Type=A
  - `name` is hostname
  - `value` is IP address
- Type=NS
  - `name` is domain (e.g. foo.com)
  - `value` is hostname of authoritative name server for this domain
- Type=CNAME
  - `name` is alias name for some "canonical" (the real) name
    `www.ibm.com` **is really** `servereast.backup2.ibm.com`
  - `value` is canonical name
- Type=MX
  - `value` is name of mailserver associated with `name`

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



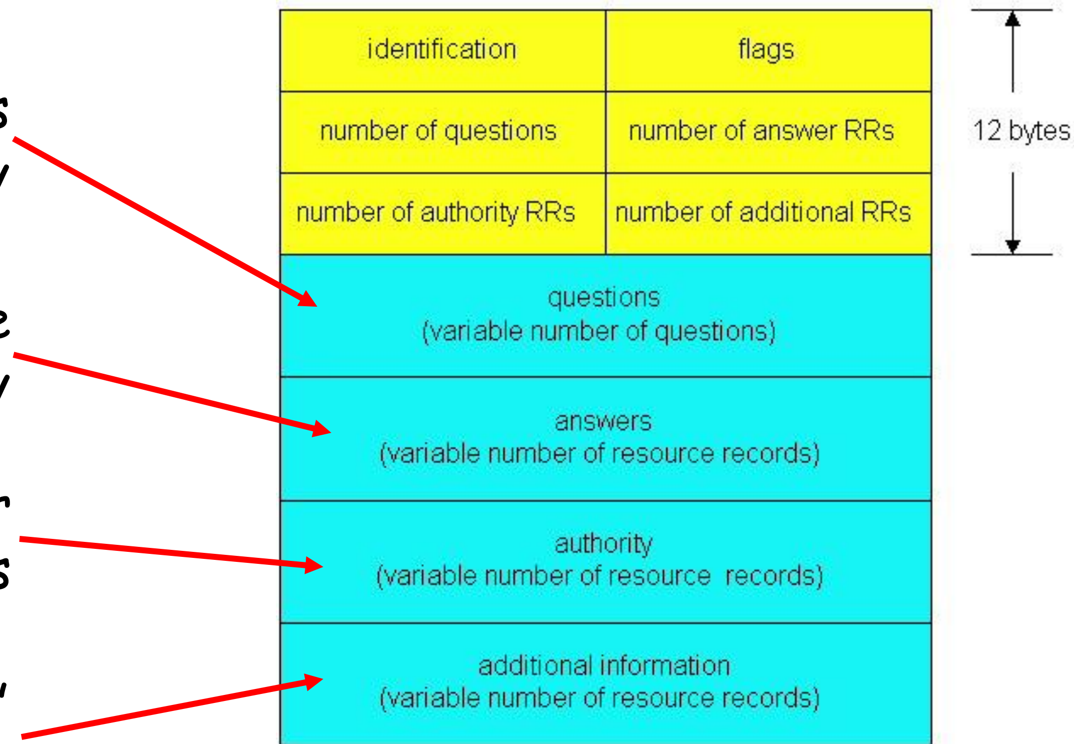
DNS protocol, messages

Name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



Host Aliasing

- A host with a complicated hostname can have *one or more alias names*, e.g.,
  - a hostname *relay1.west-coast.enterprise.com* could have, say, two aliases such as *enterprise.com* and *www.enterprise.com*
  - the hostname relay1.west-coast.enterprise.com is said to be *canonical hostname*

Host Aliasing (cont'd)

- Alias hostnames, when present, are typically more *mnemonic* than a canonical hostname.
- DNS *can* be invoked by an application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host.

Mail Server Aliasing

- It is highly desirable that email addresses be *mnemonic*, e.g.,
  - Bob has an account with Hotmail - *bob@hotmail.com*
  - the hostname of the Hotmail mail server is more complicated and much less mnemonic than simply hotmail.com (e.g., *relay1.west-coast.hotmail.com*).

Mail Server Aliasing (cont'd)

- DNS can be invoked by a mail application to obtain the *canonical hostname* for a supplied alias hostname and the *IP address* of the host.
- DNS permits a company's mail server and Web server to have *identical* (aliased) hostnames, e.g., enterprise.com.

Inserting records into DNS

- example: new startup "Network Utopia"
- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (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 NS record for networkutopia.com

- How do people get IP address of your Web site?

Load Distribution

- DNS is used to perform load distribution among *replicated servers*, e.g., web servers.
  - *Busy sites*, such as cnn.com, are replicated over multiple servers, with each server running on a different end system, and having a different IP address.
- *A set of* IP addresses is associated with *one* canonical hostname.

Load Distribution (cont'd)

- The DNS database contains *the set of IP addresses for each hostname*.
- When clients make a DNS query for a name mapped to a set of addresses,
  - the server responds with the *entire* set of IP addresses,
  - but *rotates* the ordering of the addresses within each reply.

Load Distribution (cont'd)

- A client *typically* sends its HTTP request message to the first IP address listed in the set
- *DNS rotation* distributes the traffic among all the replicated servers.
- DNS rotation also used for *email* with multiple mail servers having the same alias name.

References

- P. Mockapetris, "Domain Names - Concepts and Facilities," RFC 1034, Nov. 1987.
- P. Mockapetris, "Domain Names - Implementation and Specification," RFC 1035, Nov. 1987.
- P. Vixie, S. Thomson, Y. Rekhter, J. Bound, "Dynamic Updates in the Domain Name System," RFC 2136, April 1997.
- http://www.dns.net/dnsrd/docs/
  - a nice collection of documents pertaining to DNS

References (cont'd)

- http://www.isc.org/bind.html
- The Internet Software Consortium provides many resources for BIND, a popular public-domain name server for Unix machines
- Paul Albitz and Cricket Liu, "DNS and BIND," O'Reilly & Associates, 1993

Discussions

- DNS is *not* an application with which a user directly interacts.
- Instead, the DNS provides a core Internet name-to-address translation function for user applications and other Internet software
- Much of the "*complexity*" in the Internet architecture is located at the "*edges*" of the network

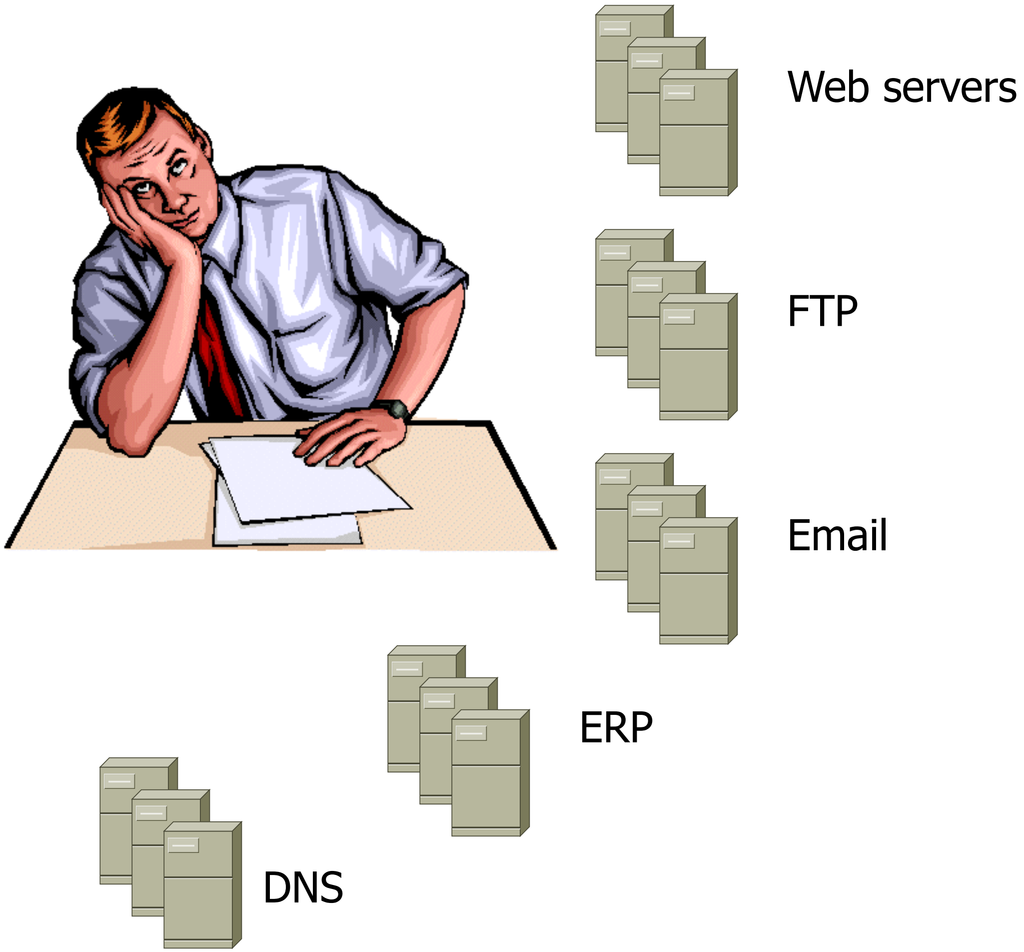
Server Farm and Web Switch

Introduction

- The Internet, in particular the World Wide Web, has experienced explosive growth and continues to extend at an amazing pace.
- Cluster-based server architecture is a successful and cost effective alternate to build a scalable, reliable, and high-performance Internet server system

