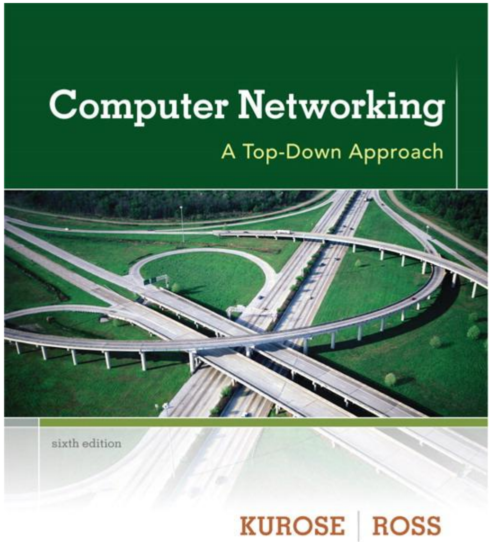
Chapter 2
Application Layer



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

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

2.7 socket programming with UDP and TCP

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
- …
- …

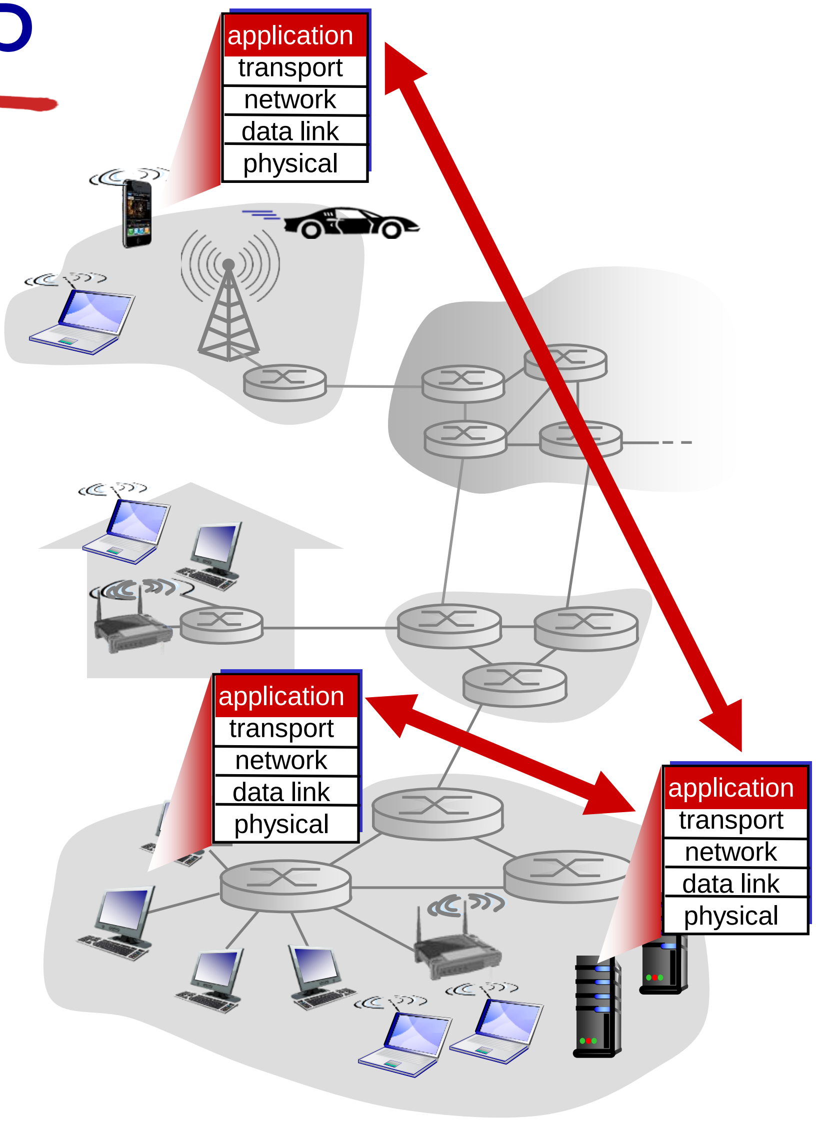
Creating a network app

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

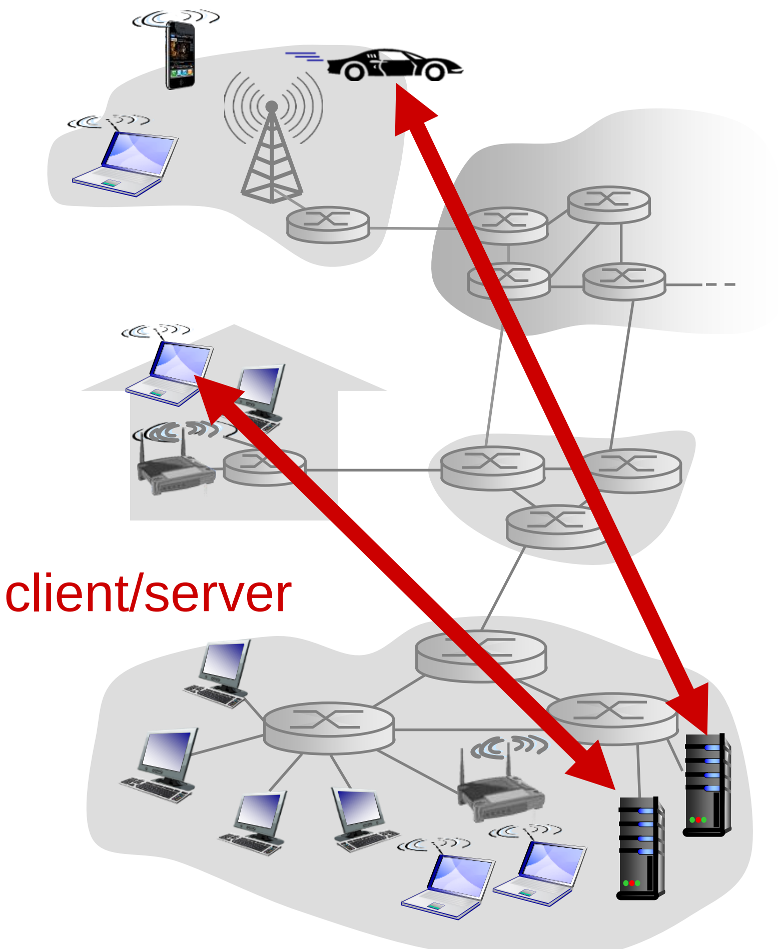


Application architectures

possible structure of applications:

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

Client-server architecture

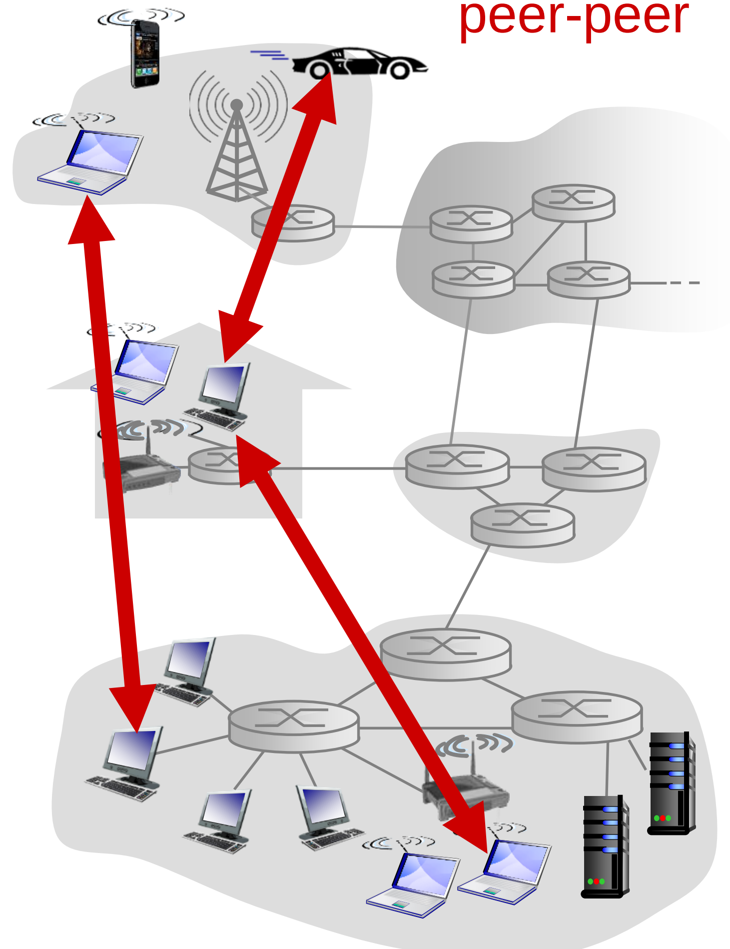


server:
- always-on host
- permanent IP address
- data centers for scaling

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

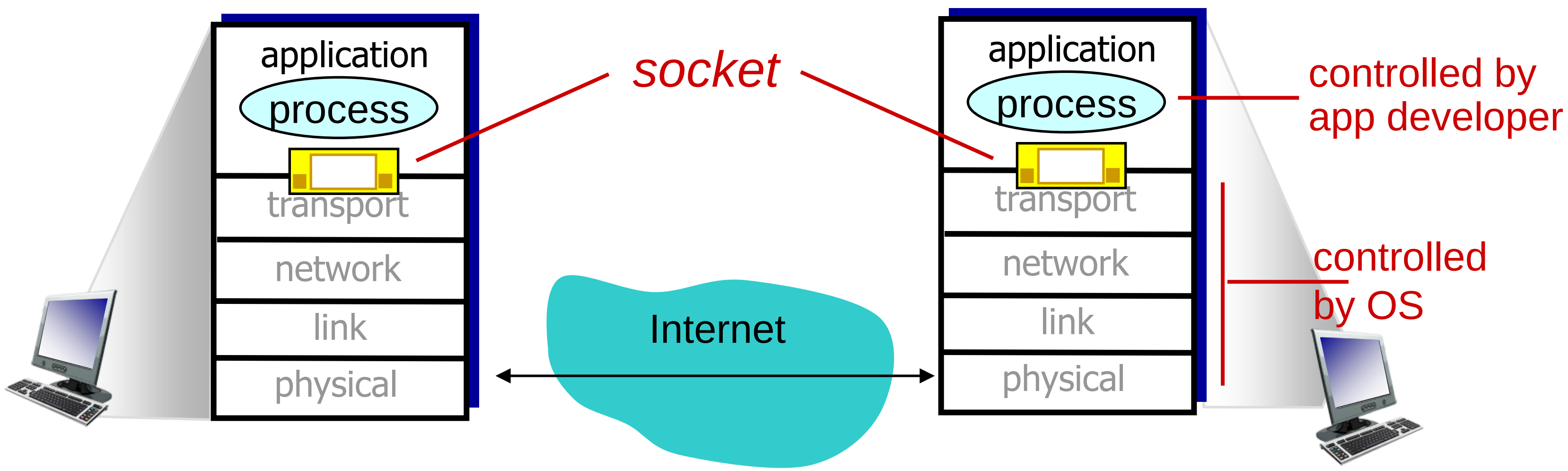
client process: process that initiates communication

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

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	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
text messaging	no loss	elastic	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

Difference between TCP and UDP Services

Description	*TCP*	*UDP*
Full Name	Transmission Control Protocol	User Datagram Protocol
Connection	TCP is a connection-oriented protocol.	UDP is a connectionless protocol.
Function	A point to point connection is established between client and server before sending message.	A point to point connection is not established before sending messages.
Usage	TCP is suited for applications that require high reliability, and transmission time is relatively less critical.	UDP is suitable for applications that need fast, efficient transmission, such as games.
Reliability	There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent.	There is no guarantee that the messages or packets sent would reach at all.
Use by other protocols	HTTP, HTTPs, FTP, SMTP, Telnet	DNS, DHCP, SNMP, RIP, VOIP
Ordering of data packets	TCP rearranges data packets in the order specified.	UDP has no inherent order as all packets are independent of each other.
Speed of transfer	The speed for TCP is slower than UDP.	UDP is faster because there is no error-checking for packets.
Header Size	TCP header size is 20 bytes	UDP Header size is 8 bytes.
Data Flow Control	TCP does Flow Control.	UDP does not have an option for flow control.
Error Checking	TCP does error checking	UDP does error checking, but no recovery options.
Acknowledgement	Acknowledgement segments	No Acknowledgment
Handshake	SYN, SYN-ACK, ACK	No handshake

