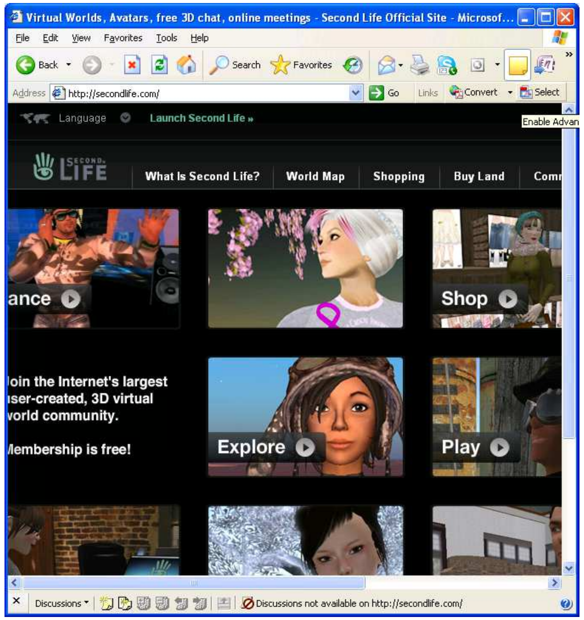
Network Applications

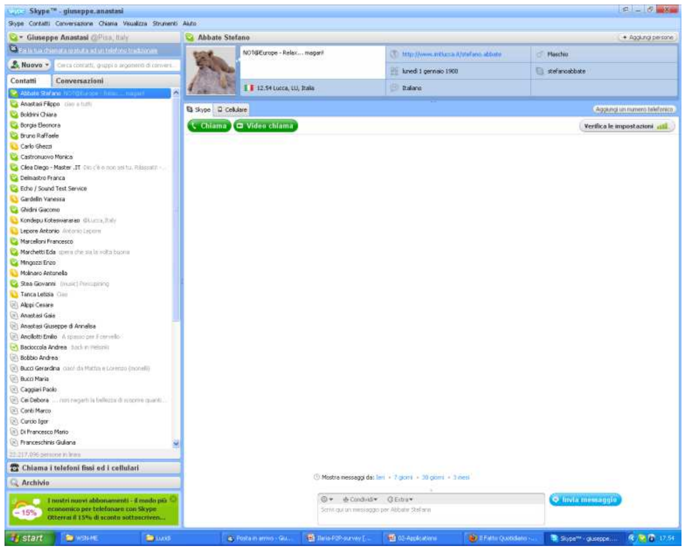
Acknowledgements

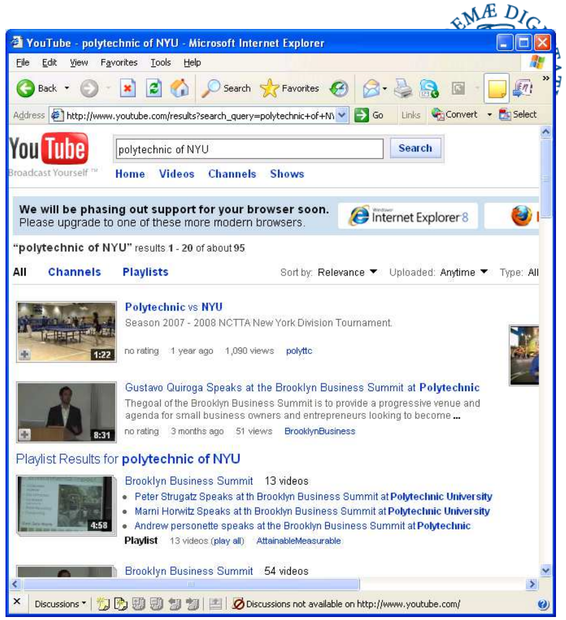
These Slides have been adapted from the originals made available by J. Kurose and K. Ross

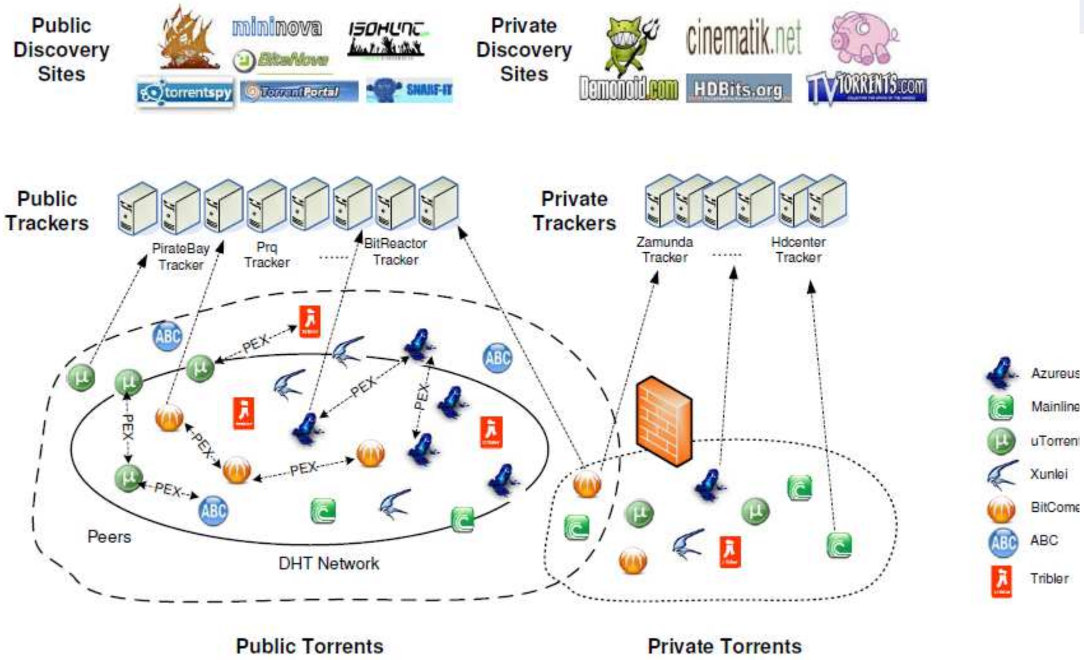
Some Network Applications

- E-mail
- Remote login
- File transfer
- instant messaging
- Web
- Streaming stored video clips
- Voice over IP
- Real-time video conferencing
- P2P file sharing
- Social networks
- Multi-user network games
- Cloud computing
- …


Virtual Worlds, Avatars, free 3D chat, online meetings - Second Life Official Site - Microsof...
What Is Second Life?
World Map
Shopping
Buy Land
Shop
Explore
Play
Join the Internet's largest user-created, 3D virtual world community.
Membership is free!




YouTube - polytechnic of NYU - Microsoft Internet Explorer
polytechnic of NYU
Search
We will be phasing out support for your browser soon.
Please upgrade to one of these more modern browsers.
"polytechnic of NYU" results 1 - 20 of about 95
Polytechnic vs NYU
Season 2007 - 2008 NCTTA New York Division Tournament
Gustavo Quiroga Speaks at the Brooklyn Business Summit at Polytechnic
Playlist Results for polytechnic of NYU
Brooklyn Business Summit 13 videos
Brooklyn Business Summit 54 videos


Public Discovery Sites
Private Discovery Sites
Public Trackers
Private Trackers
PirateBay Tracker
Prq Tracker
BitReactor Tracker
Zamunda Tracker
Hdcenter Tracker
PEX
Peers
DHT Network
Public Torrents
Private Torrents
Azureus
Mainline
uTorrent
Xunlei
BitComet
ABC
Tribler


facebook
Giuseppe Anastasi

Goals

- Conceptual and implementation issues of network applications
  - Network application paradigms
    - Client-server
    - Peer-to-peer (P2P)
  - Service required by applications
  - Service models provided by the network
- Some popular applications
  - WWW, File Transfer
  - E-mail
  - Domain Name System (DNS)
  - P2P Applications (File Sharing, Internet Telephony)
- Programming network applications
  - Application Programming Interface (API)

Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P applications
  - File Sharing, Internet Telephony
- Socket programming

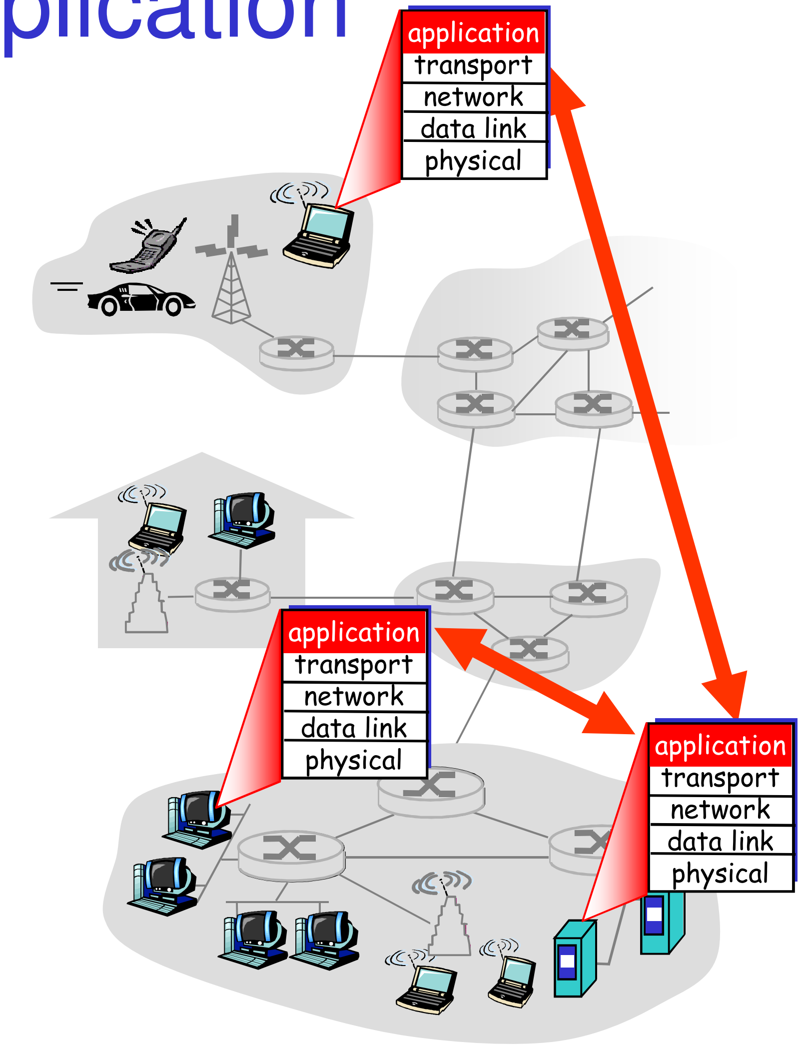
Creating a network application

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



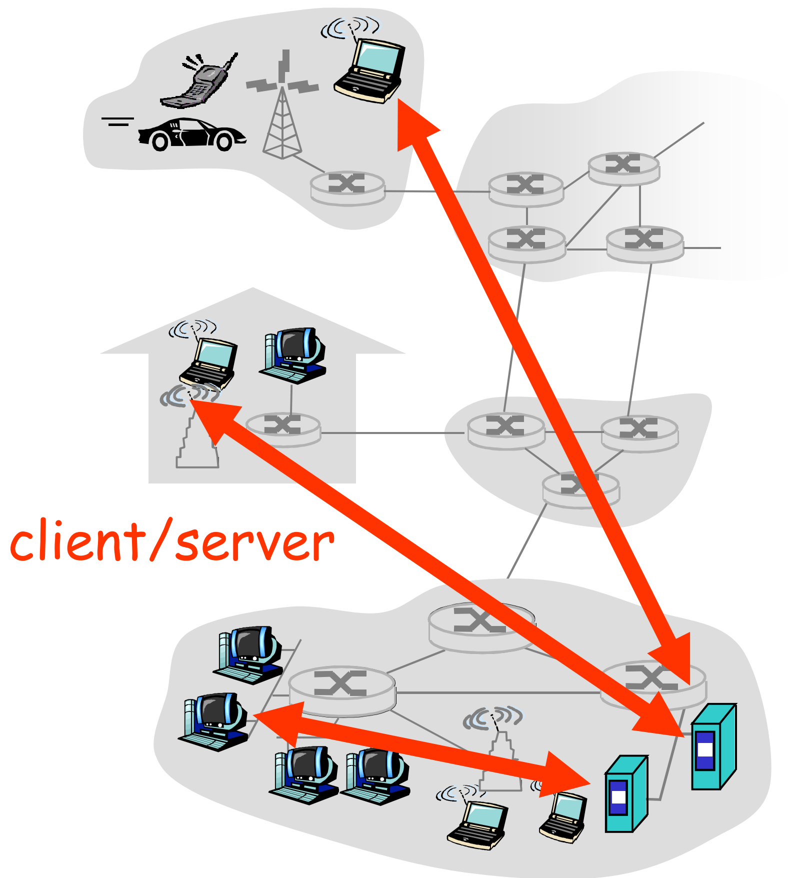
Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P applications
  - File Sharing, Internet Telephony
- Socket programming

Application architectures

- Client-server
  - Including data centers / cloud computing
- Peer-to-peer (P2P)
- Hybrid
  - Combination of client-server and P2P

Client-server architecture



server:

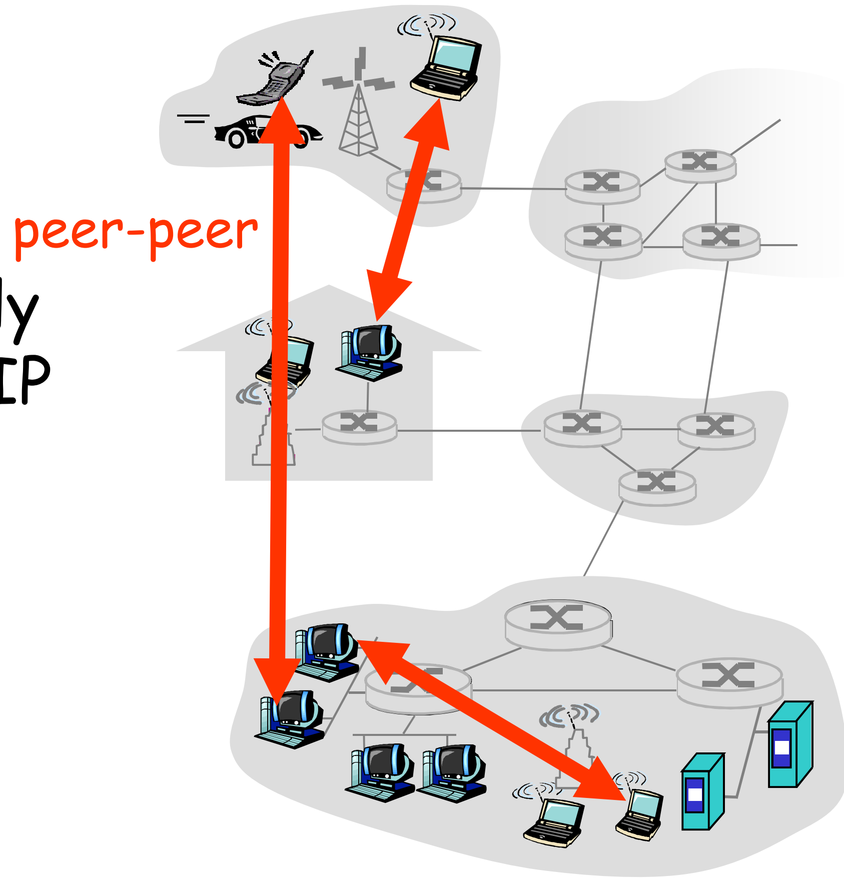
- always-on host
- permanent IP address
- server farms for scaling

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



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

Limiting factors for P2P Apps

- Asymmetrical Links
  - Most residential access networks (e.g., ADSL) provide asymmetrical bandwidth
- Security
  - P2P applications may be a challenge to security, due to their highly distributed and open nature
- Incentives
  - The success of P2P applications depends on convincing users on volunteering bandwidth, storage, CPU and energy resources

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

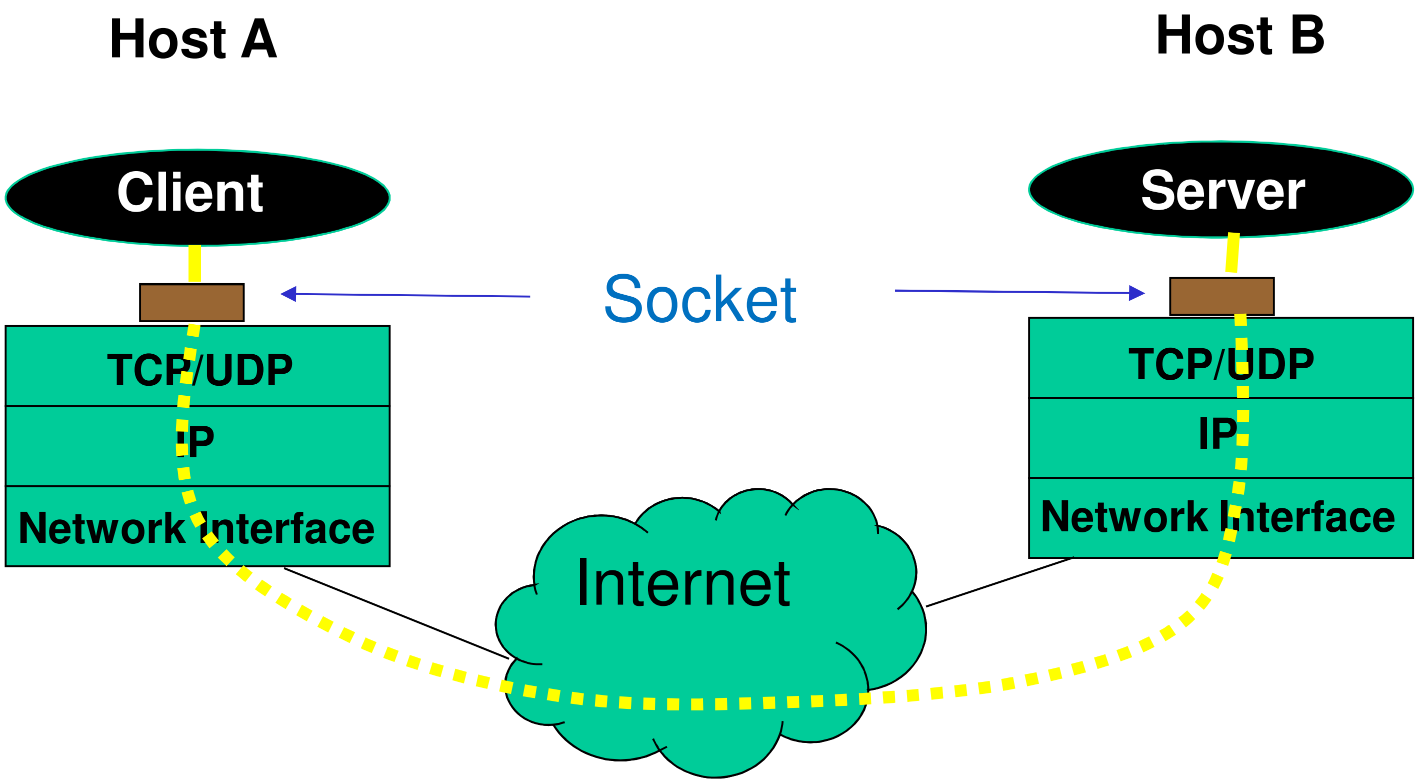
Server process: process that waits to be contacted

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

Application Programming Interface (API)

- Interface between application processes and underlying network (or OS)
  - processes send/receive messages through the API
- Socket-based API
  - The OS provides the socket abstraction for message exchange through the network
- Socket is analogous to mailbox
  - sending process drops the message into the mailbox
  - sending process relies on transport service (i.e., postal service) which brings messages to socket at receiving process
- Socket is also analogous to telephone socket
  - Sending/receiving process corresponds to user+telephone

Socket-based communication



API: (1) choice of transport protocol; (2) ability to fix a few parameters

What transport service does an app need?

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

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

Throughput
- some apps (e.g., multimedia) require minimum amount of throughput ("bandwidth sensitive" apps.)
- other apps ("elastic apps") make use of whatever throughput they get

Security
- Encryption, data integrity, …

Transport service requirements of 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
instant messaging	no loss	elastic	yes and no

Internet transport services

Stream service (TCP):

- *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

Datagram service (UDP):

- 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?