Server Environment

- Multiple Servers per application clustering
  - Scalability
  - Availability
- Manageability needs
  - Service portability
  - Transparent to users
  - No service disruption



Introduction

- An important issue is "how to dispatch and route incoming requests to the server best suited to respond?"
- Issues ignored by Existing Routing Schemes
  - Session Integrity
  - Sophisticated Load Balancing
  - Differentiated Services
  - Content Deployment

Arrowpoint

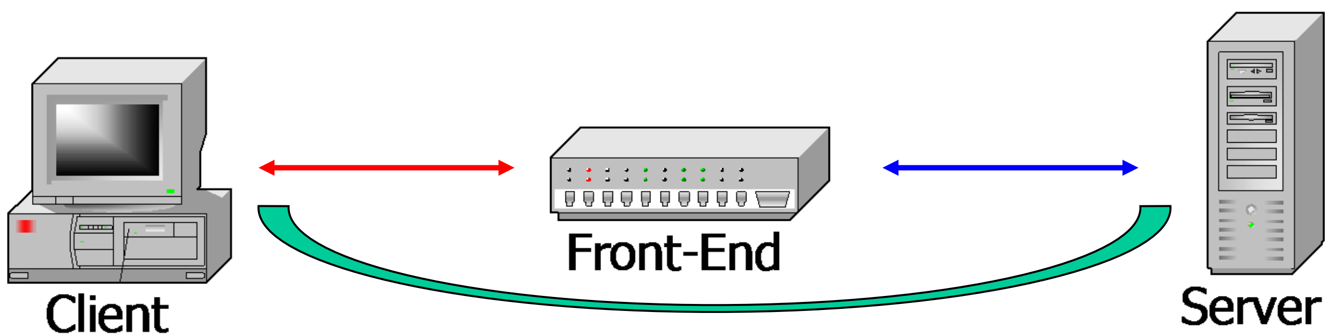
Content-aware Request Distribution

Advantages

- Increase performance due to improved hit rates
- Able to partition the server's database over the different back-end nodes
- Specialized for certain types of requests

Content-aware Request Distribution (cont'd)

- Issue of "Content Inspection"
  - "read" the content of the requests
  - a TCP connection must be established with the client prior to assigning the request to a back-end node.



Web Switch

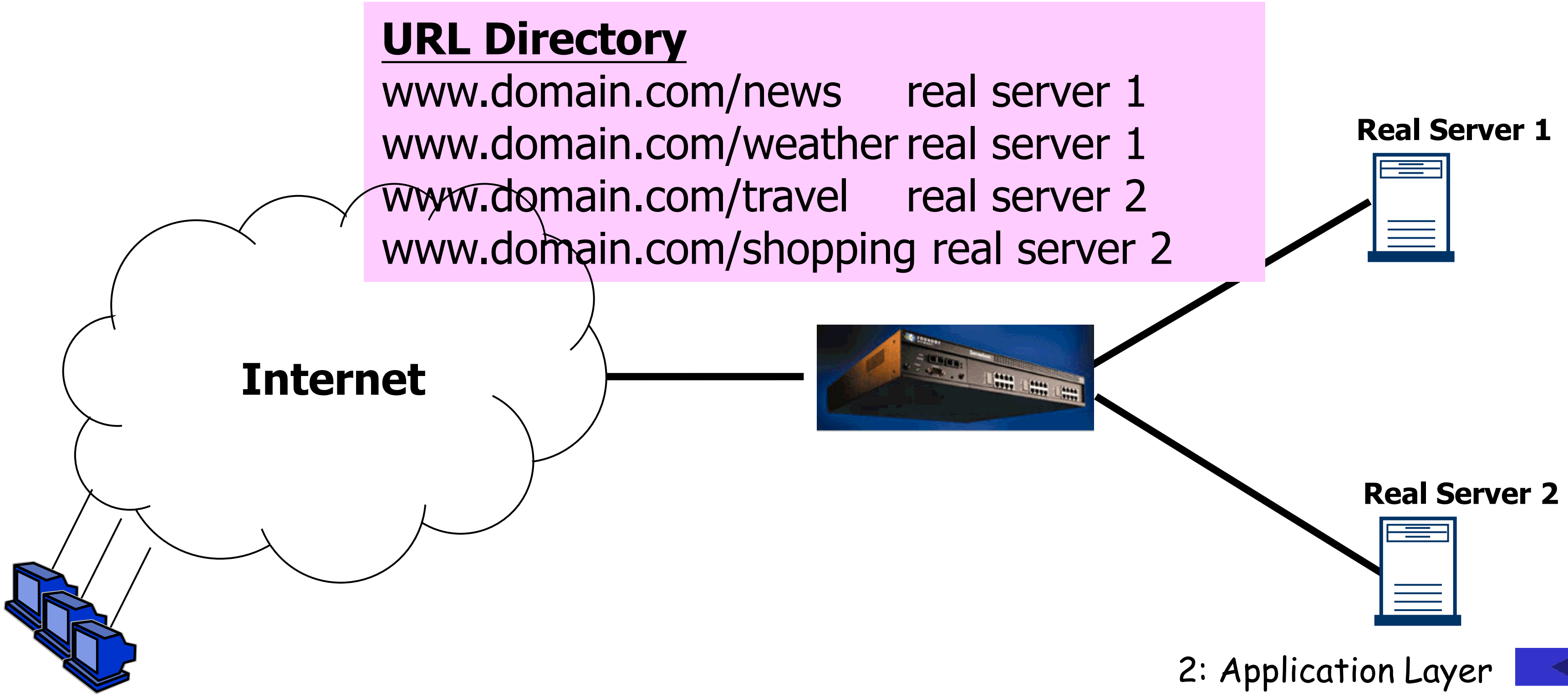
- To support virtual IP for web service
  - network address translation (VIP -> realIP)
- To support content-based switching
  - URL switching
  - Cookie switching - stateful
- To support server farm
  - load dispatching
  - Health check

HTTP Request Examples

```
GET /sports/baseball/index.shtml HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.08 [en] (X11; I; FreeBSD 4.1-Release i386)
Host: www.kimo.com.tw
Accept: image/gif, image/x-xbitmap, image/jpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: SESSIONID=123456789
```

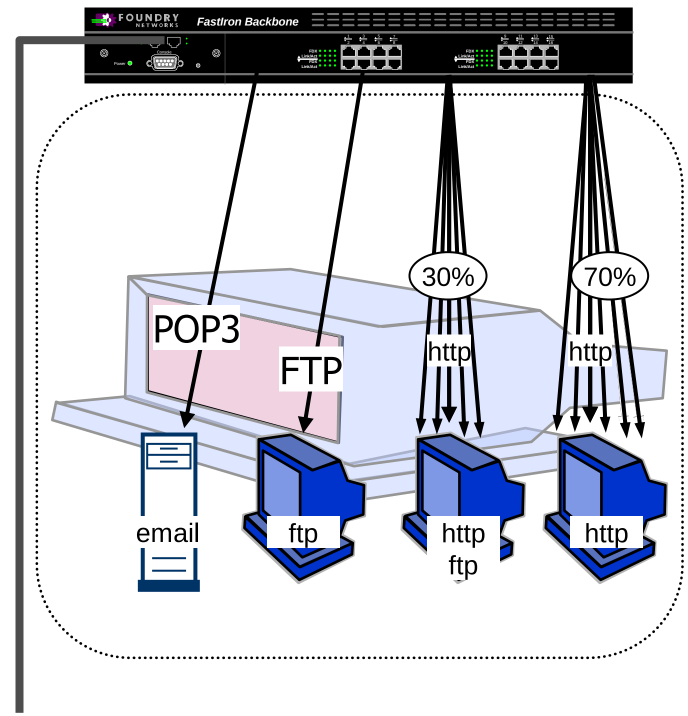
URL Switching, Content Publishing/Web Hosting

- Server load balancing based on requested content
- Reduce content replication and management overhead
- URL switching is on a per VIP basis
- High performance URL parser



Server Load Balancing

- Users connect to a Virtual IP Address but actually are served by multiple Physical Servers
- Local Load share algorithm –
  - Round-Robin
  - Least Load first (session count)
  - Least traffic first (bytes count)
  - Least weighted load (weight + session count)
  - Ping to find the most respons host

