CSE 3214: Computer Network Protocols and Applications
–Application Layer

Dr. Peter Lian, Professor
Department of Computer Science and Engineering
York University
Email: peterlian@cse.yorku.ca
Office: 1012C Lassonde Building
Course website:
http://wiki.cse.yorku.ca/course_archive/2012-13/W/3214

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

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
- creating network applications
  - socket API

Some network apps

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

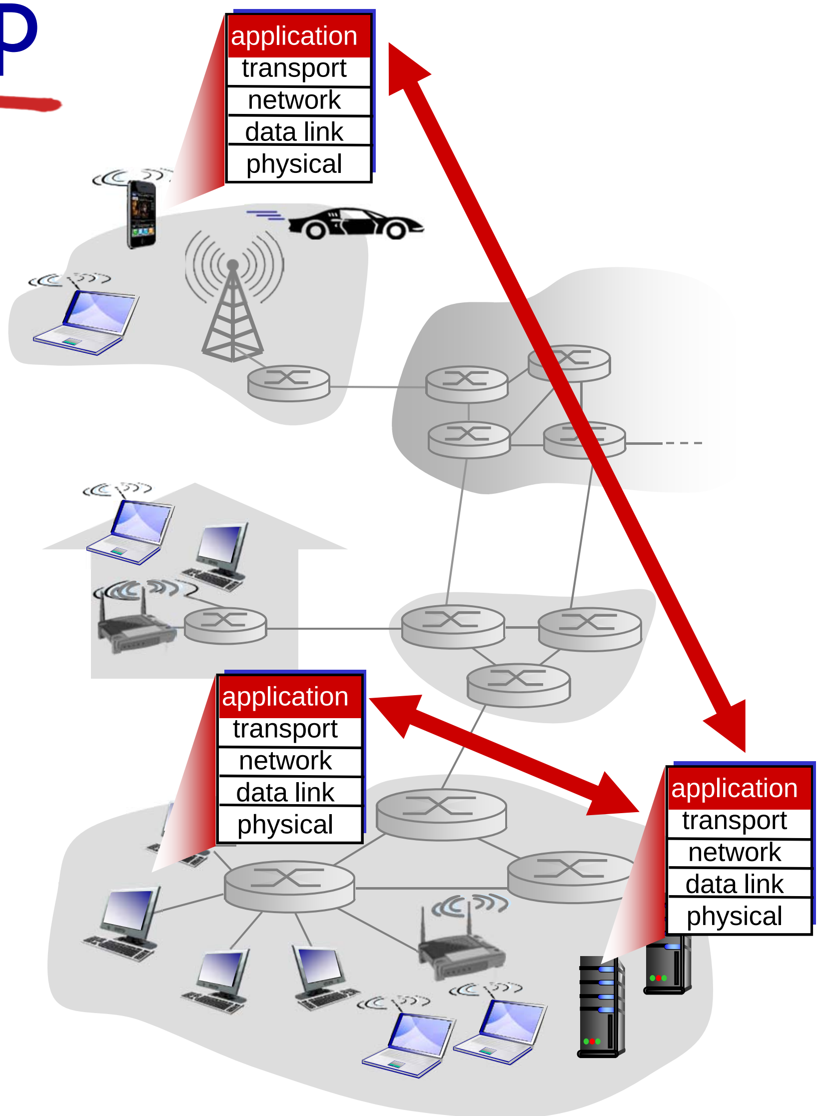
Creating a network app

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

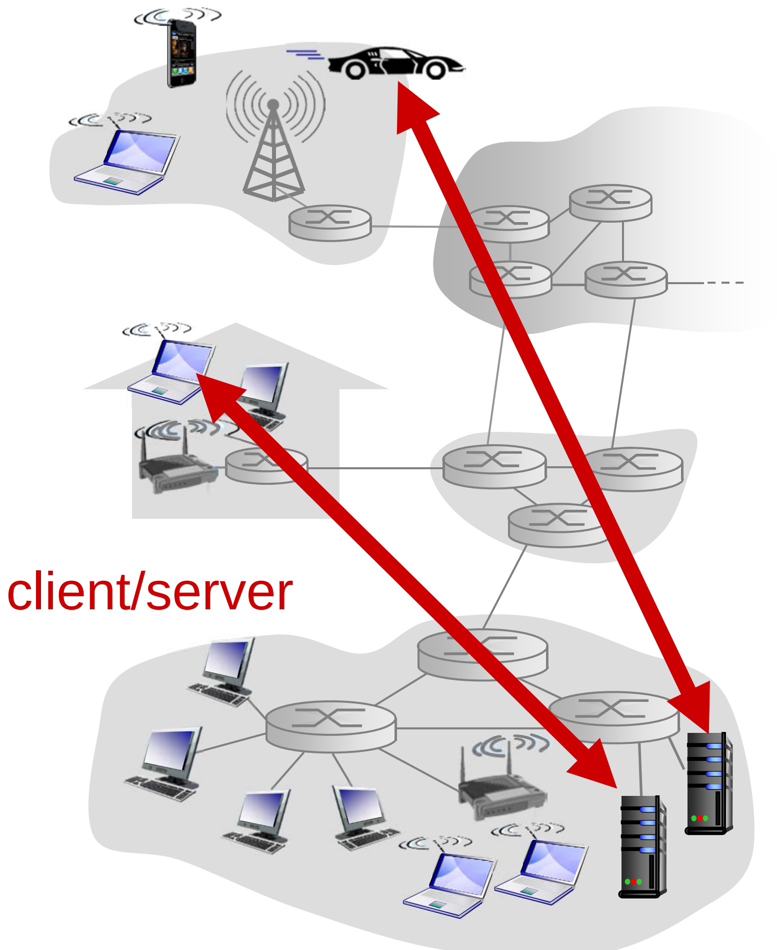


Application architectures

possible structure of applications:

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

Client-server architecture



server:

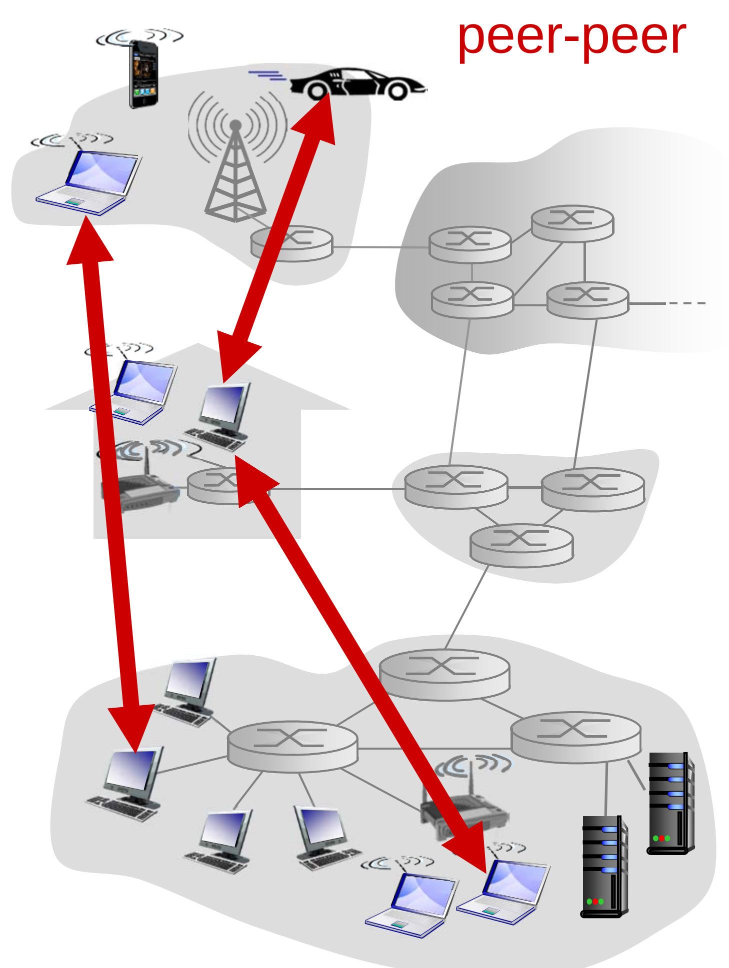
- always-on host
- permanent IP address
- data centers for scaling

clients:

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

P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



Processes communicating

process: program running within a host

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

clients, servers

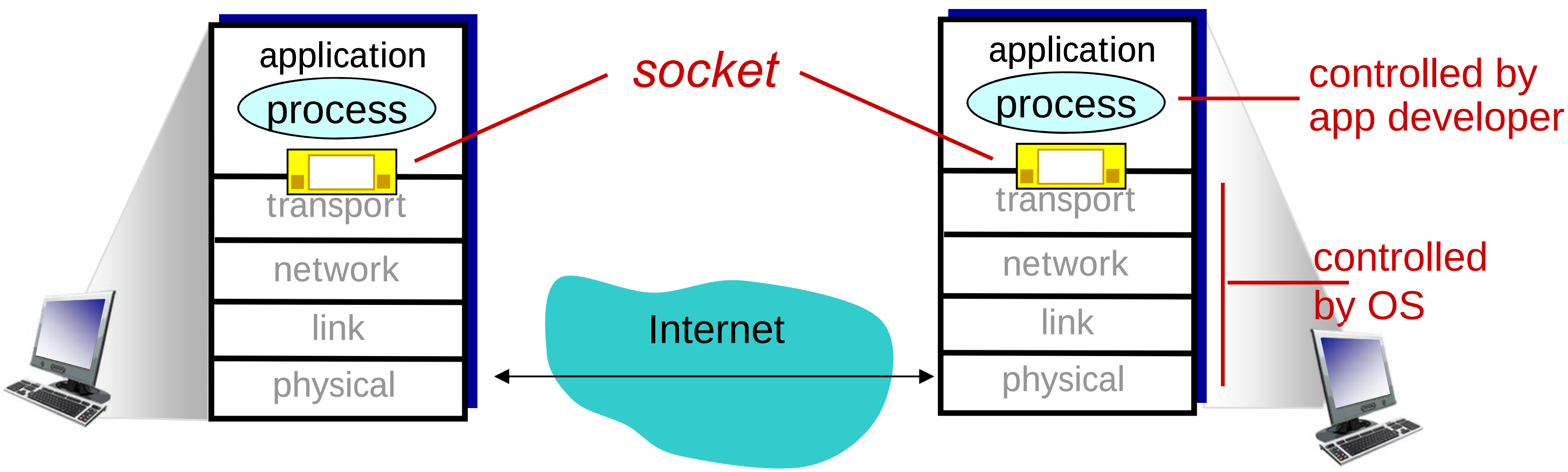
client process: process that initiates communication

server process: process that waits to be contacted

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

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 to deliver message to socket at receiving 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…

App-layer protocol defines

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

open protocols:

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

proprietary protocols:

- e.g., Skype

What transport service does an app need?

data integrity

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

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 to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

security

- encryption, data integrity, …

Transport service requirements: common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	
interactive games	loss-tolerant	few kbps up	yes, few secs
text messaging	no loss	elastic	yes, 100's
			msec yes and no

Internet transport protocols services

TCP service:

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

UDP service:

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

Q: why bother? Why is there a UDP?

Internet 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	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Securing TCP

TCP & UDP

- no encryption
- cleartext passwds sent into socket traverse Internet in cleartext

SSL

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

SSL is at app layer

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

SSL socket API

- cleartext passwds sent into socket traverse Internet encrypted

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

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,* e.g.,

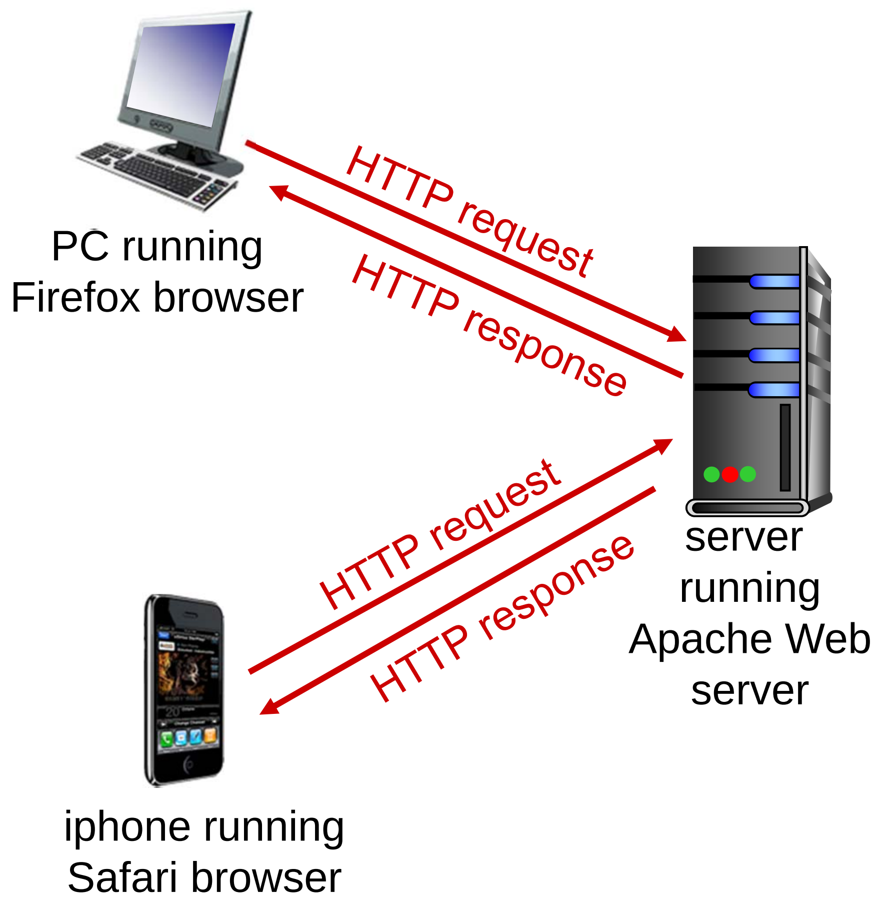
```
www.someschool.edu/someDept/pic.gif
```

host name — path name

HTTP overview

HTTP: hypertext transfer protocol

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



HTTP 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
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

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

Non-persistent 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

Non-persistent 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

time

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

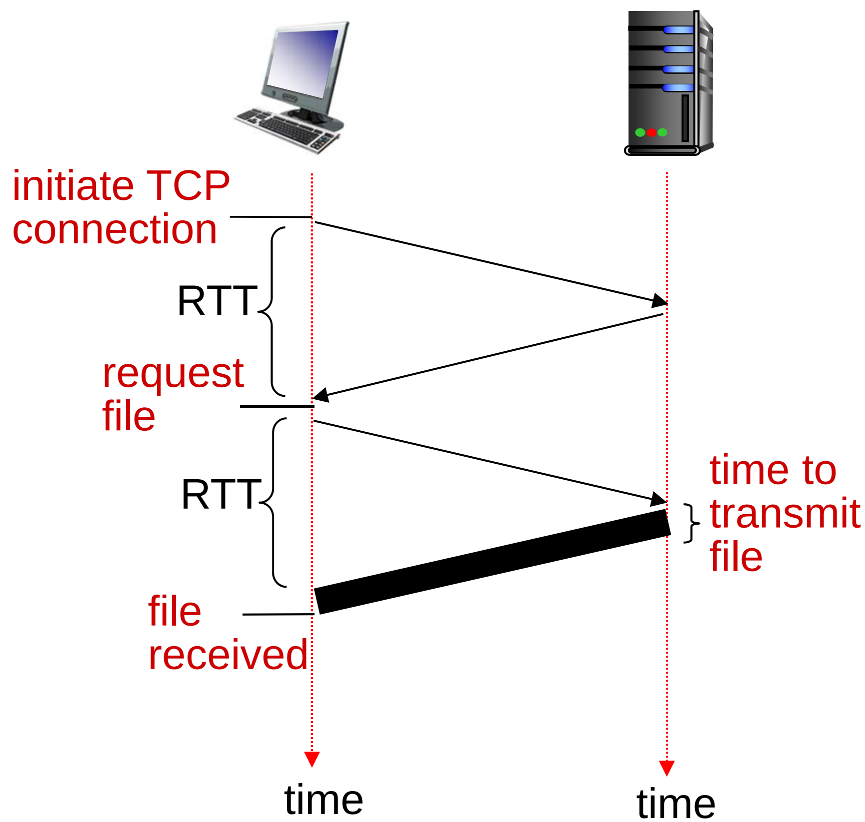
Non-persistent HTTP: response time

RTT (Round-trip time): time for a small packet to travel from client to server and back