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote login	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

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

Public-domain protocols:

- E.g., defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP, BitTorrent

Proprietary protocols:

- e.g., Skype

Applications vs.
Application-layer Protocols

Addressing Process

- Processes are executed on hosts
  - Each host executes a large number of processes concurrently
  - Processes must be addressed individually for communication taking place
- An Host is identified by its IP address
  - 32-bit sequence
- A Process is identified by its port number
  - 16-bit sequence

Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P applications
  - File Sharing, Internet Telephony
- Socket programming

Web and HTTP

First some jargon

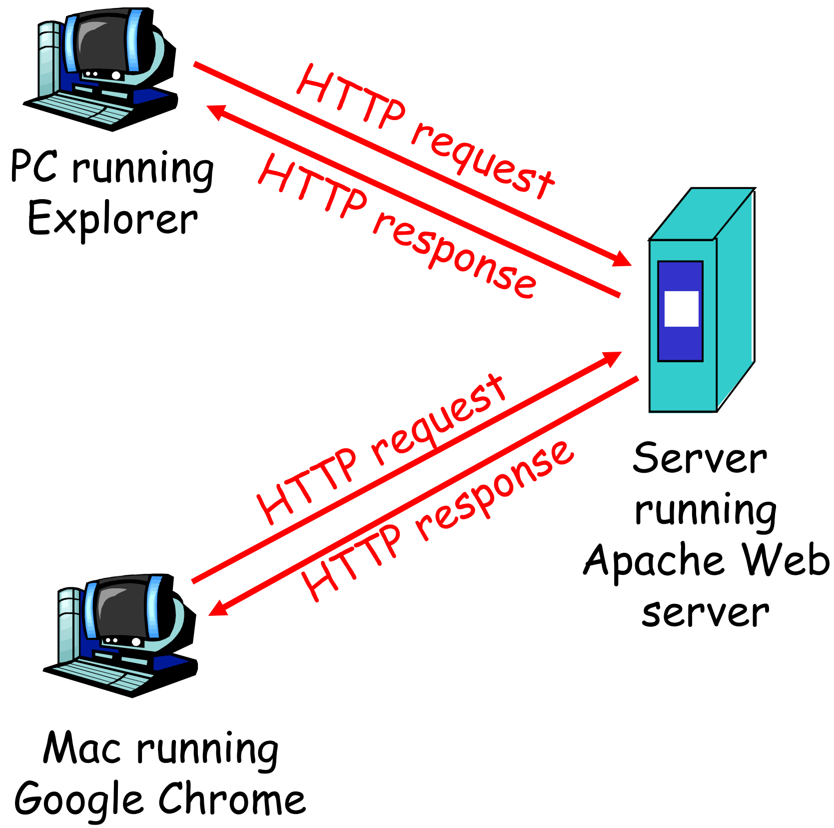
- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL
- Example URL:

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

HTTP overview [RFC 1945, 2616]

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 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.

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.

time

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

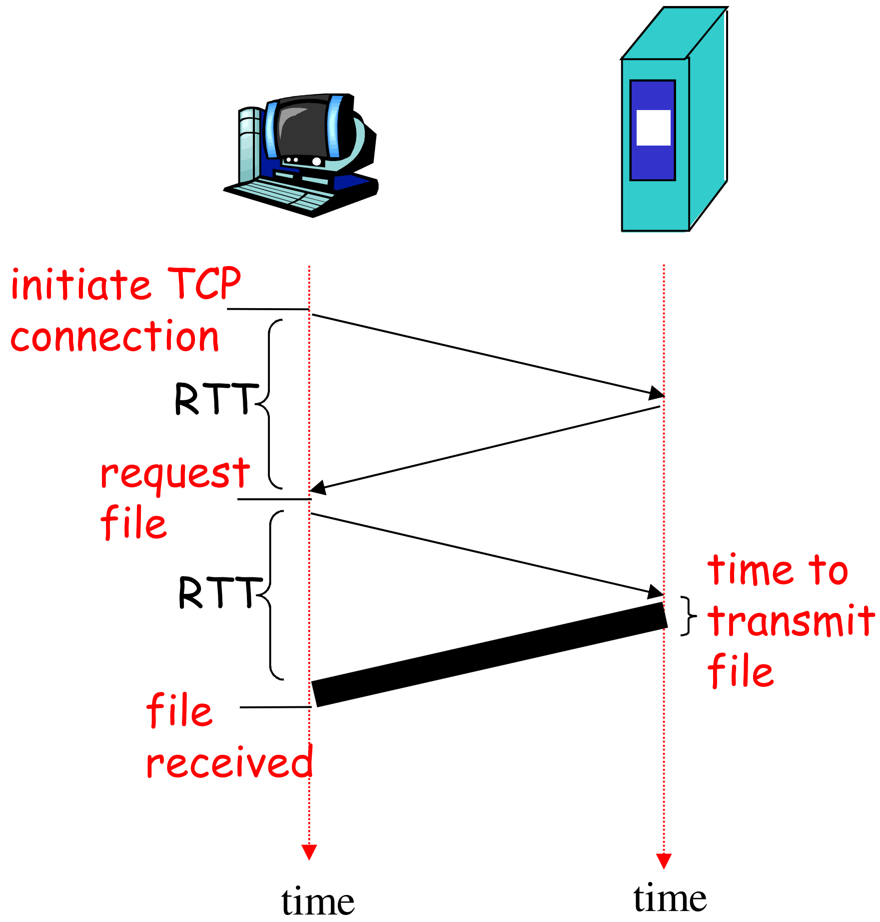
Non-Persistent HTTP: Response time

Definition of RTT: time for 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 on the server 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 [RFC 2616]

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

Suppose user entered URL
`www.someschool.edu/somedir/page.html`

request line (GET, POST, HEAD commands)

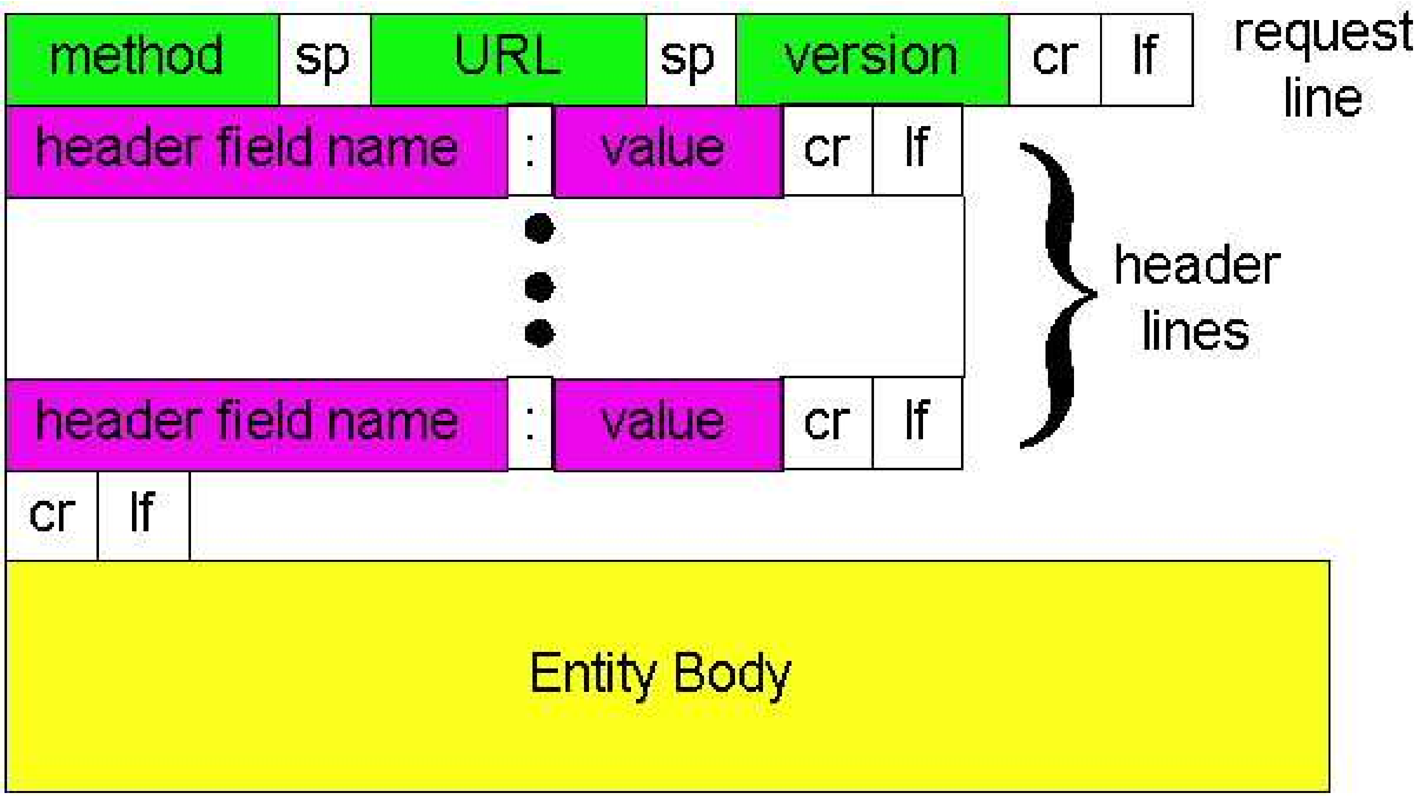
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Firefox/15.0
Connection: close
Accept-language: it
```

header lines

Carriage return, line feed indicates end of message

(extra carriage return, line feed)

HTTP request message: general format



Uploading form input

Post method:

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

GET 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 version
status code
status message)

header
lines

CR + LF

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK
Connection close
Date: Wed, 06 Aug 2013 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Sun, 22 Jun 2013 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

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 www.ing.unipi.it 80
```

Opens TCP connection to port 80 (HTTP server port) at www.ing.unipi.it. Anything typed in sent to port 80 at www.ing.unipi.it

2. Type in a GET HTTP request:

```
GET index.html HTTP/1.1
Host: www.ing.unipi.it
```

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
[RFC 2965]

Almost all Web sites use cookies

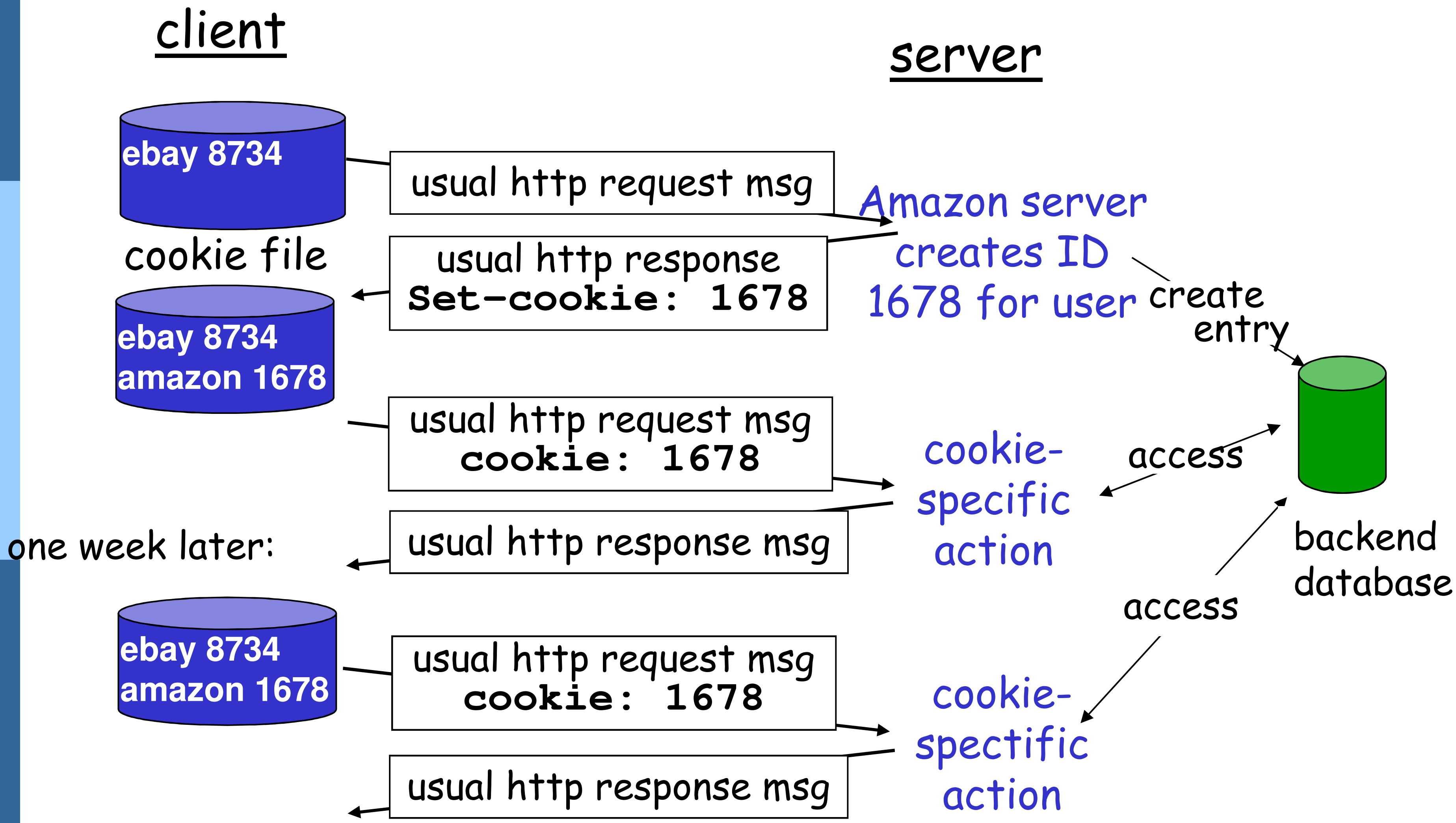
Four components:

1) cookie header line of HTTP *response* message
2) cookie header line in 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 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

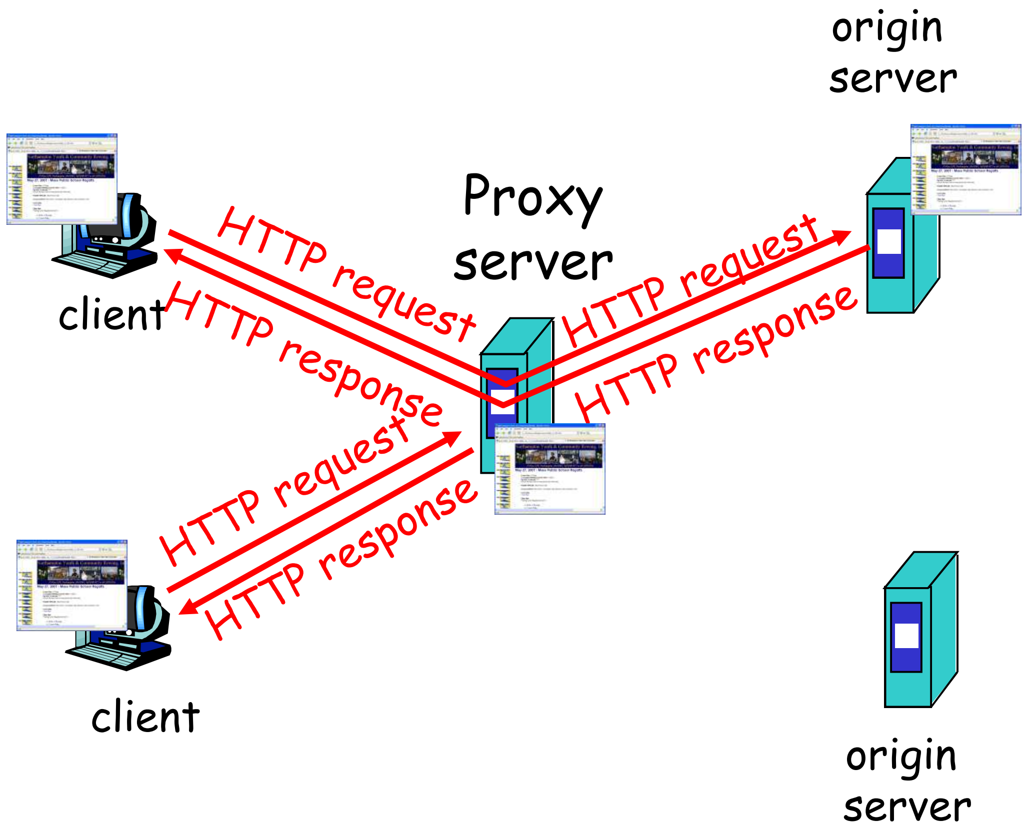
How to keep "state":

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

Caching example

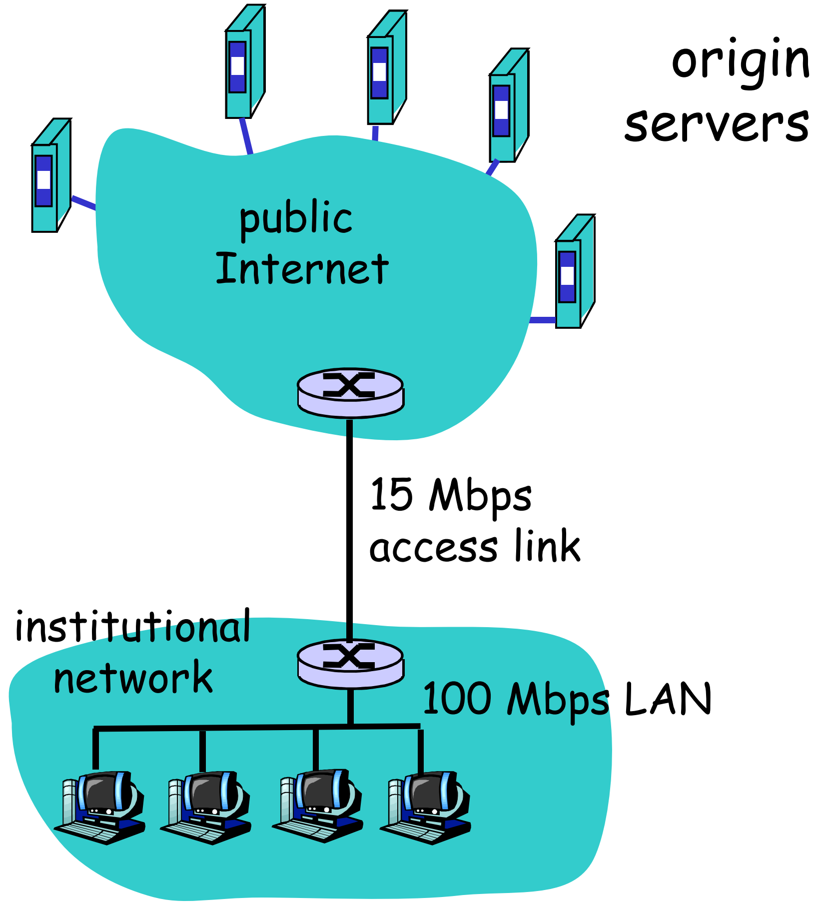
Assumptions

- average object size = 1 Mbit
- avg. request rate from institution's browsers to origin servers = 15/sec
- avg. Internet delay (from the access router on the Internet to any origin server and back) = 2 sec