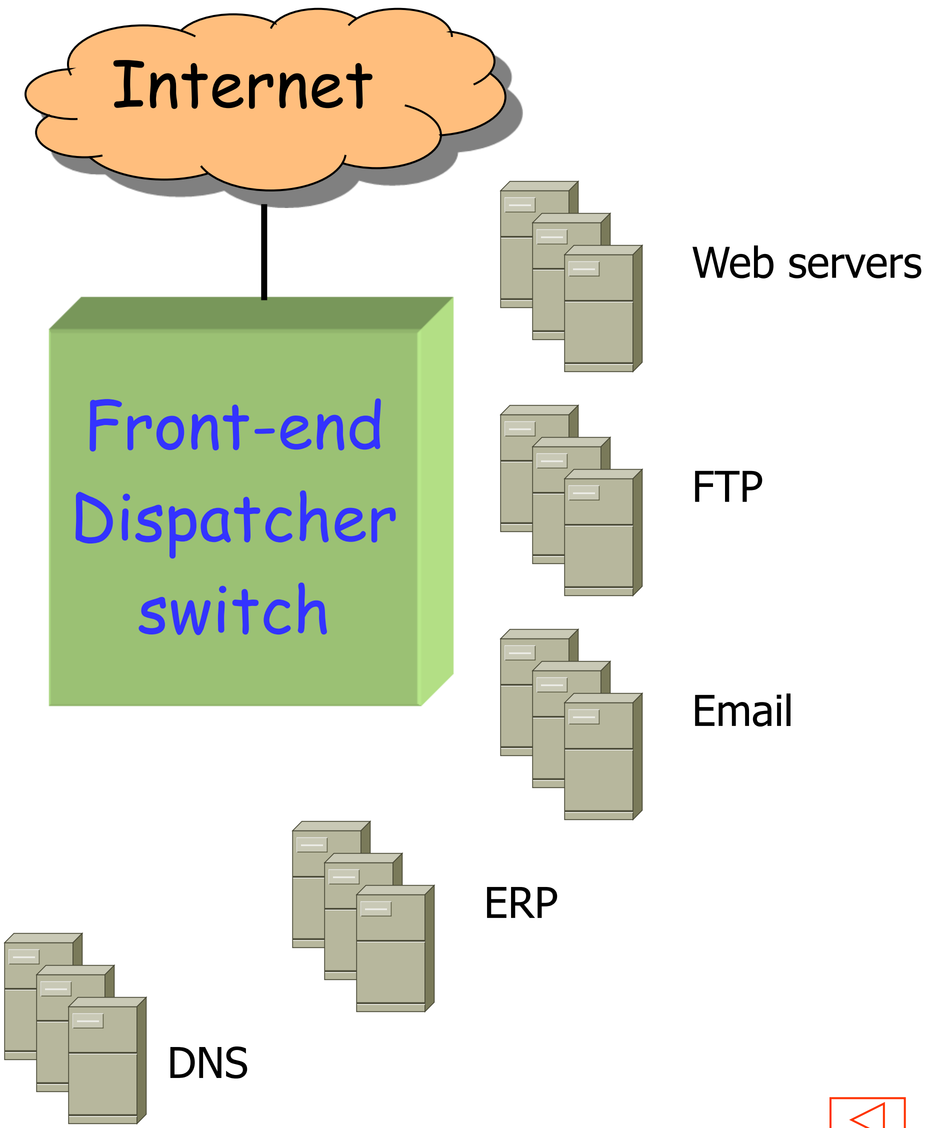


HTTP traffic to Servers 1 & 2
FTP traffic to Servers 2 & 3
Email to Server 4

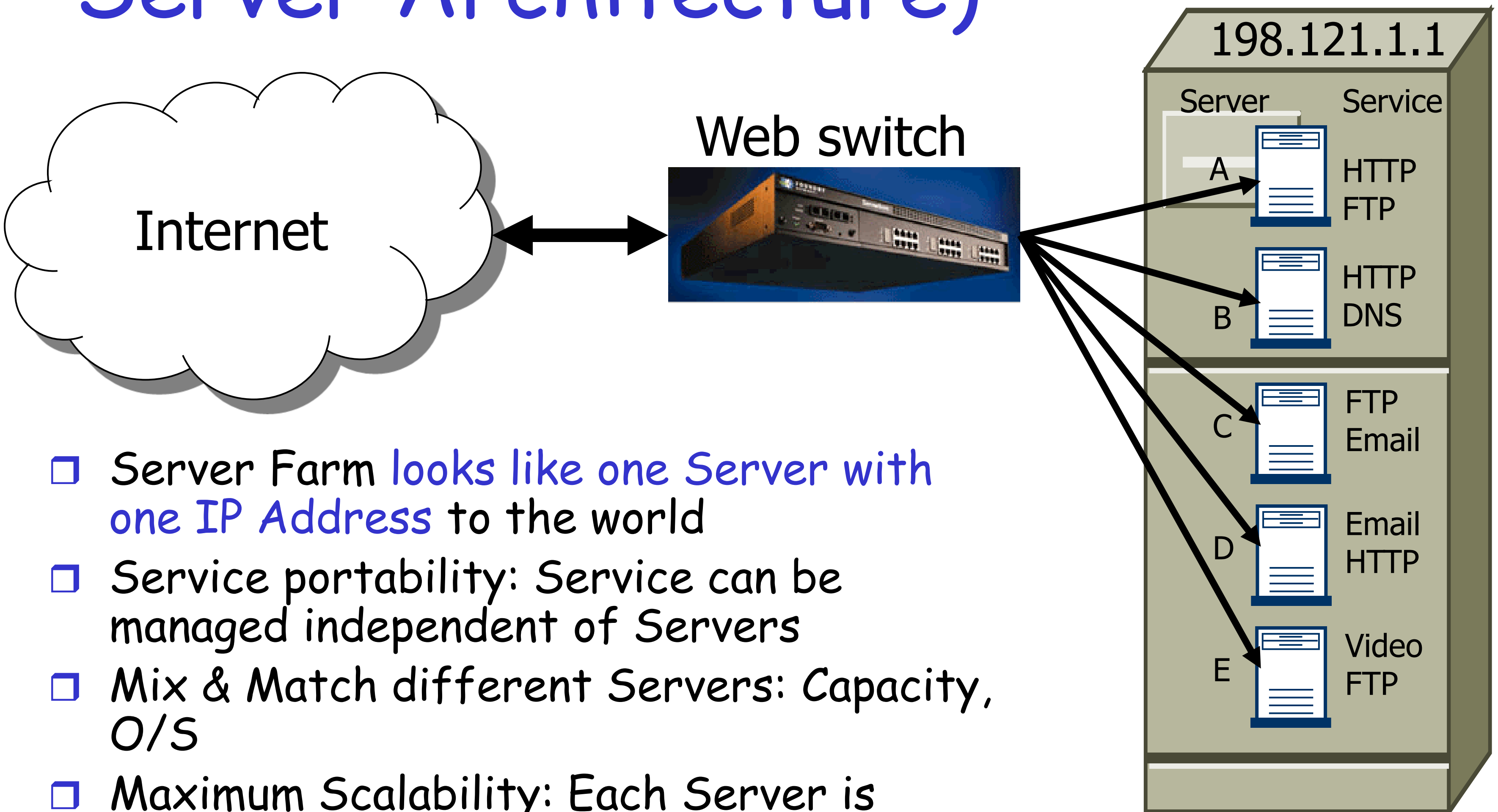
Click for next animation

Virtual IP

- Back-end nodes in the cluster share a common IP address, called VIP.
- Each node has its own unique IP (public or private) and MAC address
- client always sees VIP



Server Farms (Cluster-based Server Architecture)



- Server Farm looks like one Server with one IP Address to the world
- Service portability: Service can be managed independent of Servers
- Mix & Match different Servers: Capacity, O/S
- Maximum Scalability: Each Server is utilized to its potential

ArrowPoint technology hits Cisco jackpot

May 11, 2000 12:15 PM PDT

- a start-up that builds equipment that speeds delivery of Web content over the Internet, held a public offering March 31 that gave the company a market value of about $1 billion based on its opening stock price.
- Then last Friday, networking giant Cisco Systems acquired the start-up for about $5.7 billion in stock, based on Cisco's stock price at the time of the deal. That essentially increased ArrowPoint's value sixfold in just six weeks. Cisco shares have since dropped.
- Cisco was in desperate need of ArrowPoint's Web switches, equipment used by e-commerce Web sites and Internet service providers to manage Net traffic.
- The market is expected to grow from $260 million in 1999 to $828 million by 2002, according to a study by Internet Research Group.

