CSCI 8260 – Spring 2016
Network Attacks and Defenses

Instructor: Prof. Roberto Perdisci
perdisci@cs.uga.edu

These slides are adapted from the textbook slides by J.F. Kurose and K.W. Ross

Chapter 2: Application Layer

Our goals:

- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- 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 (YouTube)
- voice over IP
- real-time video conferencing
- cloud computing
- …
- …
-

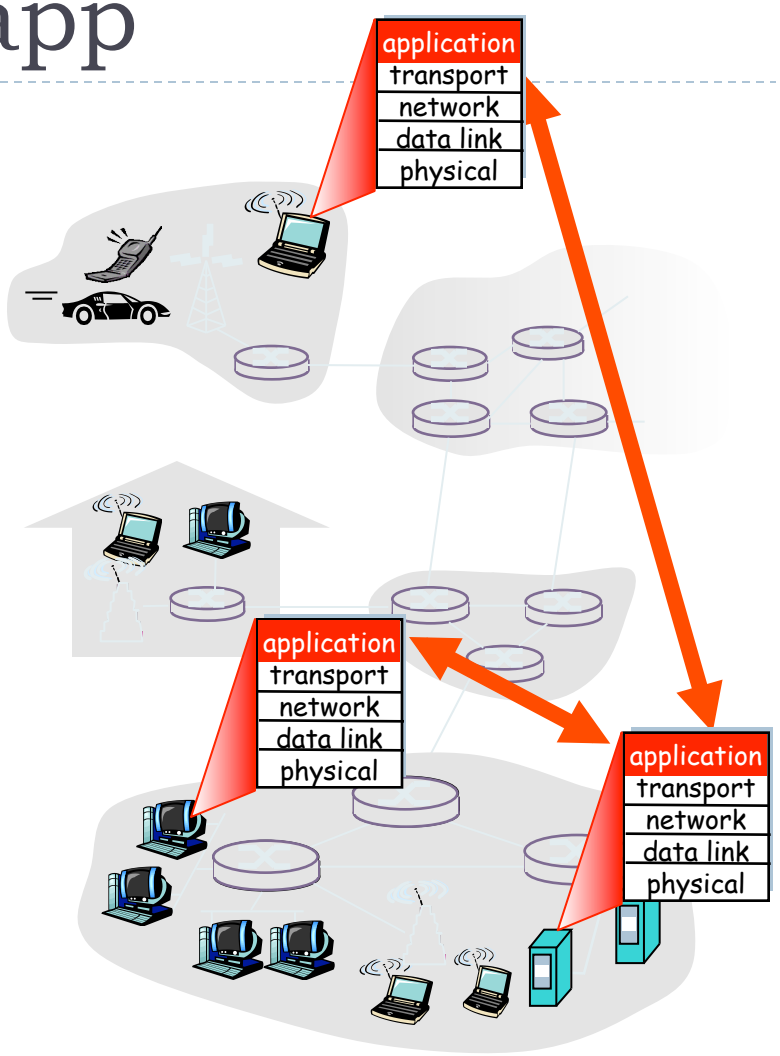
Creating a network app

write programs that

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

No need to write software for network-core devices

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



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

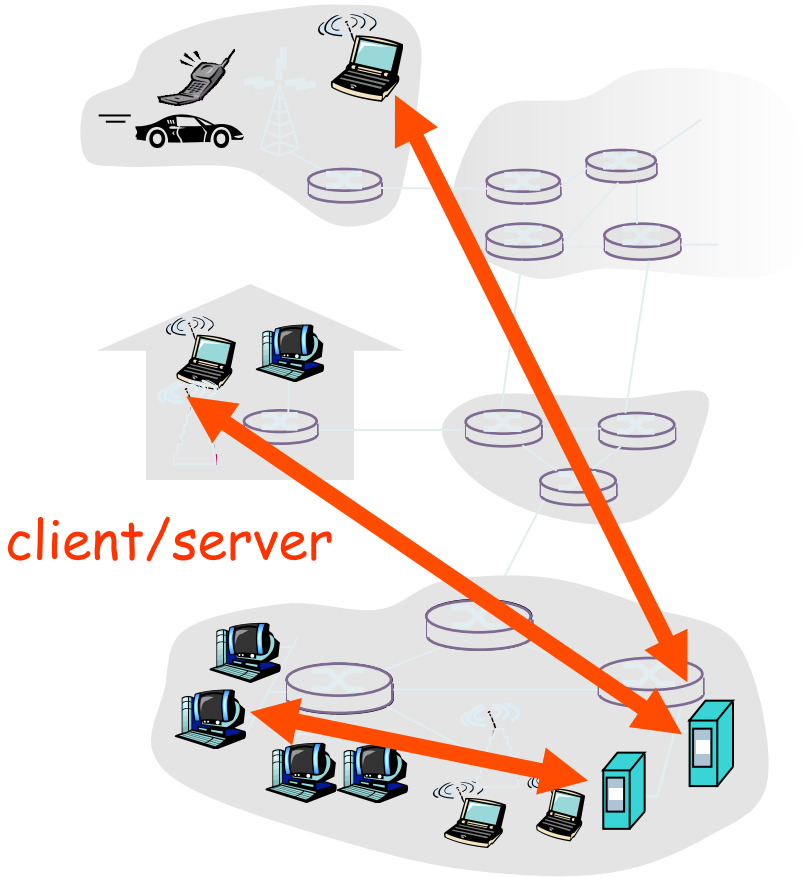
2.7 Socket programming with TCP

2.8 Socket programming with UDP

Application architectures

- client-server
- peer-to-peer (P2P)
- hybrid of client-server and P2P

Client-server architecture



server:

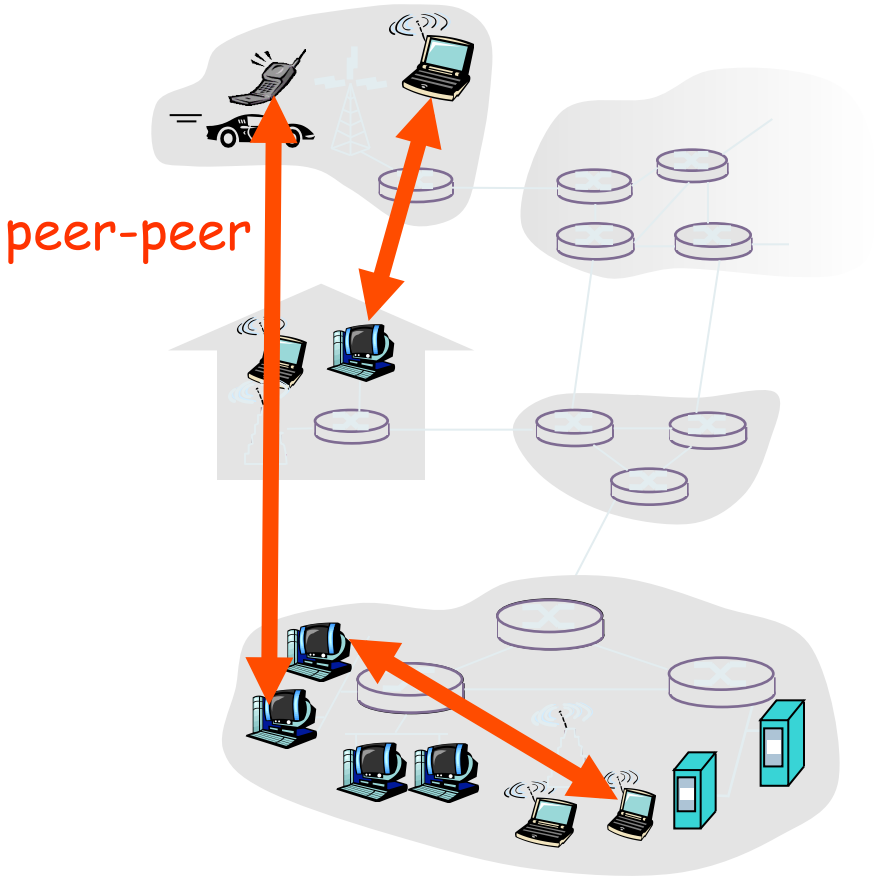
- always-on host
- permanent IP address
- server farms for scaling

clients:

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

Pure P2P architecture

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



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

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

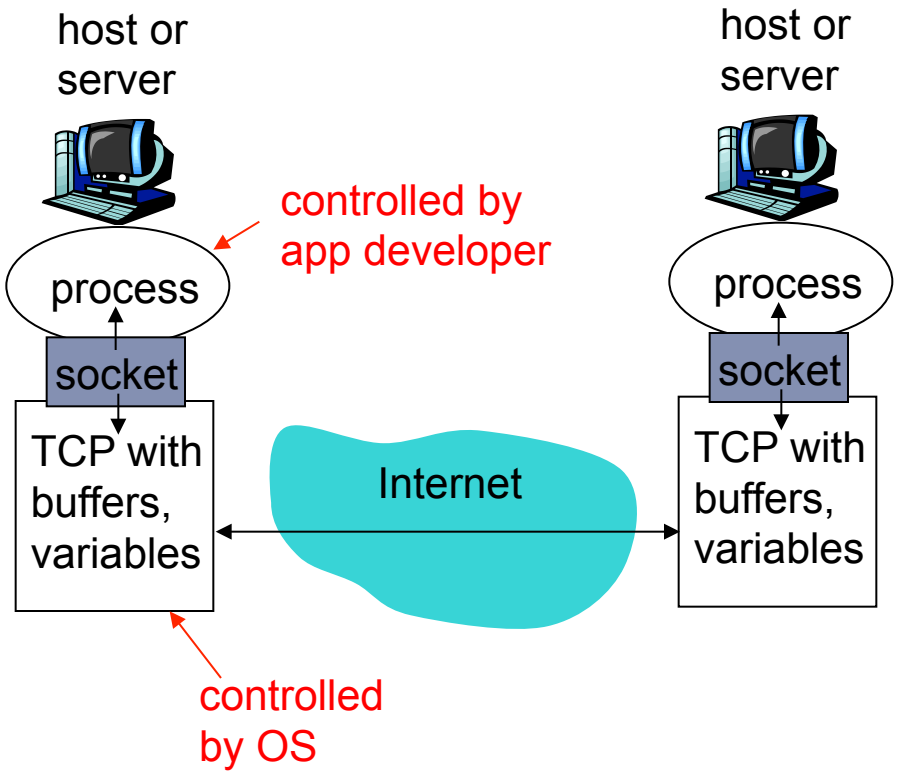
client process: process that initiates communication

server process: process that waits to be contacted

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

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves 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?

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…

Internet transport protocols services

TCP service:

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

UDP service:

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

Q: why bother? Why is there a UDP?

Chapter 2: Application layer

2.1 Principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

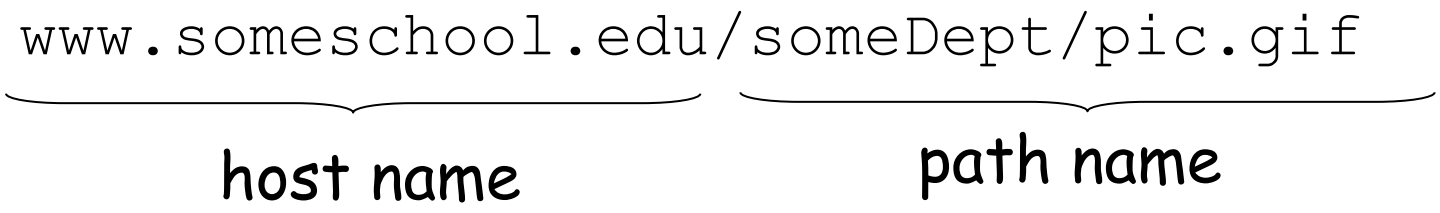
2.7 Socket programming with TCP

2.8 Socket programming with UDP

Web and HTTP

First, a review…

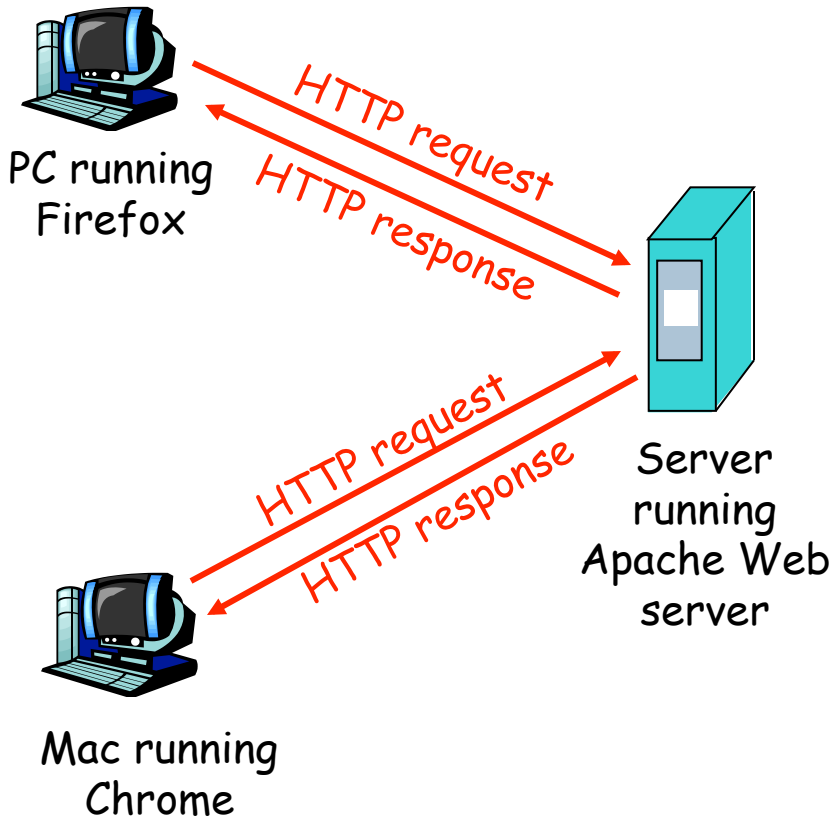
- web page consists of objects
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of base HTML-file which includes several referenced objects
- each object is addressable by a URL
- example URL:



HTTP overview

HTTP: hypertext transfer protocol

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



HTTP 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

non-persistent HTTP

- at most one object sent over TCP connection.