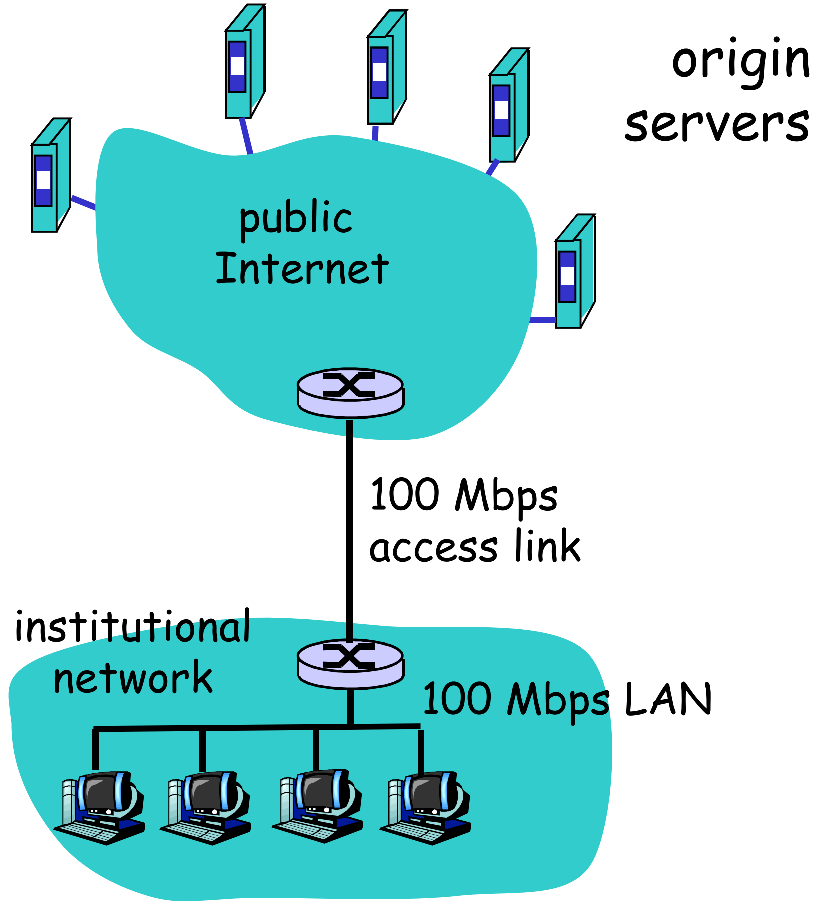
Consequences

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

= 2 sec + minutes + milliseconds



Caching example (cont)



possible solution

- increase bandwidth of access link to, say, 100 Mbps

consequence

- 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

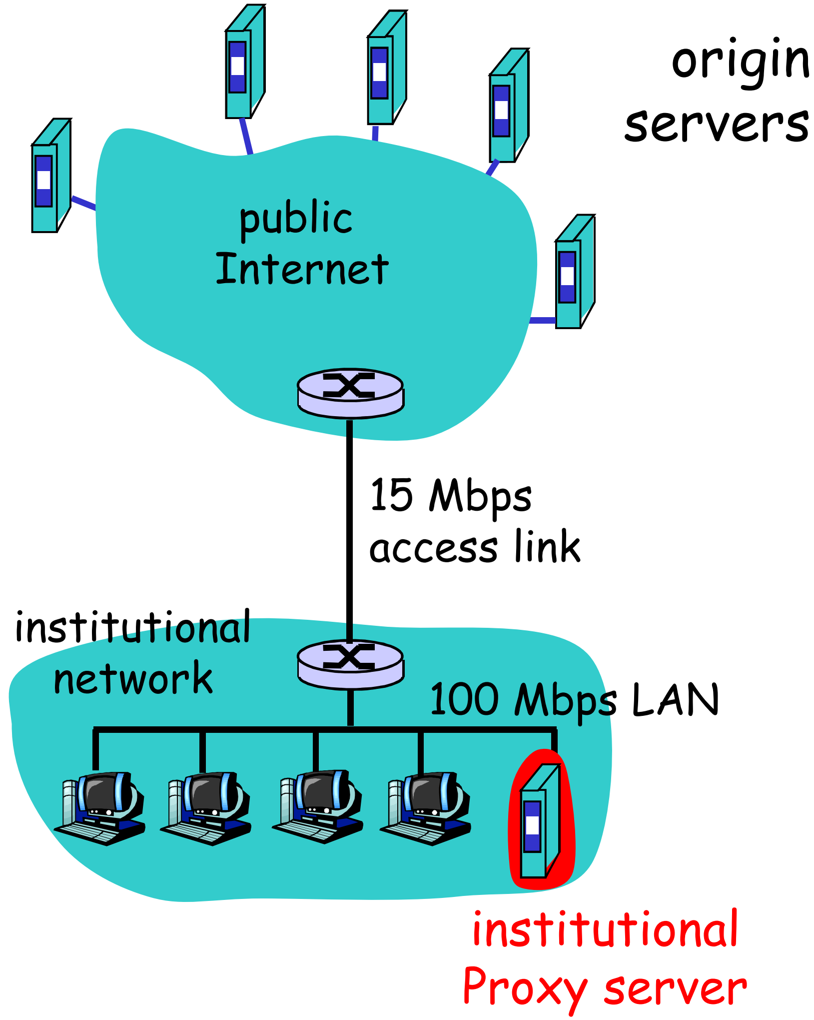
Caching example (cont)

Alternative solution:

- install a proxy server
- suppose hit rate is 0.4

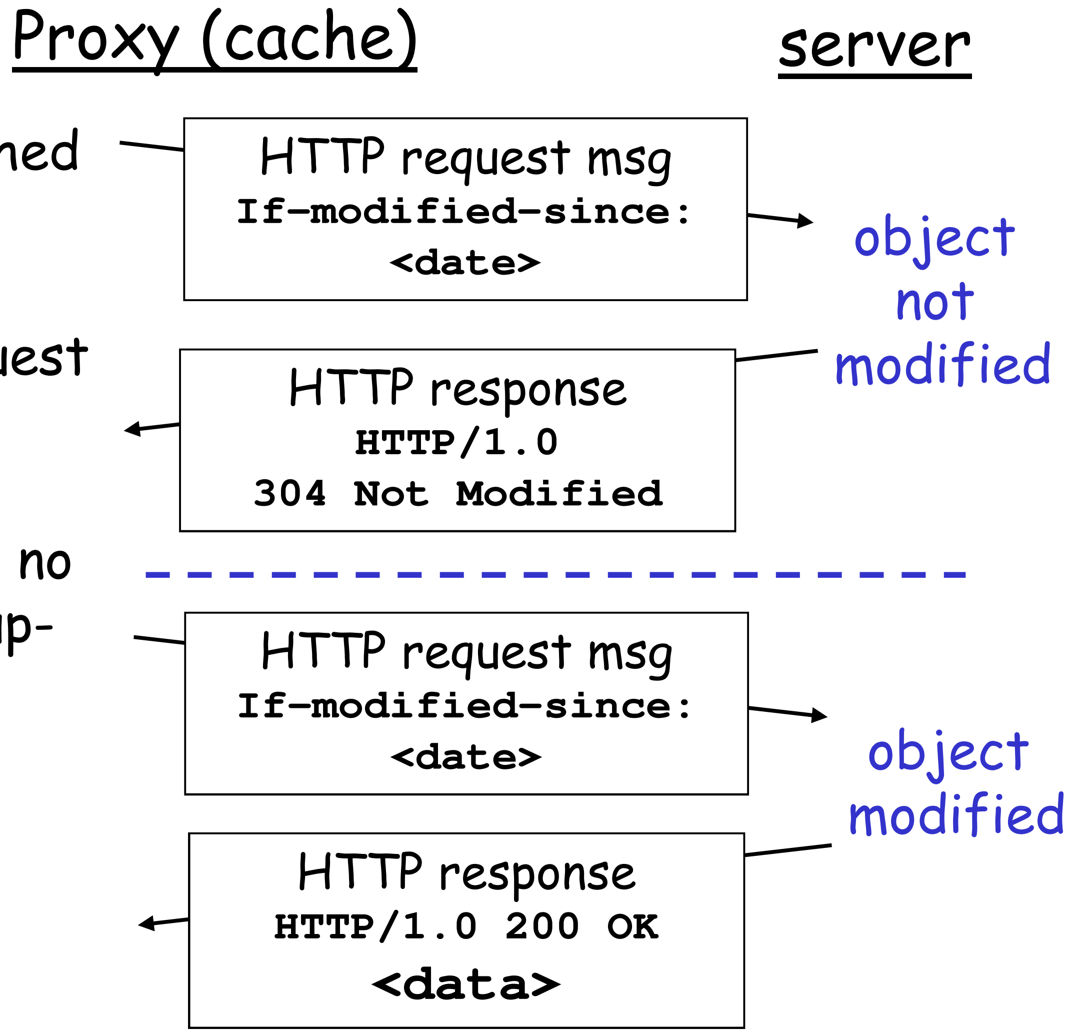
consequence

- 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 ms)
- total avg delay = Internet delay + access delay + LAN delay = 0.6*2.01 s + 0.4*0.01 s = 1.21 s



Conditional GET

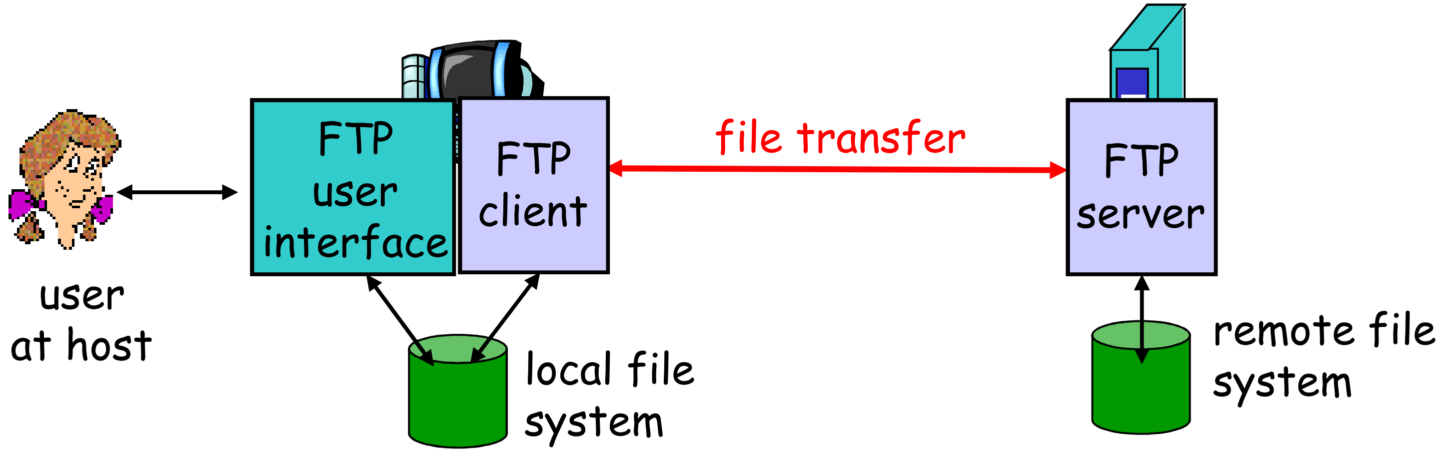
- **Goal:** don't send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request
  `If-modified-since: <date>`
- server: response contains no object if cached copy is up-to-date:
  `HTTP/1.0 304 Not Modified`



Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P applications
  - File Sharing, Internet Telephony
- Socket programming

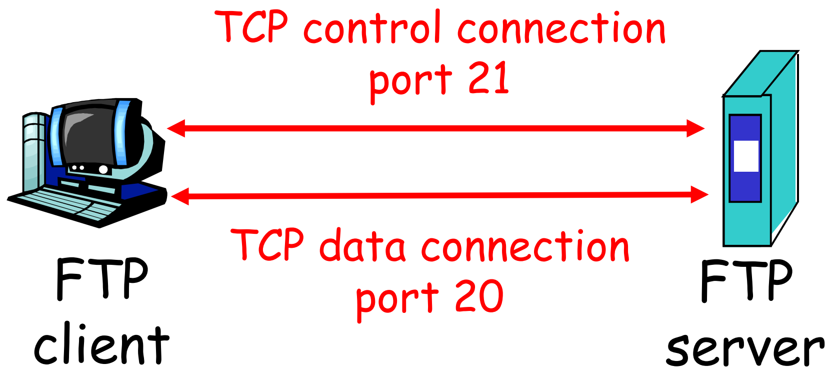
FTP: the file transfer protocol



- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp server: port 21 (port 22 for SFTP)
- ftp: RFC 959

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, TCP is transport protocol
- client authorized over control connection
- client browses remote directory by sending commands over control connection.
- when server receives file transfer command, server opens 2nd TCP connection (for file) to client
- after transferring one file, server closes data connection.



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

FTP commands, responses

Sample commands:

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

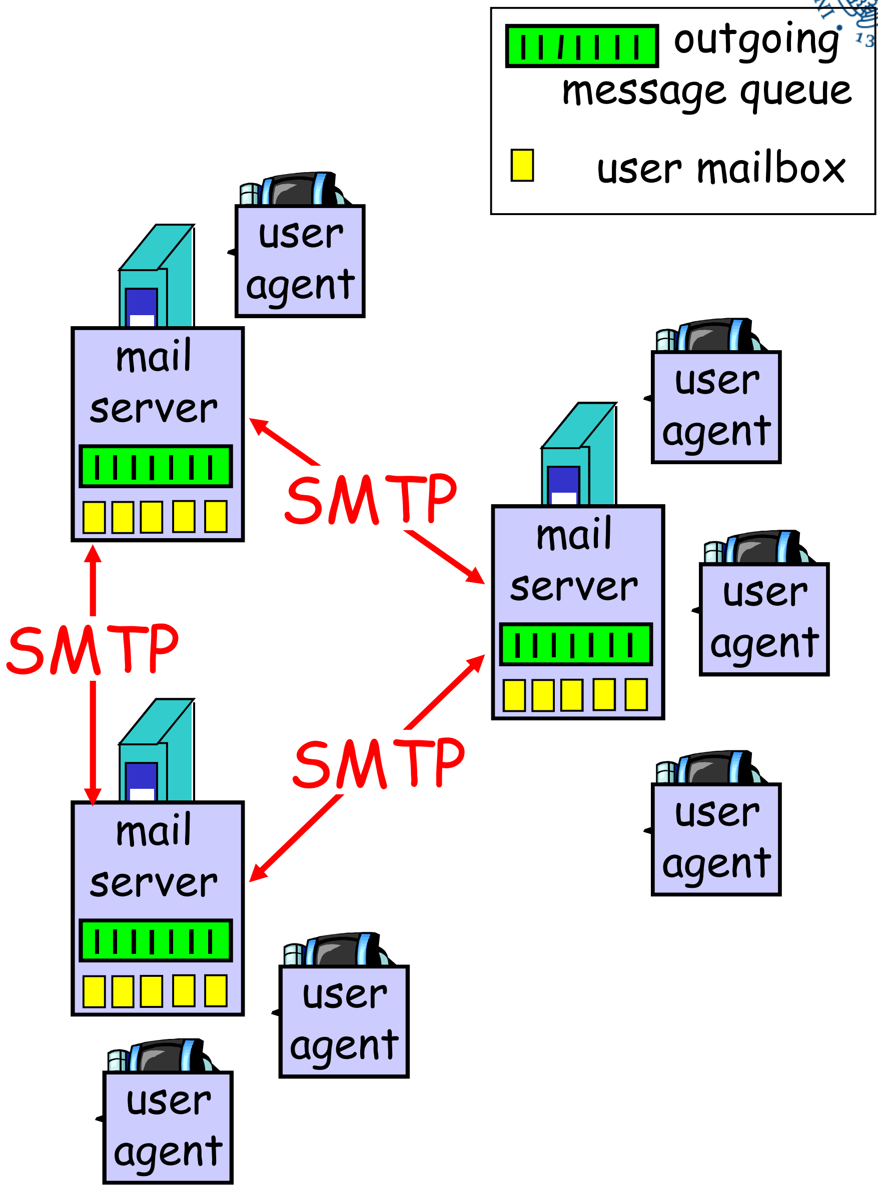
Sample return codes

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

Network Applications

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail
    - SMTP, POP3, IMAP
- DNS
- P2P applications
    - File Sharing, Internet Telephony
- Socket programming

Electronic Mail



Three major components:

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

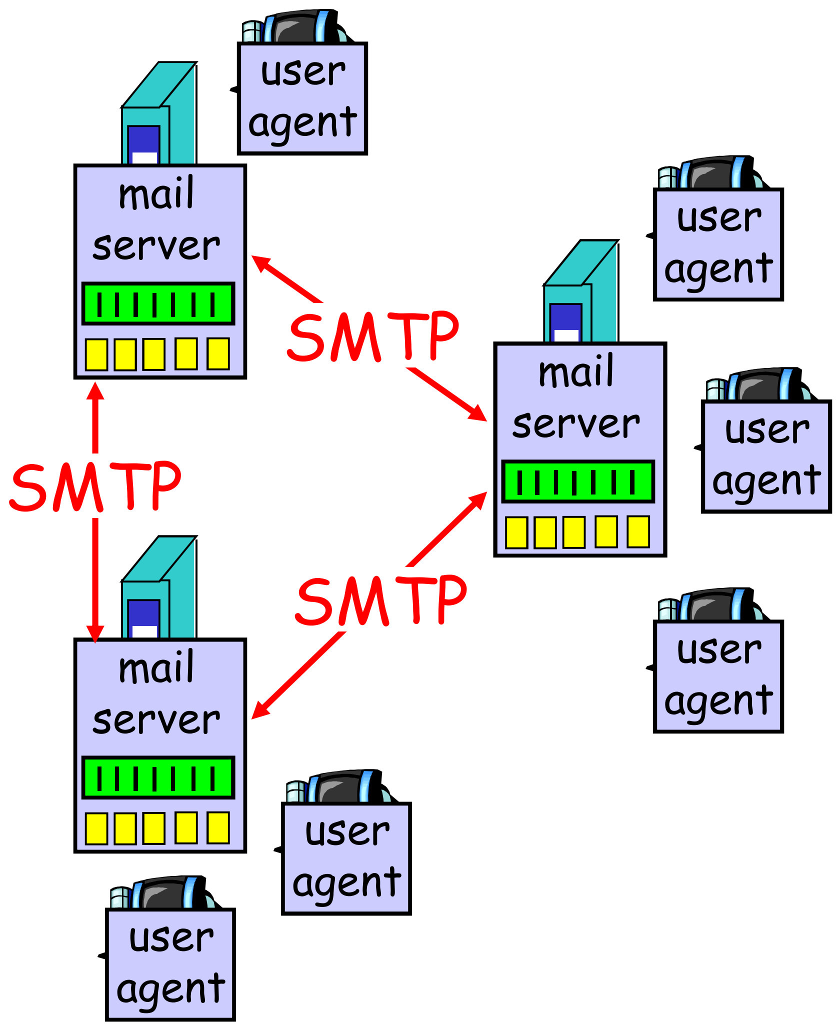
User Agent

- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Mozilla Thunderbird
- 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

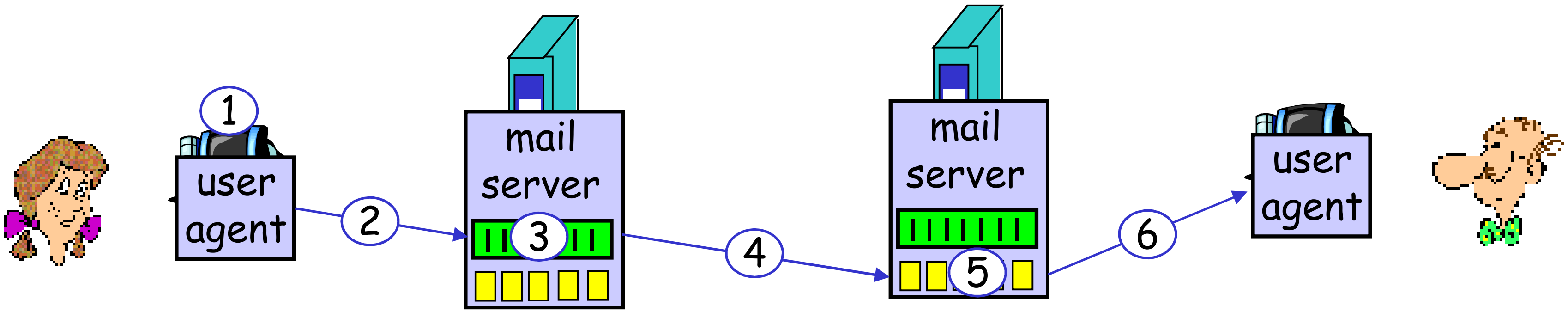


Electronic Mail: SMTP [RFC 2821, 5321]

- 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 send it 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) After some initial handshake, 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 smtp.unipi.it 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)

Comparison with HTTP

- SMTP uses persistent connections, like persistent HTTP
- both have ASCII command/response interaction, status codes
  - HTTP: pull
  - SMTP: push
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses `CRLF.CRLF` to determine end of message
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