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

2.7 socket programming with UDP and TCP

Web and HTTP

First, a review…

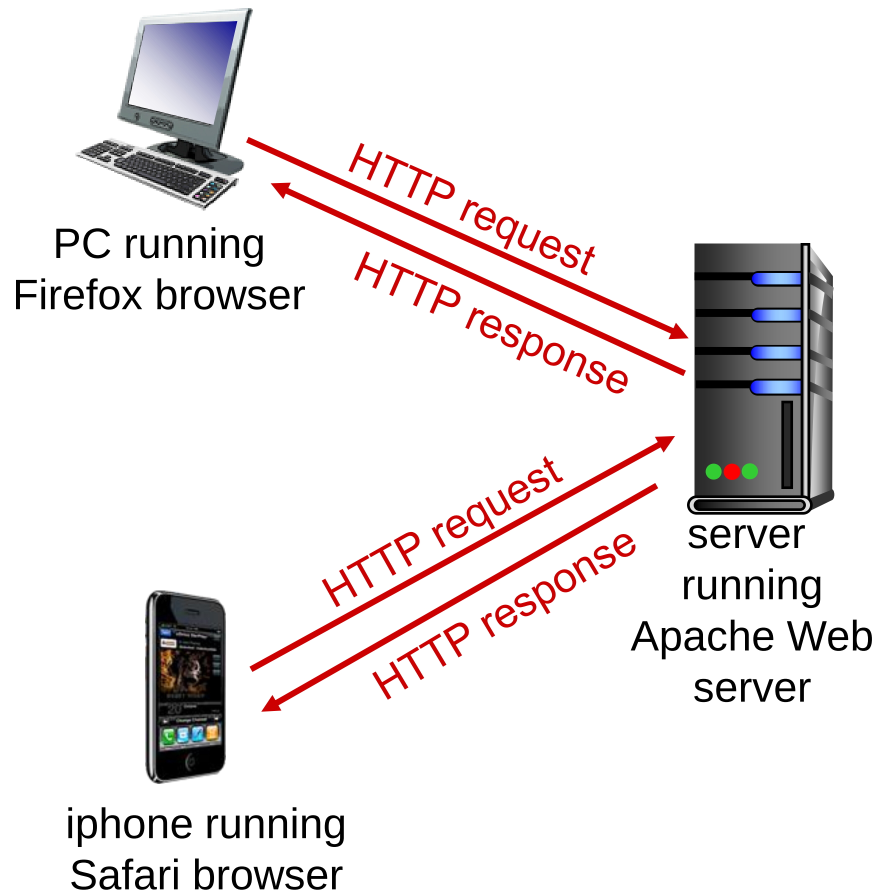
- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,

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

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (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

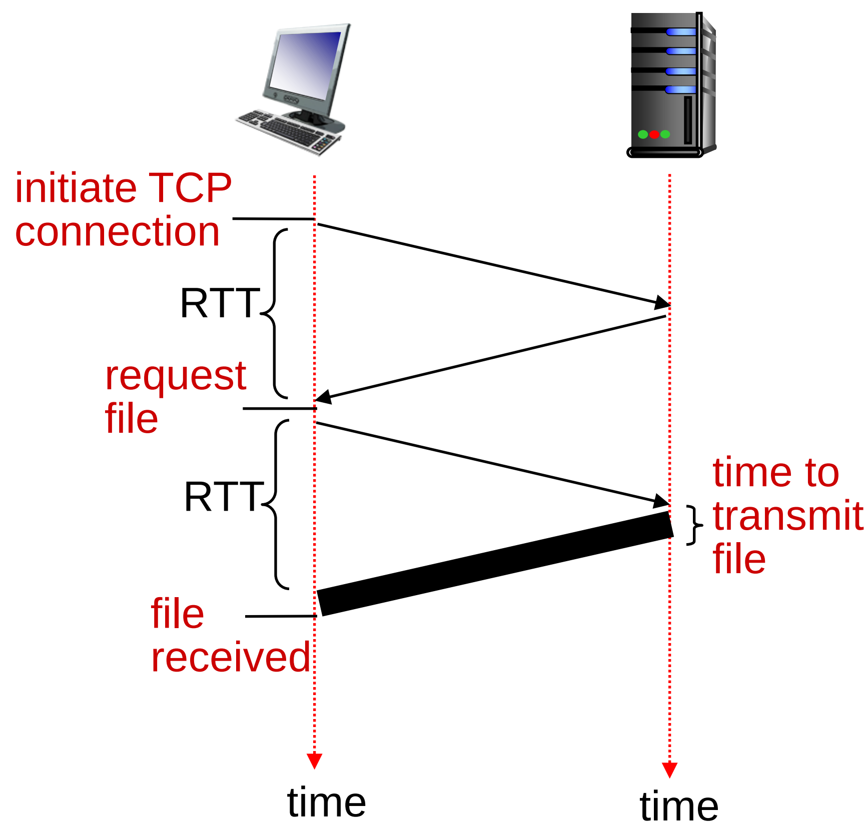
Non-persistent HTTP: response time

RTT (definition): 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 TCP connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP request message