HTTP 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
- non-persistent HTTP response time =

  2RTT+ file transmission time



Persistent HTTP

non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

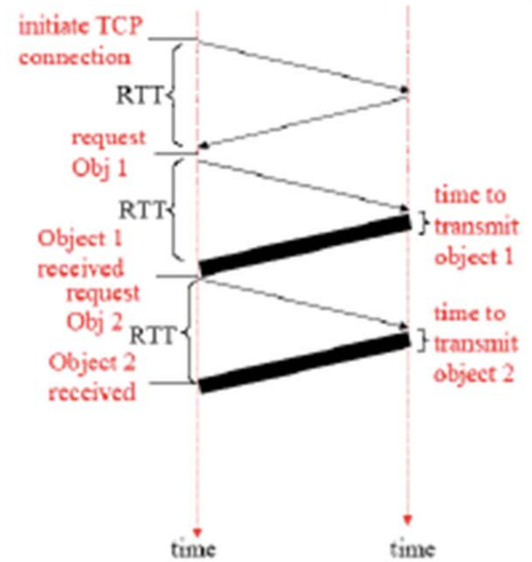
persistent HTTP:

- 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

Persistent HTTP Connection

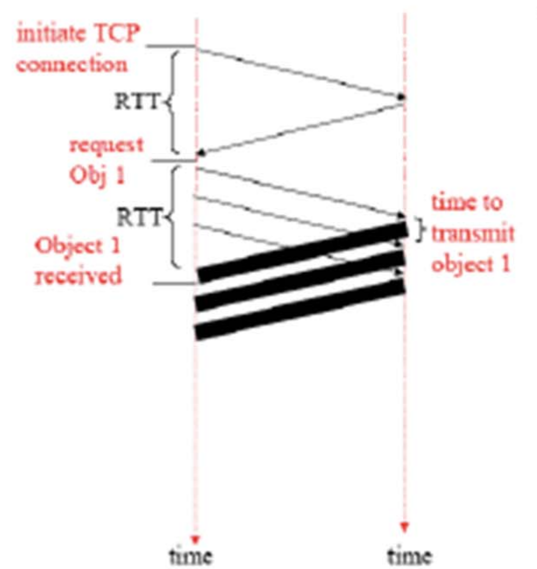
- 2 versions
  - Without pipelining – HTTP client issues a new request only when the previous response/object has been received.

retrieval time per object = RTT + transmission time



  - With pipelining – HTTP client issues a request as soon as it encounters a reference

one RTT for all objects



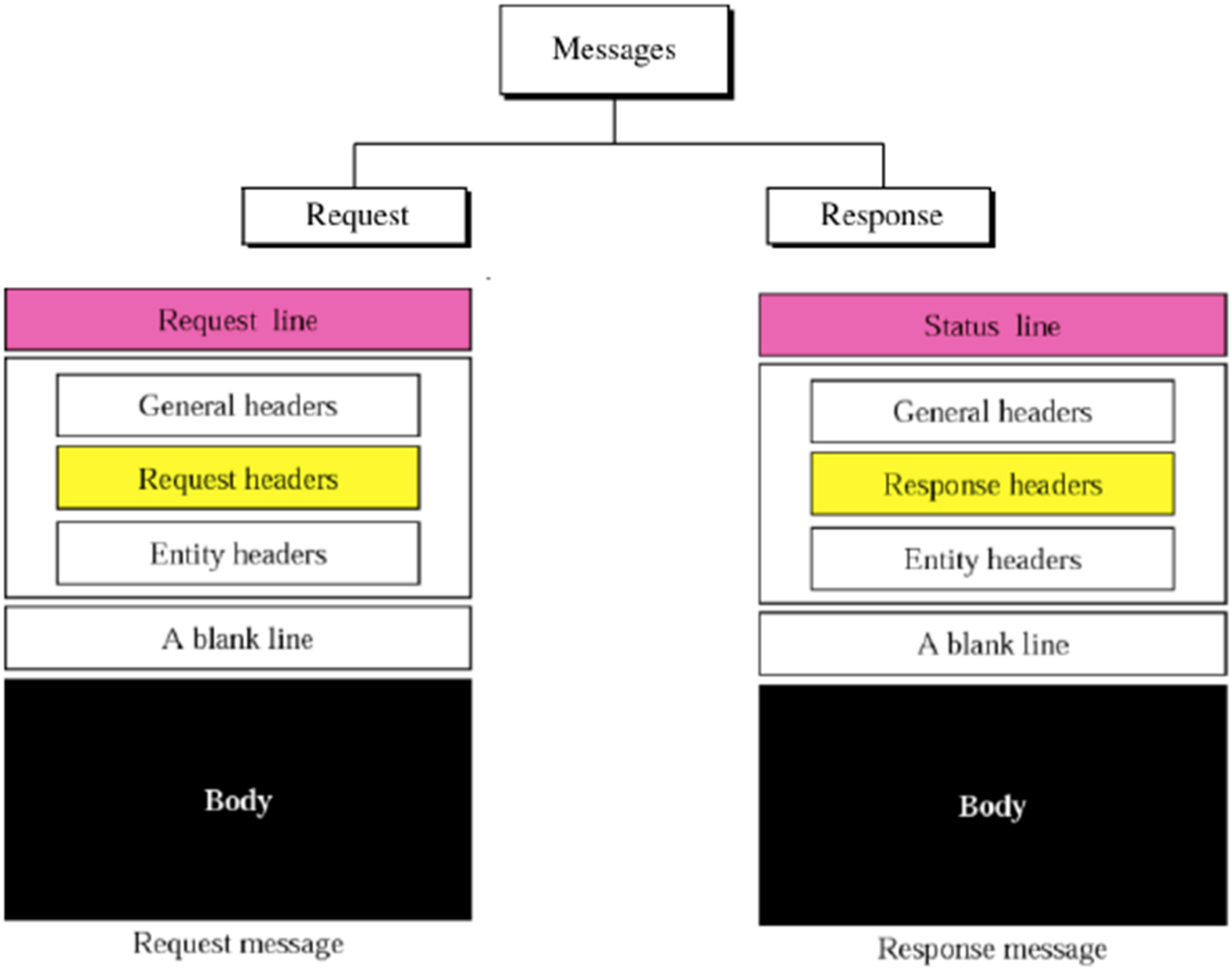
Non-Persistent vs. Persistent: Example

- Assume a Web page consists of 1 base HTML page and 10 images (each of size L bits). Data rate on the link is R bps. What is the overall retrieval time in case of:

  (a) non-persistent HTTP:

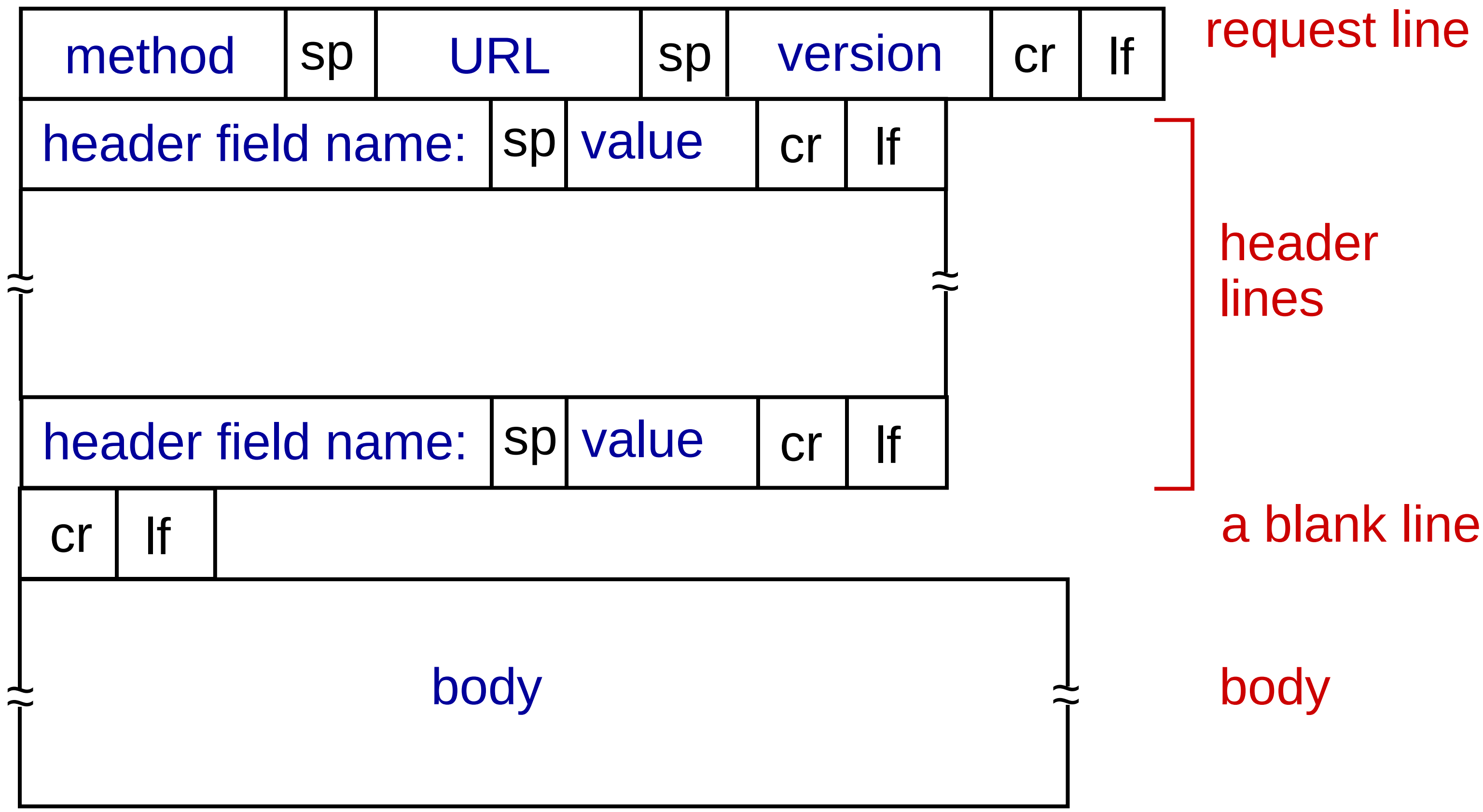
  (b) persistent HTTP with pipeline:

HTTP message format



HTTP Request Message

- From client to server
- General format



Methods

- 3 methods in HTTP/1.0: GET, POST, HEAD
- Additional 2 methods in HTTP/1.1: PUT, DELETE
  - GET – retrieves a document specified in the URL field from server
  - HEAD – get some information about document but not document itself
  - POST – provides some information for server, e.g. input to server when fills a form
  - PUT – uploads file in entity body to path specified in URL field
  - DELETE – deletes file specified in the URL field

HTTP request message example

carriage return character

line-feed character

request line
(GET, POST,
HEAD commands)

header
lines

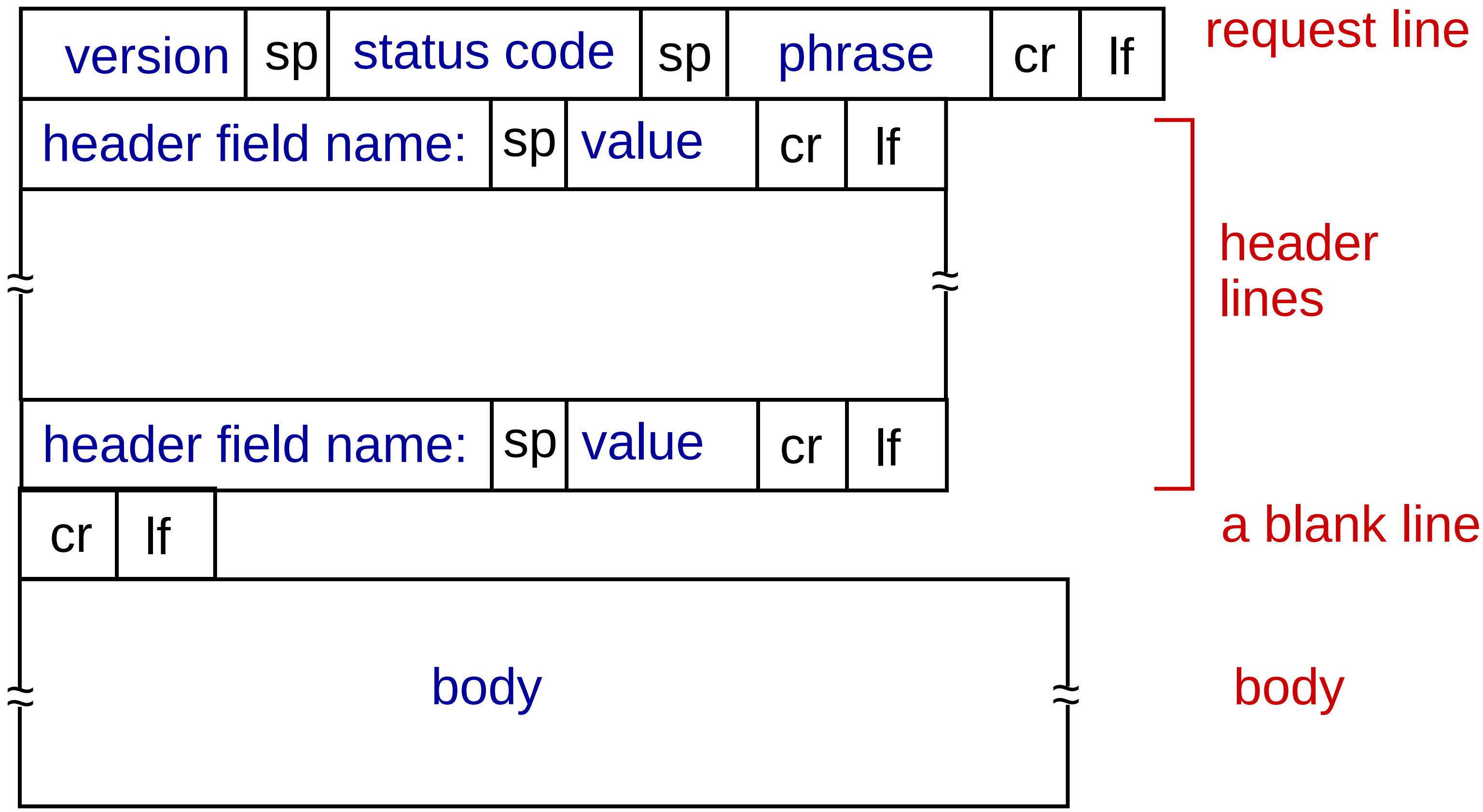
carriage return,
line feed at start
of line indicates
end of header lines

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

empty
body

HTTP Response Message

- From server to client
- General format



HTTP response status codes

- status code is 3-digit integer that indicates the response to a received request; status phrase gives short textual explanation of the status code