Content Delivery Network (CDN)

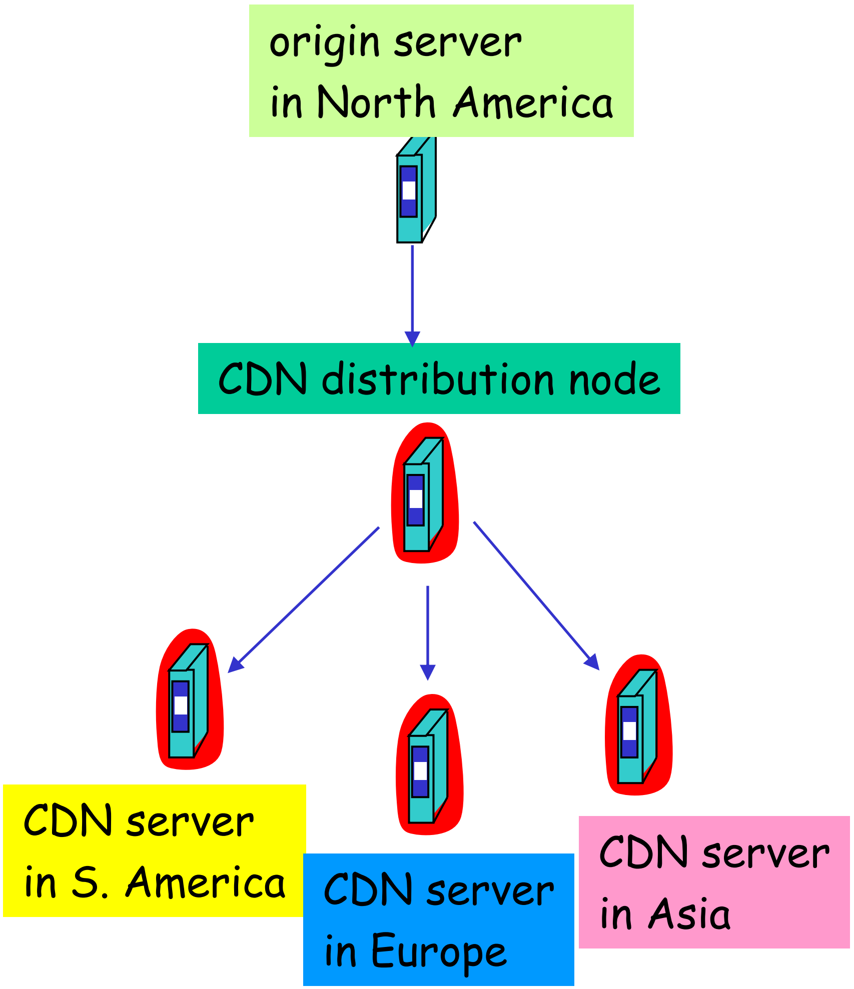
- Deliver Web-based content from geographically dispersed servers that sit on the edge of various networks
- Deliver content according to the proximity of the Web surfer.
- Example
  - A Web surfer viewing a Web site on a computer in California most likely will get content delivered from servers on the West Coast;
  - a Boston viewer would get images from a server on the East Coast.

Content distribution networks (CDNs)

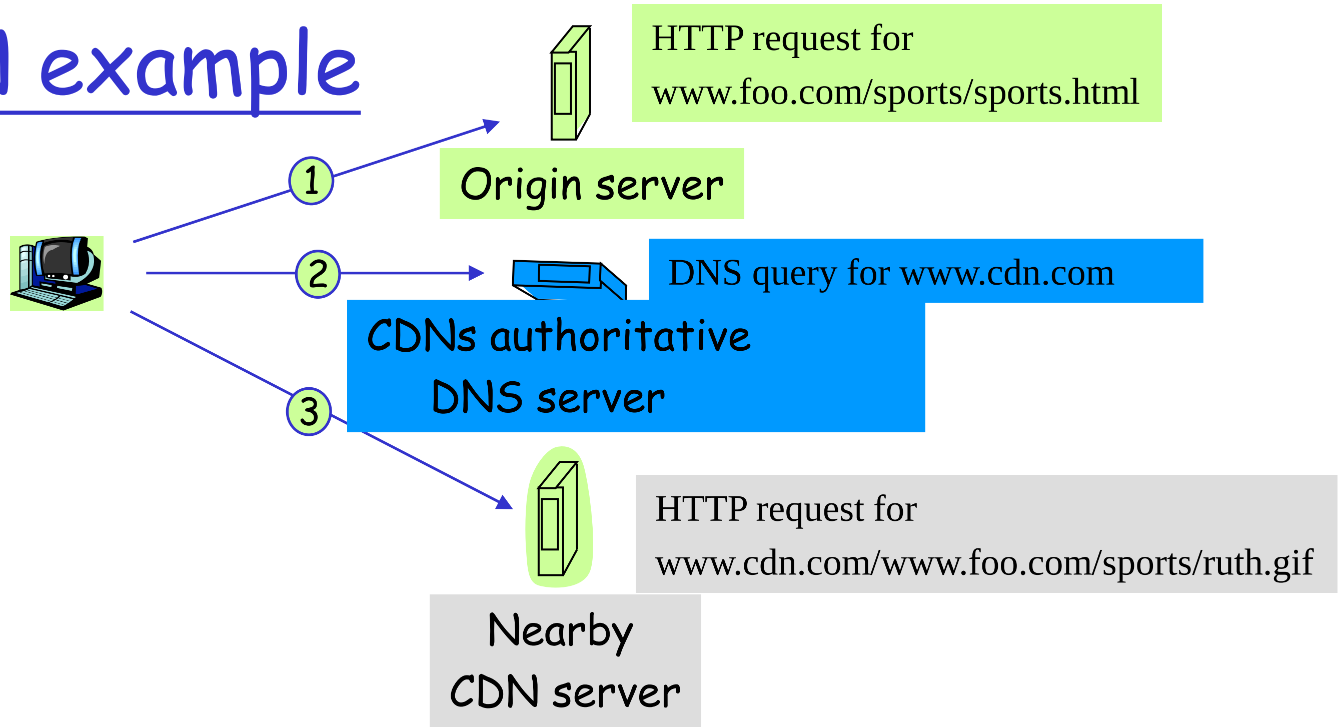
- The content providers are the CDN customers.

Content replication

- CDN company installs hundreds of CDN servers throughout Internet
  - in lower-tier ISPs, close to users
- CDN replicates its customers' content in CDN servers. When provider updates content, CDN updates servers



CDN example



origin server

- www.foo.com
- distributes HTML
- Replaces:

  http://www.foo.com/sports.ruth.gif

  with

  http://www.cdn.com/www.foo.com/sports/ruth.gif

CDN company

- cdn.com
- distributes gif files
- uses its authoritative DNS server to route redirect requests

More about CDNs

routing requests

- CDN creates a "map", indicating distances from leaf ISPs and CDN nodes
- when query arrives at authoritative DNS server:
  - server determines ISP from which query originates
  - uses "map" to determine best CDN server

not just Web pages

- streaming stored audio/video
- streaming real-time audio/video
  - CDN nodes create application-layer overlay network

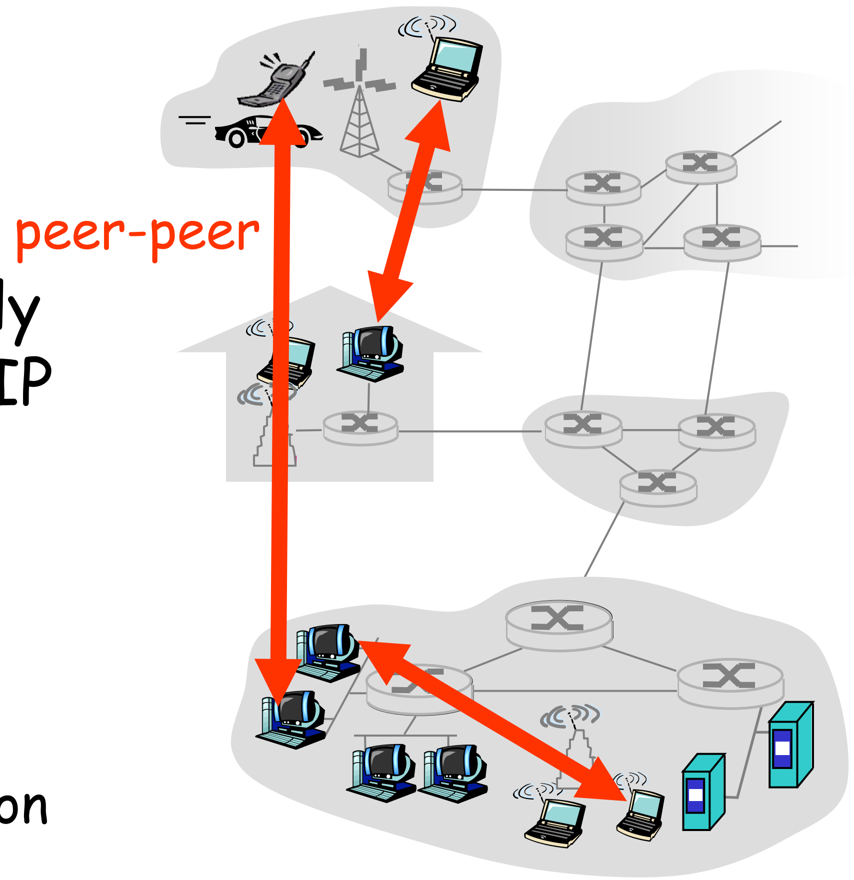
Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP

Pure P2P architecture

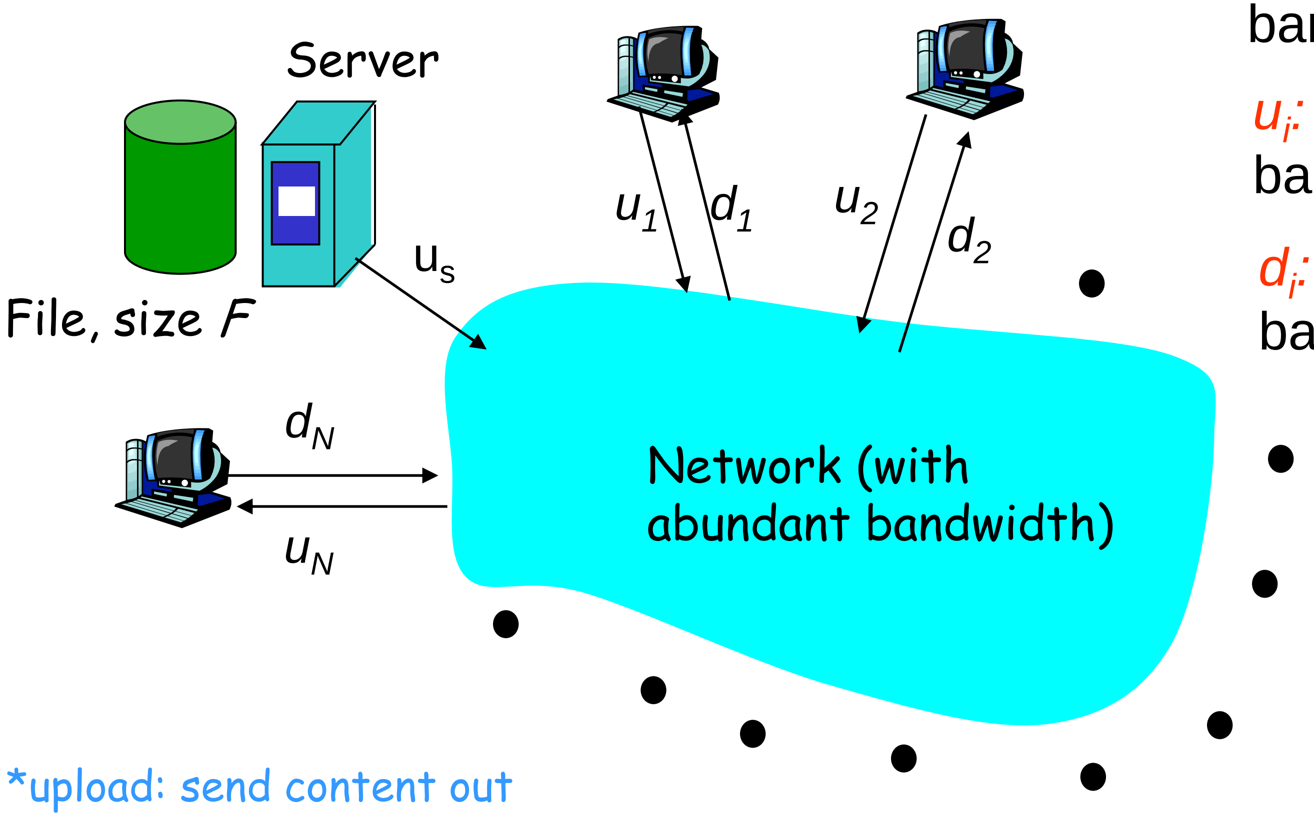
- *no* always-on server
- *arbitrary* end systems *directly* communicate
- peers are intermittently connected and change IP addresses

- 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*?



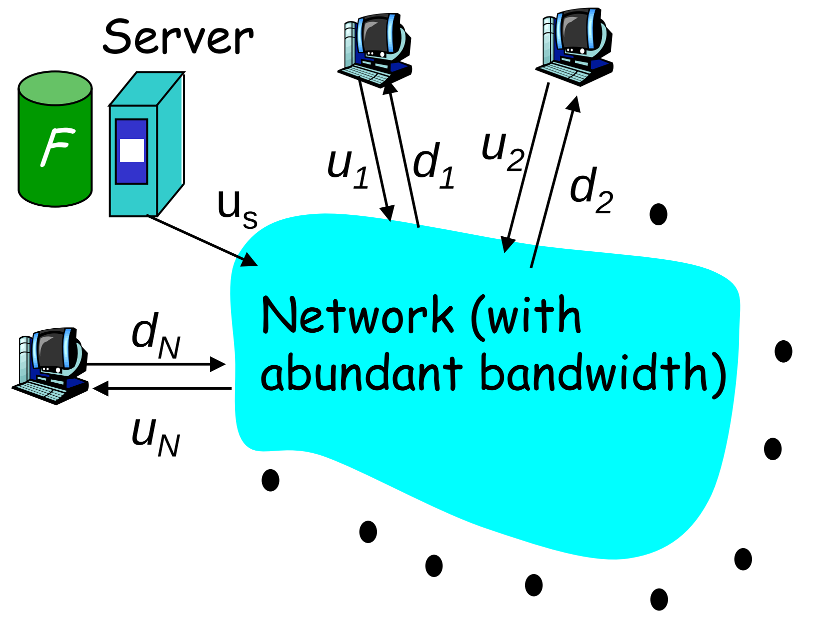
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

*upload: send content out

File distribution time: server-client



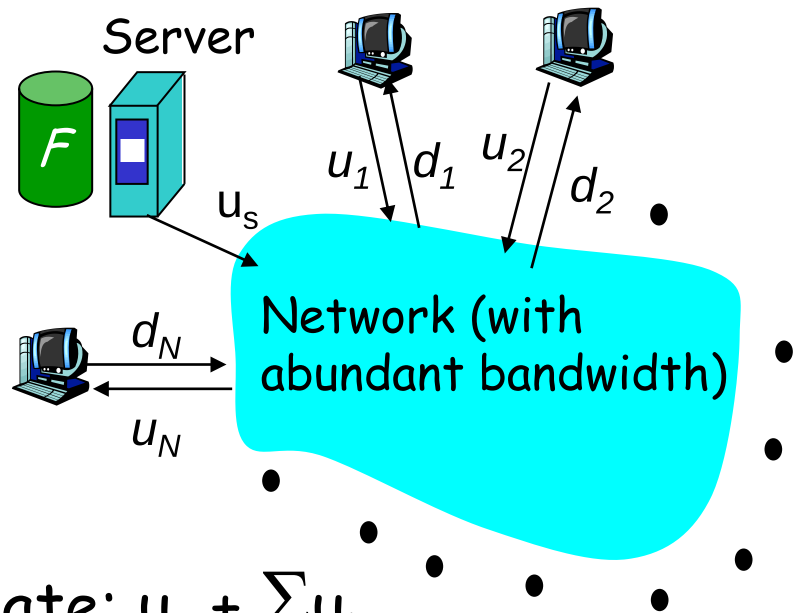
- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download

$$\text{Time to distribute } F \text{ to } N \text{ clients using client/server approach} = d_{cs} = \max\left\{ NF/u_s,\ F/\min_i(d_i) \right\}$$

increases linearly in N (for large N)

File distribution time: P2P

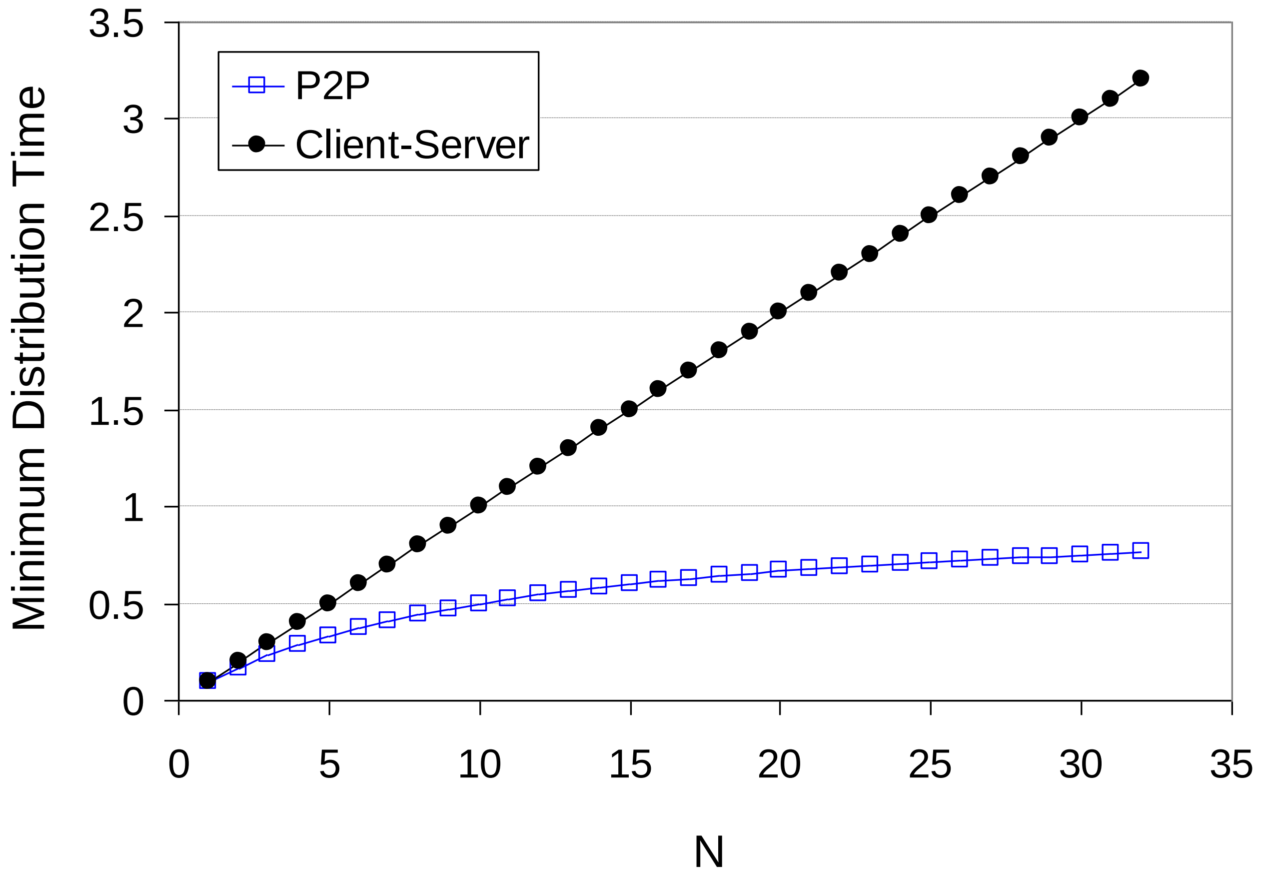
- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \sum u_i$