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

carriage return character

line-feed character

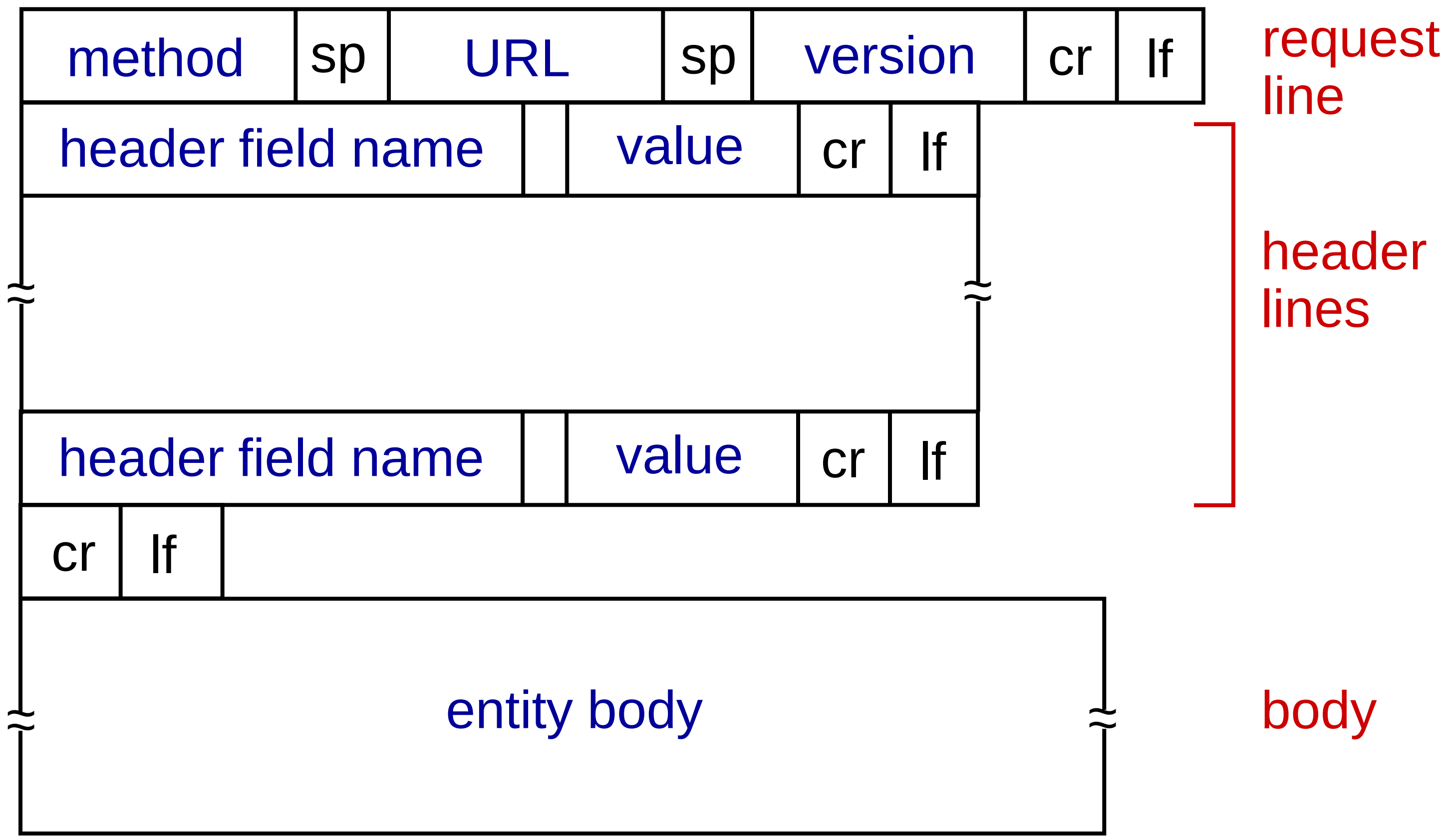
request line (GET, POST, HEAD commands)