`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`

HTTP Response Message Example

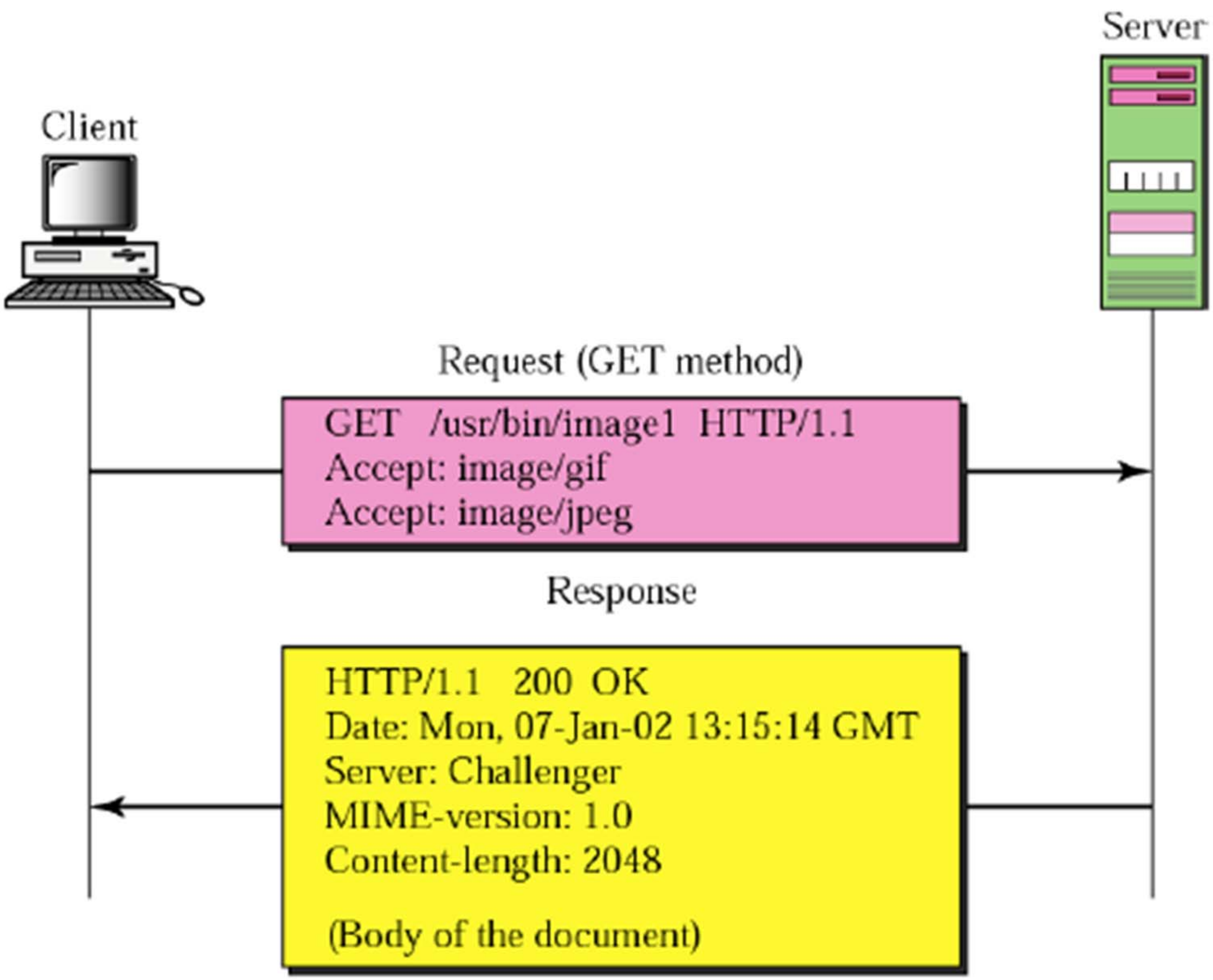
status line (protocol status code status phrase)

```
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 ...
```

header lines

data, e.g., requested HTML file

HTTP messaging example



HTTP Headers

- Exchange additional information between the client and the server

header field name:	sp	value	cr	lf

- General Header – gives general information about the message and can be present in both a request and response

Header	Description
cache-control	Specifies info about caching
connection	Specifies whether connection should be closed or not
date	Shows the date and time at which the message originated
MIME-version	Shows the MIME version used
...	

HTTP Request Headers

- REQUEST HEADER – can be present only in a request message – it specifies the client's configuration and the client's preferred document format

Header	Description
accept	Shows the media format the client can accept
accept-language	Shows the language the client can accept
host	Specifies the Internet host of the requested resource
if-modified-since	Send the document if newer than specified date
user-agent	Identifies the client program
…	

HTTP Response Header

- RESPONSE HEADER – can be present only in a response message – it specifies the server's configuration and special information about the request

Header	Description
public	Shows the list of HTTP methods supported by this server
retry-after	Shows how long the service is expected be unavailable
server	Shows the server name and version number
set-cookie	Define a name – value pair associated with this URL
…	

HTTP Entity Header

- ENTITY HEADER – gives information about the body of the document/message – mostly present in response message

Header	Description
content-encoding	Specifies the encoding scheme
content-language	Specifies the language
content-length	Shows the length of the document
content-type	Specifies the media type
expires	Gives the date and time when contents may change
location	Specifies the location of the created or moved document
…	

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.cse.yorku.ca 80
```

opens TCP connection to port 80 (default HTTP server port) at cse website. anything typed in sent to port 80 at www.cse.yorku.ca

2. type in a GET HTTP request:

```
GET /cshome/index.html HTTP/1.1
Host: www.cse.yorku.ca
```

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

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

Trying out HTTP (client side) for yourself

eleliany — bash — 102×31

```
Yong-MacBook-Air:~ eleliany$ telnet www.cse.yorku.ca 80
Trying 130.63.92.30...
Connected to gold-cse.cse.yorku.ca.
Escape character is '^]'.
GET /cshome/index.html HTTP/1.1
Host: www.cse.yorku.ca

HTTP/1.1 200 OK
Date: Sun, 13 Jan 2013 19:39:38 GMT
Server: Apache/2.2.22 (Unix) DAV/2 mod_ssl/2.2.22 OpenSSL/1.0.0d PHP/5.2.17
X-Powered-By: PHP/5.2.17
Transfer-Encoding: chunked
Content-Type: text/html