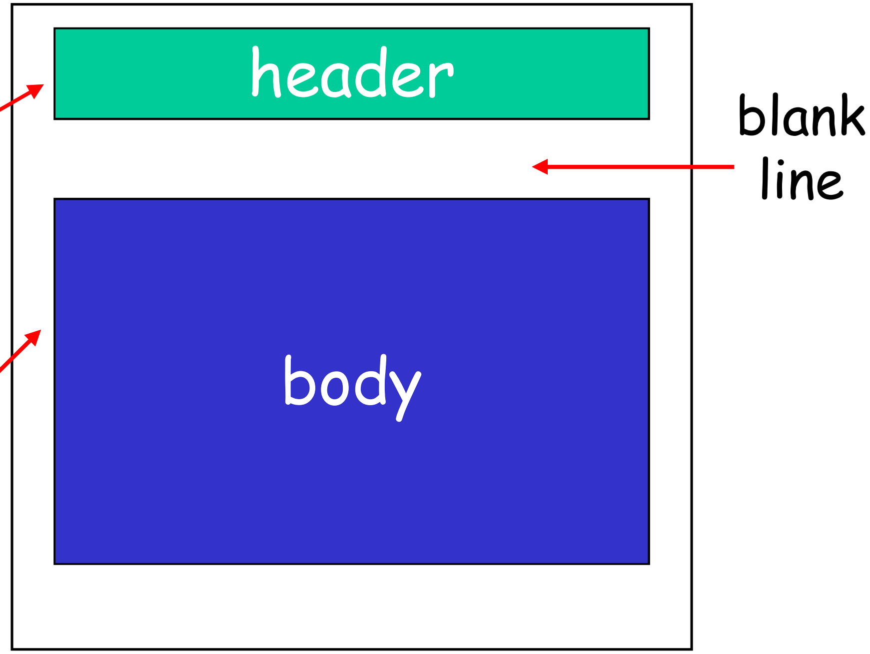
Mail message format

SMTP: protocol for exchanging email msgs

RFC 5322: standard for text message format:

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

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



MIME Extensions (RFC 2045, 2046]

- Message headers defined in RFC 5322 are not sufficient for multimedia contents and non ASCII text
- Multipurpose Internet Mail Extensions (MIME) defines additional headers
  - Content-Transfer-Encoding:
    - Indicates the encoding technique that was used (e.g., base64 or quoted-printed content-transfer encoding)
  - Content-Type:
    - Allows the receiving user agent to take an appropriate action on the message (e.g., send the message to a decompression routine)

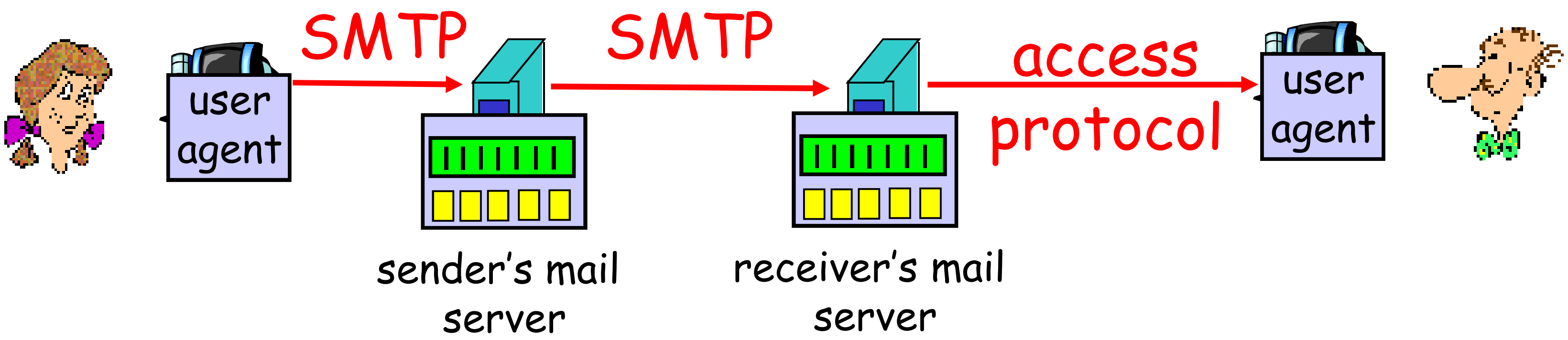
MIME Extensions (RFC 2045, 2046]

- Alice sends a JPEG image to Bob
  - E.g., as an attachment

From: alice@crepes.fr
To: bob@hamburger.edu
Subject: A picture of mine
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

(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 [RFC 1939]

authorization phase

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

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

transaction phase, client:

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

```
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
```

Try POP3 interaction for yourself:

- `telnet mailbox.unipi.it 110`
- `+ OK mail server ready`
- `User anastasi`
- `+ OK`
- `Pass xxxxxxx`
- `-ERR authentication failed`
- `Quit`
- `+OK Logging out`

POP3 (more)

- 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 [RFC 3501]

- Mail Access Protocol
  - like POP3, but more complex
- 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

Webmail

- The user agent is a browser
- Browser-Mail Server Communications through HTTP
  - Messages from the mail server to the browser are sent through HTTP, instead of POP3 or IMAP
  - Messages from the browser to the mail server are sent through HTTP, instead of SMTP

Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P applications
  - Content Search and Location, File Sharing, Internet Telephony
- Socket programming

How to identify Internet nodes

People: many identifiers:

- name, SSN, passport #

Internet hosts, routers:

- hostname, e.g., www.yahoo.com - used by humans
- IP address (32 bit) - used for addressing datagrams

Q: map between hostnames and IP addresses?

DNS: Domain Name System
[RFC 1034, 1035, and subsequent updates]

- distributed database
  - implemented in hierarchy of many *name servers (DNS servers)*
- application-layer protocol
  - Used by hosts, routers, name servers to query the distributed database and *resolve* names
  - core Internet function, implemented as application-layer protocol (complexity at network's "edge")
  - DNS runs over UDP and used port 53
  - DNS servers are often UNIX machine with Berkeley Internet Name Domain (BIND) software

DNS Services

- hostname to IP address translation
  - Often used by other protocols
    - HTTP, FTP, SMTP
- host aliasing
  - Canonical, alias names
  - Example www.iet.unipi.it, canonical name: info.iet.unipi.it
- mail server aliasing
  - bob@hotmal.com (mail server: server-1.hotmail.com)
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

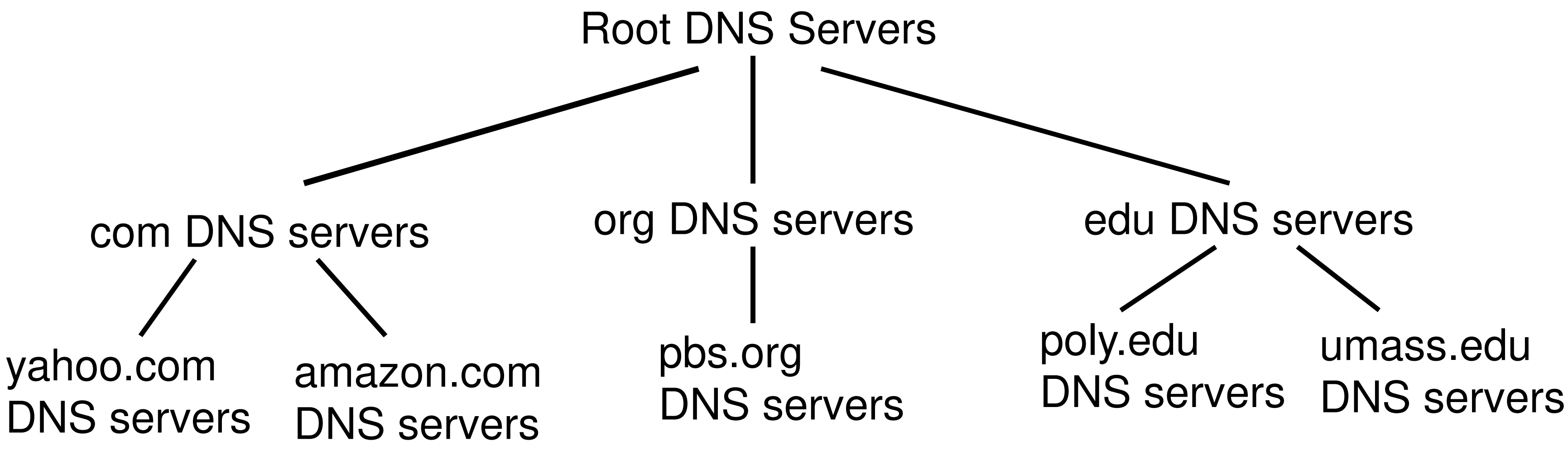
Key question

Why not a centralized DNS?

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

doesn't *scale!*

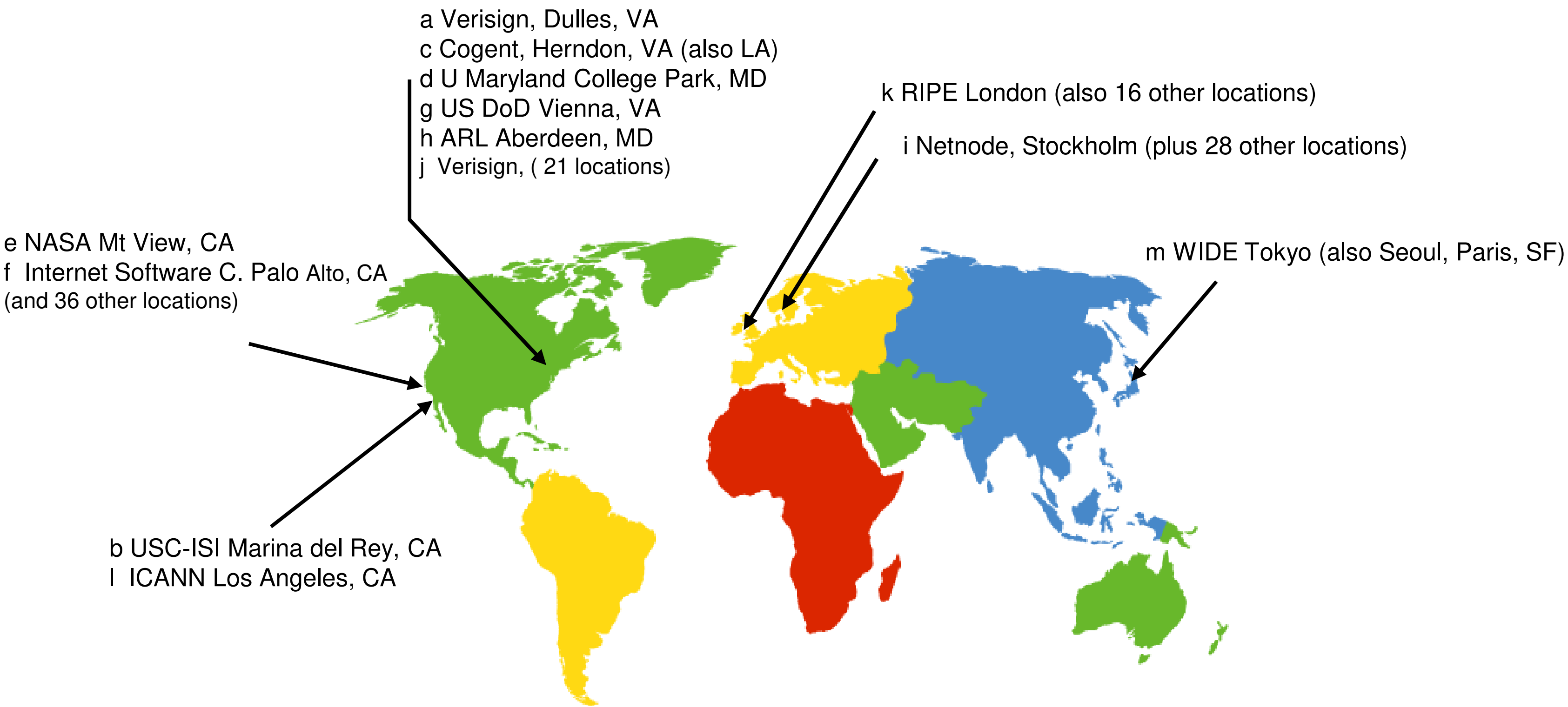
Distributed, Hierarchical Database



Client wants IP for www.amazon.com (1st approximation)

- 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



13 root name servers worldwide

Complete list available at: www.root-servers.org

TLD Servers

- responsible for com, org, net, edu, etc, and all country top-level domains such as it, uk, fr, ca, jp, …
  - Verisign Global Registry Services maintains servers for com TLD
  - Educause for edu TLD
  - Registro .IT (Pisa) maintains servers for .it TLCD
    - Located at IIT-CNR, Pisa

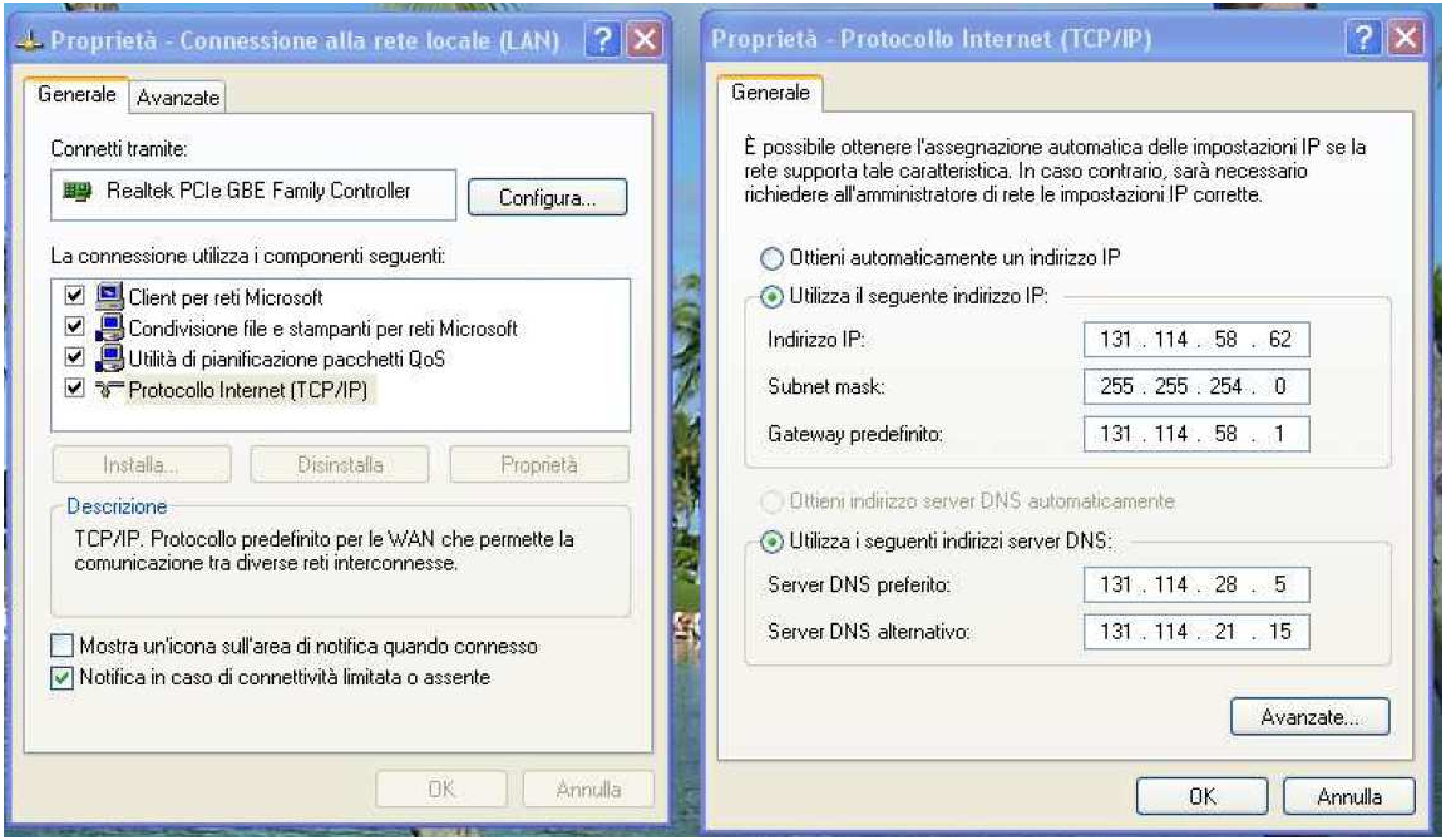
Authoritative Servers

- 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 DNS Server (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

Local DNS Server

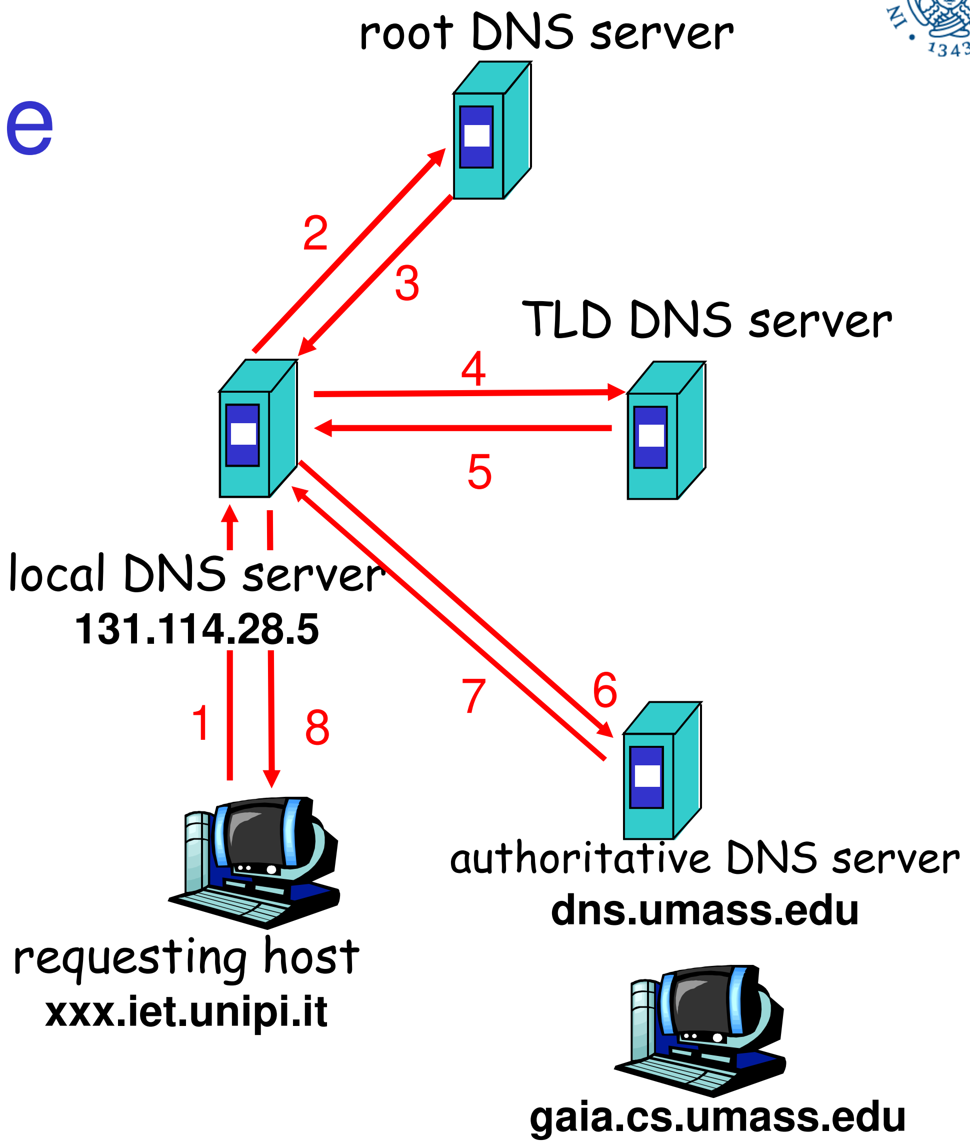


DNS name resolution example

- Host at xxx.iet.unipi.it wants IP address for gaia.cs.umass.edu

iterated query:

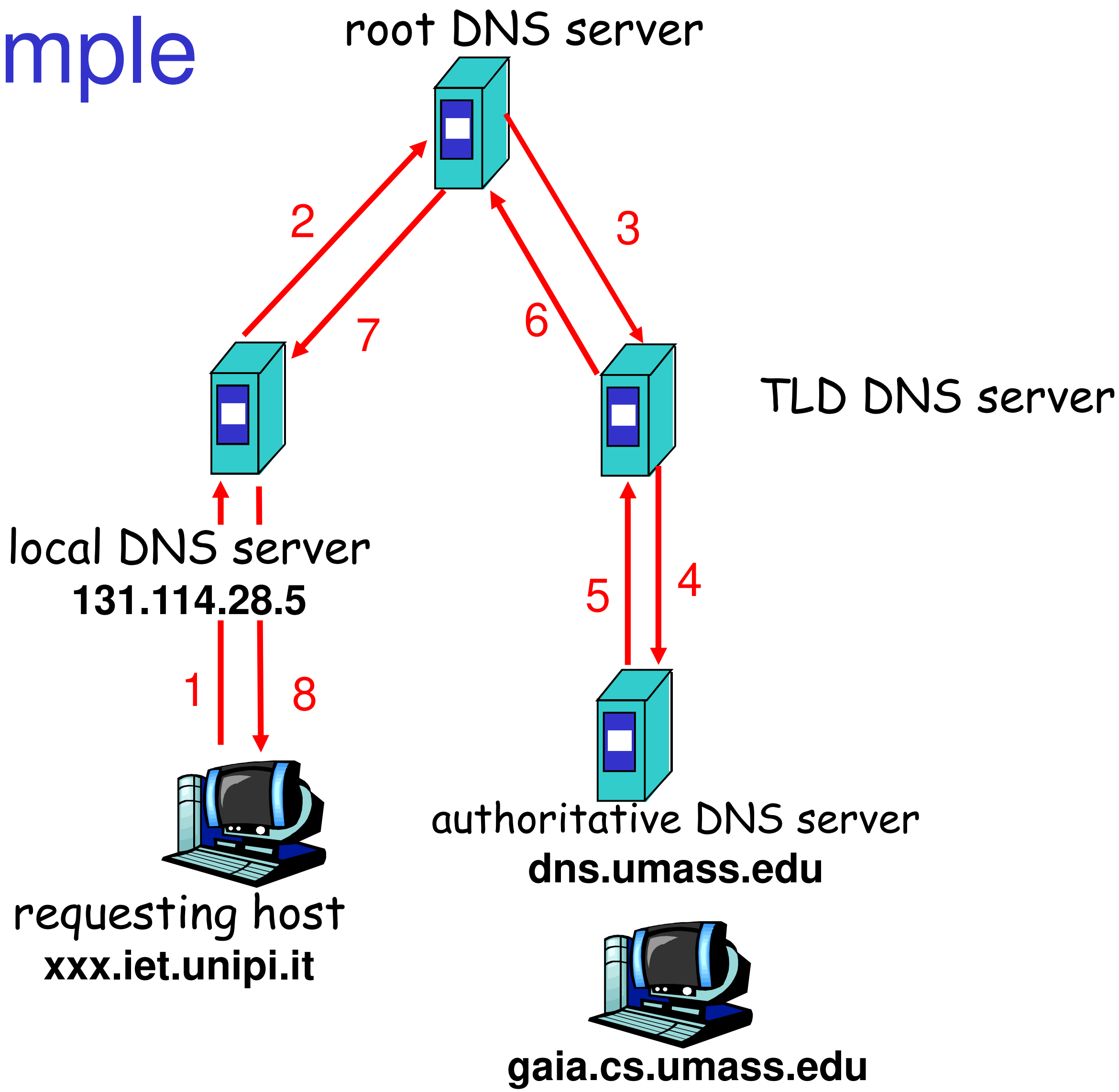
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

<u>recursive query:</u>

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



DNS: caching and updating records

- Once (any) name server learns mapping, it *caches* mapping
  - Reduced delays
  - Reduced amount of Internet traffic
  - cache entries timeout (disappear) after some time
    - The timeout is often set to 2 days
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited

DNS records [RFC 1034, 1035]

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 a domain (e.g. hal.com)
  - **value** is hostname of authoritative DNS server for that domain
  - (hal.com, dns.hal.com, NS)
- Type=CNAME
  - **name** is the alias name for some "canonical" (i.e., real) name, e.g., www.hal.com
  - **value** is canonical name, e.g., servereast.backup2.hal.com
- Type=MX
  - **value** is the canonical name of mailserver associated with **name**
  - (hal.com, mail.server1.hal.com, MX)

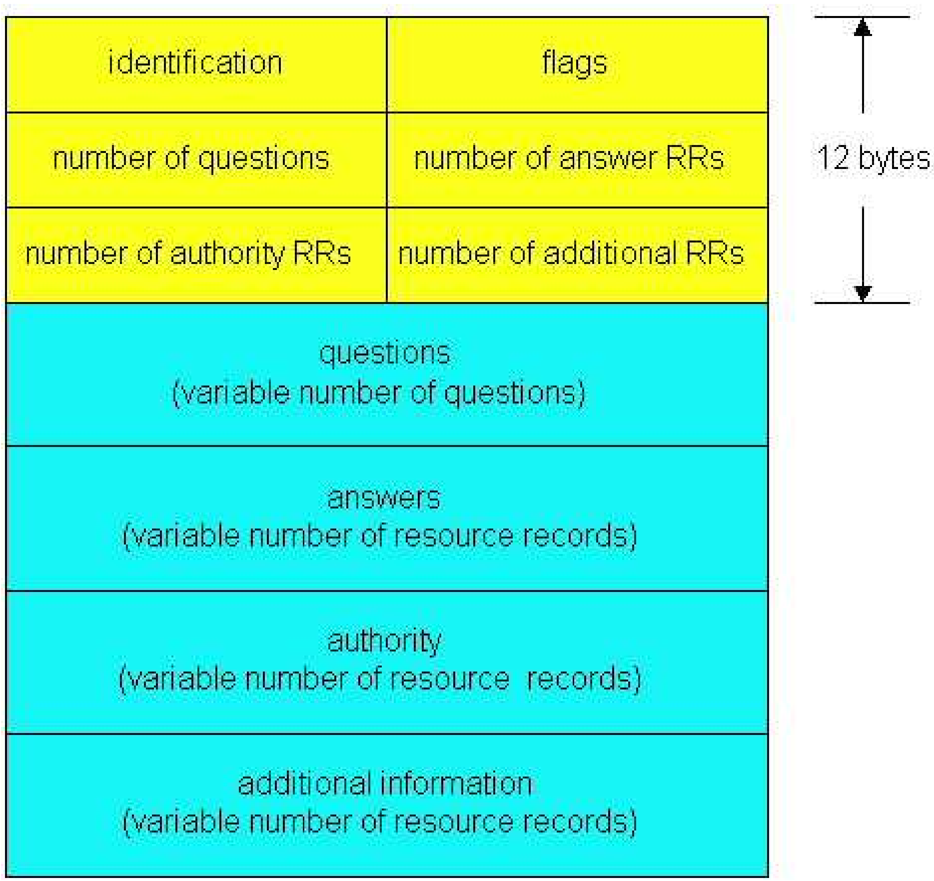
DNS records

- Example 1
  - dns.umass.edu **is authoritative** for gaia.cs.umass.edu
  - dns.umass.edu **contains** the following record

    (gaia.cs.umass.edu, 128.119.40.205, *A*)
- Example 2
  - The edu TLD server **is not authoritative** for gaia.cs.umass.edu
  - It **doesn't contain** a record for gaia.cs.umass.edu
  - It **does contain** the following two records
    - (umass.edu, dns.umass.edu, NS)
    - (dns.umass.edu, 128.119.40.111, *A*)

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 (0/1)
  - recursion desired (query)
  - recursion available (reply)
  - server is authoritative for the queried name (reply)

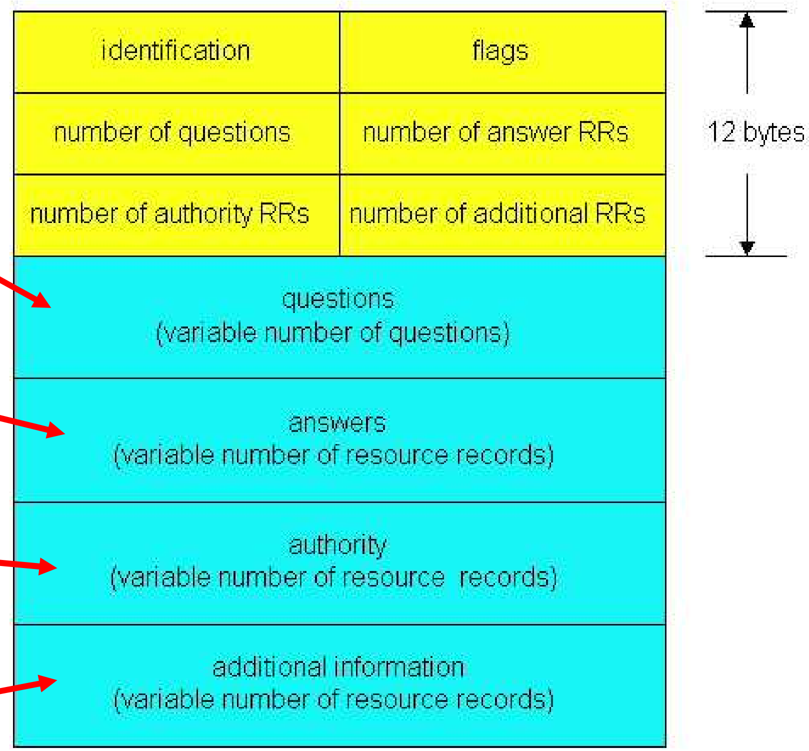
DNS protocol, messages

Name, type fields for a query

RRs in response to query

records for other authoritative servers

additional "helpful" info that may be used



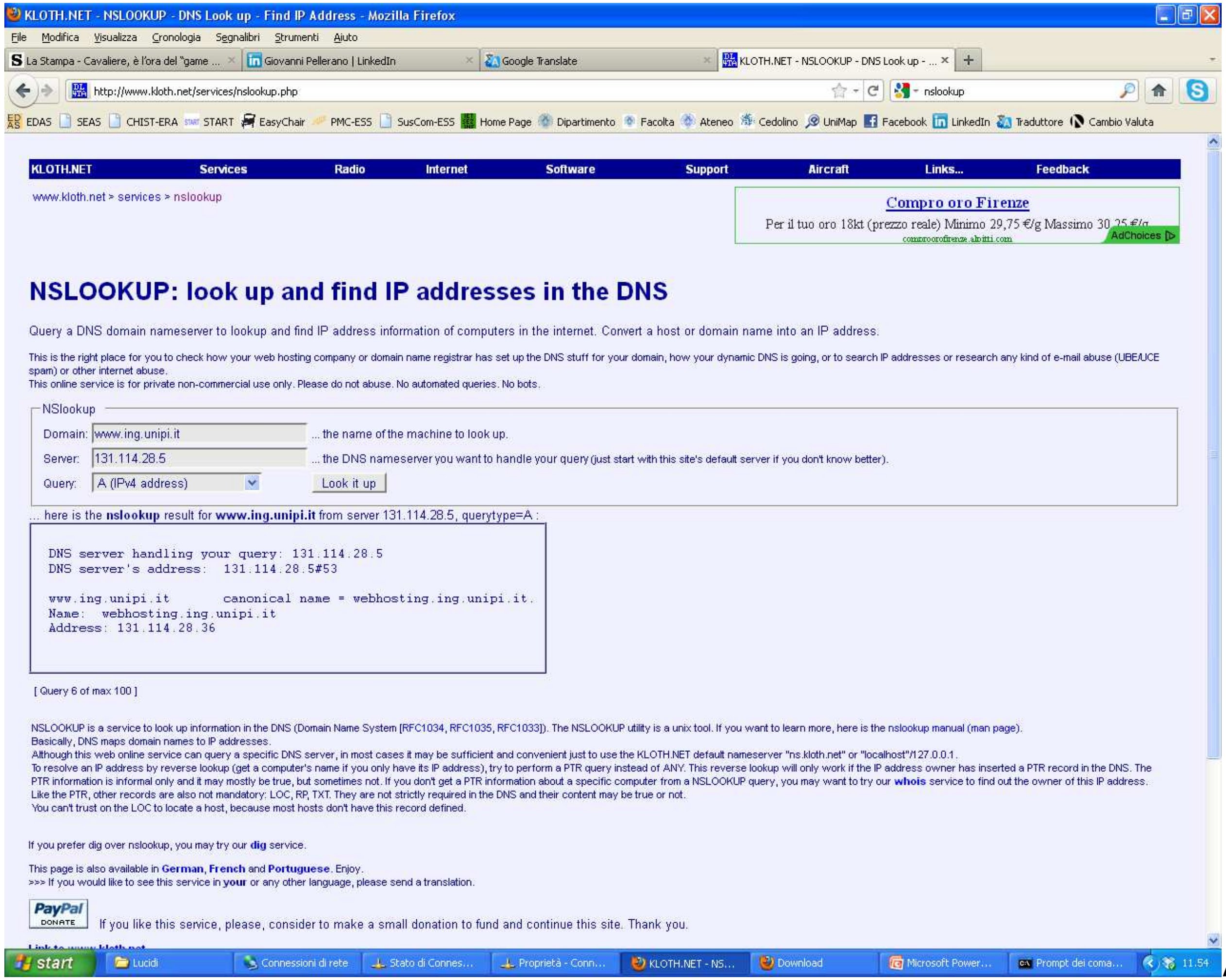
nslookup

Prompt dei comandi - nslookup

```
Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Giuseppe>nslookup
Server predefinito:  ns1.ing.unipi.it
Address:  131.114.28.5

> www.ing.unipi.it
Server:  ns1.ing.unipi.it
Address:  131.114.28.5

Nome:    webhosting.ing.unipi.it
Address:  131.114.28.36
Aliases:  www.ing.unipi.it

>
```

nslookup

- http://www.kloth.net/services/nslookup.php



Inserting records into DNS

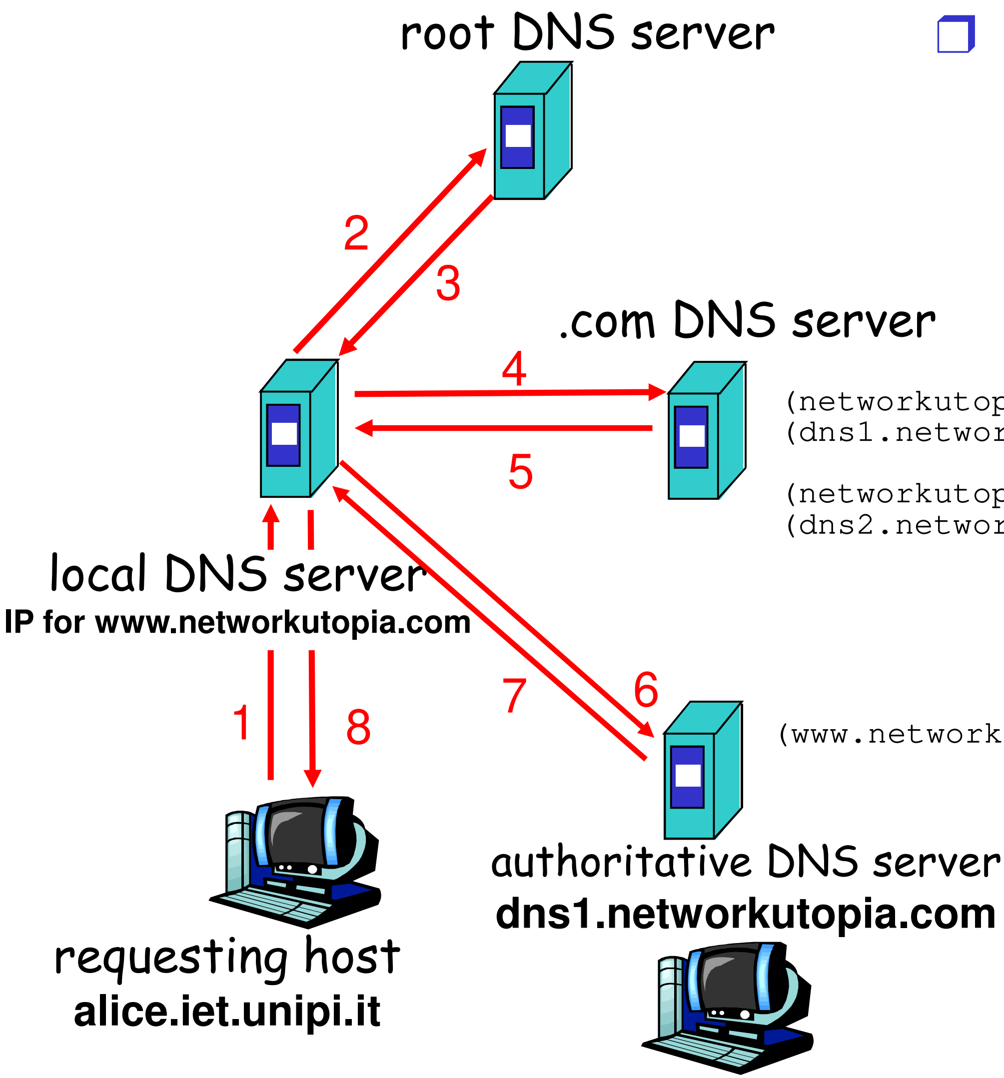
Example: new startup “Network Utopia” founded

- register name networkutopia.com at *DNS registrar*
  - (e.g., Network Solutions, see www.internic.net for a list)
  - 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)
```

- Then, insert in the authoritative DNS server
  - Type A record for www.networkuptopia.com;
  - Type MX record for networkutopia.com
- How do people get IP address of your Web site?

How do people get IP address of your Web site?



- Host at alice.iet.unipi.it wants to visit www.networkutopia.com

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

(networkutopia.com, dns2.networkutopia.com, NS)
(dns2.networkutopia.com, 212.212.212.2, A)
```

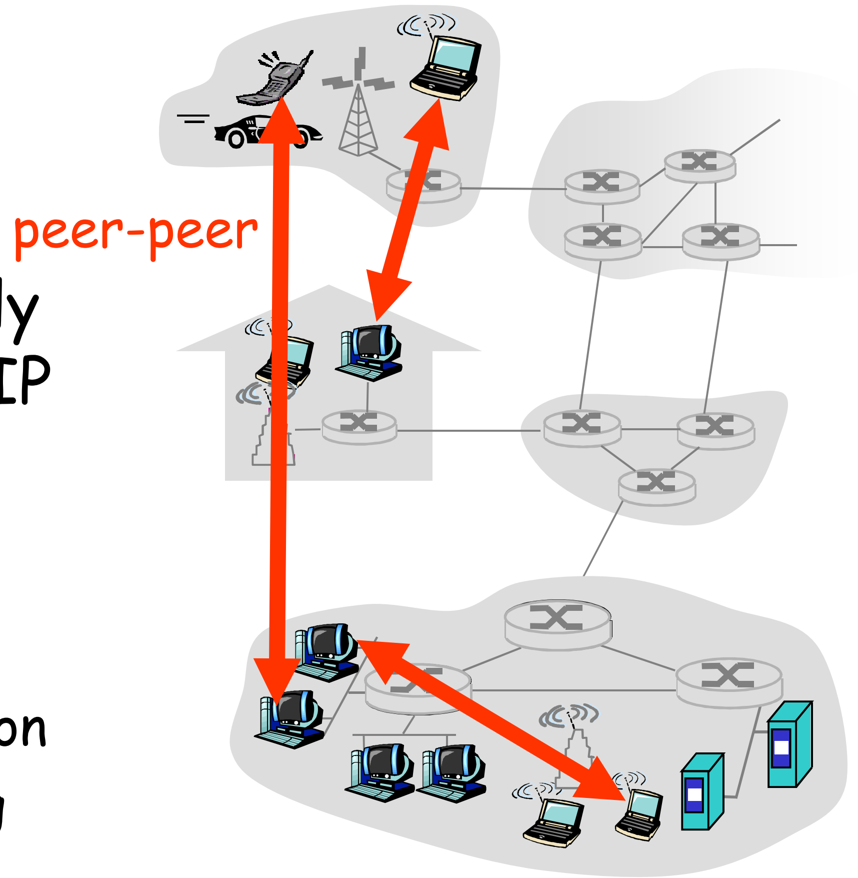
```
(www.networkutopia.com, 212.212.212.4, A)
```

Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P applications
  - Content Search/Location, File Distribution, Internet Telephony
- Socket programming

Pure P2P architecture

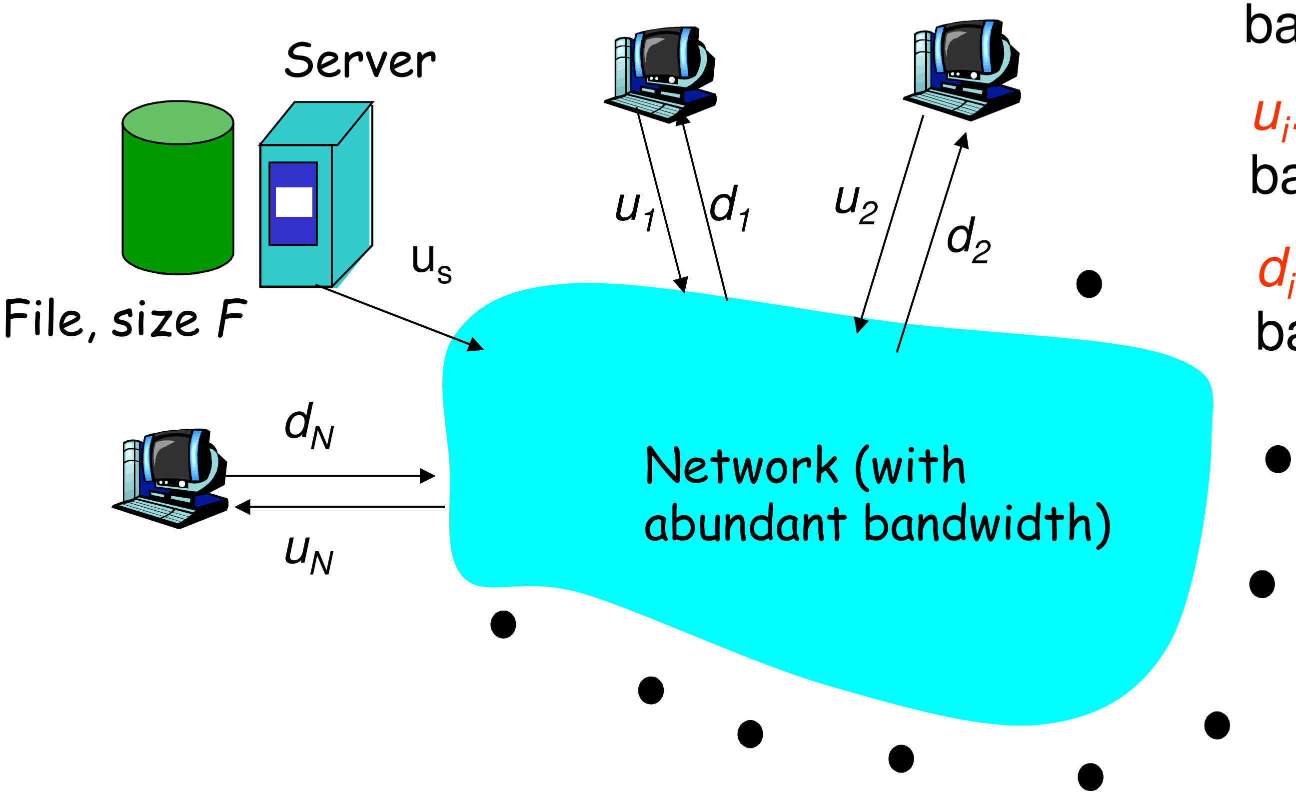
- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses



- <u>Three topics:</u>
  - Searching for information
  - File distribution/sharing
  - Internet Telephony

File Distribution: Server-Client vs P2P

<u>*Question*</u> : How much time to distribute file from one server to *N* *peers*?