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
```

HTTP request message: general format



Uploading form input

POST method:

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

URL method:

- uses GET method
- input is uploaded in URL field of request line:

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

Method types

HTTP/1.0:

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

HTTP/1.1:

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

HTTP response message

status line (protocol status code status phrase)

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-to-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`

User-server state: cookies

many Web sites use cookies

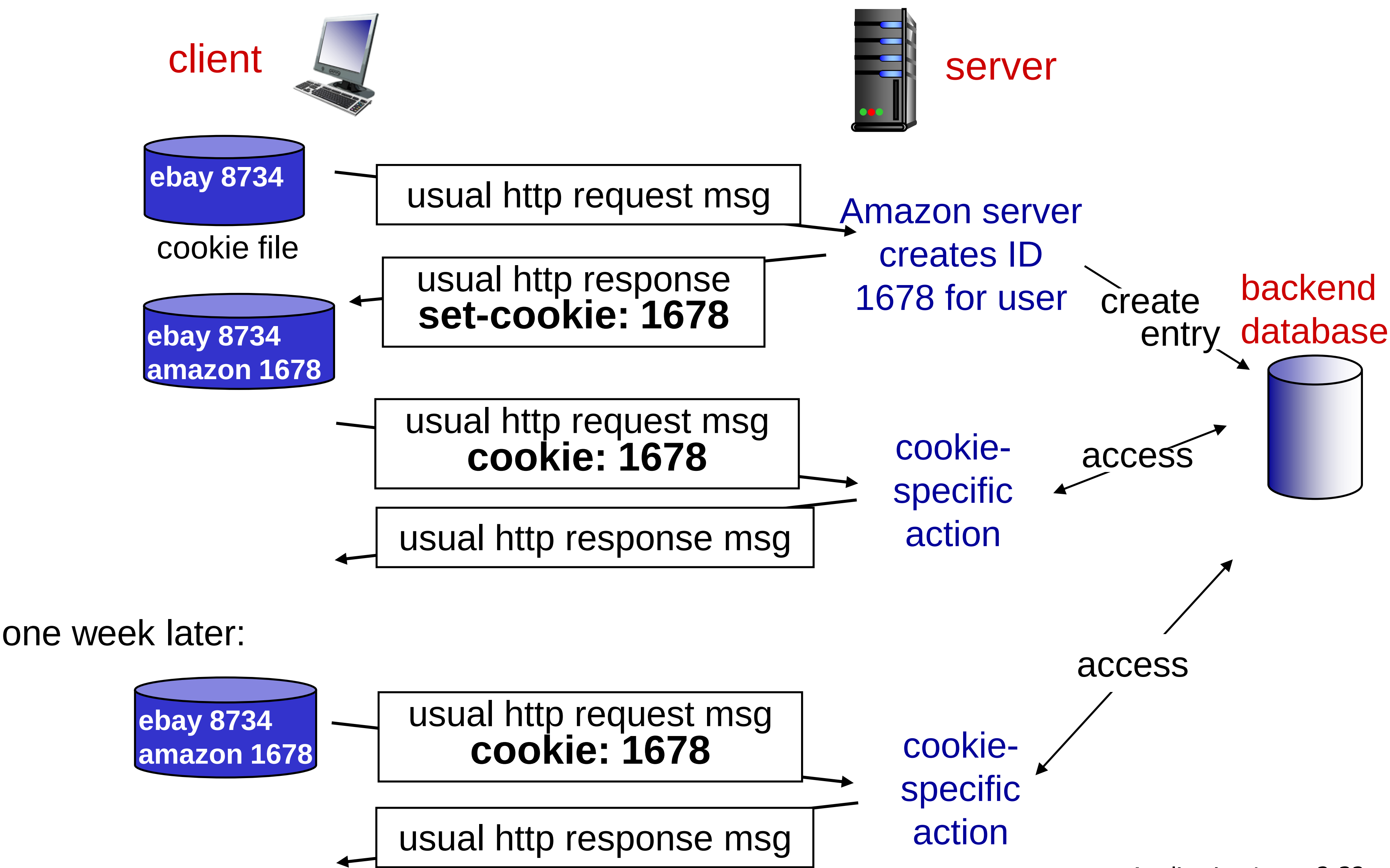
four components:

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

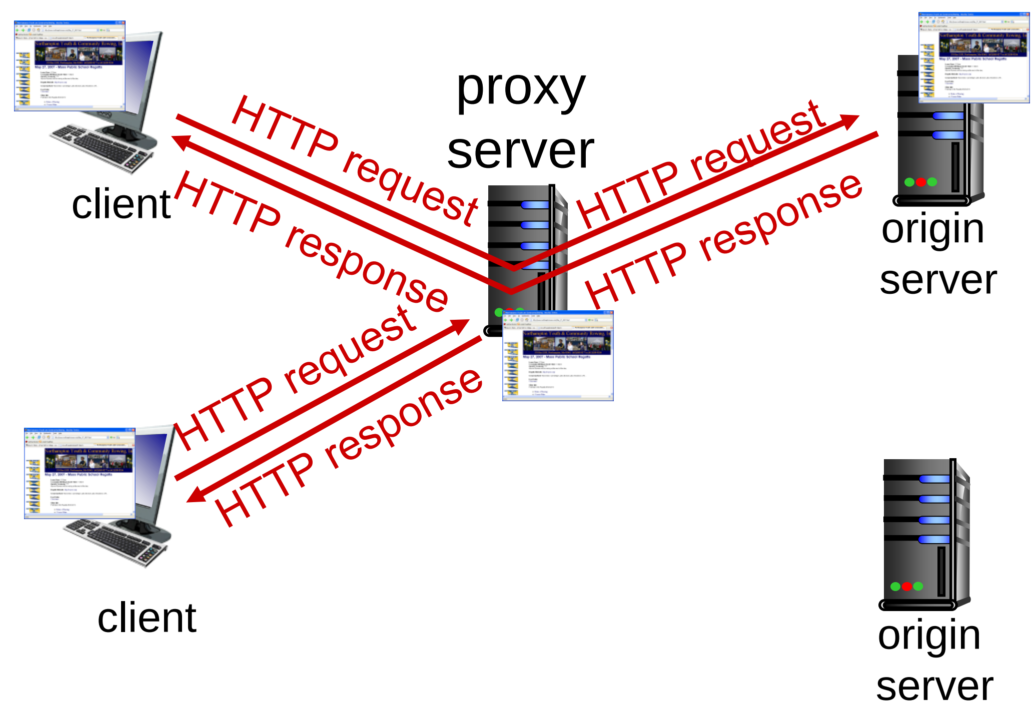
how to keep "state":