206d
<html>
    <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

        <meta name="Author" content="York University">

        <meta name="GENERATOR" content="Palomino WebPal/CMS - www.palominosys.com">

        <meta name="Classification" content="">
        <script src="../_global/jquery.min.js"></script>
        <title>Department of Computer Science and Engineering - Welcome - Home
        </title>
          <script type="text/javascript" src="../_javascript/oodomimagerollover.js"></script>
          <script type="text/javascript" src="../_javascript/webpal_helpers.js"></script>
          <script language="javascript" type="text/javascript">
            function handleError()
```

Cookie

- HTTP is a stateless protocol – server forgets about each client as soon as it delivers response
  - Stateless behavior is an issue when:
    - Server wants to have accurate count of site visitors
    - Server wants to restrict user access, etc.
    - Server wants to personalize pages for each client, or remember selections they made
- Cookie Technology allows site to keep track of users
  - A cookie is a short piece of data, not code. It is not an executable program and cannot directly harm the machine

User-server state

many Web sites use cookies

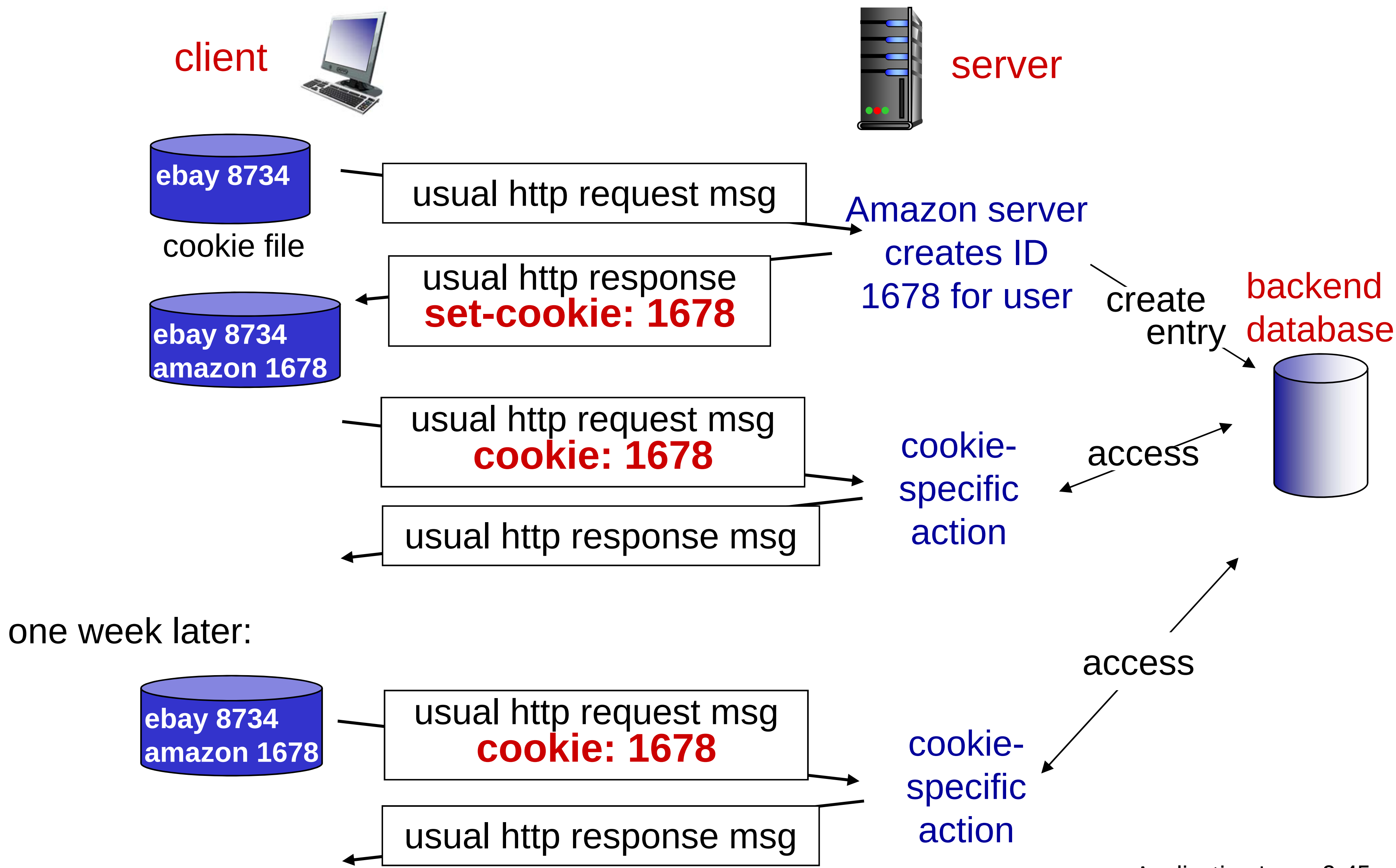
four components:

1) For new user, server adds Set-Cookie header to its response with an identifier
2) Client stores the ID in a cookie file kept on its disk and managed by user's browser
3) Back-end database keeps the ID on server
4) Client uses the ID in all subsequent requests

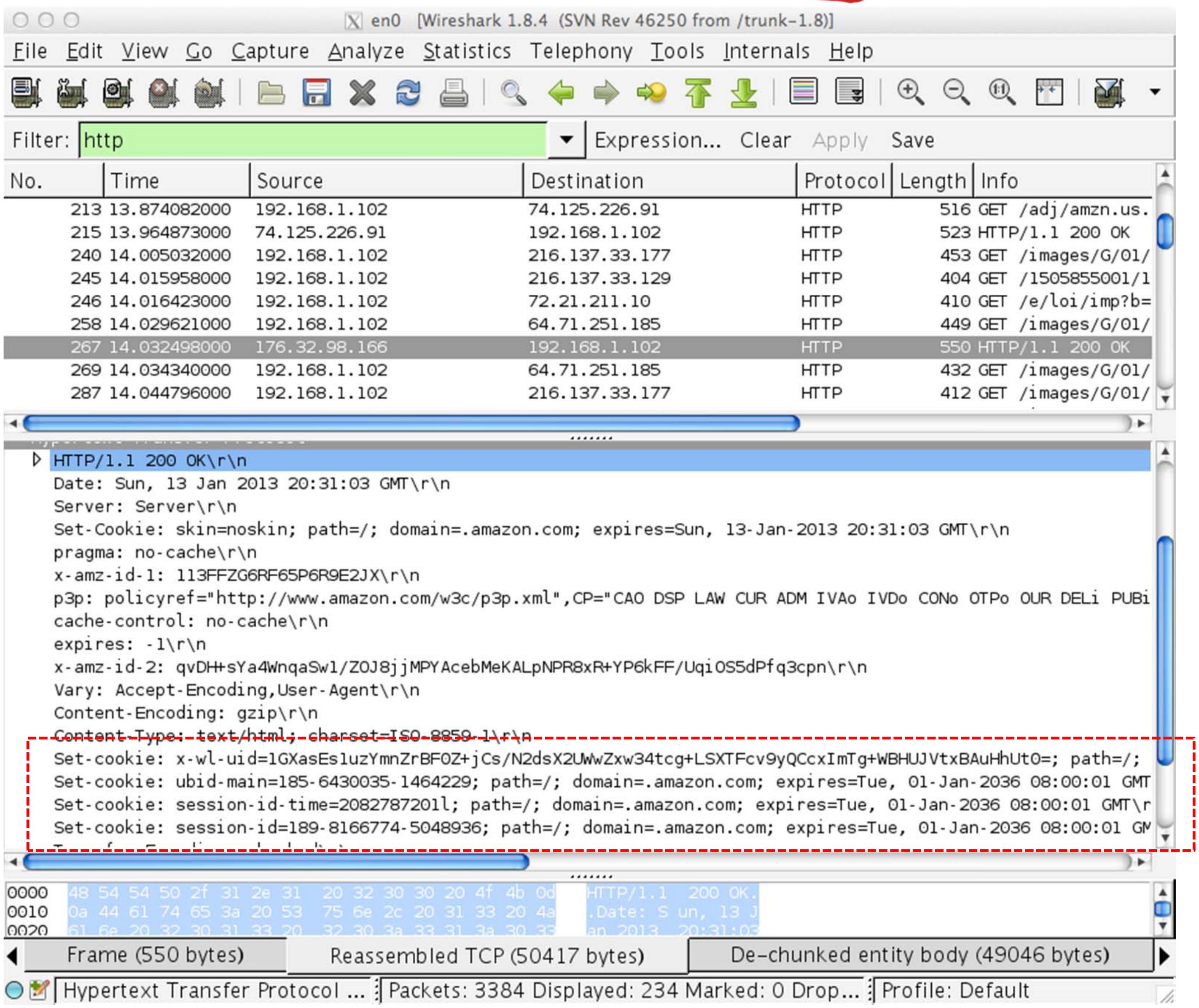
example:

- 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"



Cookies Example



Issues with Cookies

what cookies can be used for:

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

aside

cookies and privacy:

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

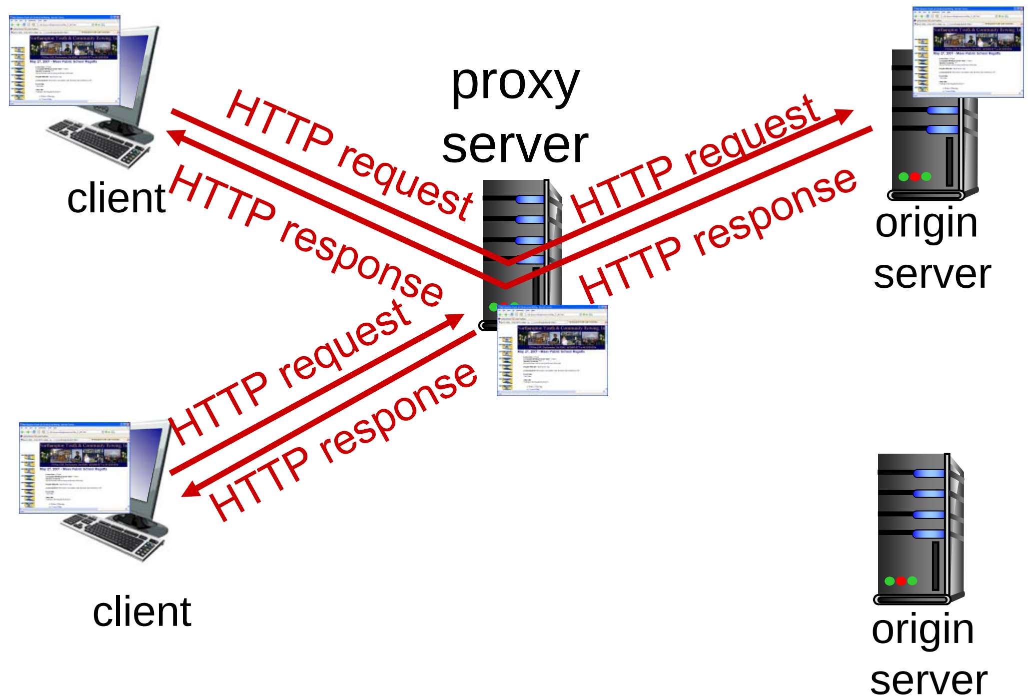
Issues with cookies:

- Undesirable cookies: any server can set a cookie for any reason. User may not even be informed that this is happening

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
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

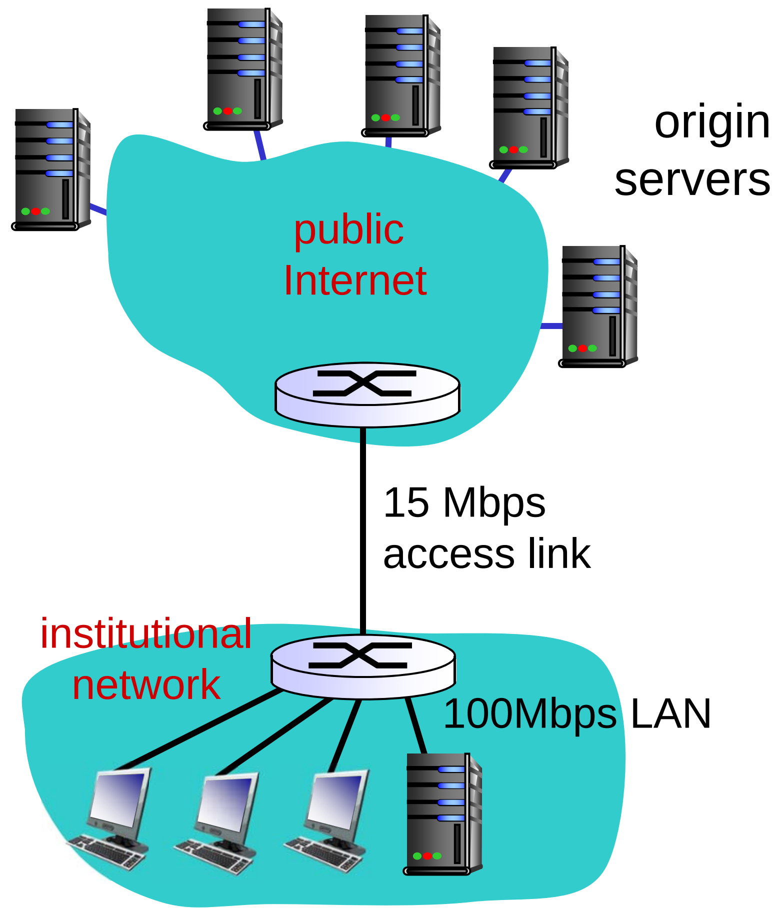
Caching example:

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

consequences:

- LAN traffic intensity=(15req/s*1Mb/req)/100Mbps =0.15
- WAN traffic intensity=(15req/s*1Mb/req)/15Mbps= 1 *problem!*
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + msecs



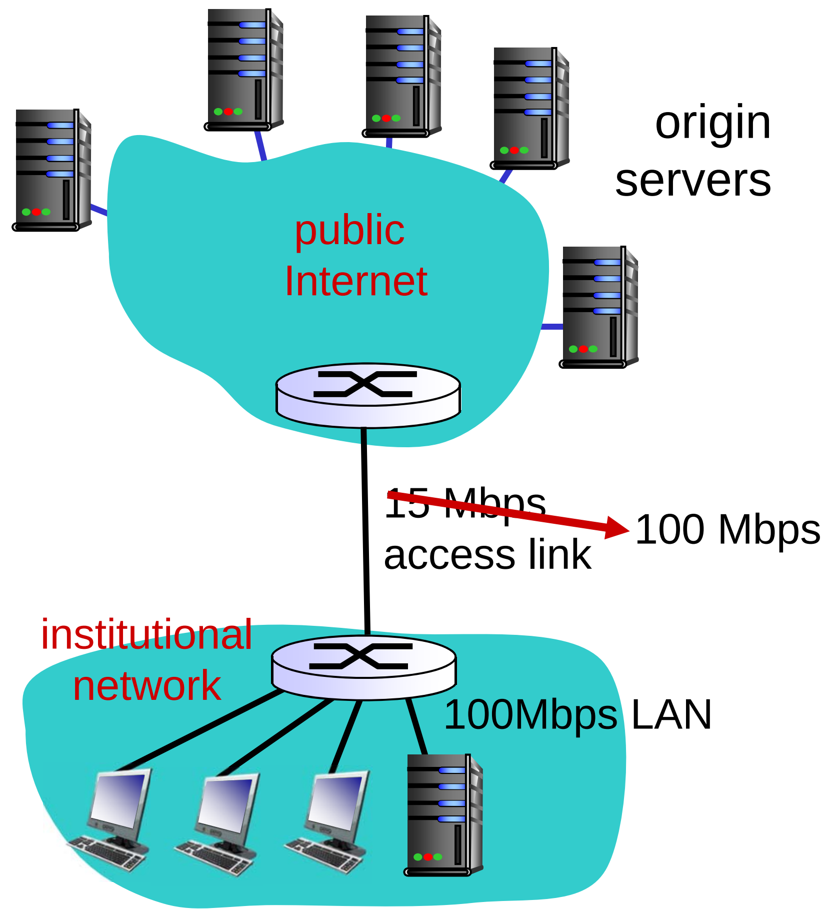
Caching example: fatter access link

assumptions:

- avg object size: 1Mbits
- avg request rate from browsers to origin servers:15/sec
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~15Mbps~~ → 100Mbps

consequences:

- LAN TI = 0.15
- WAN TI = ~~1~~ → 0.15
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ + msecs → msecs



Cost: increased access link speed (not cheap!)

Caching example: install local cache

assumptions:

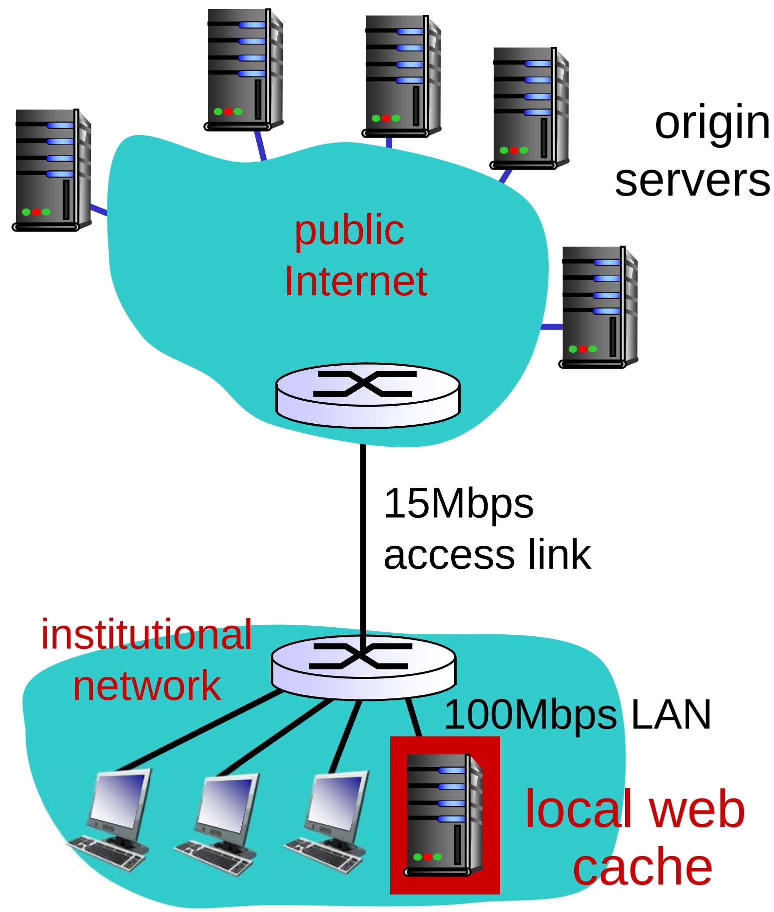
- avg object size: 1 Mbits
- avg request rate from browsers to origin servers:15/sec
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

consequences:

- LAN TI: 0.15
- access link utilization = 1
- total delay = ?

How to compute link utilization, delay?

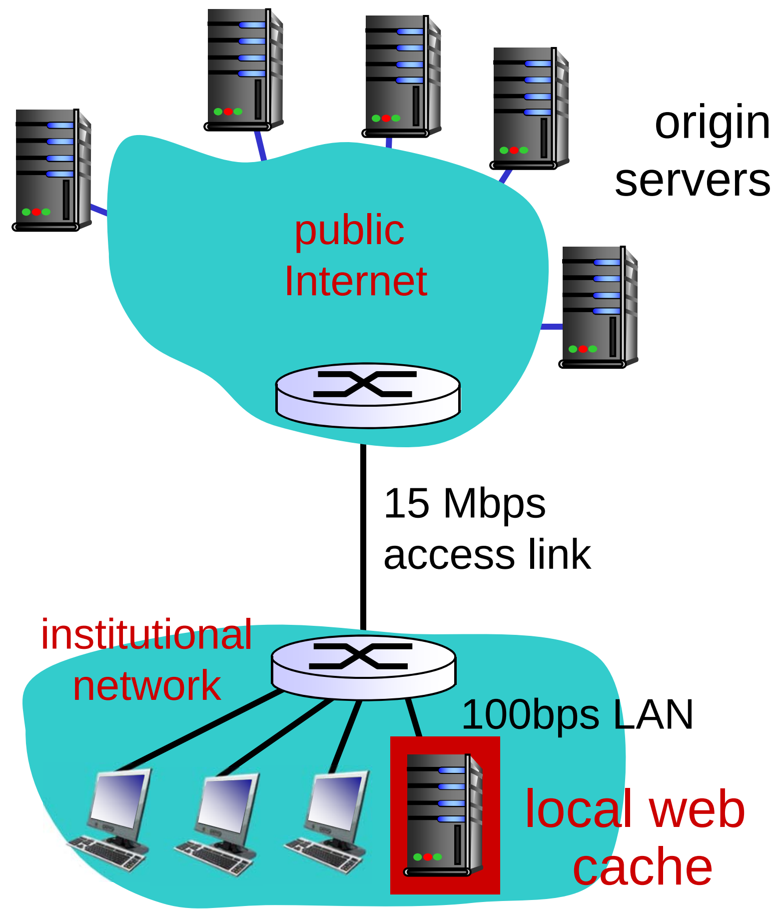
Cost: web cache (cheap!)



Caching example: install local cache

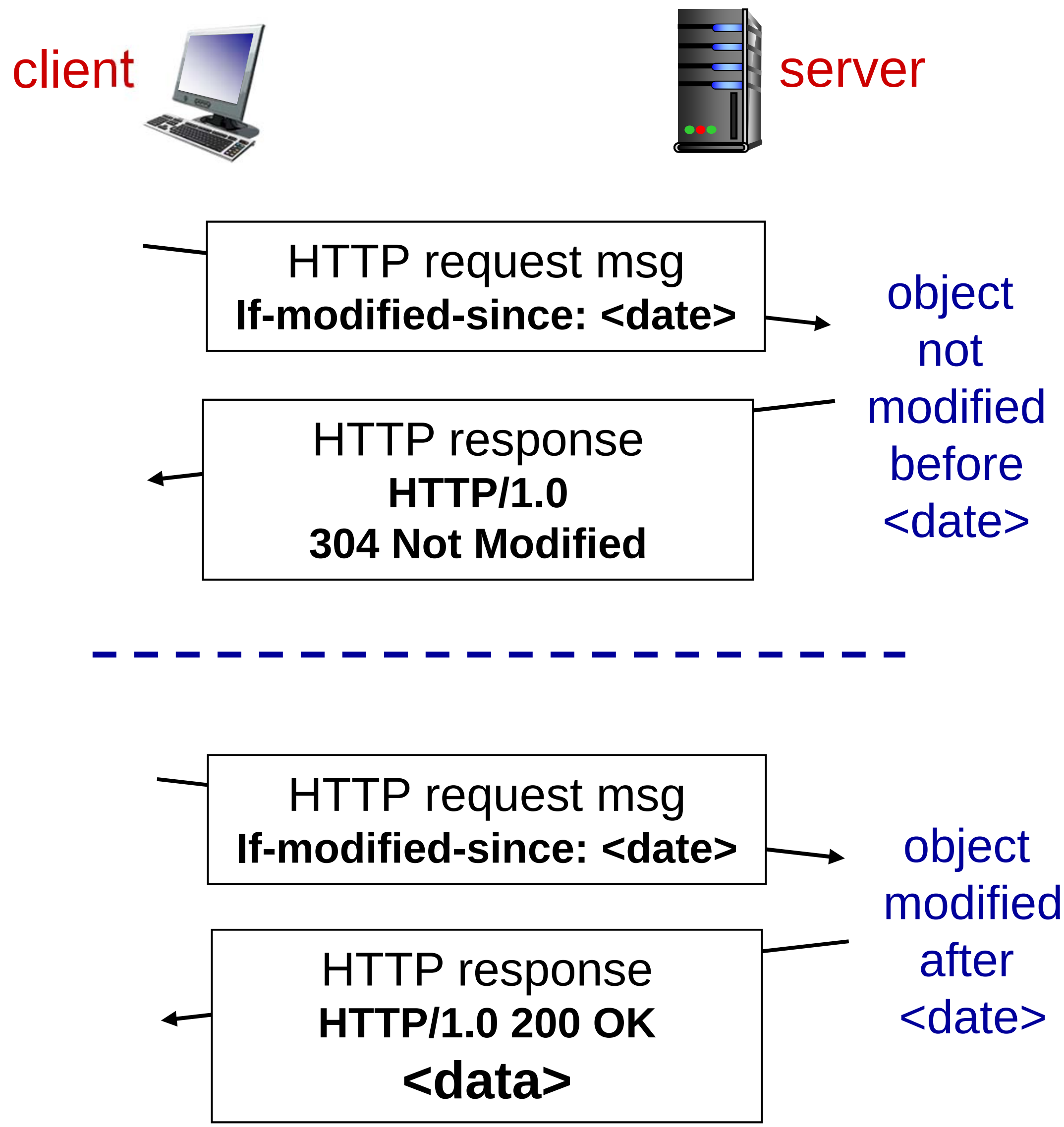
Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4(typical 0.2~0.7)
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link = 0.6*15req/s*1Mbps = 9 Mbps
  - TI = 9/15 = .6
- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs)
  - = ~ 1.2 secs
  - less than with 100 Mbps link (and cheaper too!)



Web Cache Challenge

- *Goal:* do not send object if cache has up-to-date cached version
- What if cached data is changed?
- Solution: use conditional GET in HTTP message
  `If-modified-since: <date>`
- *server:* response contains no object if cached copy is up-to-date:
  `HTTP/1.0 304 Not Modified`



Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

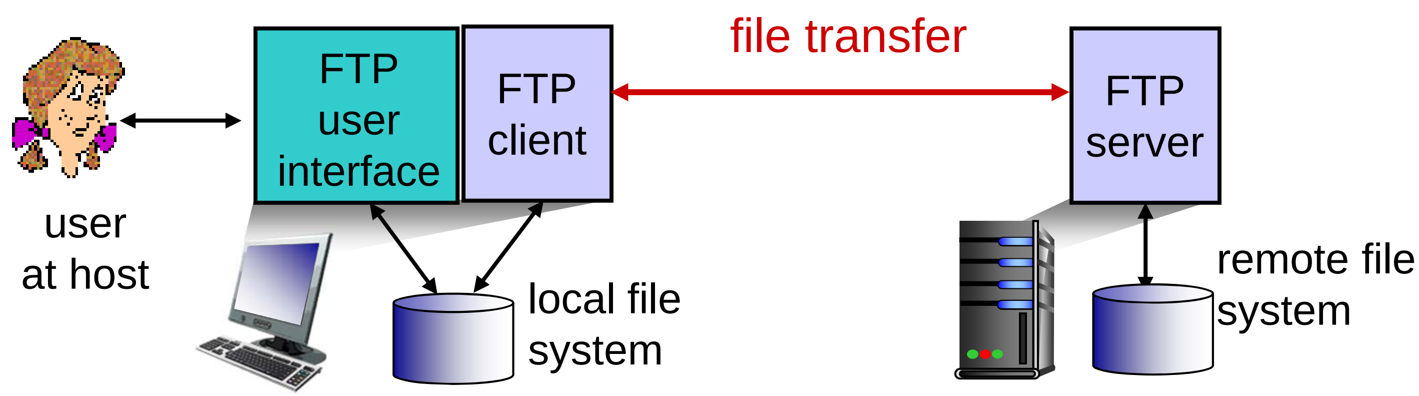
2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

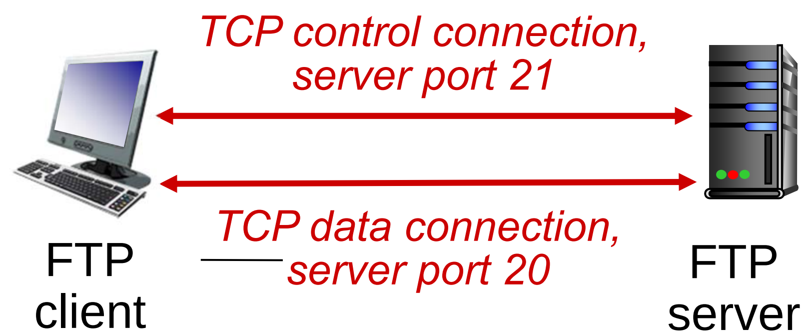
FTP: the file transfer protocol



- 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, using TCP
- client authorized over control connection
- client browses remote directory, sends commands over control connection
- when server receives file transfer command, such as get or put, *server* opens 2nd TCP data connection (for file) *to* client
- after transferring one file, server closes data connection



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

FTP commands

sample commands:

- asc - sent as ASCII text over control channel
- bin – sent as binary
- ls – list of file
- cd – change directory
- get filename – retrieves a file from remote host
- put filename stores file onto remote host
- ye - quit

- Examples
  - ftp my@cse.yorku.ca
  - ls –al
  - cd prism
  - get index.html
  - put myfile

FTP Example

eleliany — ftp — 102×31