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

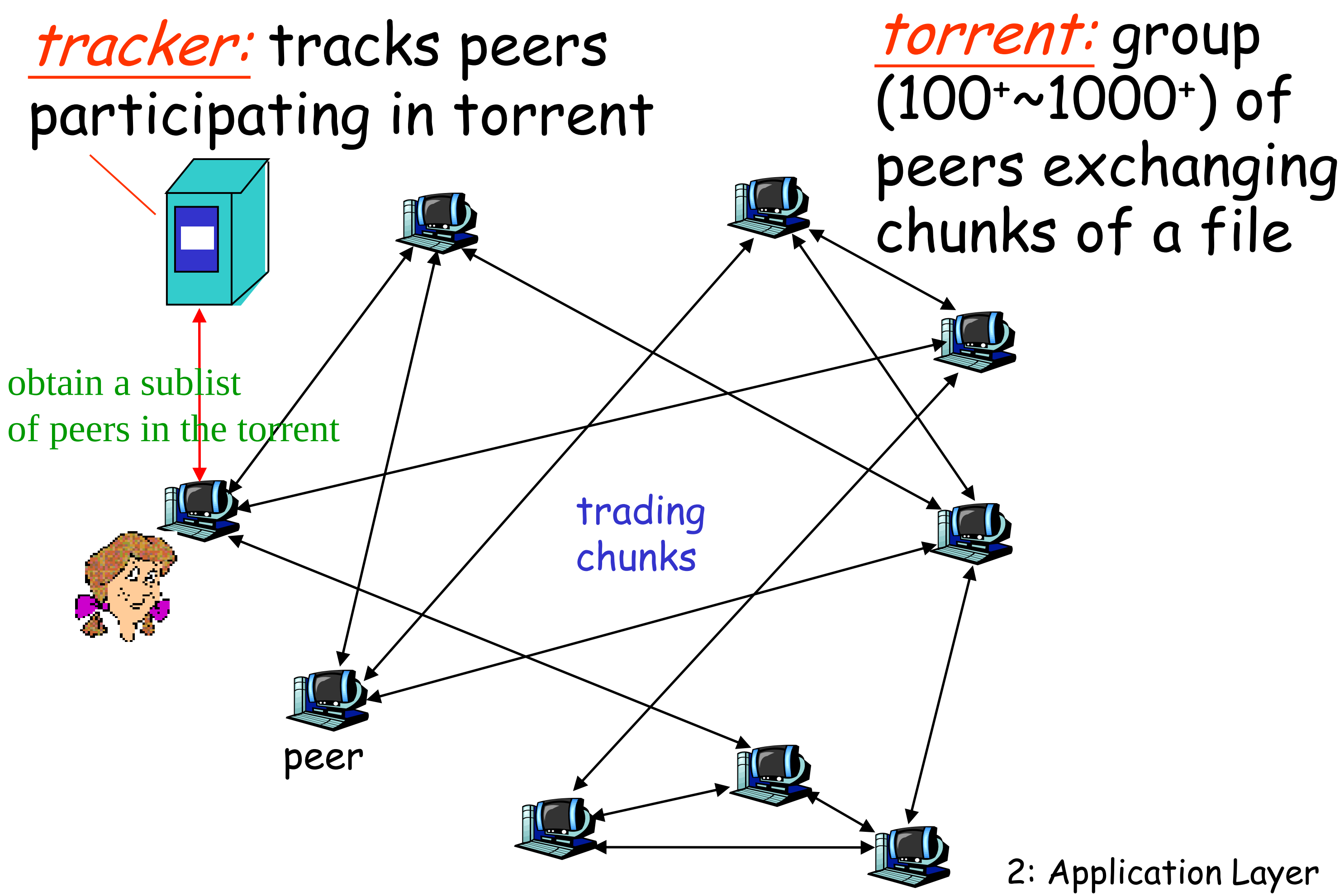
Server-client vs. P2P: example

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

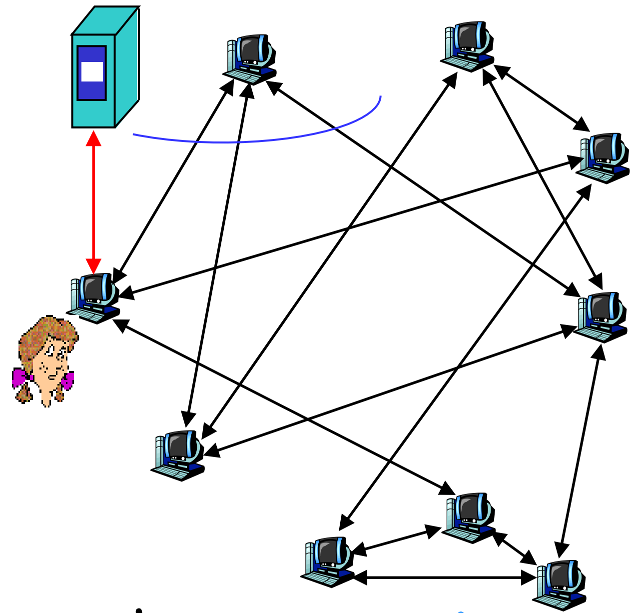


P2P Case Study: BitTorrent

- 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

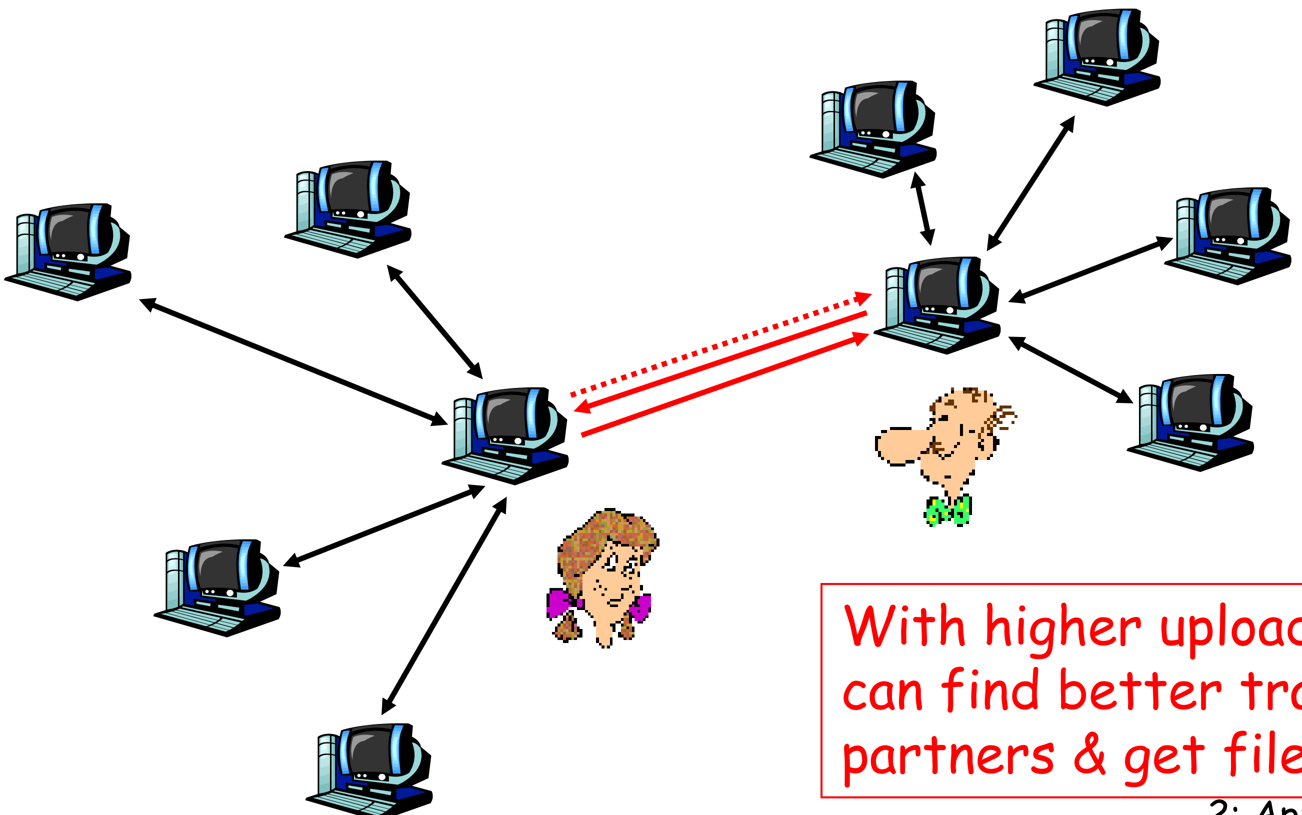
- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