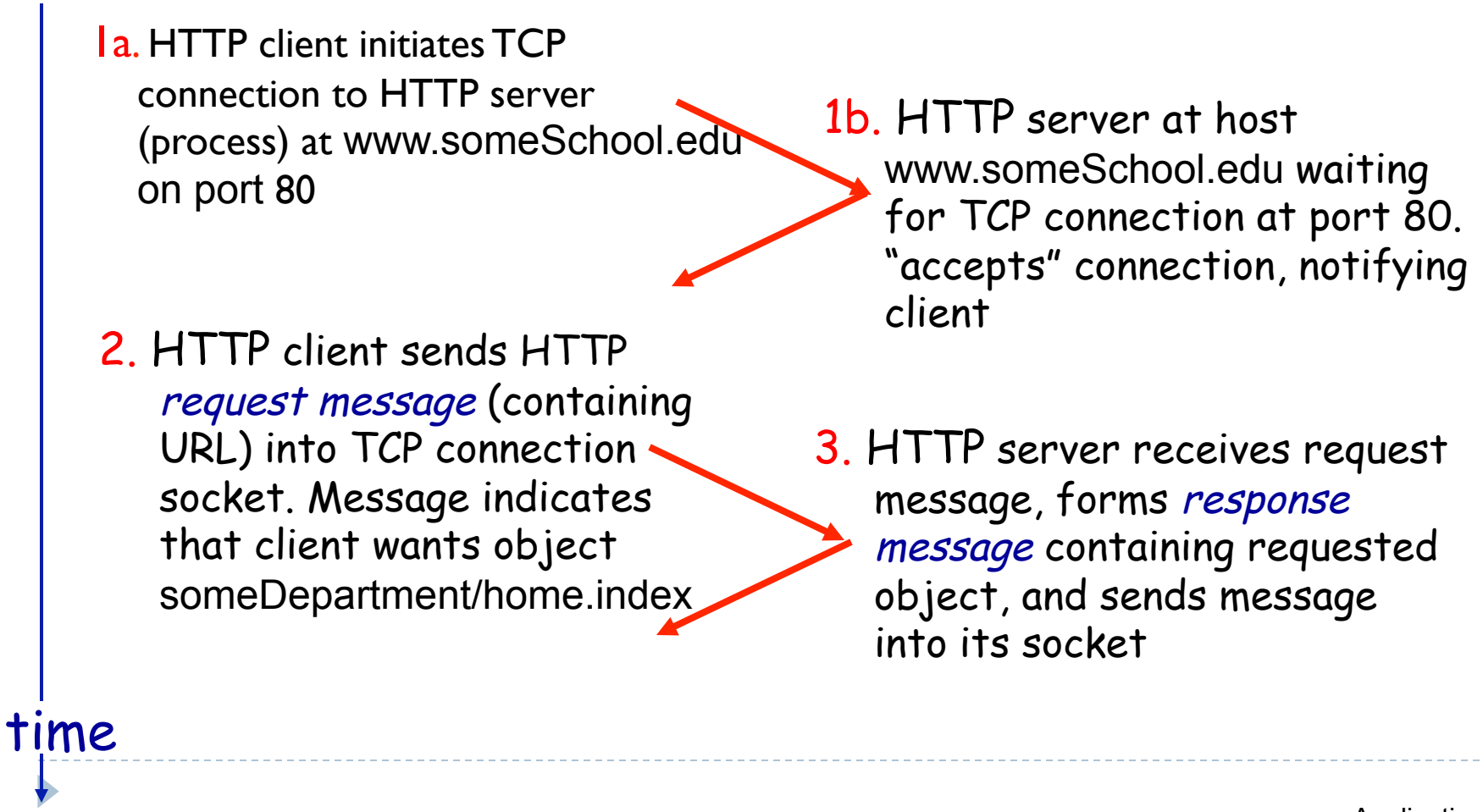
persistent HTTP

- multiple objects can be sent over single TCP connection between client, server.

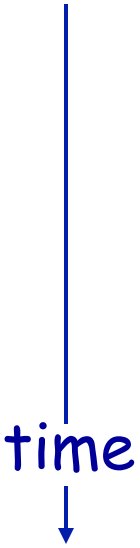
Nonpersistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)



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

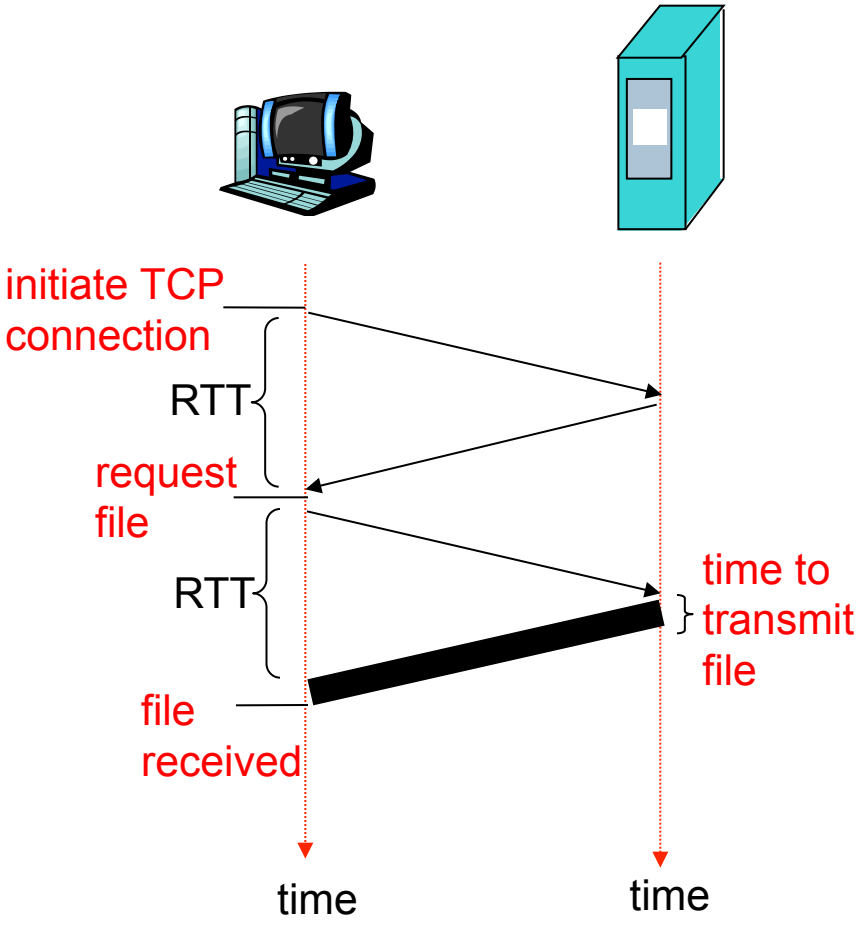
Non-Persistent HTTP: Response time

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

non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection

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

Advantage of non-persistent HTTP

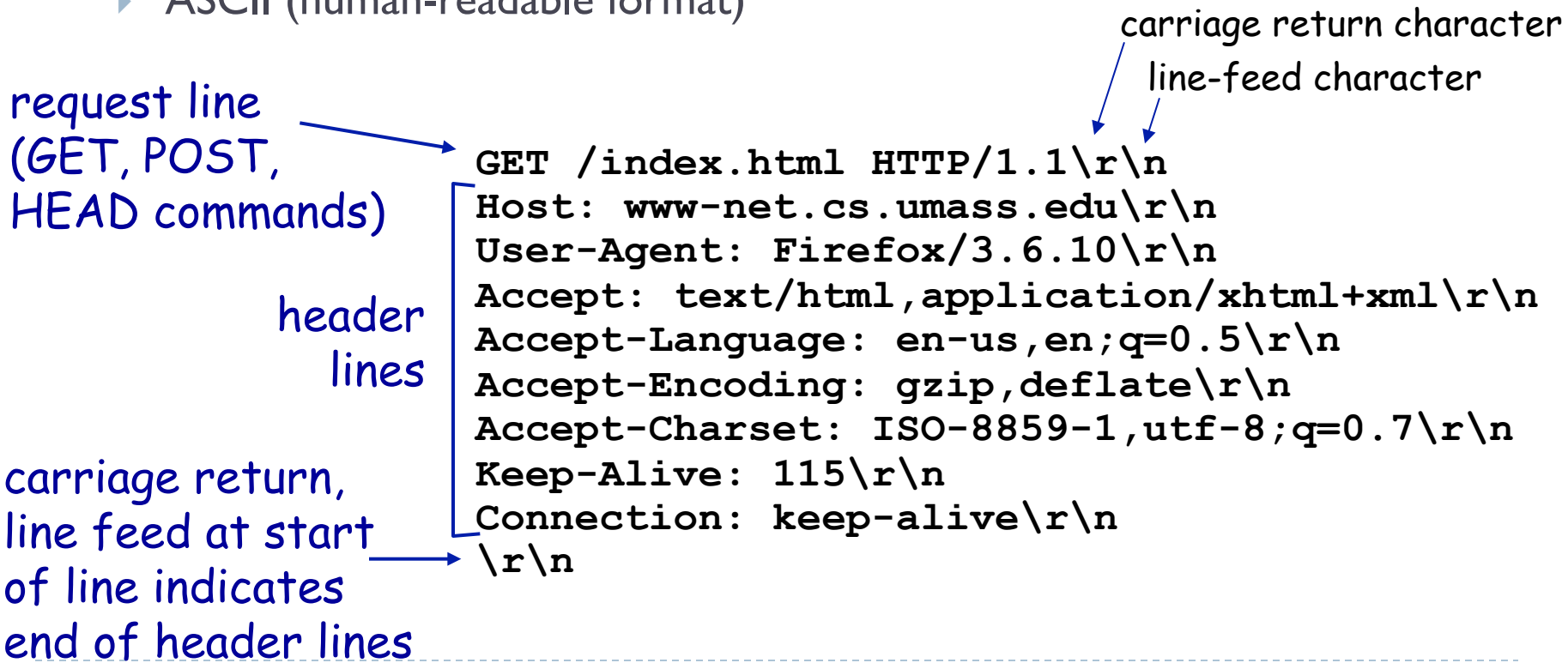
non-persistent HTTP:

- browsers can open parallel TCP connections to fetch referenced objects "at the same time"
  - Has advantages and disadvantages

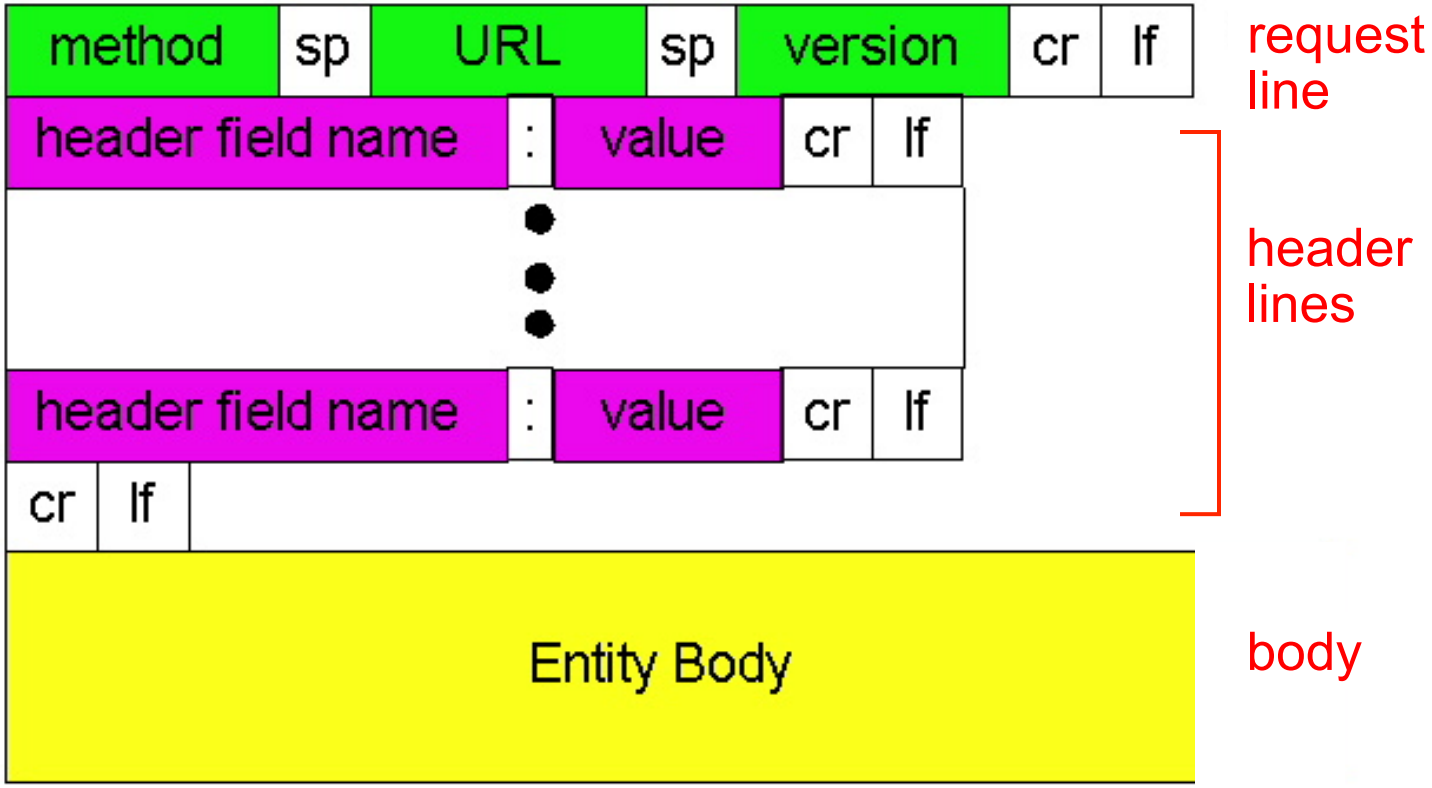
HTTP request message

http://www-net.cs.umass.edu:8080/index.html

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



HTTP request message: general format



A simple test… ****

- $ nc –l 12345
- Point your browser to http://127.0.0.1:12345/testme

- If your user-agent looks strange and you curious to know why, read this:
  - http://webaim.org/blog/user-agent-string-history/

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

www.example.com/animalsearch.php?name=monkeys&age=10

Method types

HTTP/1.0

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

HTTP/1.1

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

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

HTTP response status codes

- status code appears in 1st line in server->client response message.
- some sample codes:

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

- requested object moved, new location specified later in this msg (Location:)