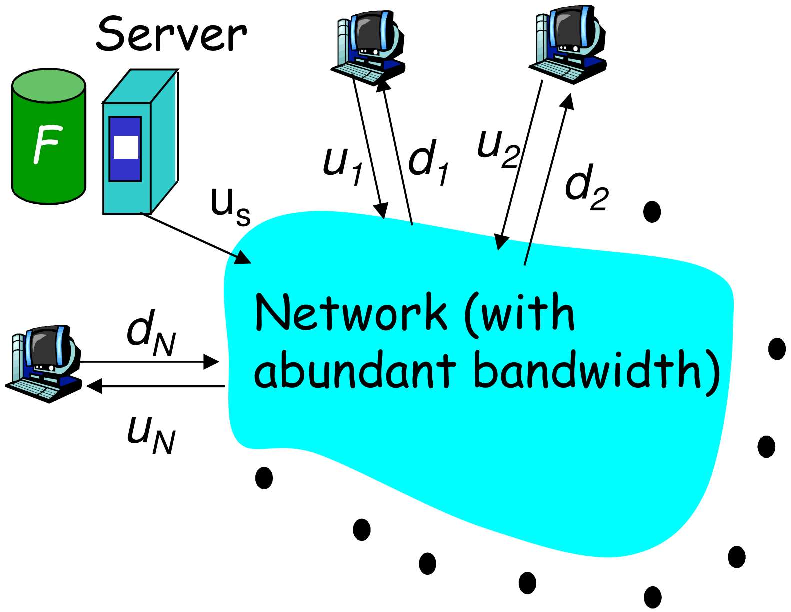
u_s: server upload bandwidth

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

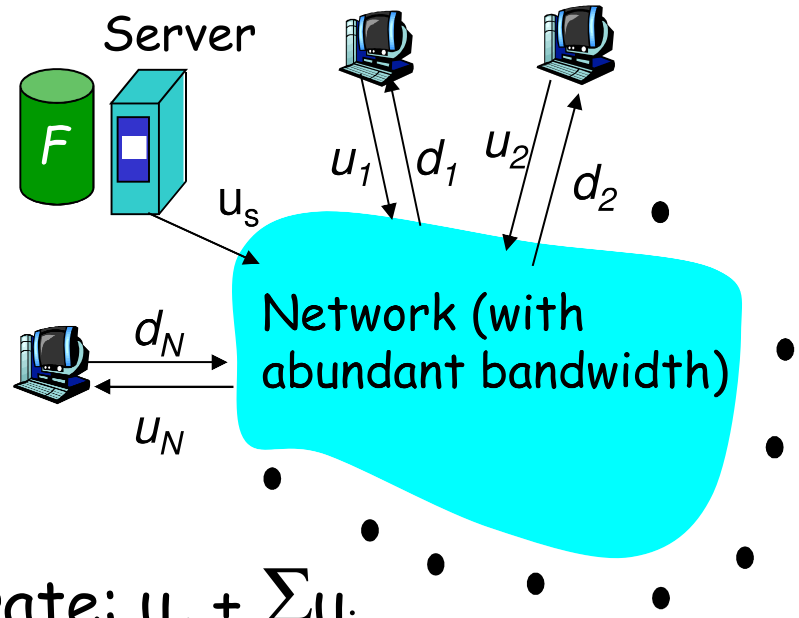


Time to distribute F to N clients using client/server approach $= d_{cs} = \max \{ NF/u_s, F/\min_i(d_i) \}$

increases linearly in N (for large N)

File distribution time: P2P

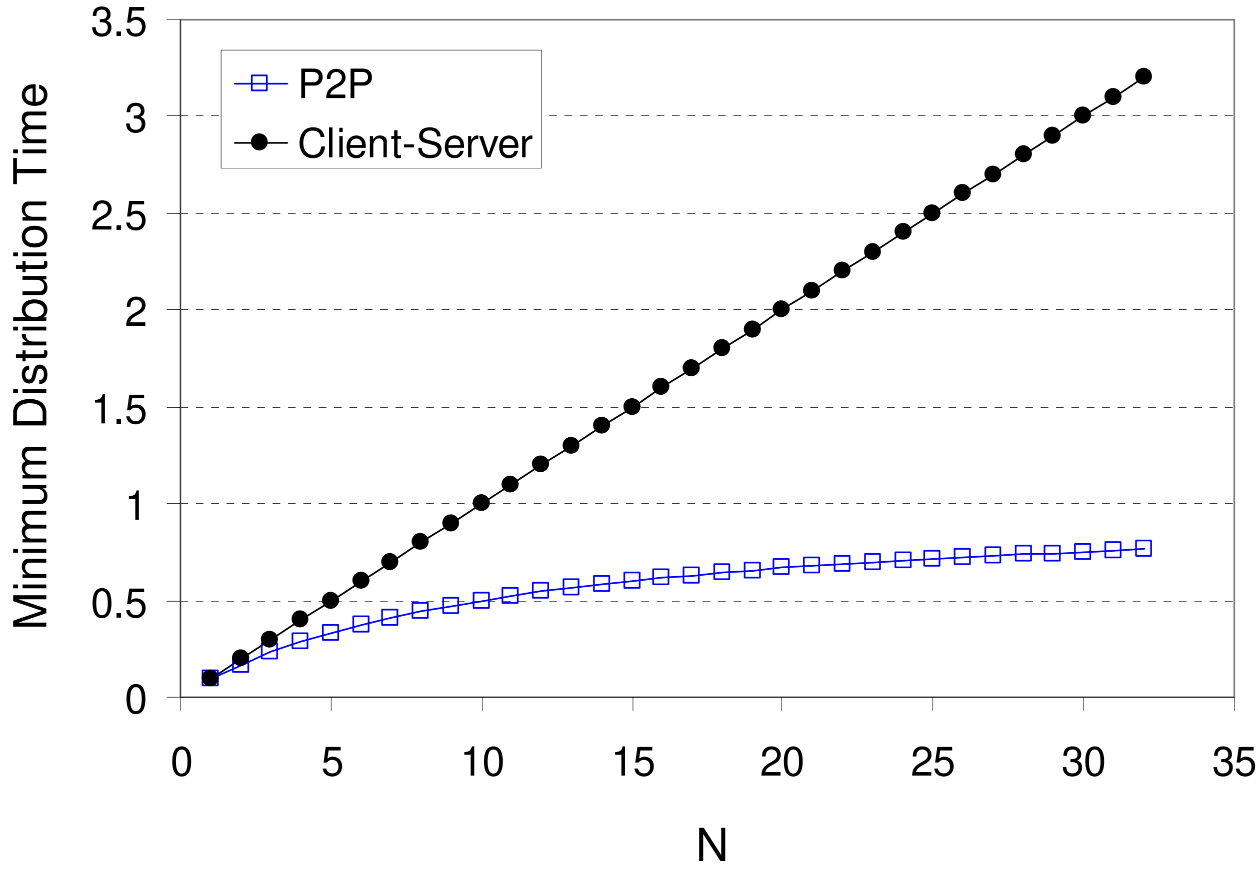
- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \Sigma u_i$



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

Server-client vs. P2P: example

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



Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P applications
  - Content Search/Location, File Distribution, Internet Telephony
- Socket programming

Searching for Information

- P2P File Sharing
  - Peers make files available to share (songs, movies, ..)
  - Interested peers need to know where a desired file can be downloaded from
- Instant Messaging
  - Usernames have to be mapped to IP addresses
  - When user A goes online the index is notified with A's IP address
  - When user B starts the client it searches the index and learns that A is online at a certain IP address
- Indexes are needed
  - Mapping of information to host locations
  - Search and Update Operations

Content Index

- Database with (key, value) pairs;
  - key: content type; value: IP address
  - E.g., (Led Zeppelin IV, 203.17.123.38)
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) pairs

Index Design Approaches

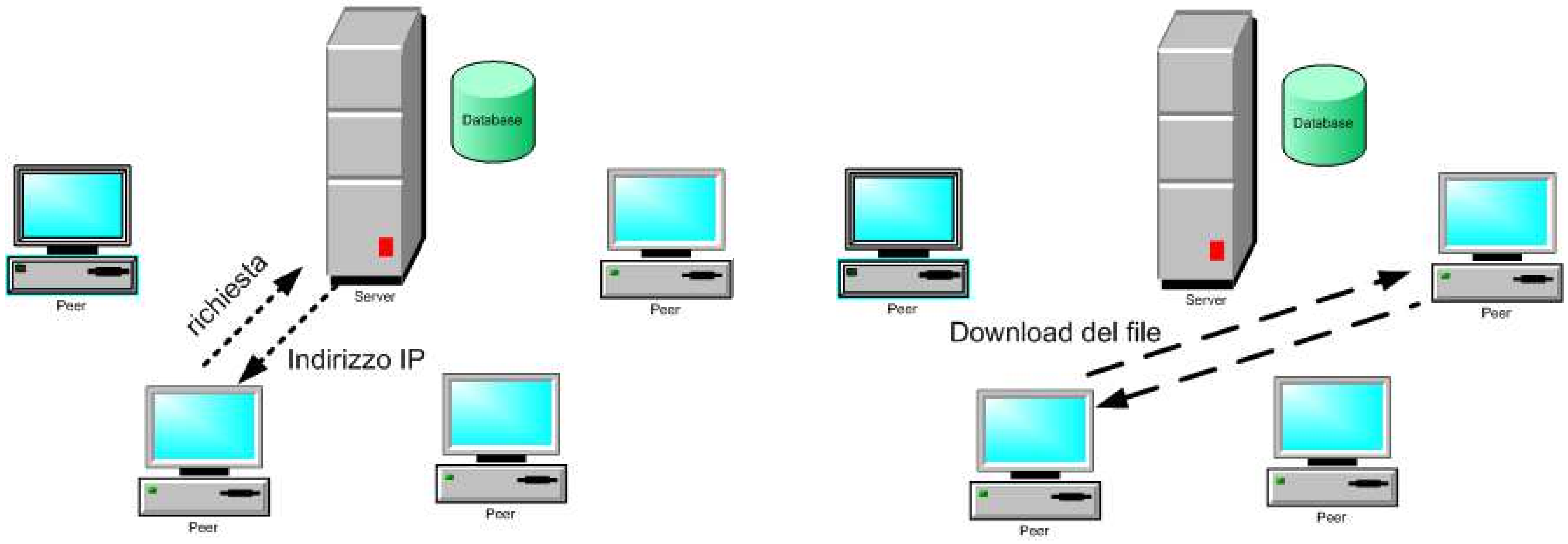
- Centralized Index
- Query Flooding
- Hierarchical Overlay
- Distributed Hash Table (DHT)

Centralized Index

- Service provided by a server (or server farm)
  - Used in Napster
  - When a user becomes active the application notifies the index with its IP address and list of available files
- Hybrid Approach
  - File distribution is P2P
  - Search is client-server

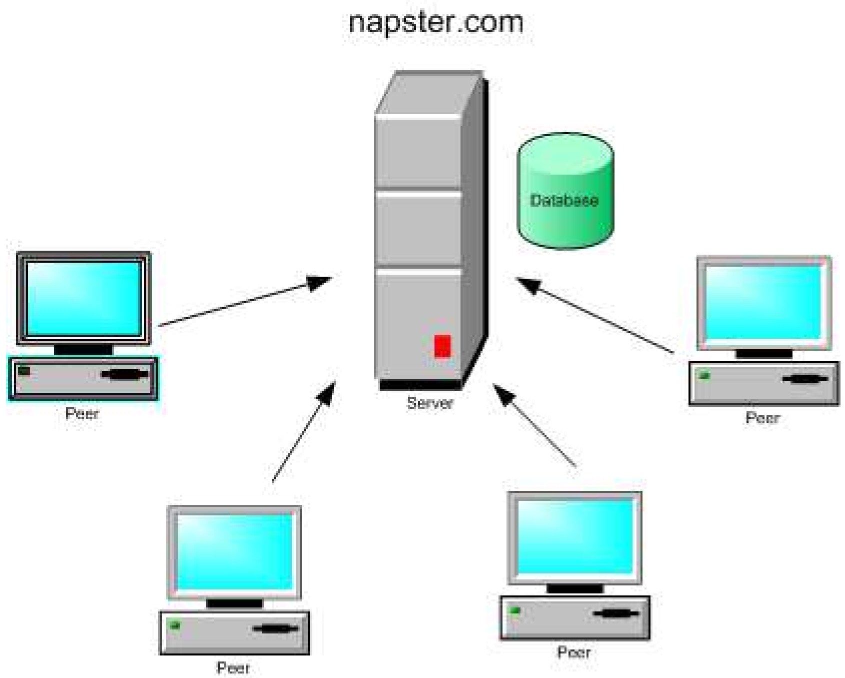
Napster

- P2P system for music sharing
- Centralized index
  - napster.com



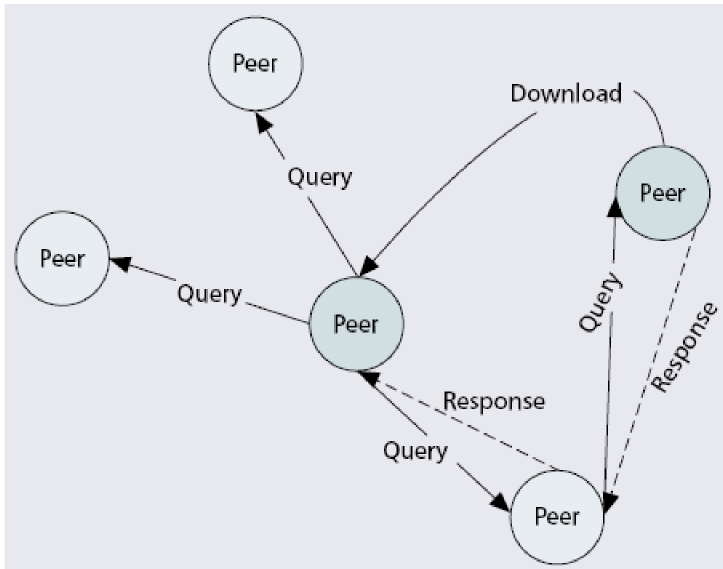
Napster

- Drawbacks
  - Single point of failure
  - Performance bottleneck
  - Copyright Infringement



Query Flooding

- Completely decentralized approach
  - Used in the original Gnutella version (LimeWire)
  - Overlay network
    - Graph formed of all active peers (nodes) and TCP connections among them (edges)
  - Query Flooding + Unicast Responses
  - File download from a single peer
- Scalability
  - Limited-scope query flooding
    - Reduces query traffic
    - Decreases the probability to locate the content

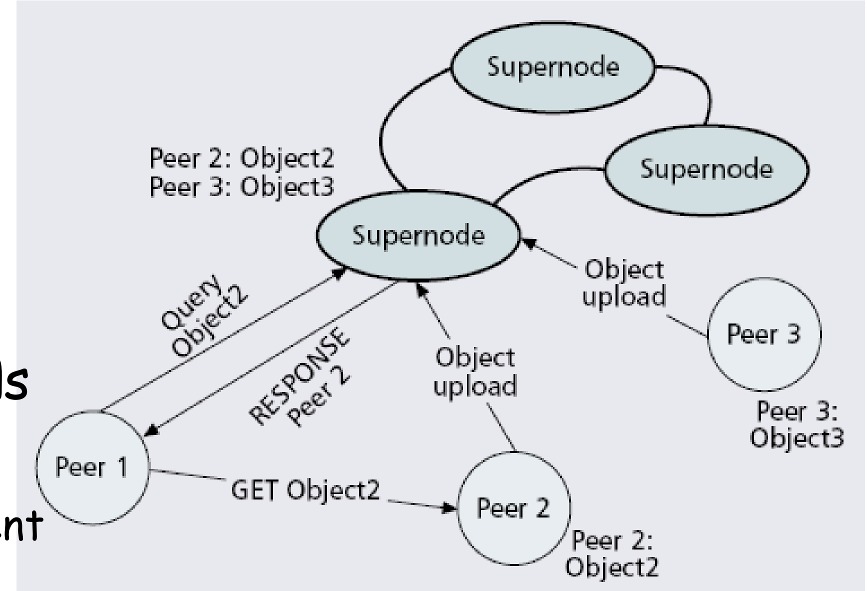


Query Flooding

- Dynamic overlay construction (Gnutella)
  - Peer X becomes active and needs to joins the overlay
  - X uses a list of often-active peers or contact a tracker site
  - X starts setting up TCP connections with peers in the list
    - E.g., X establish a TCP connection with Y
  - X sends a ping message to Y (including peer-count field)
  - Y forwards the ping message to its neighbors
    - All other peers continue forwarding the ping message until peer-count reaches 0
  - Whenever a peer Z receive a ping replies with a pong message
    - The pong message includes the Z's IP address
  - Upon receiving pong messages X can establish TCP connections with the corresponding peers

Hierarchical Overlay

- Combines best features of previous approaches
  - First proposed in FastTrack (Kazaa, Morpheus), also used in modern Gnutella
  - No central server
  - Not all peers are equal
- Super Nodes (SN)
  - Peers with high-bandwidth connection and high availability
  - Ordinary peers connects to SNs
  - Each SN has a local index
    - Peers inform their SN about content
    - SNs form an SN overlay net
  - Peers query their SN
  - Download from a single peer



Distributed Hash Table (DHT)

- DHT = distributed P2P database
  - Used in eMule
  - Also used in BitTorrent for distributed tracker
    - Kademlia DHT