```
Yong-MacBook-Air:~ eleliany$ ftp peterlian@cse.yorku.ca
Connected to cse.yorku.ca.
220-York University Department of Computer Science and Engineering FTP Server
220 FTP Server ready.
331 Password required for peterlian
Password:
230 User peterlian logged in
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
229 Entering Extended Passive Mode (|||48402|)
150 Opening ASCII mode data connection for file list
drwx------   2 peterlian faculty      4096 Dec 10 13:10 prism
drwx--x--x   2 peterlian faculty      4096 Dec 10 13:10 www
226 Transfer complete
ftp> cd prism
250 CWD command successful
ftp> ls -al
229 Entering Extended Passive Mode (|||43956|)
150 Opening ASCII mode data connection for file list
drwx------   2 peterlian faculty      4096 Dec 10 13:10 .
drwx--x--x   4 peterlian faculty      4096 Jan  8 12:22 ..
-rwx------   1 peterlian faculty      1040 Dec 10 13:10 .cshrc
226 Transfer complete
ftp> get .cshrc
local: .cshrc remote: .cshrc
229 Entering Extended Passive Mode (|||4929|)
150 Opening BINARY mode data connection for .cshrc (1040 bytes)
100% |*********************************************************|  1040        1.75 MiB/s    00:00 ETA
226 Transfer complete
1040 bytes received in 00:00 (65.63 KiB/s)
```

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

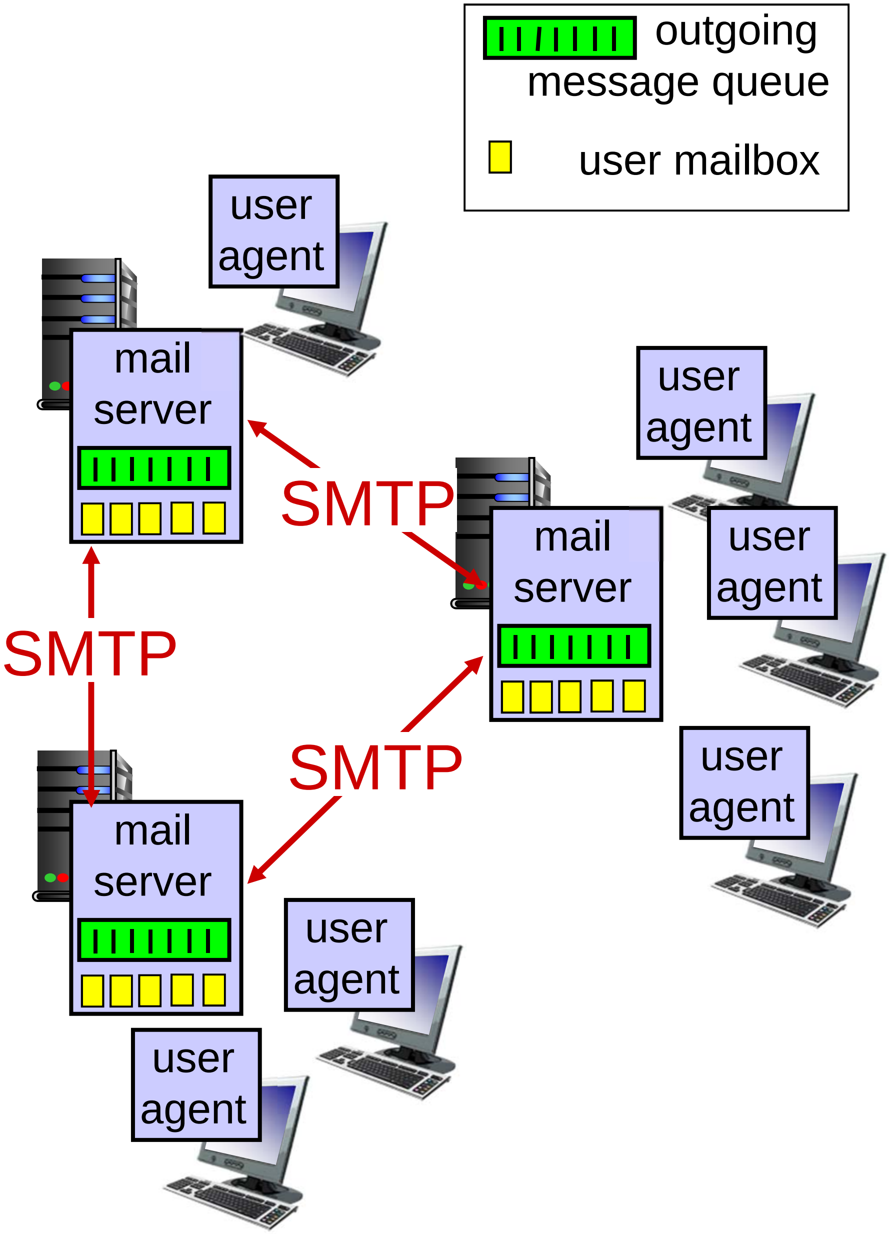
Electronic mail

Three major components:

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

User Agent

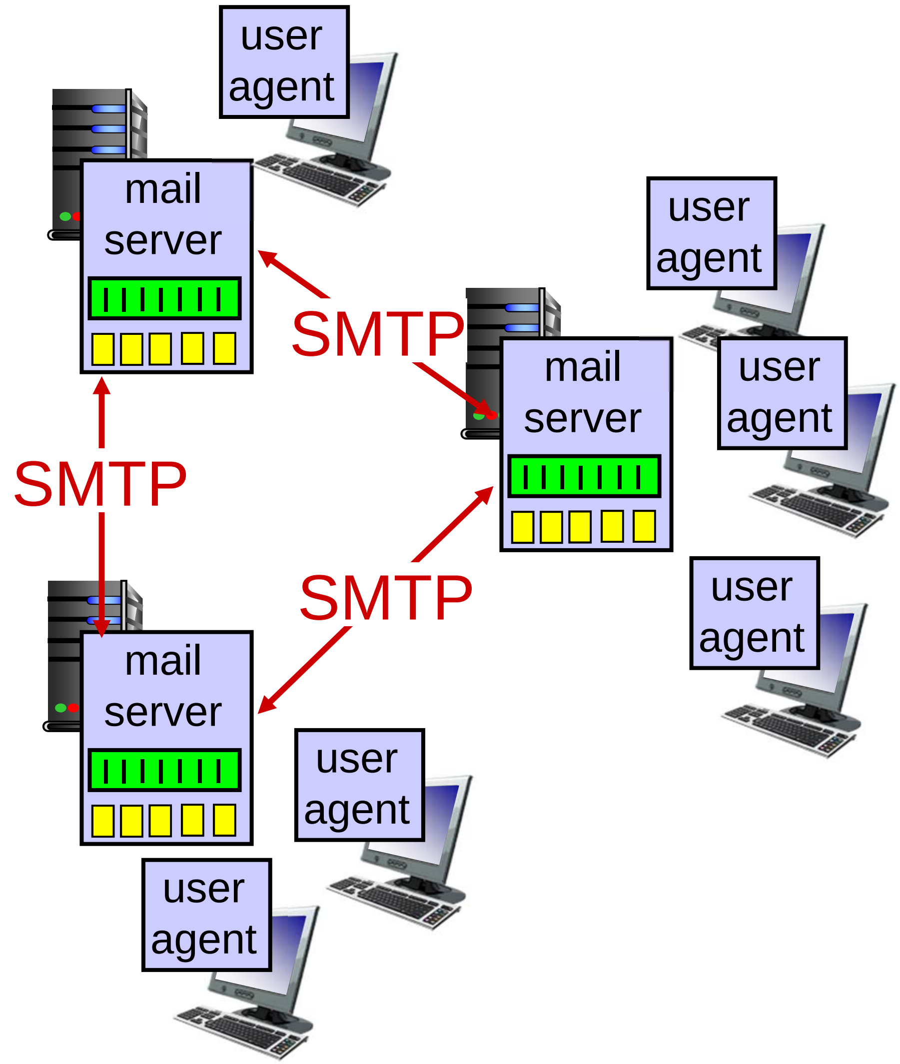
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



Electronic mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server

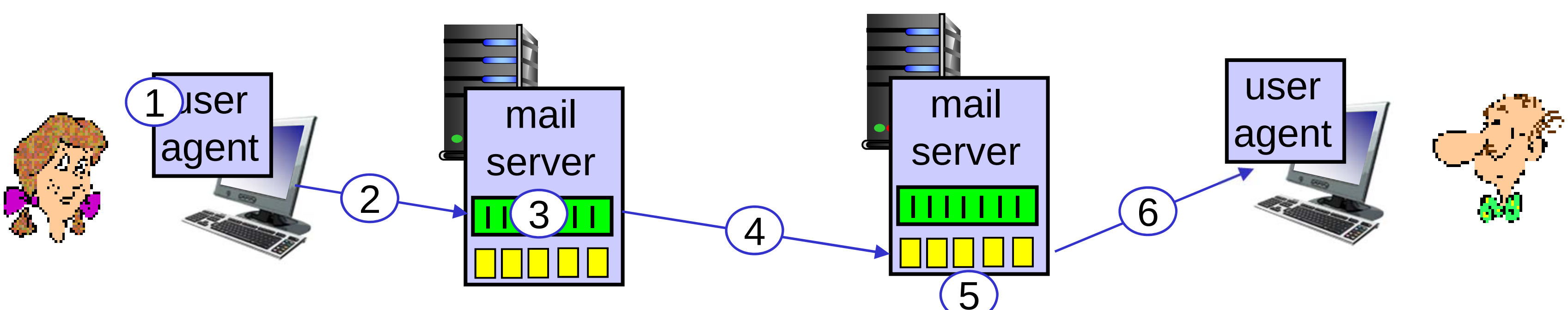


Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP, FTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCI

Scenario: Alice sends message to Bob

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



Sample SMTP interaction