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)

- The goal is to provide a way of indexing for information search and update in distributed database.
- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name (-> peer)
  - key: content type; value: IP address (-> content)
- Peers query DB with key (-> search)
  - DB returns values that match the key
- Peers can also insert (key, value) peers

Centralized vs. Distributed Approach

- Early P2P systems such as Napster took the centralized approach.
- Distributed database approach, e.g.,
  - Distribute (e.g., randomly) the database across all the peers; each peer only holds a small subset of the totality of the pairs.
  - Each peer maintains a list of the IP addresses of all participating peers.
  - Send a query to all other peers and the one has the pairs respond .
  - Not scalable!

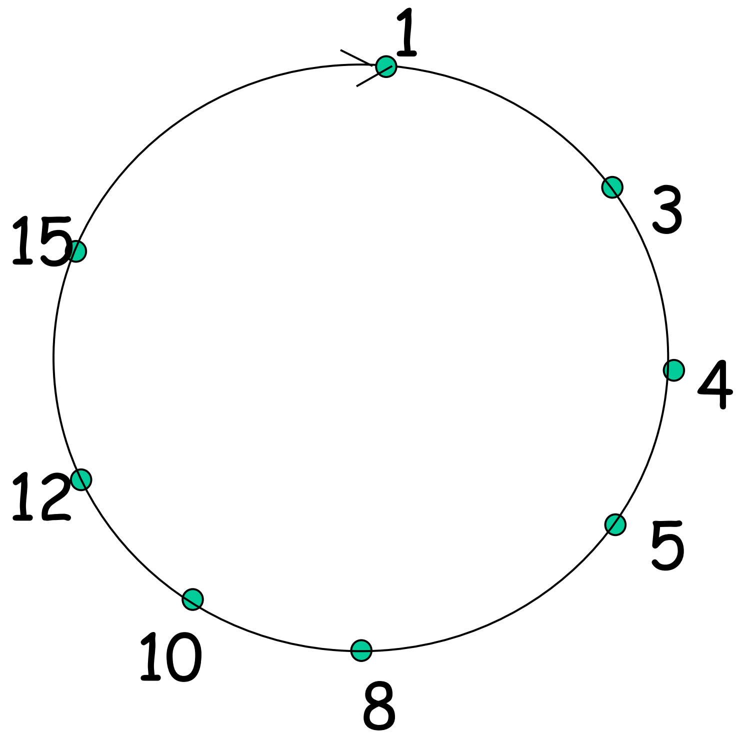
Design a P2P database: DHT Identifiers

- Assign integer identifier to each peer in the range $[0,2^n-1]$.
  - Each identifier can be represented by n bits.
- Require each key to be an integer in the same range.
- To get integer keys for contents, 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: (distribute pairs among peers)
  - 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

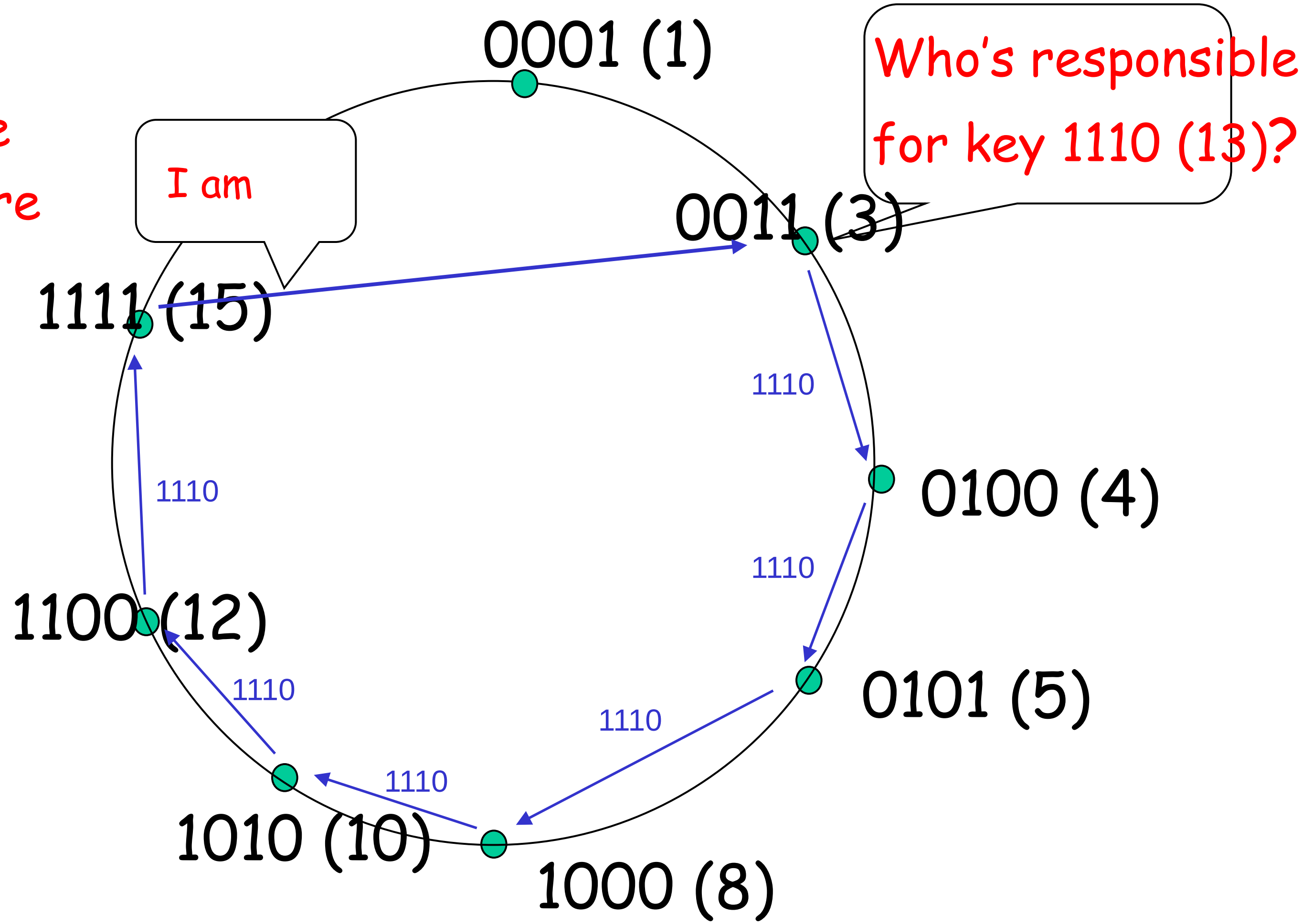
Circular DHT (1)



- Each peer *only* aware of immediate successor and predecessor.
- "Overlay network" (specifies abstract logical relationship between peers)

Circle DHT (2)