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.uga.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 /profile/mission HTTP/1.1
Host: www.uga.edu
```

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

3. look at response message sent by HTTP server!

(or use wireshark!)

User-server state: cookies

many Web sites use cookies

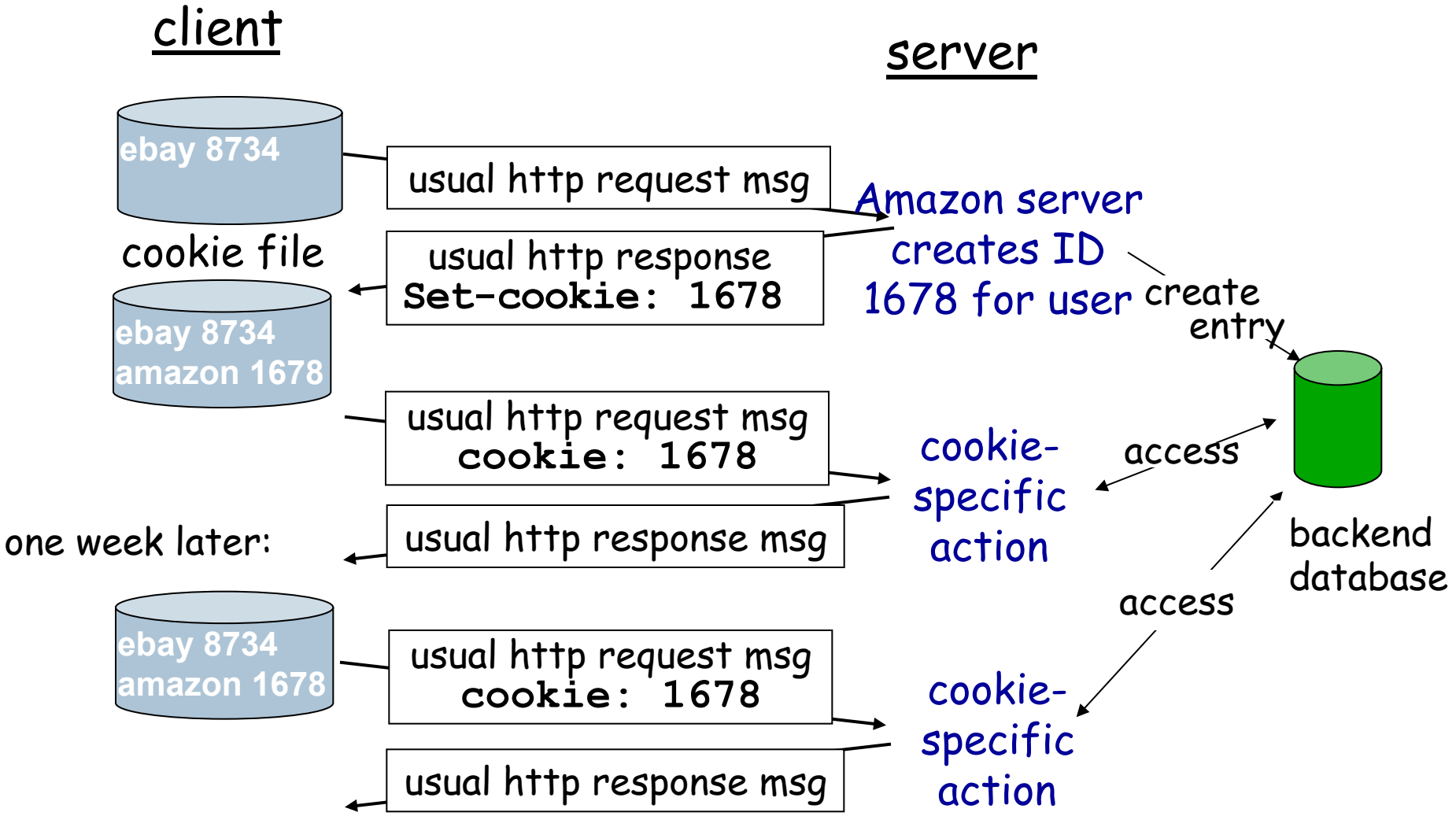
four components:

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

Cookies and Privacy

- Two types of cookies
  - Session cookies
  - Permanent cookies (tracking cookies)

- Third-party cookies (see http://tools.ietf.org/html/rfc2965)
  - You visit www.example.com, which contains a banner from ads.clicks-for-me.net
    - in simple terms ads.clicks-for-me.net is third-party because it does not match the domain showed on the URL bar
    - third-party sites should be denied setting or reading cookies
  - The browser allows ads.clicks-for-me.net to drop a third-party cookie
  - Then you visit www.another-example.com , which also loads ads from ads.clicks-for-me.net
  - ads.clicks-for-me.net can track the fact that you visited both www.example.com and www.another-example.com !!!

Cookies and Security

- Authentication Cookies can be stolen
  - An attacker may be able to "sniff" your authentication cookies
  - The attacker will be able to login as you on a website (e.g., Facebook, Twitter, etc…)
  - See FireSheep for a concrete example!
    - http://codebutler.com/firesheep