```
S-SMTP server, C-SMTP client

    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
```

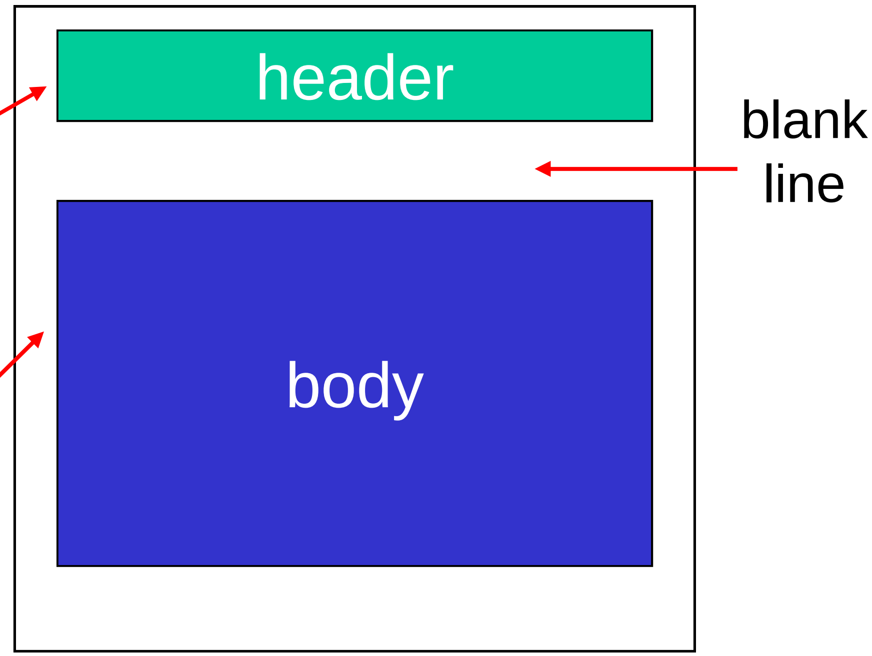
Mail message format

SMTP: protocol for exchanging email msgs

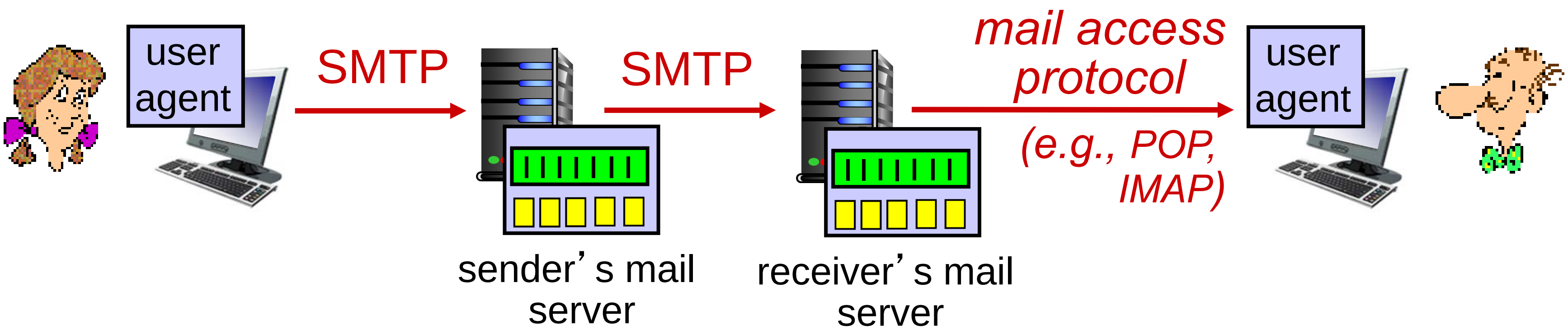
RFC 822: standard for text message format:

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

  different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only



Mail access protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]: authorization, download
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

authorization phase

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

transaction phase, client:

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

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

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (more) and IMAP

more about POP3

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

IMAP

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

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

DNS: domain name system

- Internet-host identifiers
  - IP addresses
    - unique, universal identifiers, e.g. 74.125.226.50
    - Scanning IP address from left to right more and more information about specific location of host can be obtained
    - Difficult to remember
  - Symbolic (DNS) names
    - Unique user friendly name, e.g. www.google.com
    - Easy to remember – preferred by humans
    - Provide little information about host location – difficult to aggregate by routers
    - Consist of variable number of alphanumeric characters – difficult to process by routers
- DNS enables IP address to Symbolic name translation and vice versa

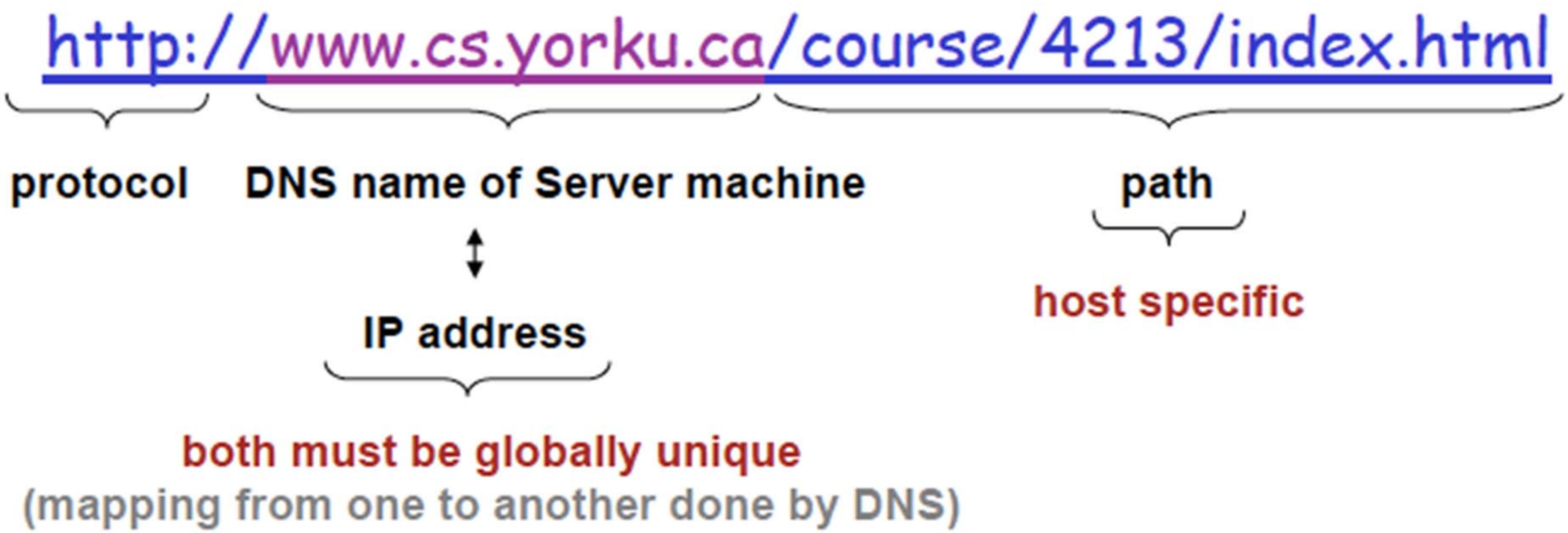
Domain Name Label

Label	*Description*
aero	Airlines and aerospace companies
biz	Businesses or firms (similar to "com")
com	Commercial organizations
coop	Cooperative business organizations
edu	Educational institutions
gov	Government institutions
info	Information service providers
int	International organizations
mil	Military groups
museum	Museums and other non-profit organizations
name	Personal names (individuals)
net	Network support centers
org	Nonprofit organizations
pro	Professional individual organizations

DNS Names vs. URLs

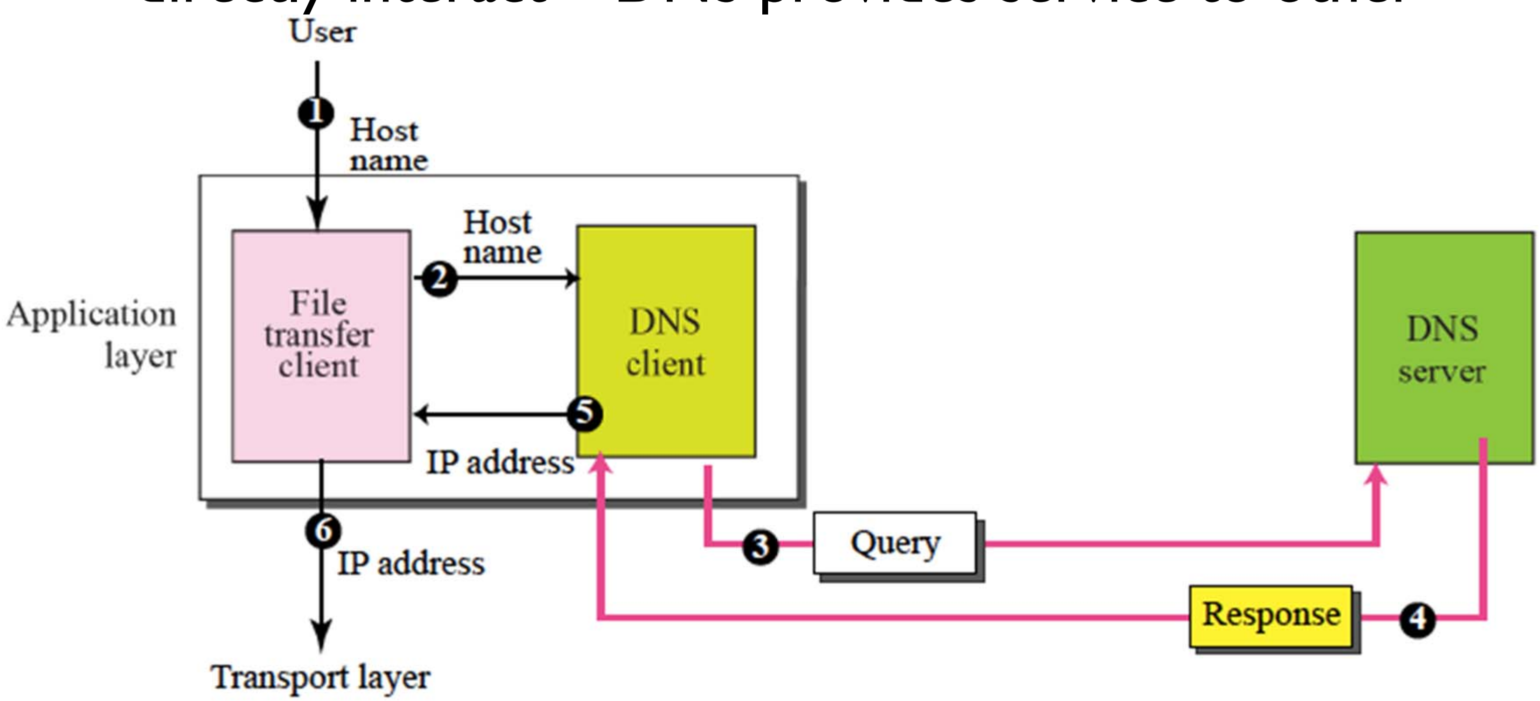