- 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
  - 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)

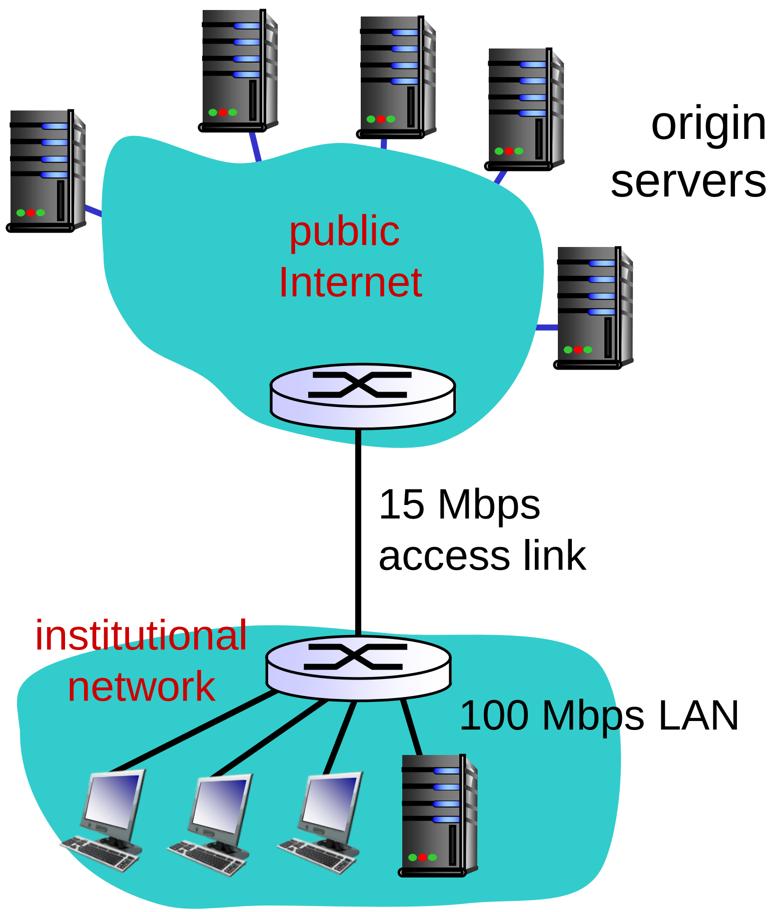
Caching example:

assumptions:

- avg object size: 1Mbits
- avg request rate from browsers to origin servers:15 requests/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = 100% *problem!*
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs



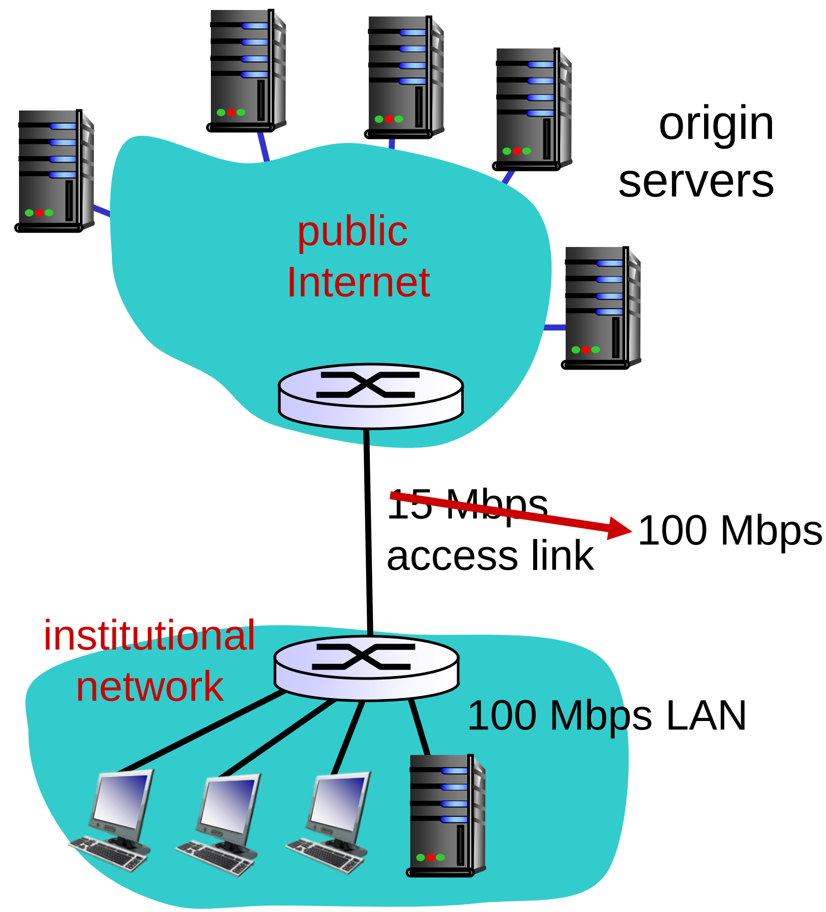
Caching example: fatter access link

assumptions:

- avg object size: 1Mbits
- avg request rate from browsers to origin servers:15 requests/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~15 Mbps~~ 100 Mbps

consequences:

- LAN utilization: 100%
- access link utilization = ~~100%~~ 15%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ + usecs
  msecs



Cost: increased access link speed (not cheap!)

Caching example: install local cache