DHT Identifiers

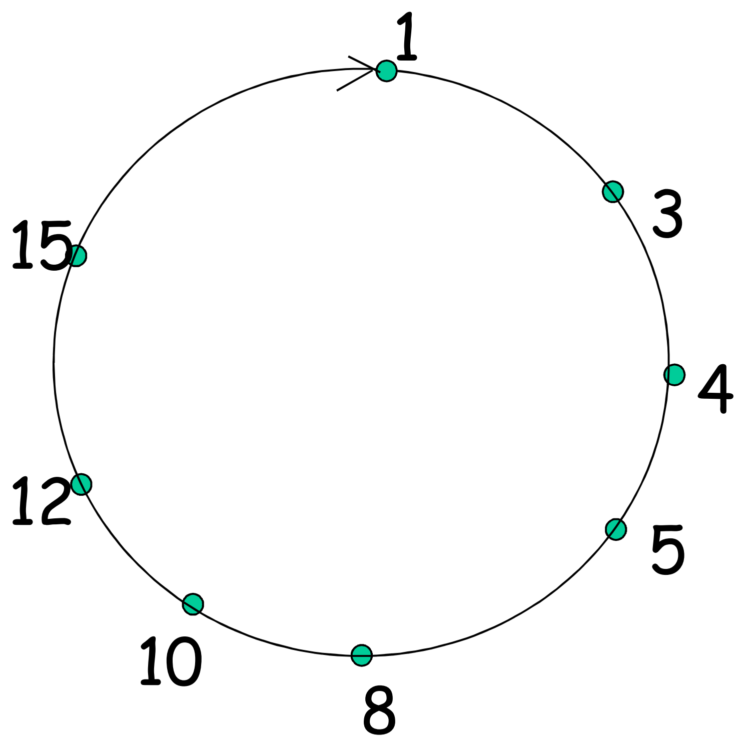
- Assign integer identifier to each peer in range $[0,2^n-1]$.
  - Each identifier can be represented by n bits.
- Require each key to be an integer in same range.
- To get integer keys, hash original key.
  - eg, key = h("Led Zeppelin IV")
  - This is why they call it a distributed "hash" table

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: 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

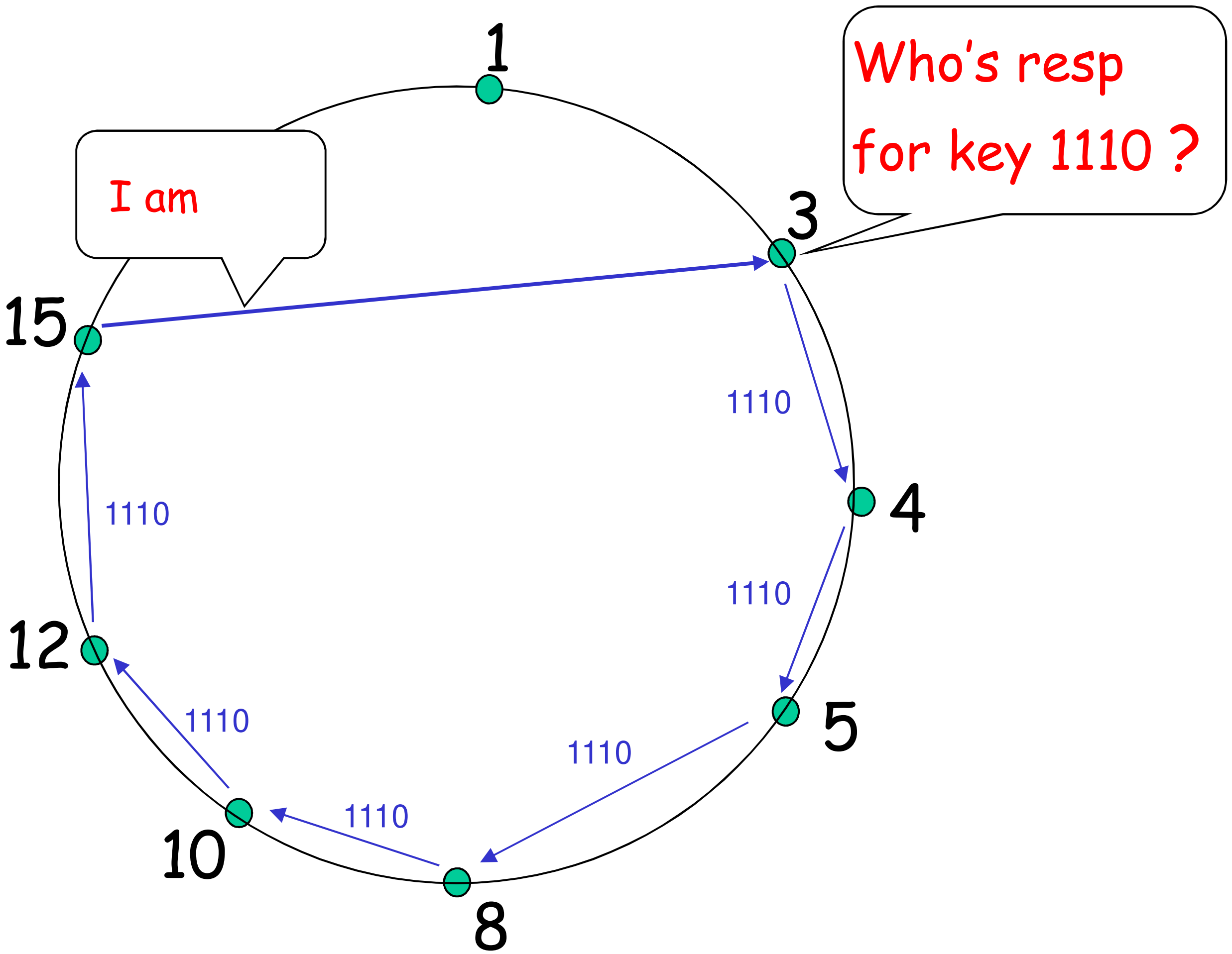
Circular DHT (1)

- Alice wants to insert a (key, value) pair.
- How can she know the IP address of the immediate successor?
- Each peer *only* aware of immediate successor and predecessor.
- "Overlay network"



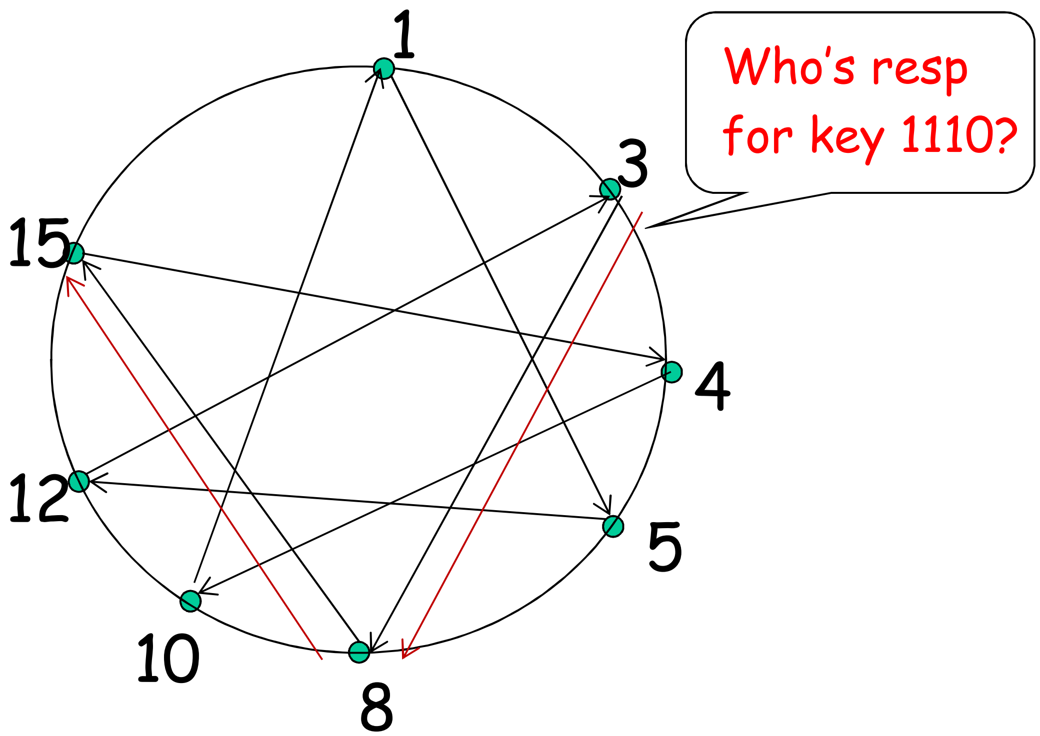
Circle DHT (2)

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



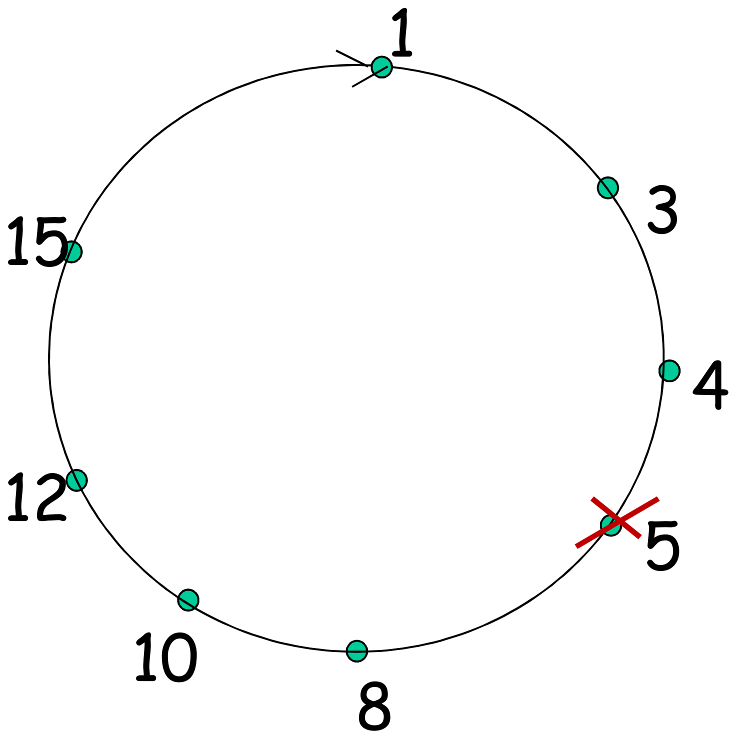
Define closest as closest successor

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.

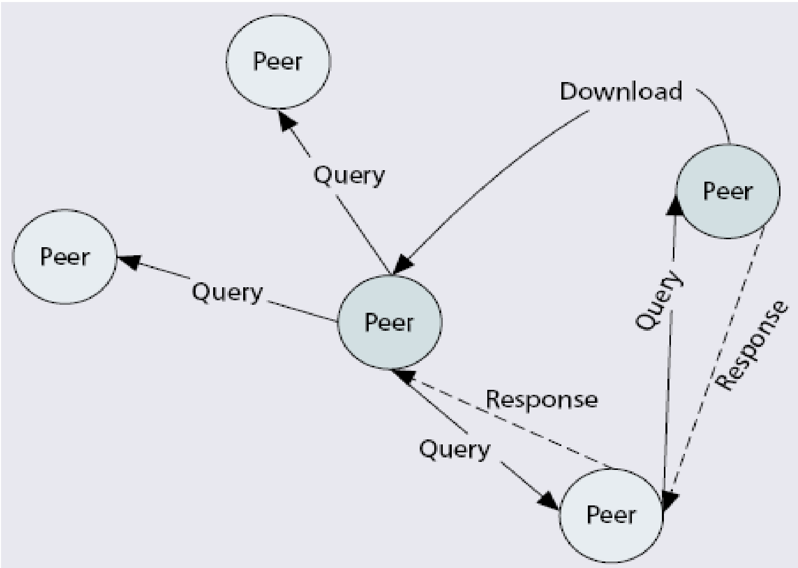
- Peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
- What if peer 13 wants to join?

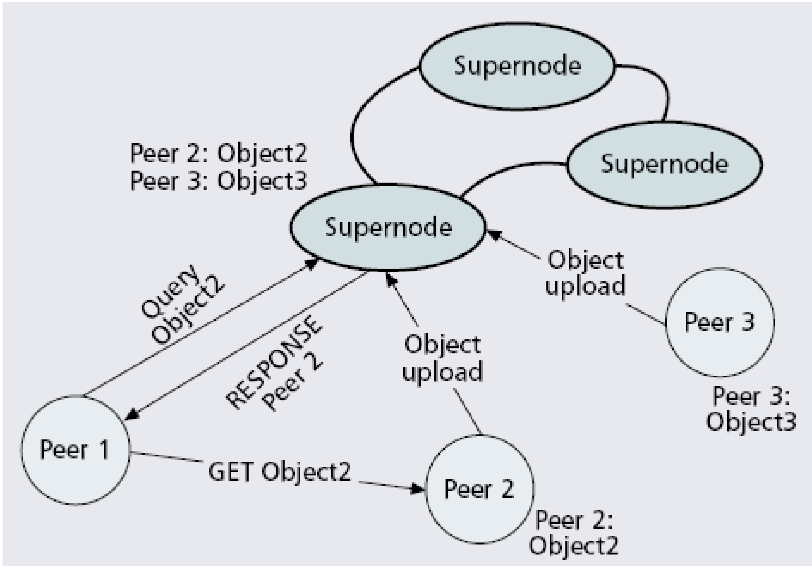
Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P applications
  - Content Search/Location, File Distribution, Internet Telephony
- Socket programming

File Download

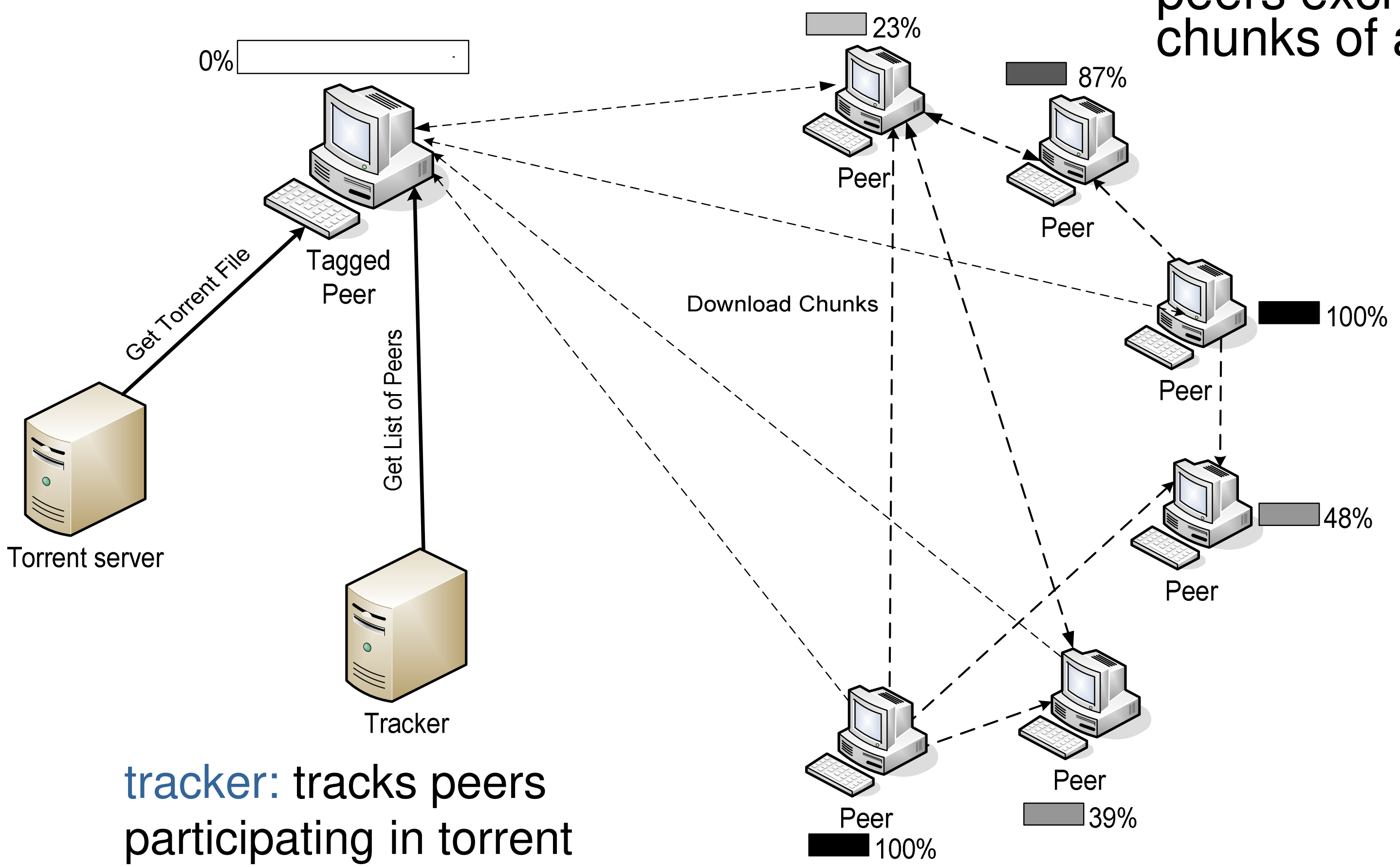
- Download from a single peer
  - Napster
  - Gnutella
  - FastTrack
- Download from a many peers
  - BitTorrent





BitTorrent

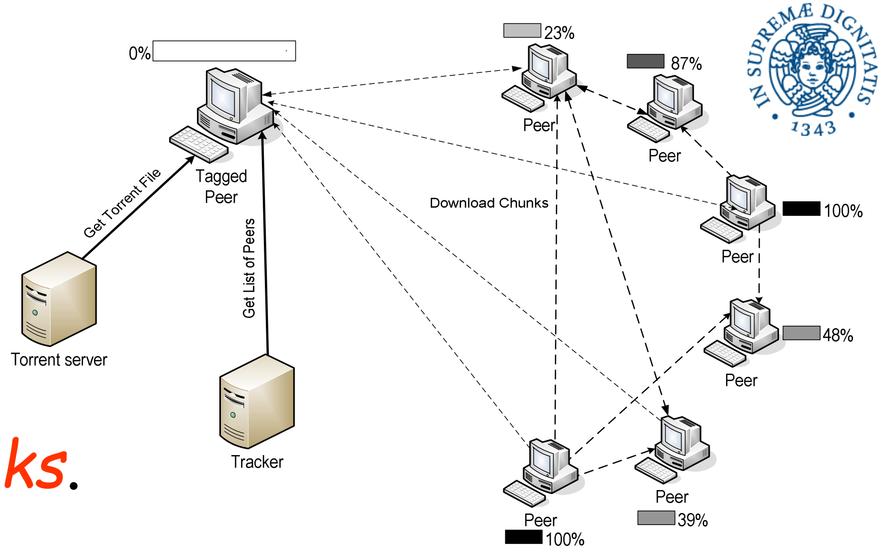
- P2P file distribution



torrent: group of peers exchanging chunks of a file

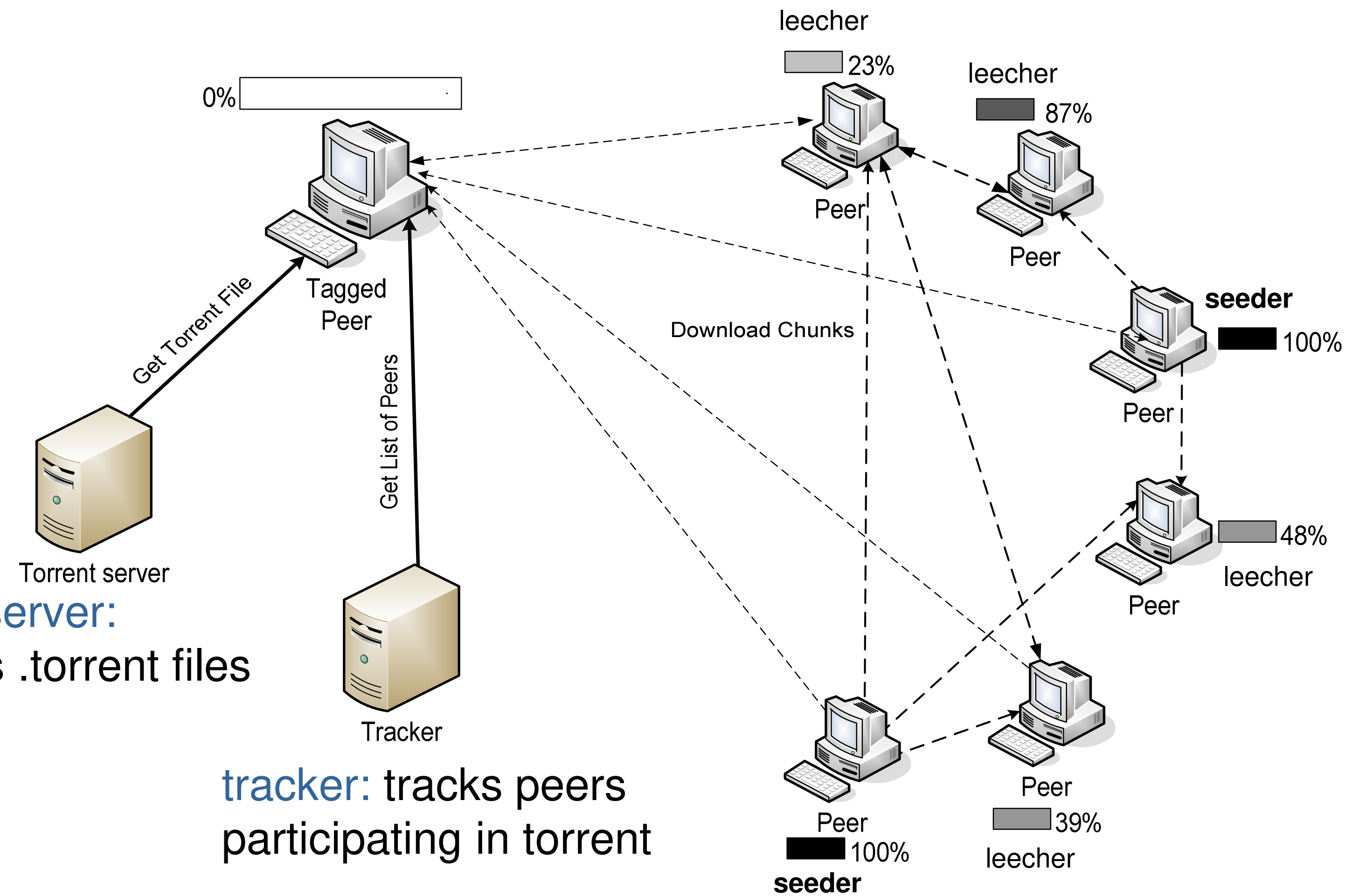
tracker: tracks peers participating in torrent