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



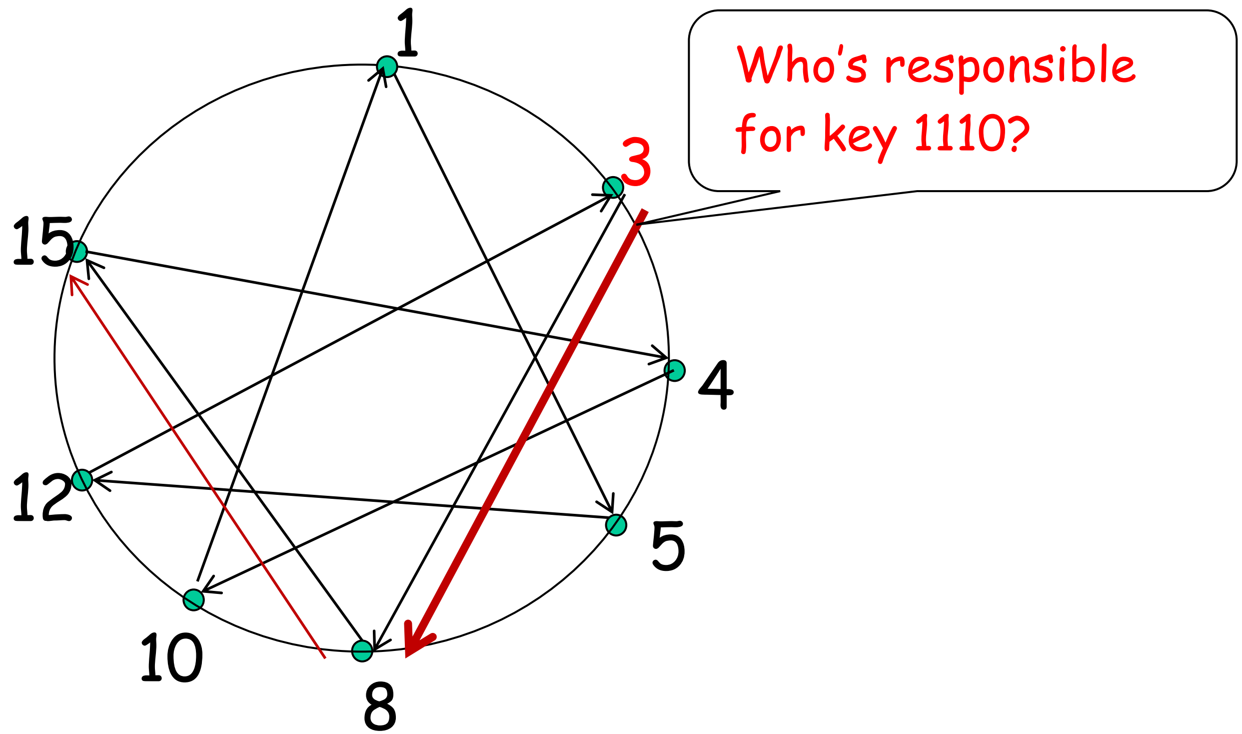
Define closest as immediate successor

Discussion

- Issue of tradeoff between the number of neighbors each peer has to track and the number of messages that the DHT needs to send to resolve a single query.
- Need to refine the design of DHT to keep the two numbers to an acceptable size.

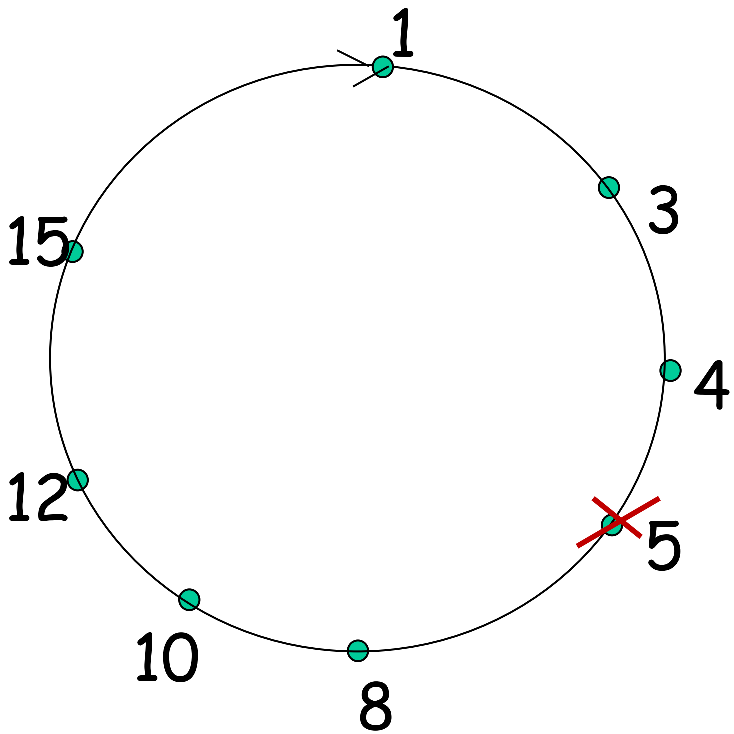
-> add shortcuts!

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
- Can significantly reduce the number of messages used to process a query.

Peer Churn

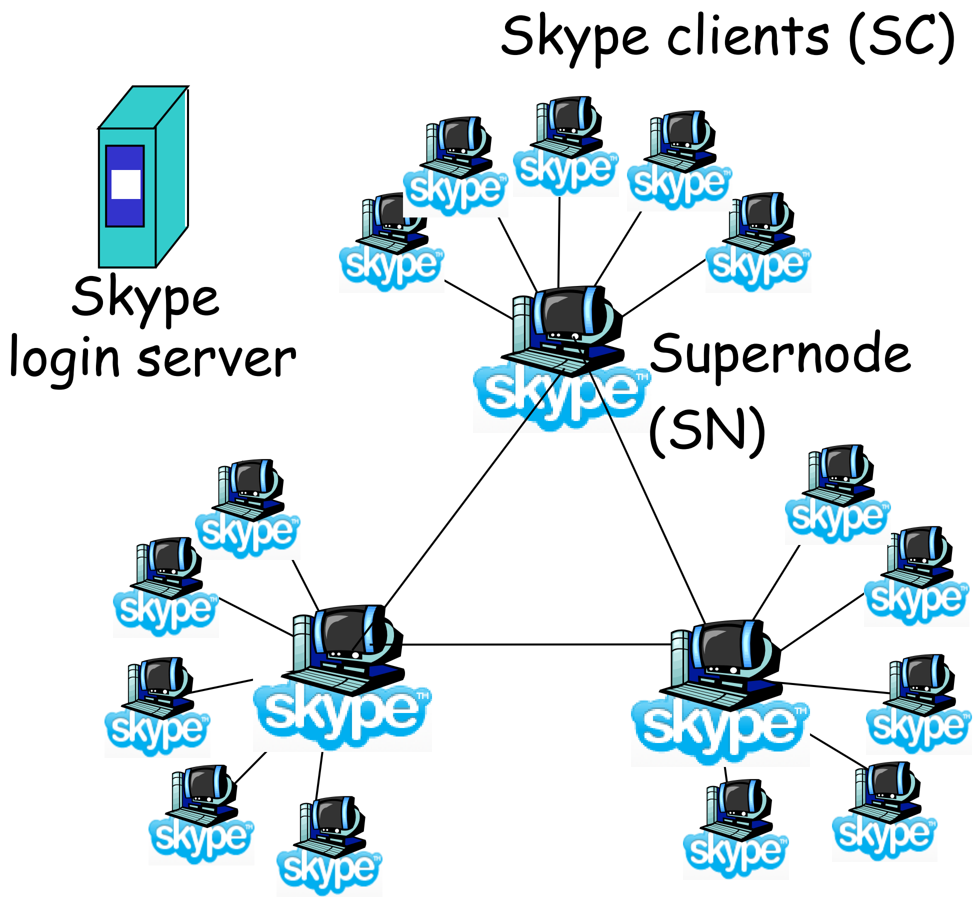


- A peer can come and go arbitrarily.
- To handle peer churn, require each peer to know the IP address of its two successors.
- Each peer periodically pings its two successors to see if they are still alive.

- 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?
  (assume it knows peer 1's (or any) existence in the DHT.

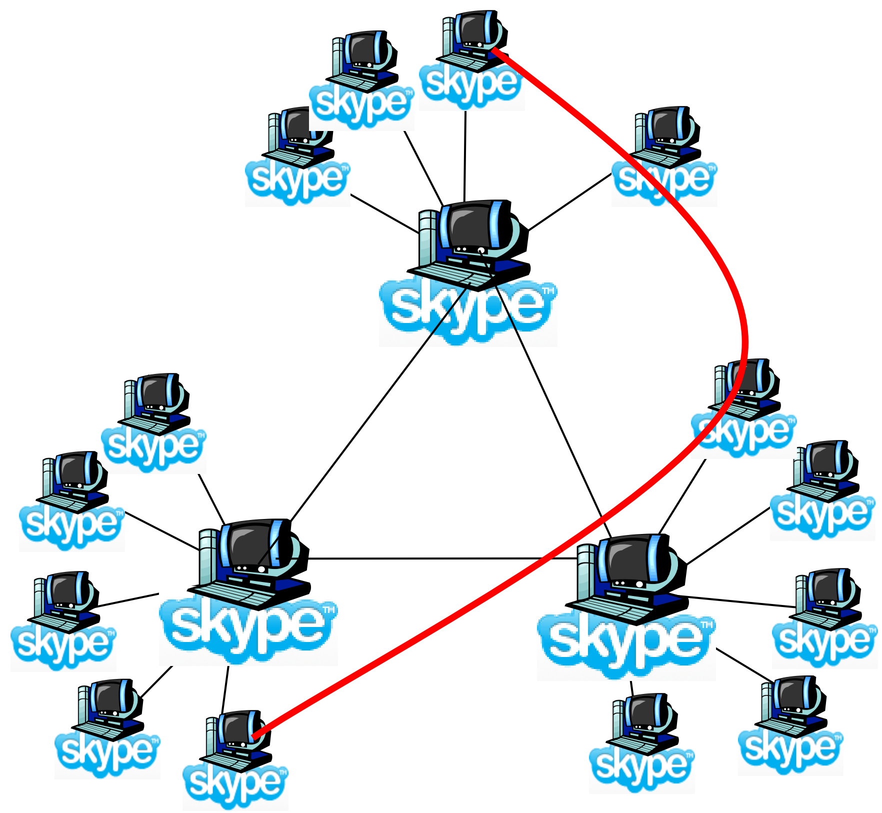
P2P Case study: Skype

- 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
  - P2P: BitTorrent, Skype
- socket programming

The end. ☺