- *DNS name ≠ URL*
  - Typical URL contains three parts:

URL = protocol + DNS name + path

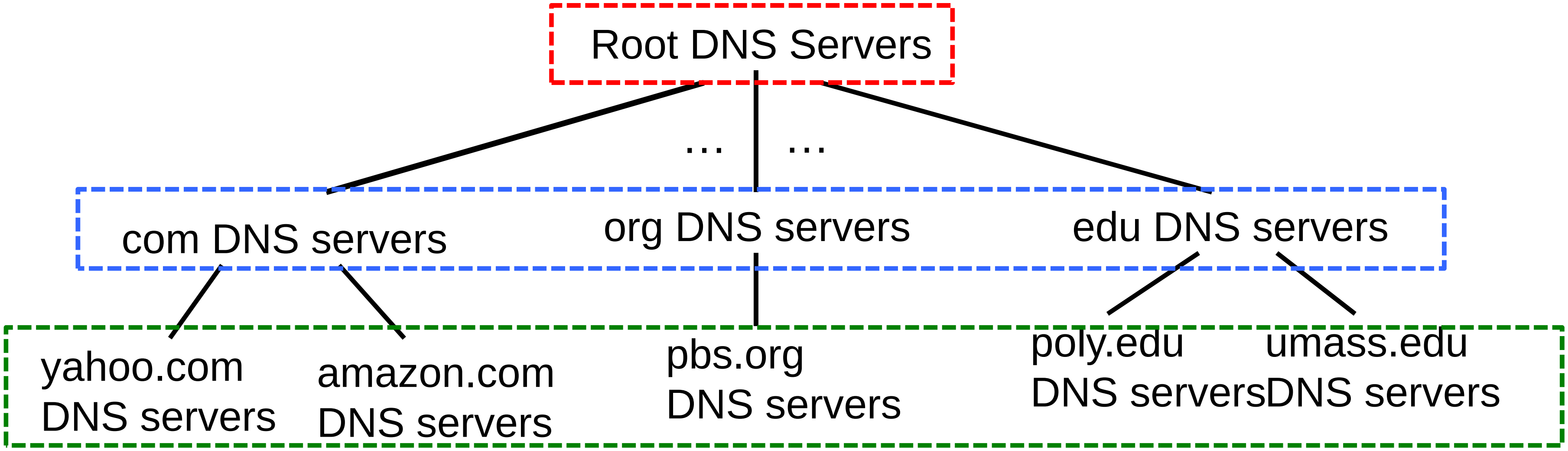


Elements of DNS

- Distributed database – implemented as a hierarchy of many name (DNS) servers
- Application-layer protocol – allows hosts to query distributed database
  - Runs over UDP on port 53
  - Unlike HTTP, DNS is not an application with which users directly interact – DNS provides service to other



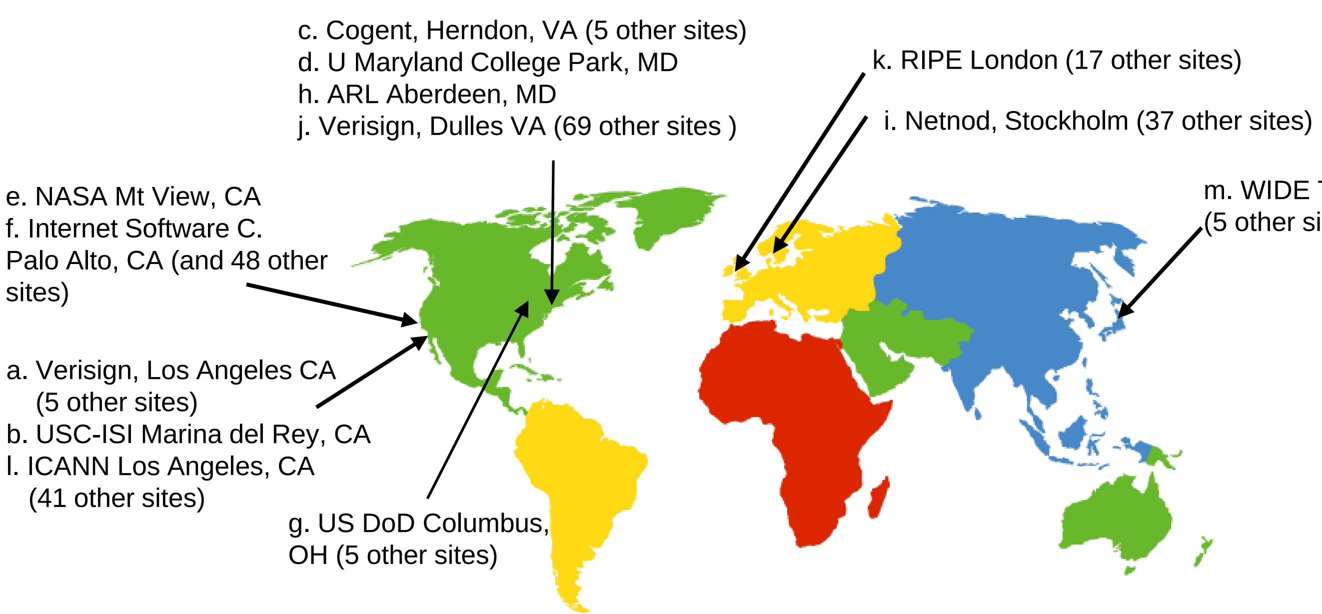
DNS: a distributed, hierarchical database



- 3 types of DNS servers – Root DNS server, Top-Level Domain (TLD) server, Authoritative DNS server
- No single DNS server has all mappings for all hosts – mappings are divided and distributed across DNS servers

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, 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 own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Local DNS name server

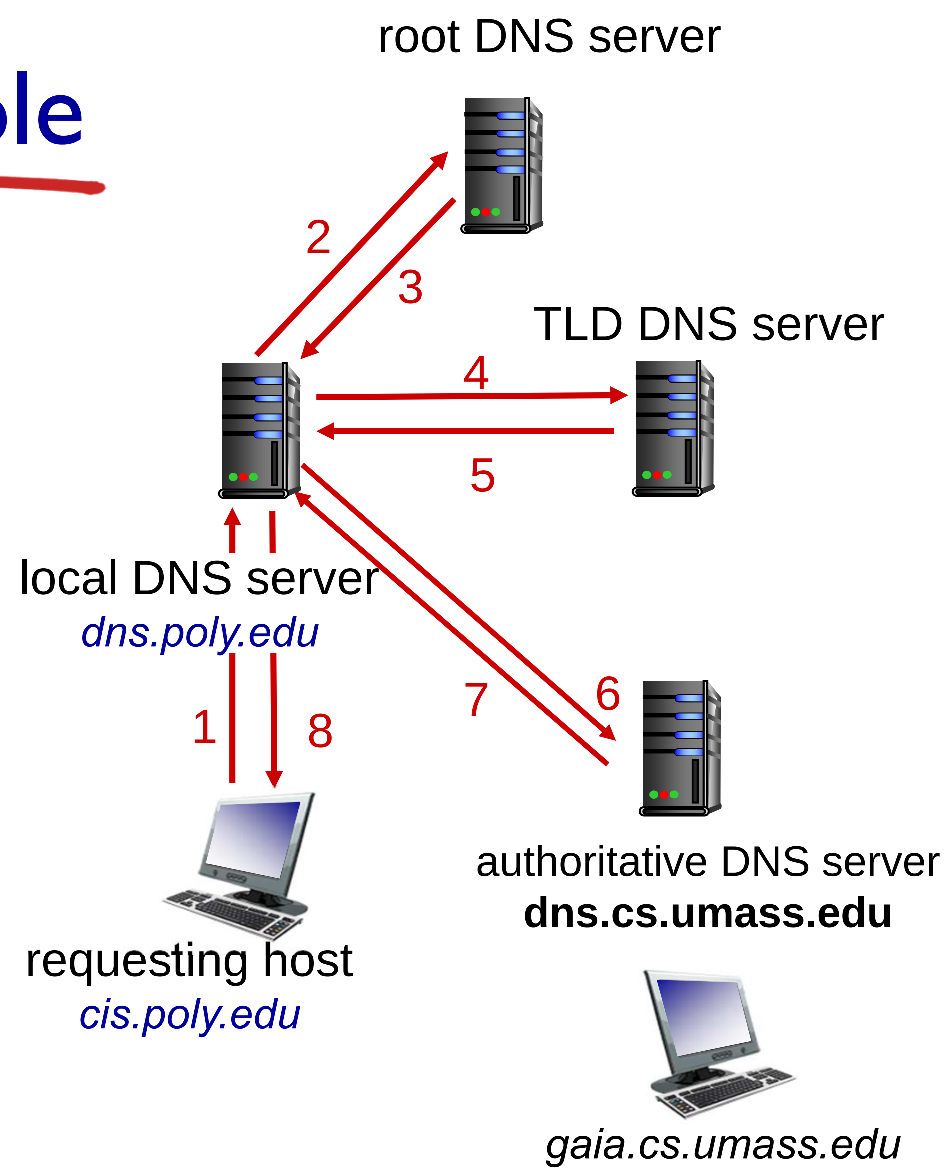
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

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

iterated query:

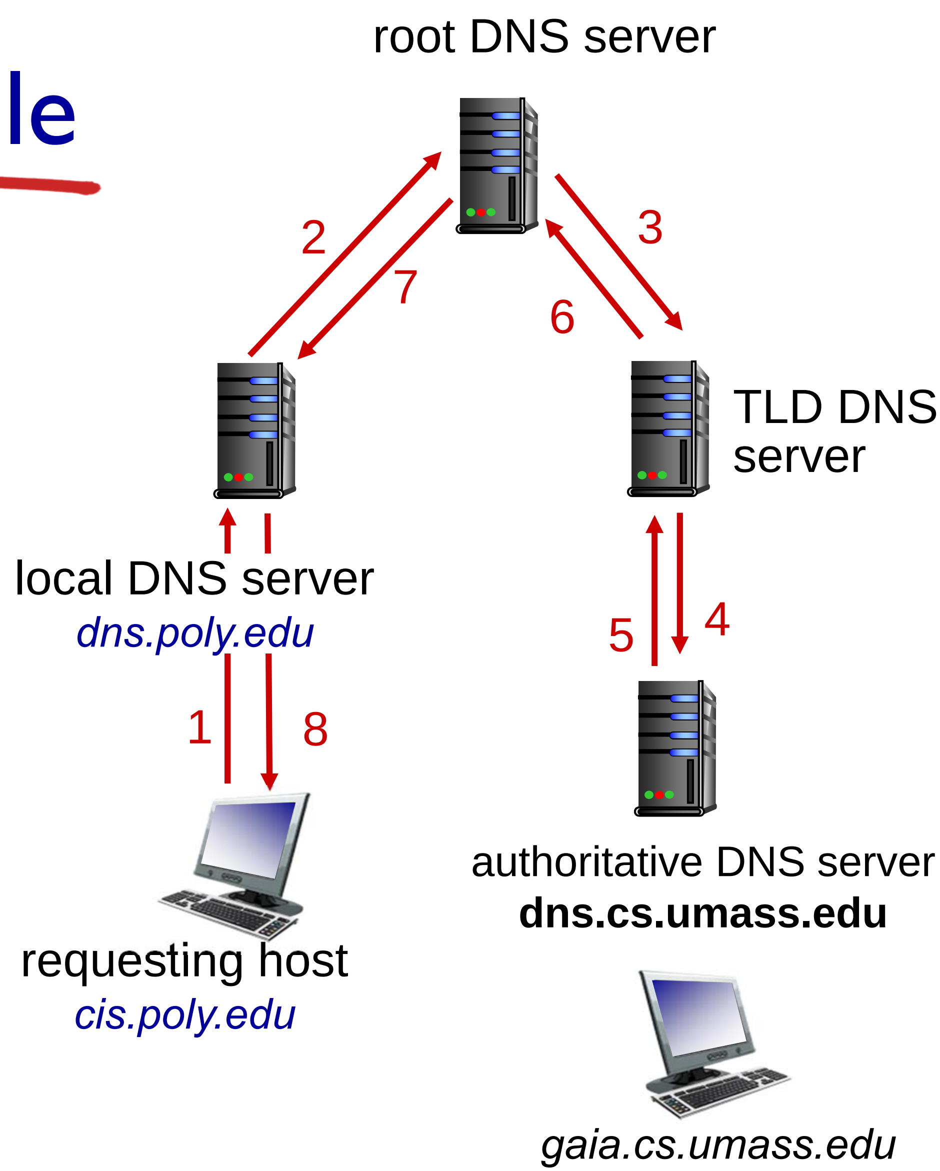
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



DNS: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard
  - RFC 2136

DNS records

DNS: distributed 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

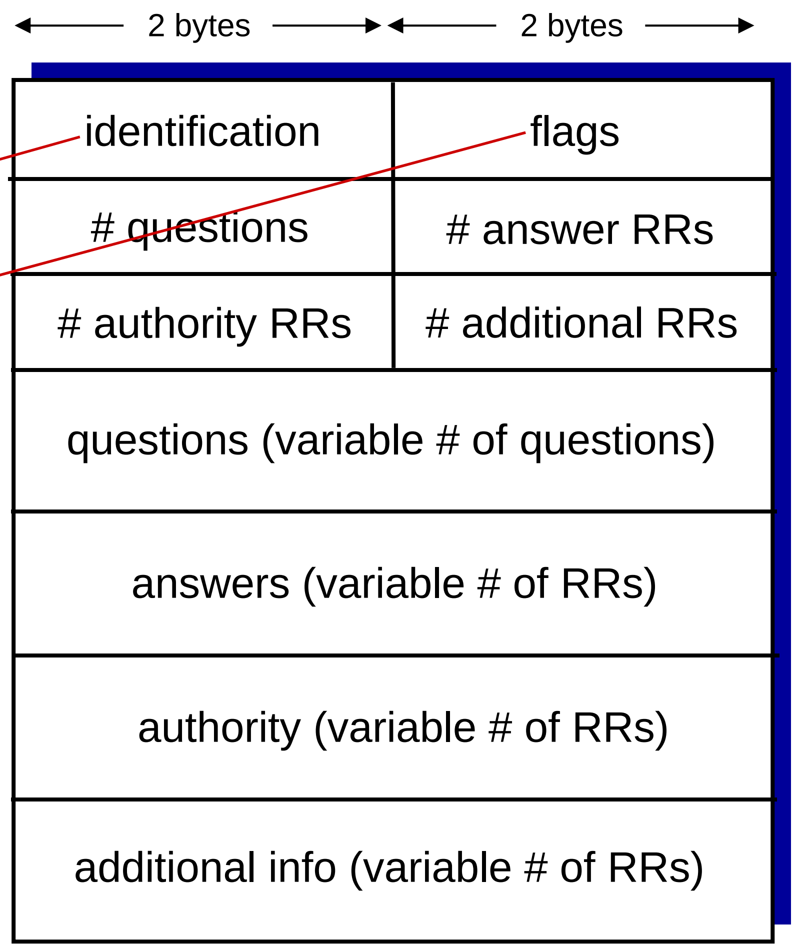
type=MX
- **`value`** is name of mailserver associated with **`name`**

DNS protocol, messages

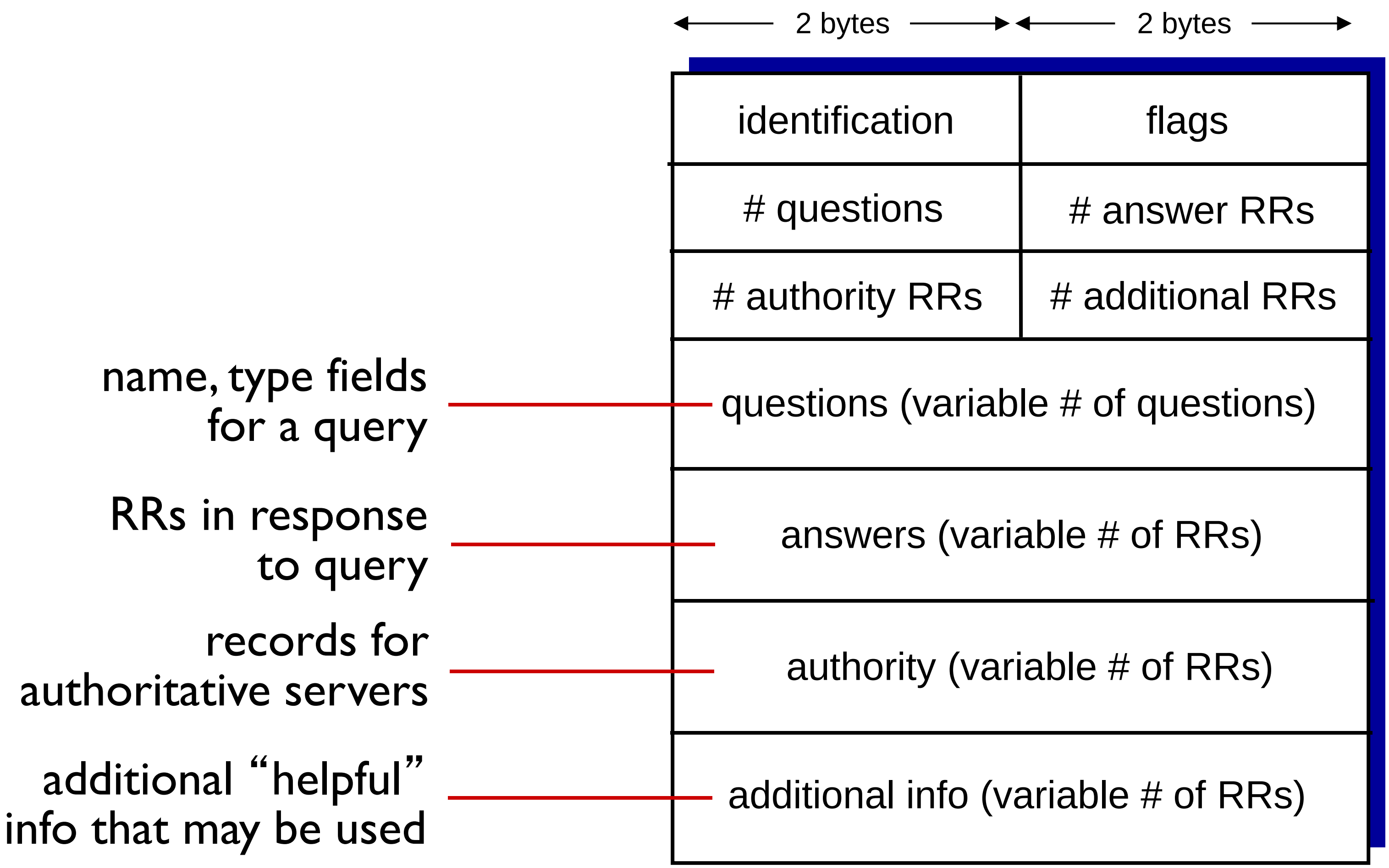
- *query* and *reply* messages, both with same *message format*

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



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

Attacking DNS

DDoS attacks

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

Redirect attacks

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

Exploit DNS for DDoS

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

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

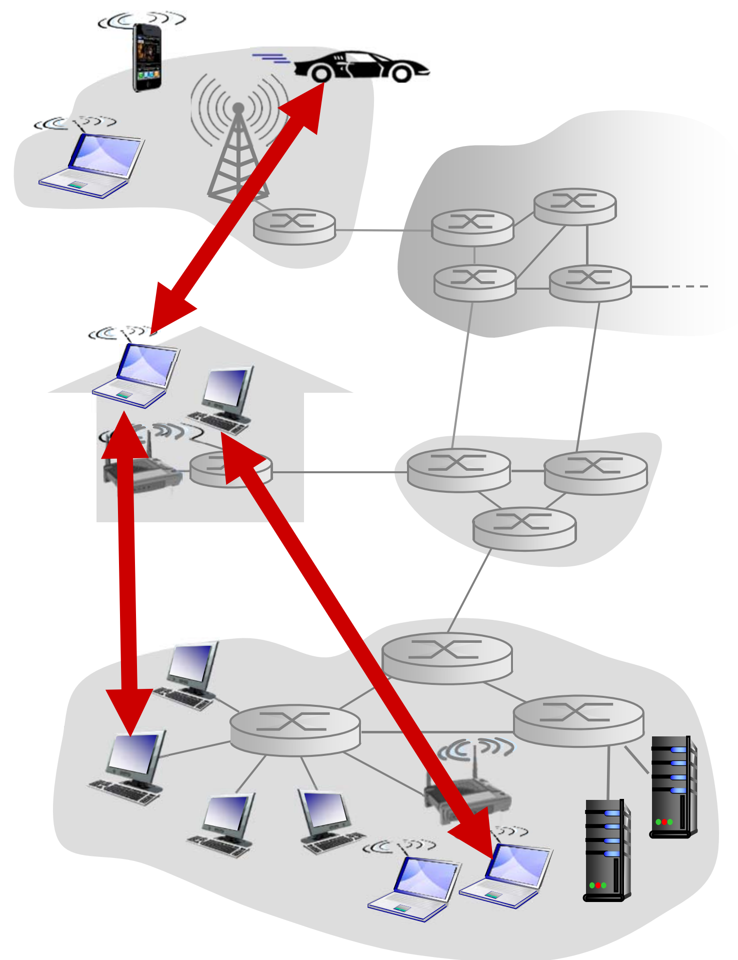
2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

Pure P2P architecture

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



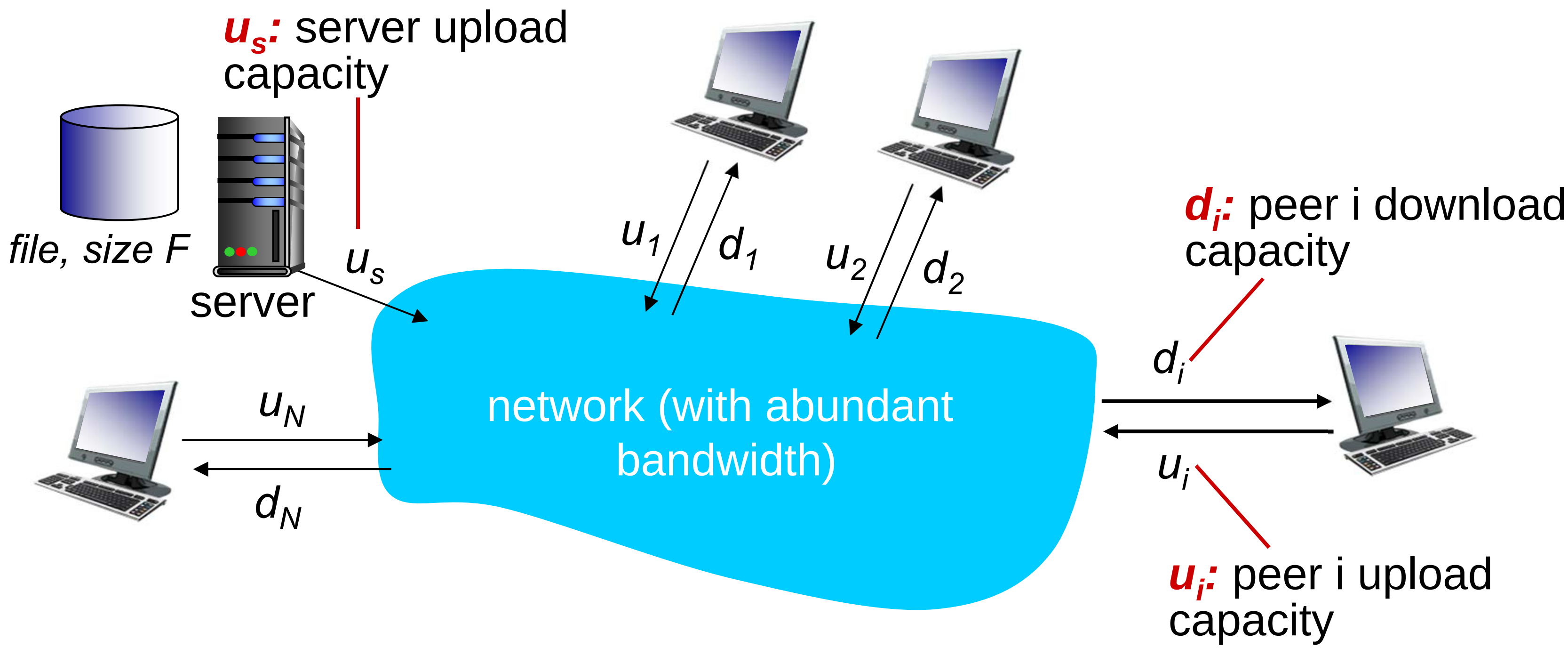
examples:

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