Session IDs

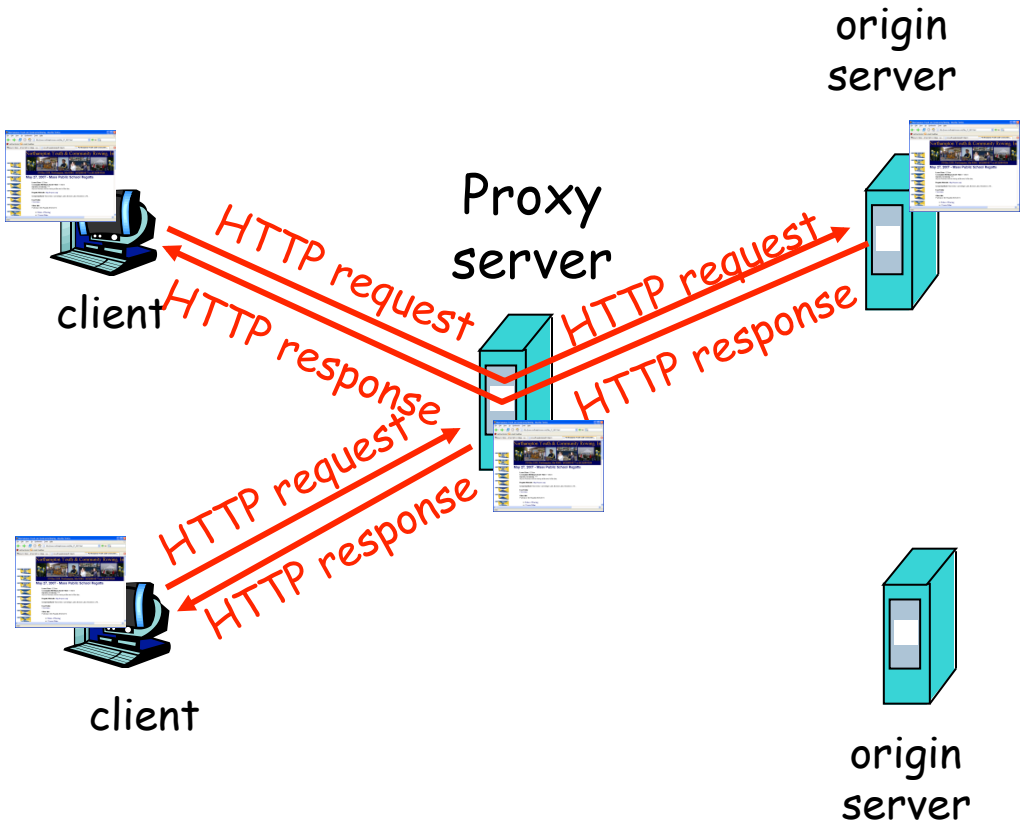
- Cookies are not the only way you can keep state
  - Session IDs are commonly used by web applications
    - http://example.com/index.php?user_id=0F4C26A1&topic=networking

- What are the main difference between cookies and Session IDs?
  - Session IDs are typically passed in the URL (added to web app links)
  - Cookies are passed through HTTP req/resp headers
  - Cookies are stored in the browser's cache and have an expiration date
  - Session IDs are volatile: never stored, only used until end of session

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
  - Splits the TCP connection!
- 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 in HTTP

http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html

HTTP Pipelining and Range

- Pipelining
  - The client sends multiple HTTP request without waiting for server response
  - The server sends the response one after the other

- Range
  - HTTP allows downloading pieces of objects
  - Example:
    - 10MB image to be downloaded
    - We can open 10 different TCP connection and send 10 HTTP requests in parallel
    - Download 1MB of data from each connection and stitch them back together

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

DNS: Domain Name System

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

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

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