BitTorrent protocol



- 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

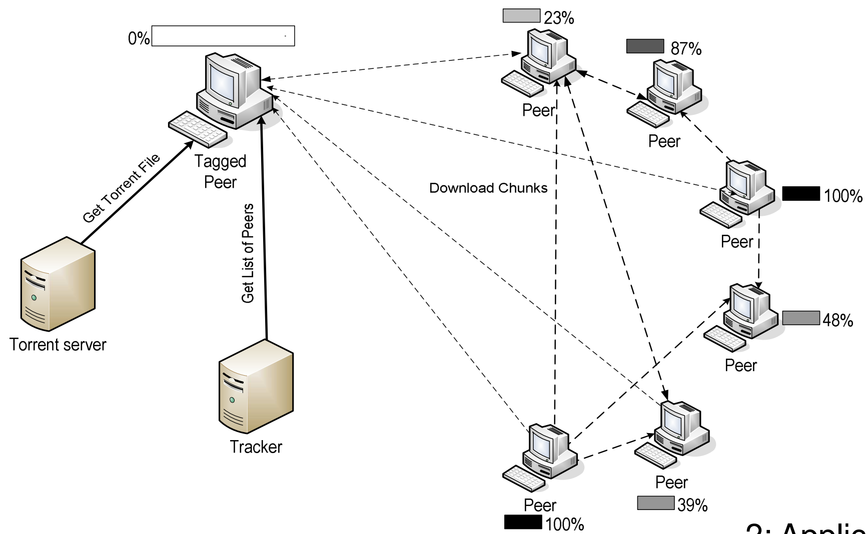


torrent server:
contains .torrent files

tracker: tracks peers
participating in torrent

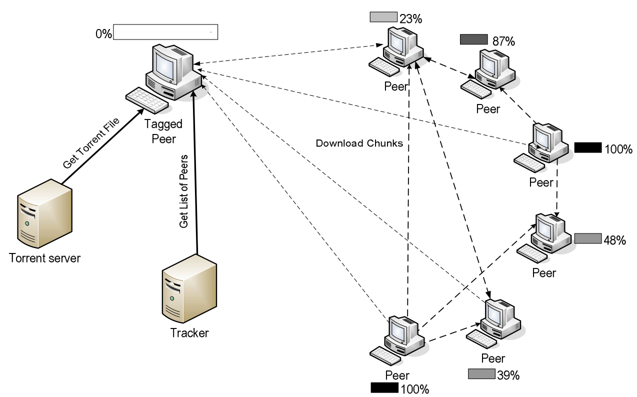
BitTorrent: Pulling Chunks

- 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



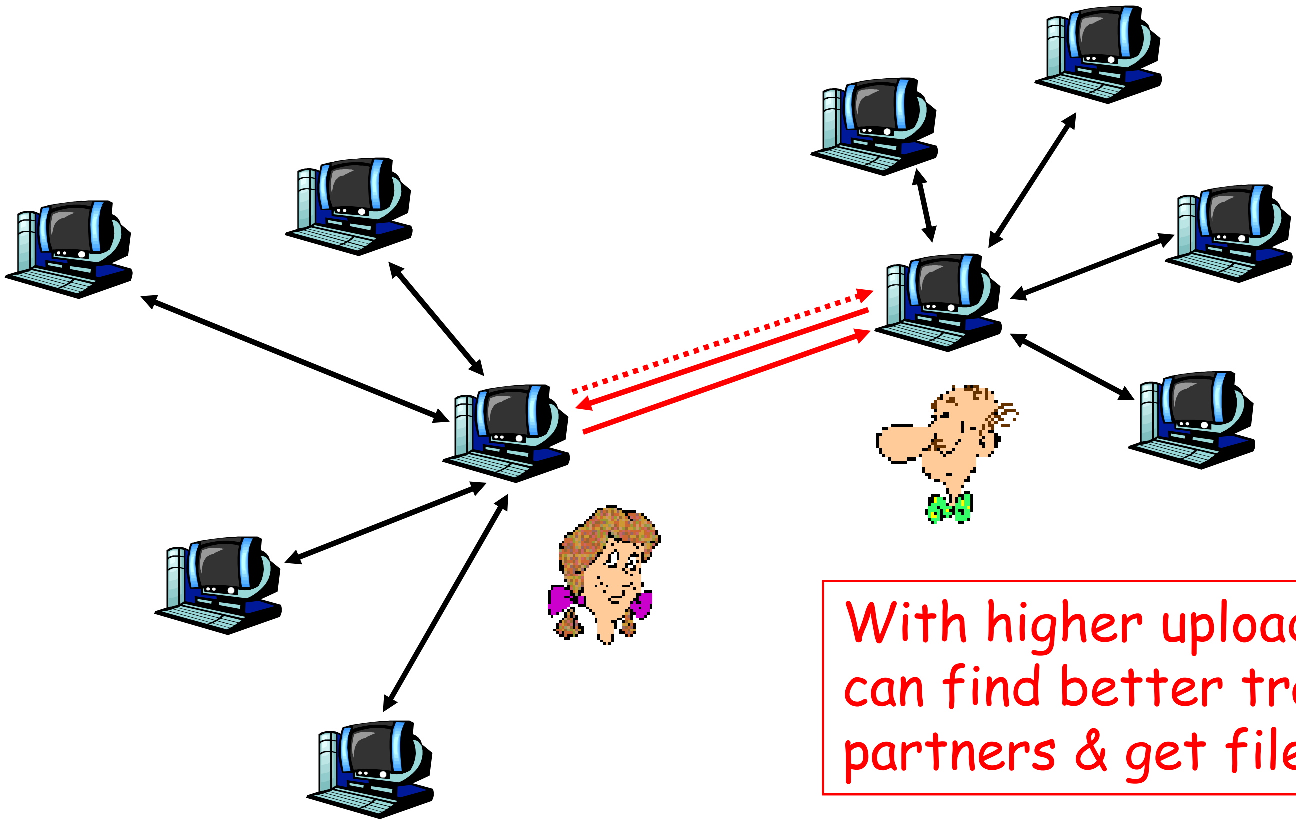
BitTorrent: Tit-for-tat

- Alice sends chunks to 4 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



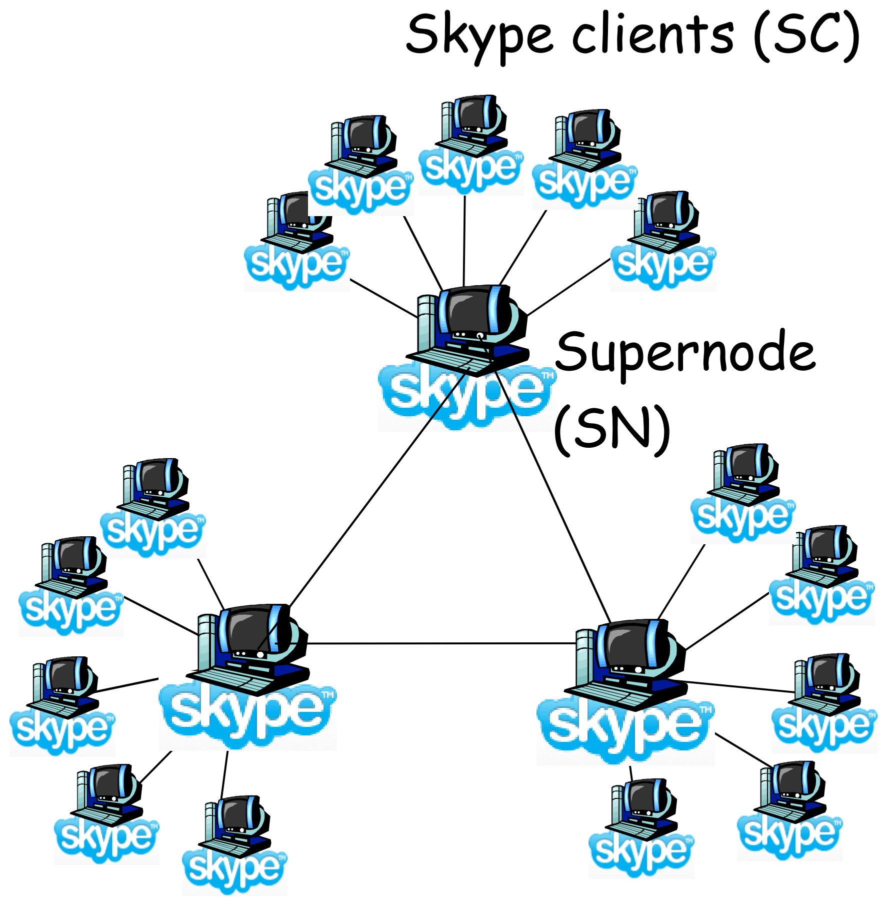
With higher upload rate, can find better trading partners & get file faster!

Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P applications
  - Content Search/Location, File Distribution, Internet Telephony
- Socket programming

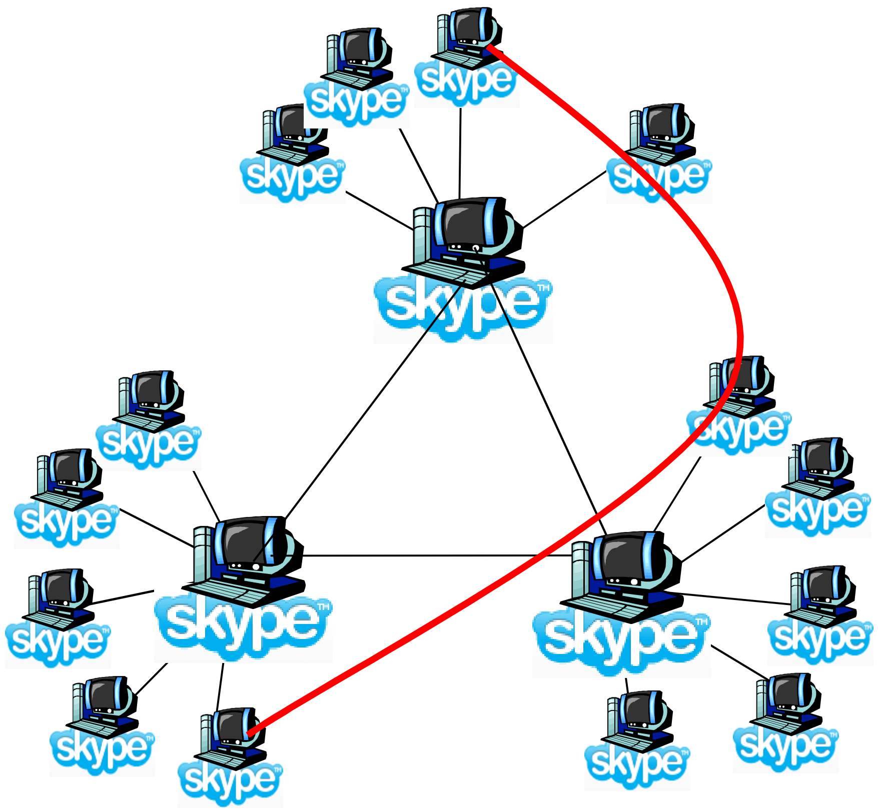
P2P Internet Telephony: Skype

- 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



Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- P2P applications
  - File Sharing, Internet Telephony
- Socket programming
  - UDP/TCP

Socket API

Goal: learn how to build client/server application that communicate using sockets

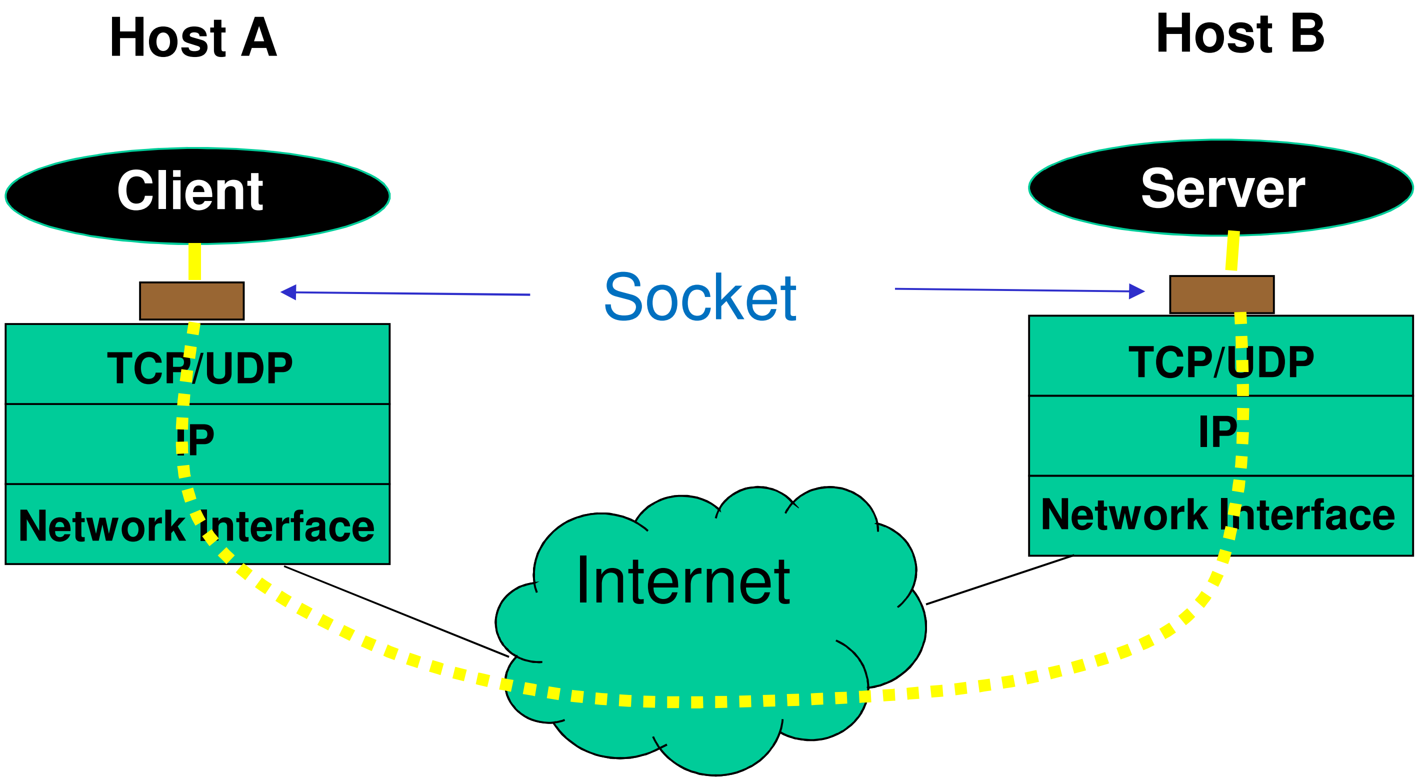
Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - Datagram (UDP)
  - Stream (TCP)

Socket

- Socket
  - An *application-created*, *OS-controlled* interface (a "door") into which application process can both send/receive messages to/from another application process
- Socket Address
  - IP address
  - Port Number
- Process communication
  - Communication involve a couple of sockets, at the two endpoints

Socket-based communication



Socket programming basics

- Server must be running before client can send anything to it.
- Server must have a socket (door) through which it receives and sends segments
- Similarly client needs a socket
- Socket is locally identified with
  - IP address
  - Port number (HTTP server: 80, Mail server: 25)
- Client needs to know server IP address and socket port number.

Socket programming with UDP

UDP: no "connection" between client and server

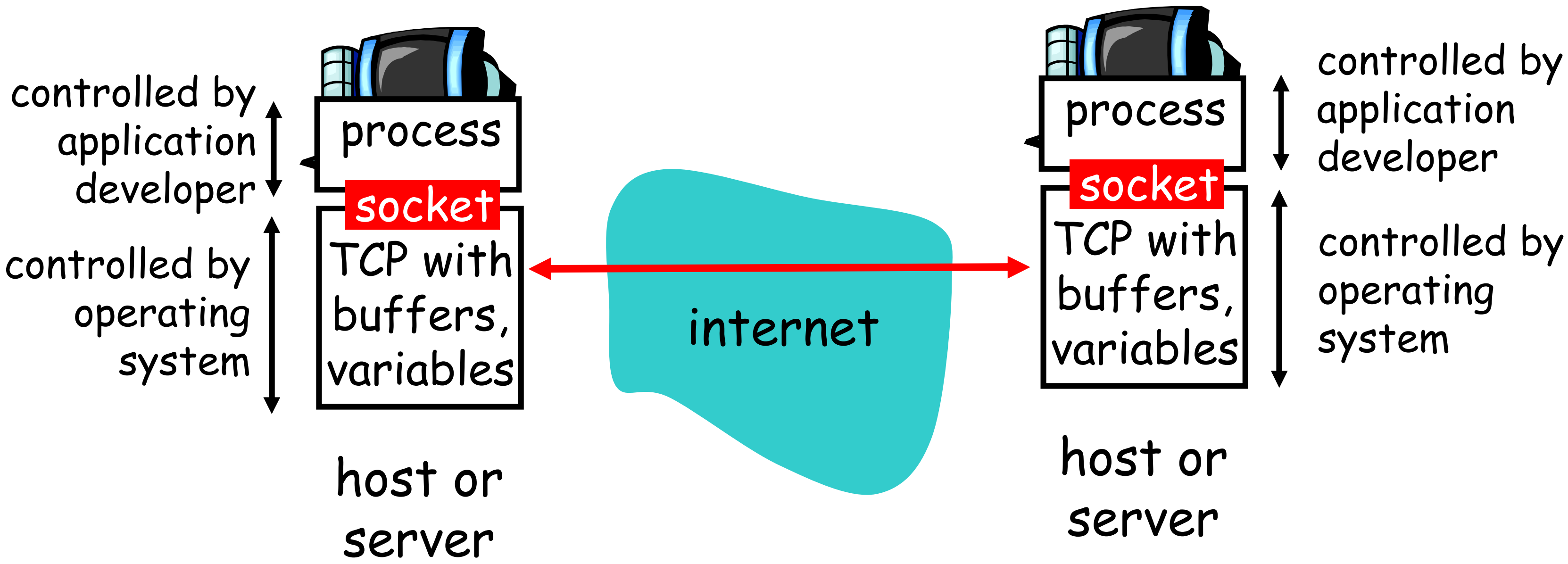
- no handshaking
- sender explicitly attaches IP address and port of destination to each message
- OS attaches IP address and port of sending socket to each message
- Server can extract IP address, port of sender from received message

Application viewpoint

UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server

Socket Programming with TCP

TCP service: reliable transfer of **bytes** from one process to another



Socket programming with TCP

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients

application viewpoint

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Summary

- Application paradigms
  - 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

Lessons 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
- Control vs. data messages
  - in-band, out-of-band
- Centralized vs. decentralized
- Stateless vs. stateful
- Reliable vs. unreliable message transfer
- “Complexity” at network edge