File distribution: client-server vs P2P

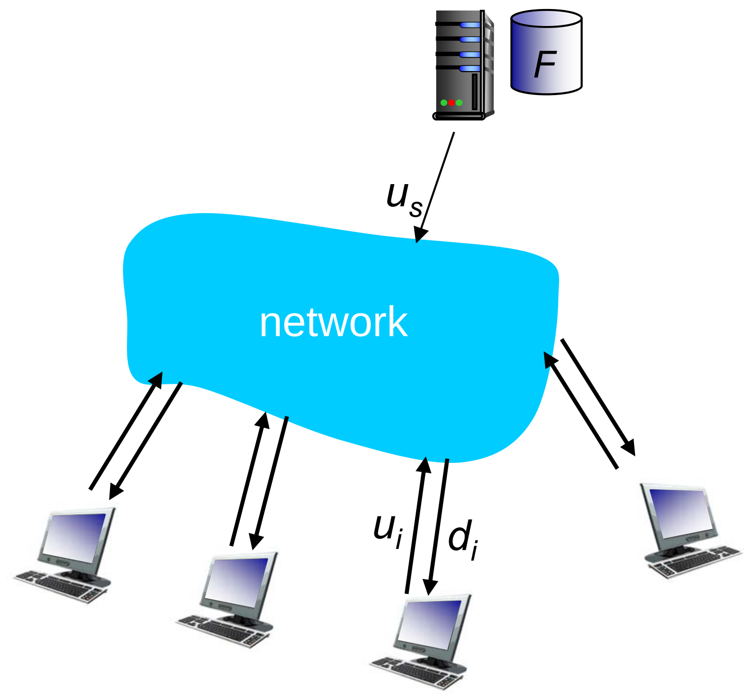
Question: how much time to distribute file (size *F*) from one server to *N* peers?

- peer upload/download capacity is limited resource



File distribution time: client-server

- *server transmission:* must sequentially send (upload) *N* file copies:
  - time to send one copy: F/u_s
  - time to send N copies: NF/u_s
- *client:* each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min}

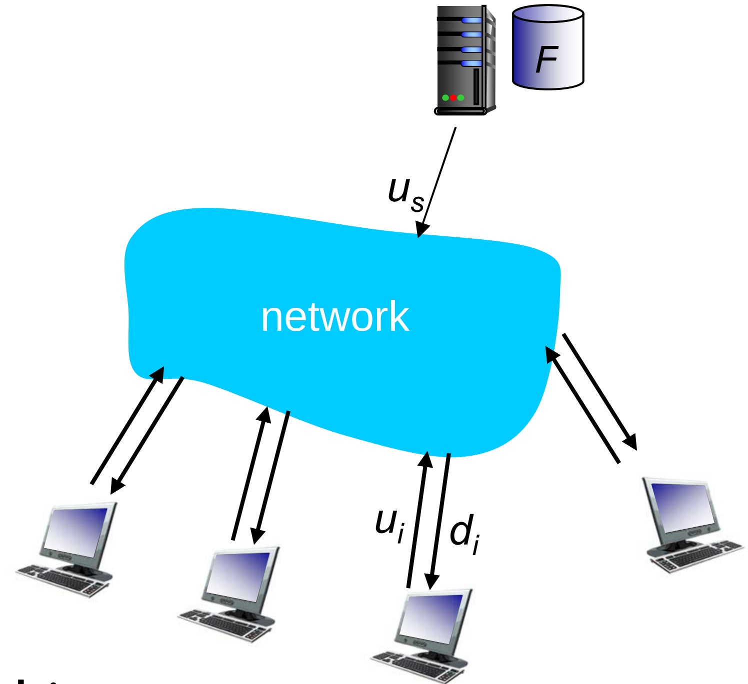


time to distribute F to N clients using client-server approach

$$D_{c\text{-}s} \geq max\{NF/u_{s,},F/d_{min}\}$$

increases linearly in N

File distribution time: P2P



- *server transmission:* must upload at least one copy
  - time to send one copy: F/u_s
- *client:* each client must download file copy
  - min client download time: F/d_{min}
- *clients:* as aggregate must download *NF* bits
  - max upload rate (limting max download rate) is $u_s + \Sigma u_i$

time to distribute F to N clients using P2P approach

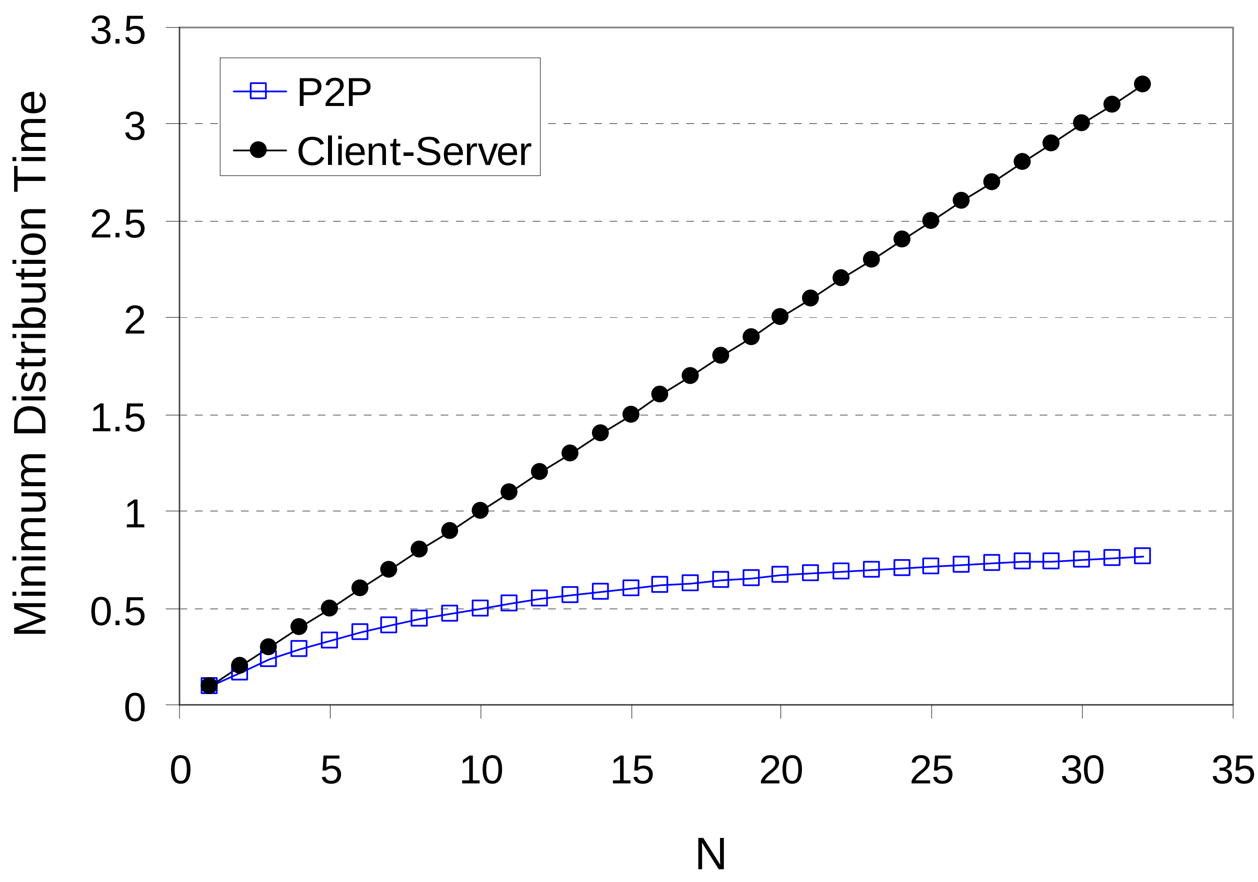
$$D_{P2P} \geq max\{F/u_{s,},F/d_{min,},NF/(u_s + \Sigma u_i)\}$$

increases linearly in *N* …

… but so does this, as each peer brings service capacity

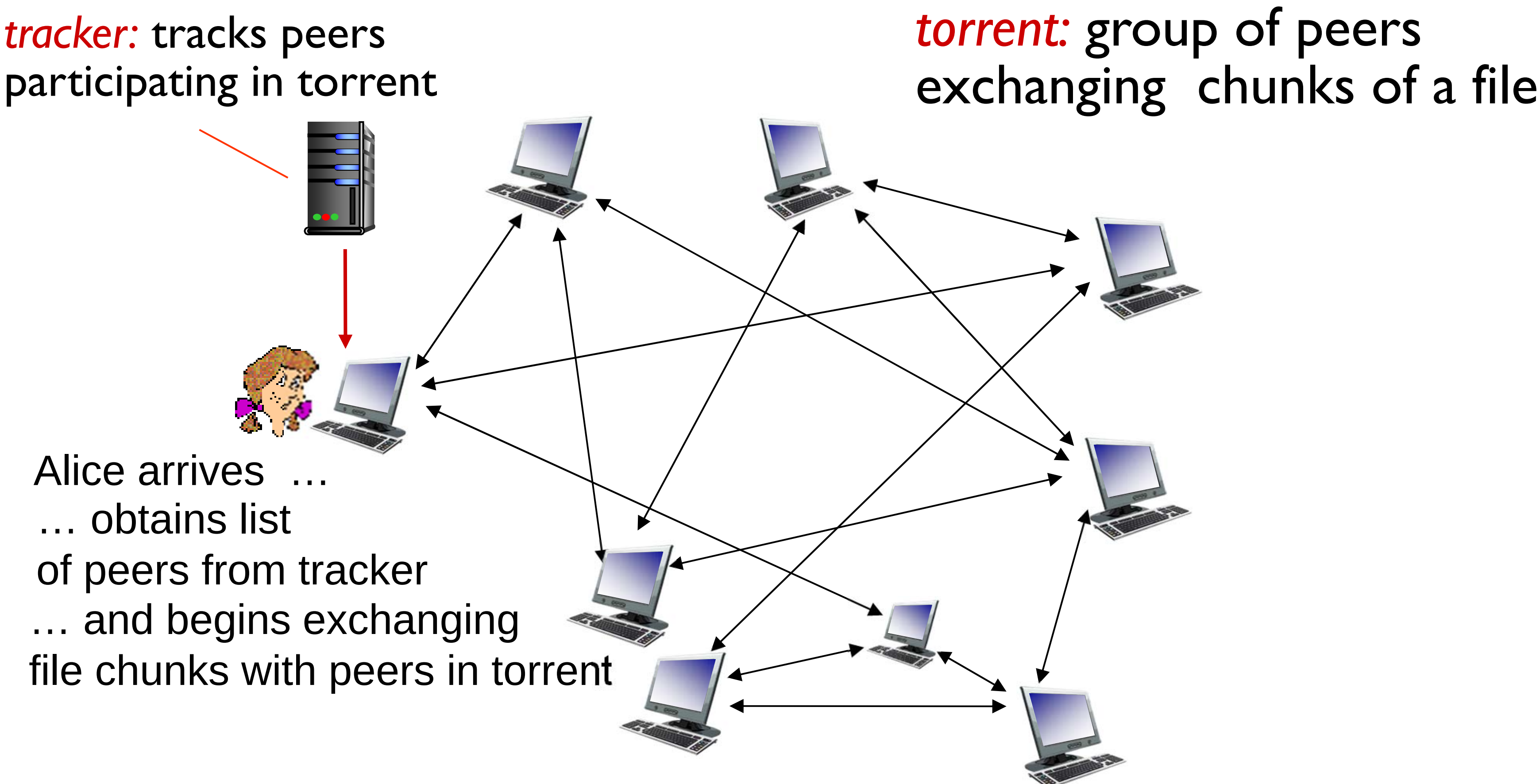
Client-server vs. P2P: example

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



P2P file distribution: BitTorrent

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



Chapter 2: summary

our study of network apps now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP

- specific protocols:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent

Chapter 2: summary

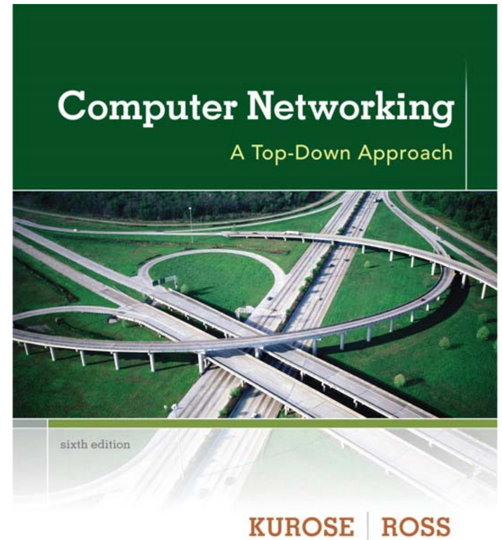
most importantly: learned about protocols!

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - headers: fields giving info about data
  - data: info being communicated

important themes:

- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable msg transfer
- "complexity at network edge"

A note on these slides



Part of PPT slides were adopted from Prof. Natalija Vlajic' early CSE3214 course and the rest were adopted from the book "Computer Networking: A Top Down Approach" 6th Edition by Jim Kurose and Keith Ross

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012