Domain Name System:

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

DNS

DNS services

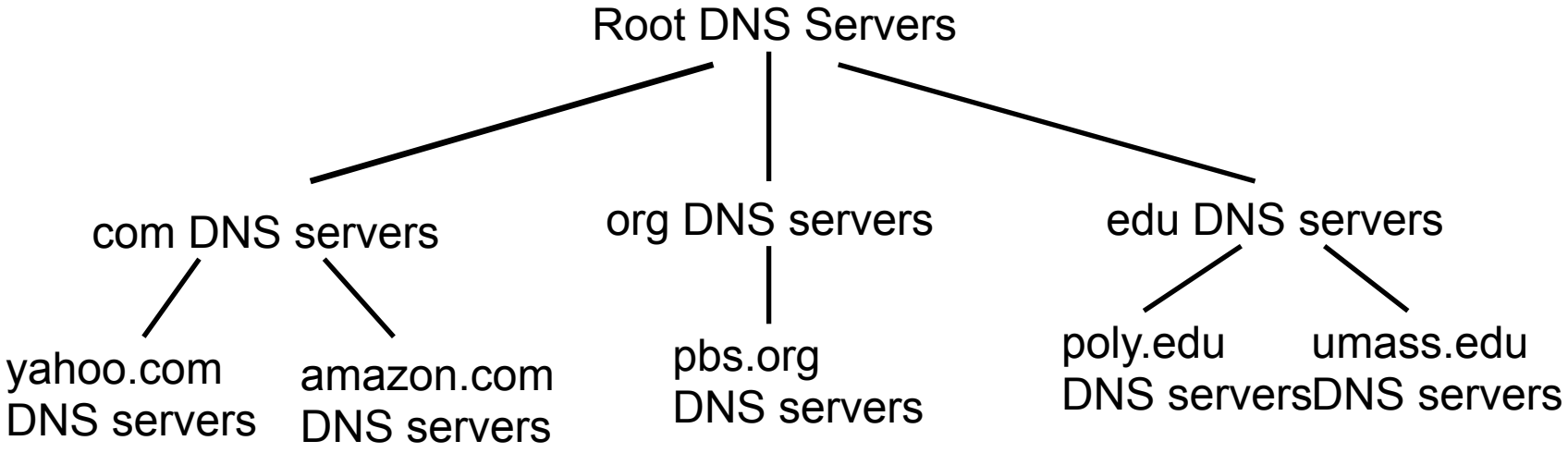
- hostname to IP address translation
- host aliasing
  - Canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

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

doesn't *scale!*

Distributed, Hierarchical Database

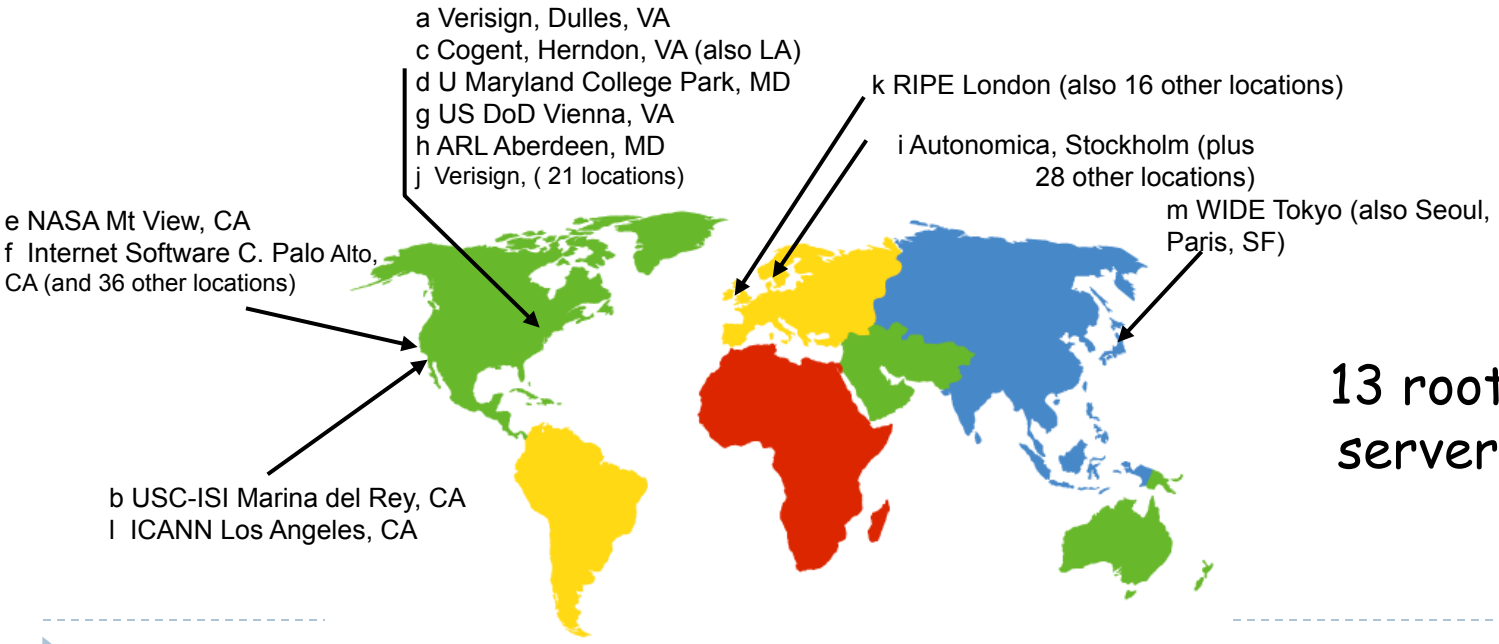


client wants IP for www.amazon.com; 1st approx:

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

Authoritative DNS servers:

- organization's 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

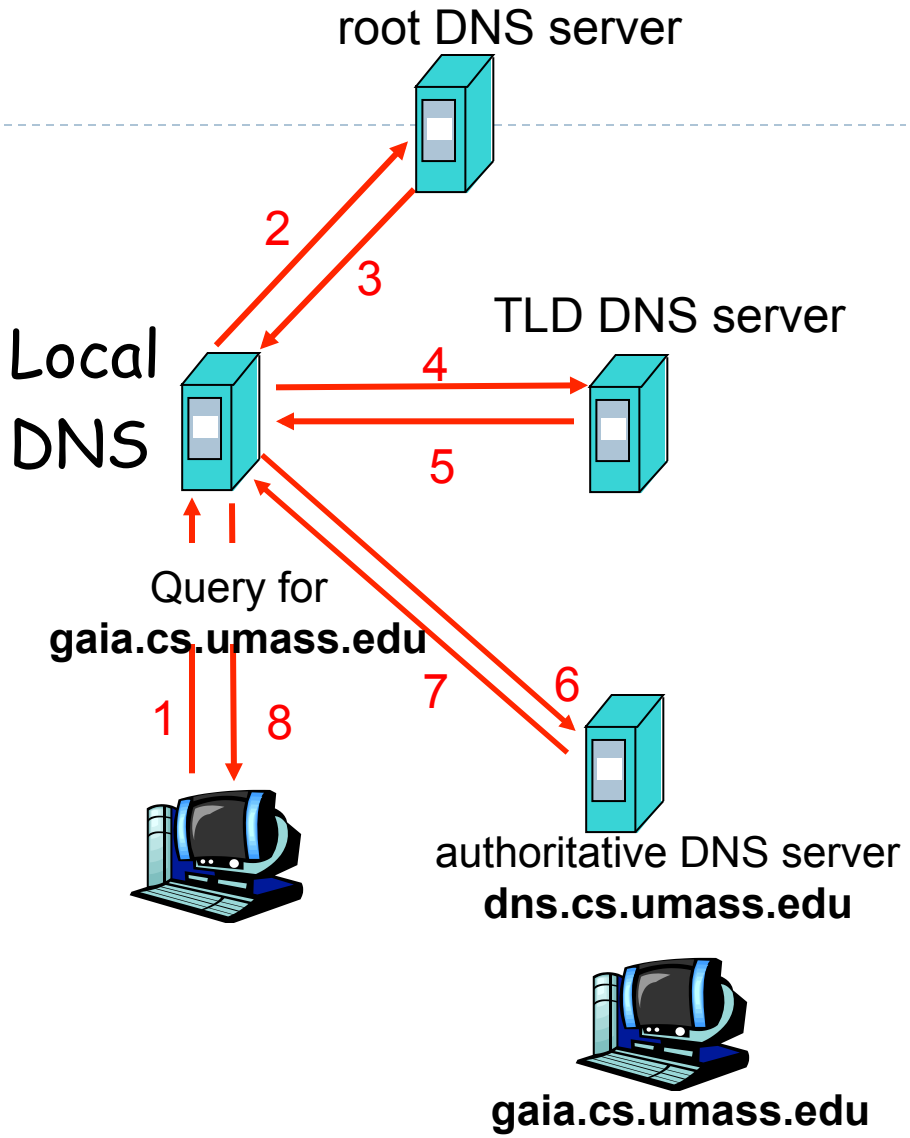
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one.
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - acts as proxy, forwards query into hierarchy

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: caching and updating records

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

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

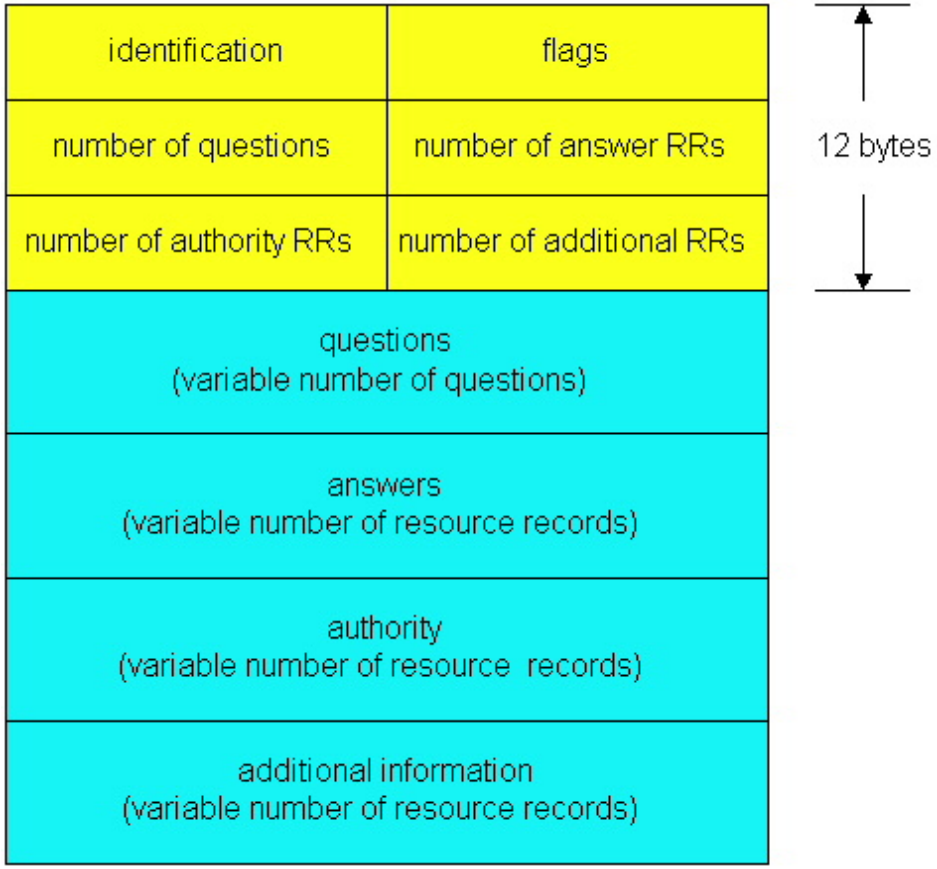
- **value** is name of mailserver associated with **name**

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



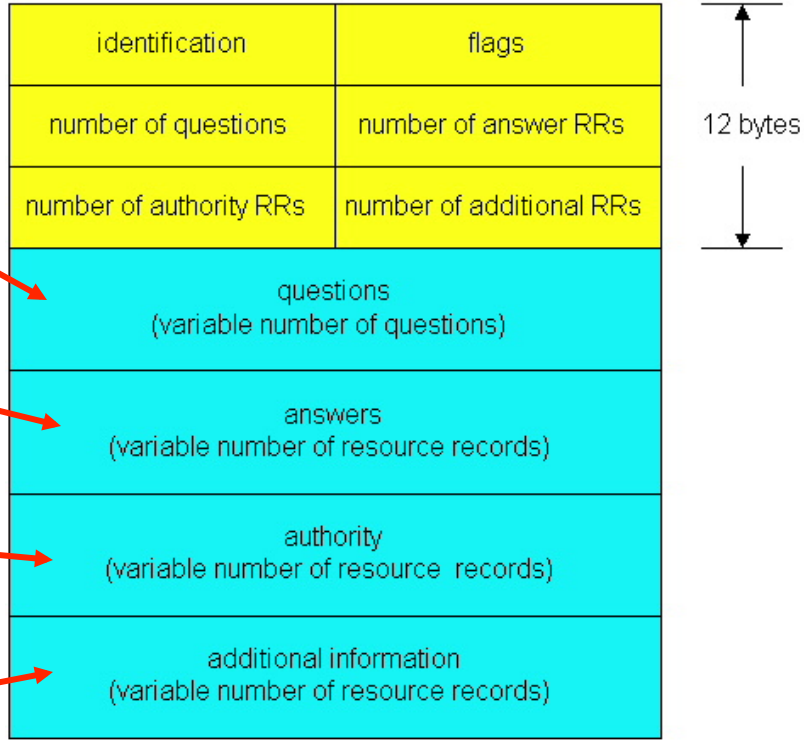
DNS protocol, messages

Name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



Inserting records into DNS

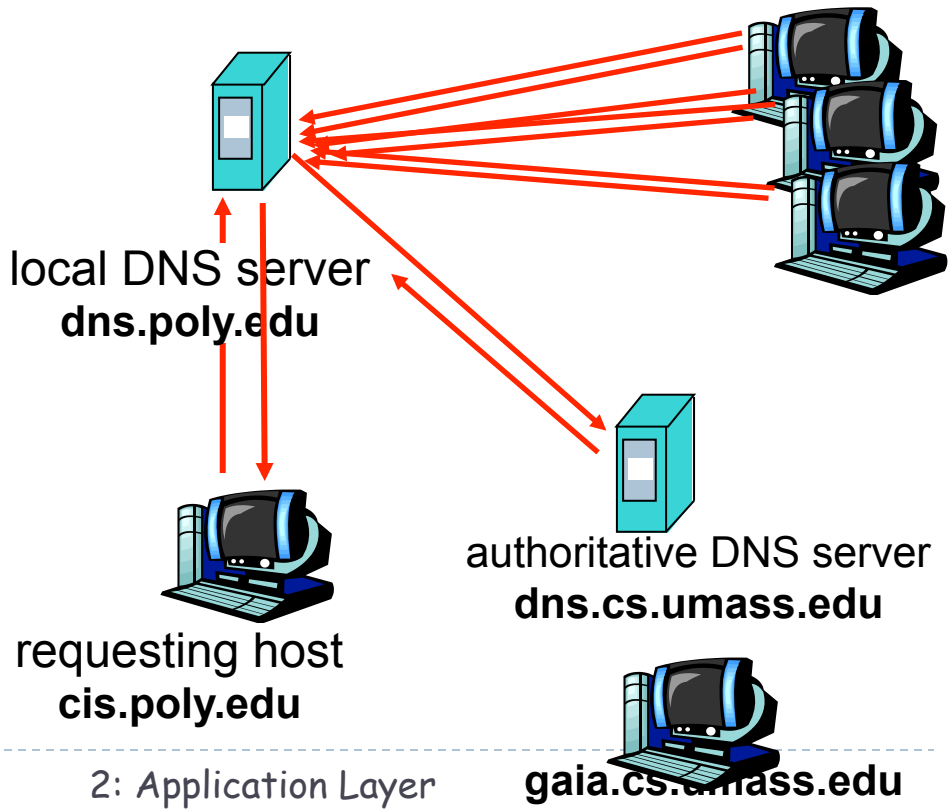
- example: new startup "Network Utopia"
- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts two RRs into com TLD server:

```
(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 MX record for networkutopia.com
- How do people get IP address of your Web site?

DNS Poisoning

- DNS uses UDP
- Source IP address can be spoofed
- Responses are accepted with a "First Comes First Wins" policy, subsequent
  - Only check is on TXID
- What consequences?



DNSSEC

- DNS "patches"
  - Port randomization
  - 0x20-Bit encoding

- Better solution: DNSSEC
  - Responses are digitally signed
  - They can be verified by following a *chain of trust* anchored at the roots
  - Not yet fully deployed

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

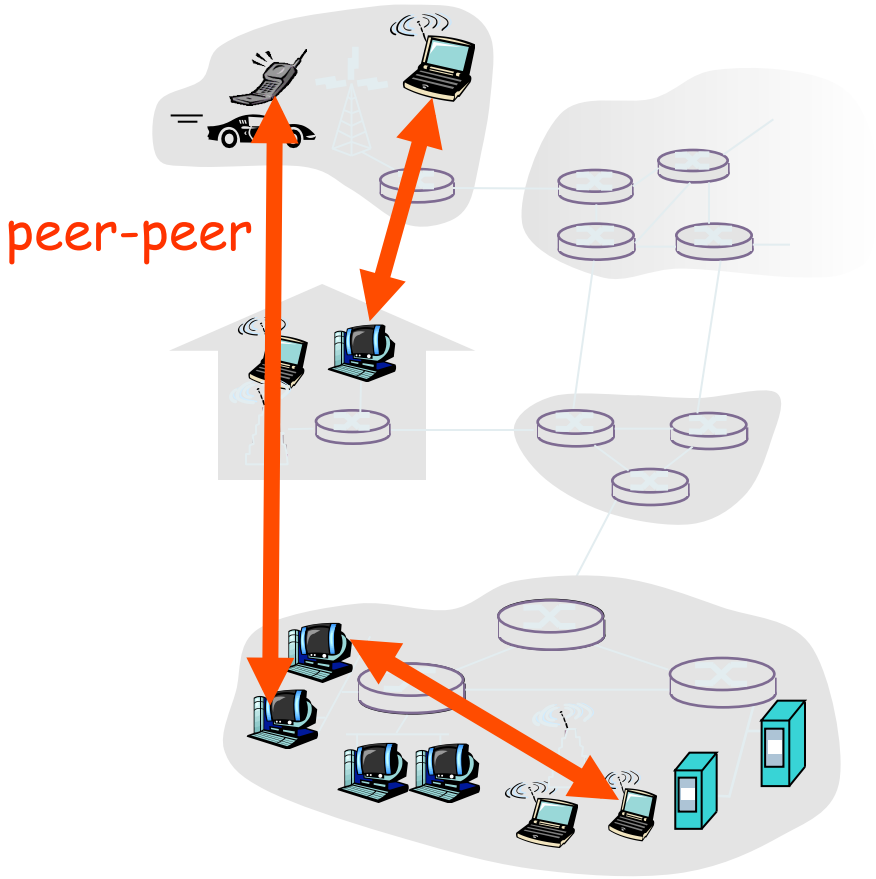
2.6 P2P applications

2.7 Socket programming with TCP

2.8 Socket programming with UDP

Pure P2P architecture

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



Three topics:

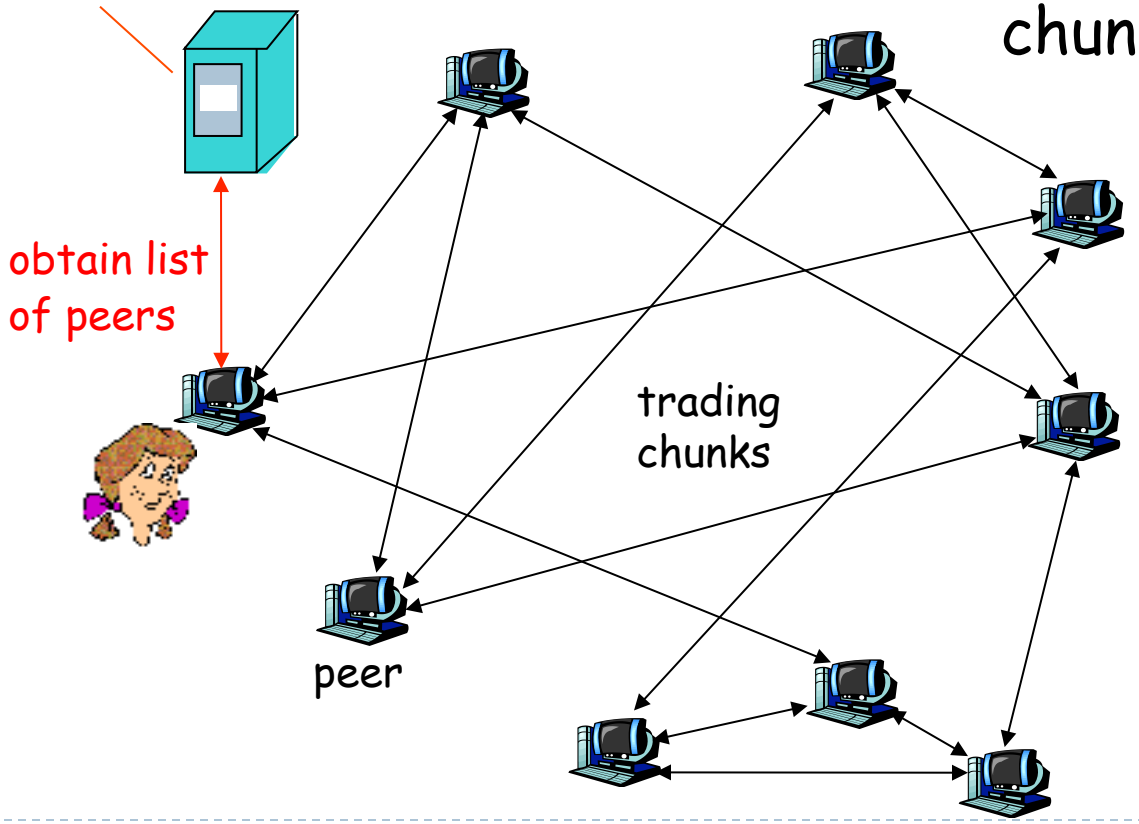
- file distribution
- searching for information
- case Study: Skype

File distribution: BitTorrent ****

P2P file distribution

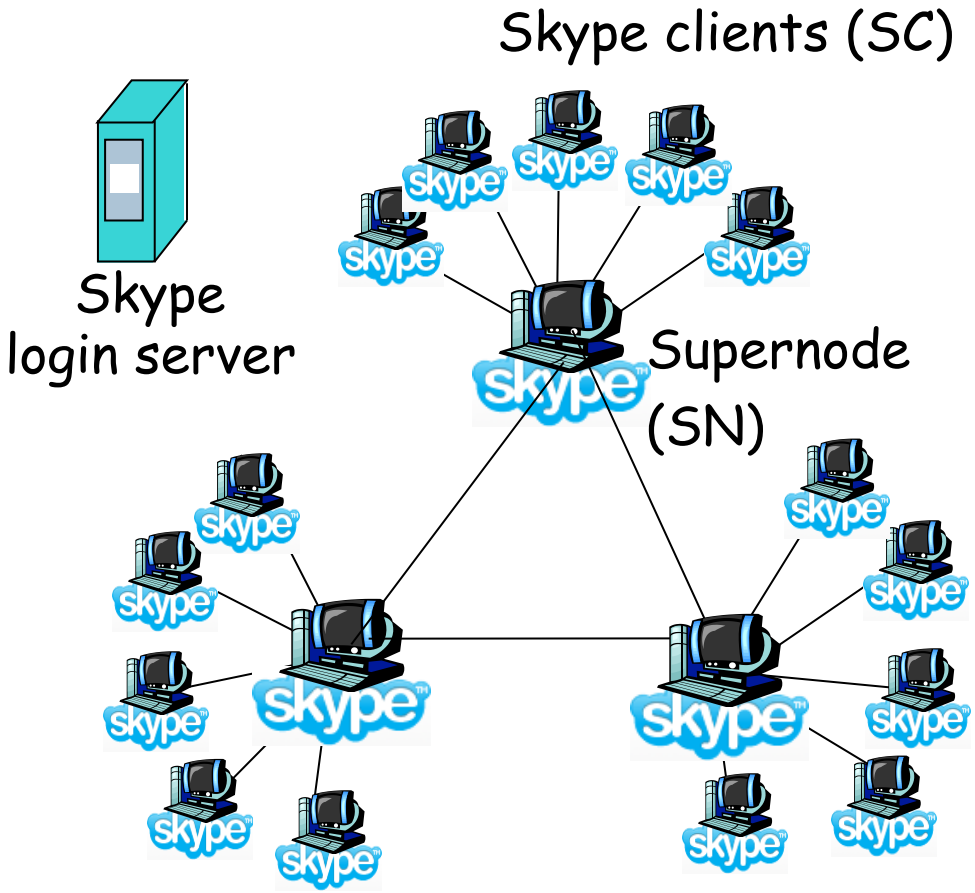
tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



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