assumptions:

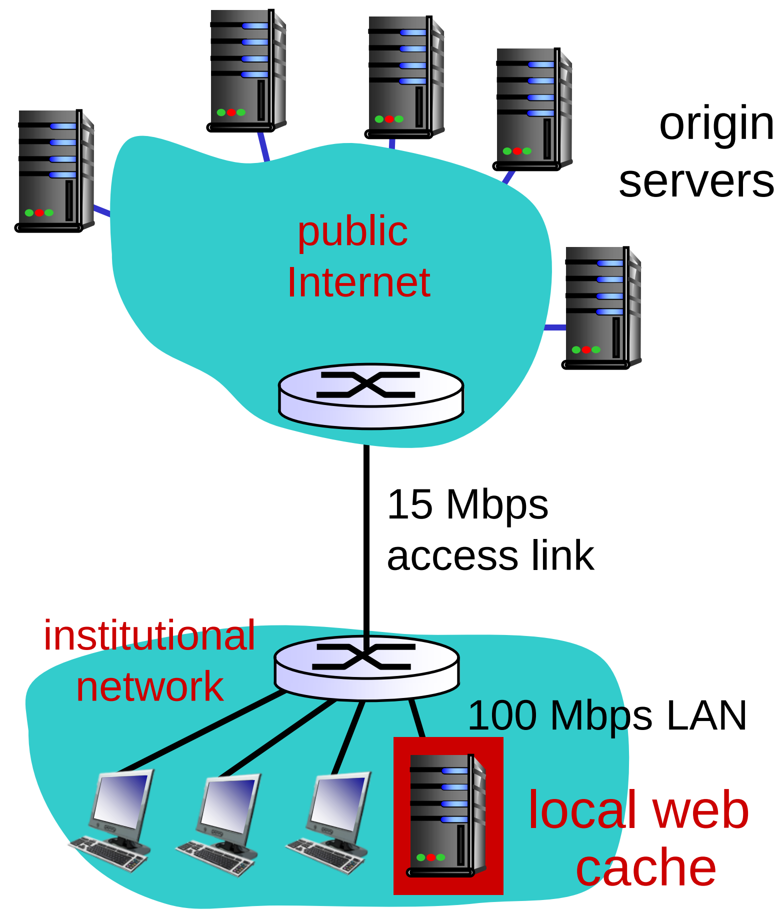
- avg object size: 1Mbits
- avg request rate from browsers to origin servers:15 requests/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ?
- total delay = ?

How to compute link utilization, delay?

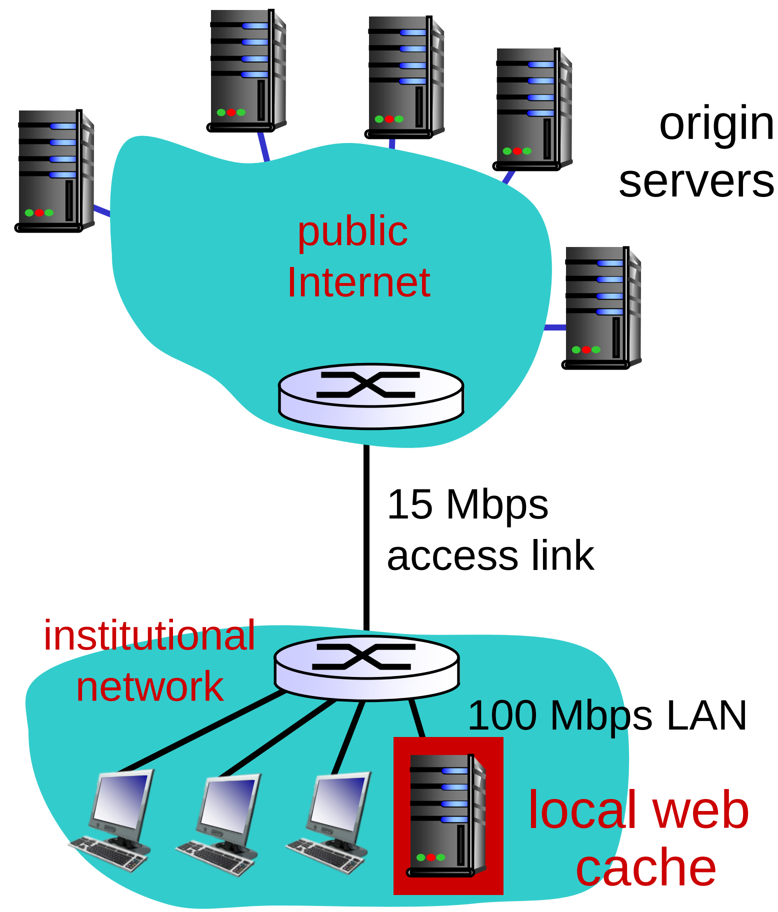
Cost: web cache (cheap!)



Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 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*15 Mbps = 9 Mbps
  - utilization = 9/15 = .60
- 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!)



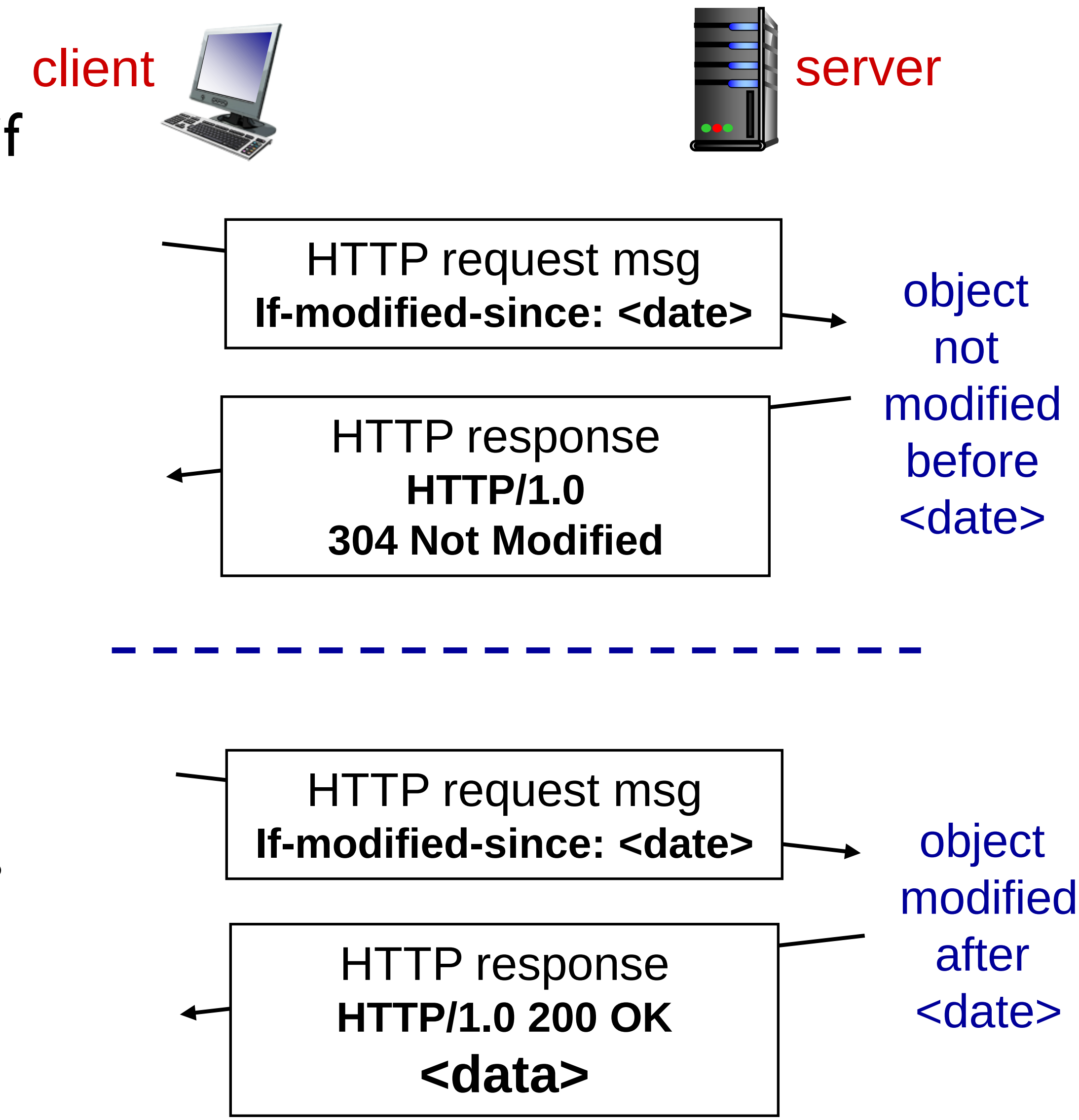
Conditional GET

"It's a HTTP mechanism that allows a cache to verify that its object are up to date."

- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *cache:* specify date of cached copy in HTTP request

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

  `HTTP/1.0 304 Not Modified`



Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

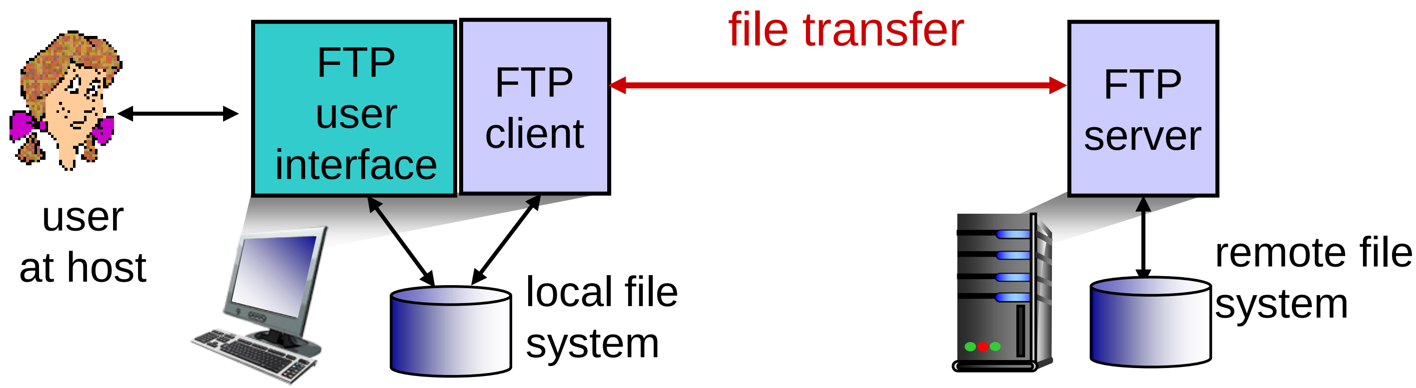
2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.7 socket programming with UDP and TCP

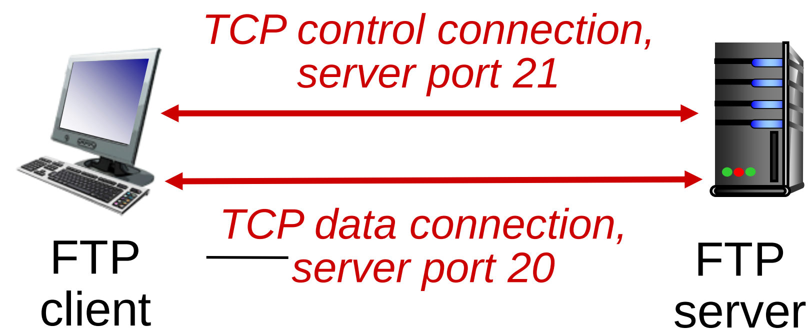
FTP: the file transfer protocol



- 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, *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
- control connection: *"out of band"*
- FTP server maintains "state": current directory, earlier authentication

FTP commands, responses

sample commands:

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

sample return codes

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

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.7 socket programming with UDP and TCP

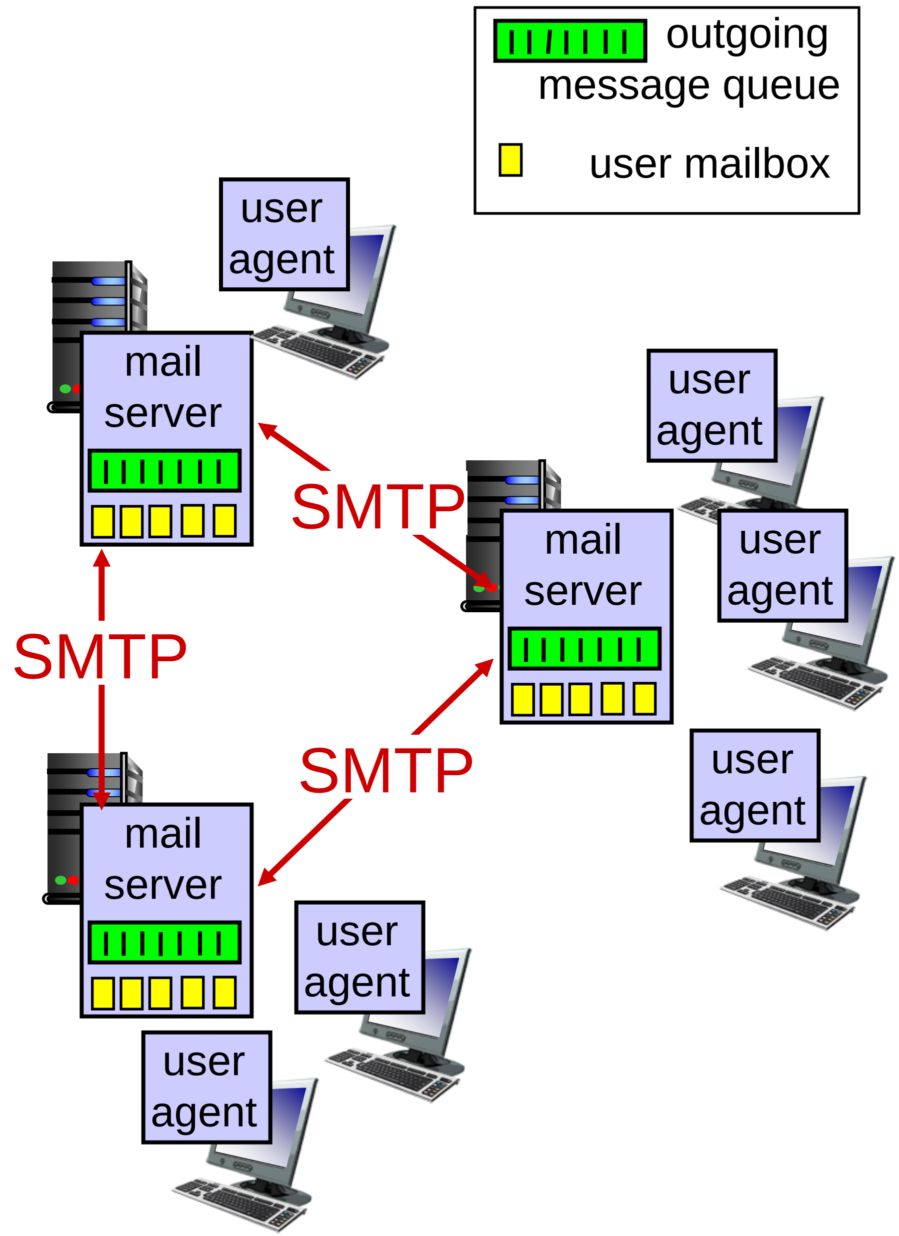
Electronic mail

Three major components:

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

User Agent

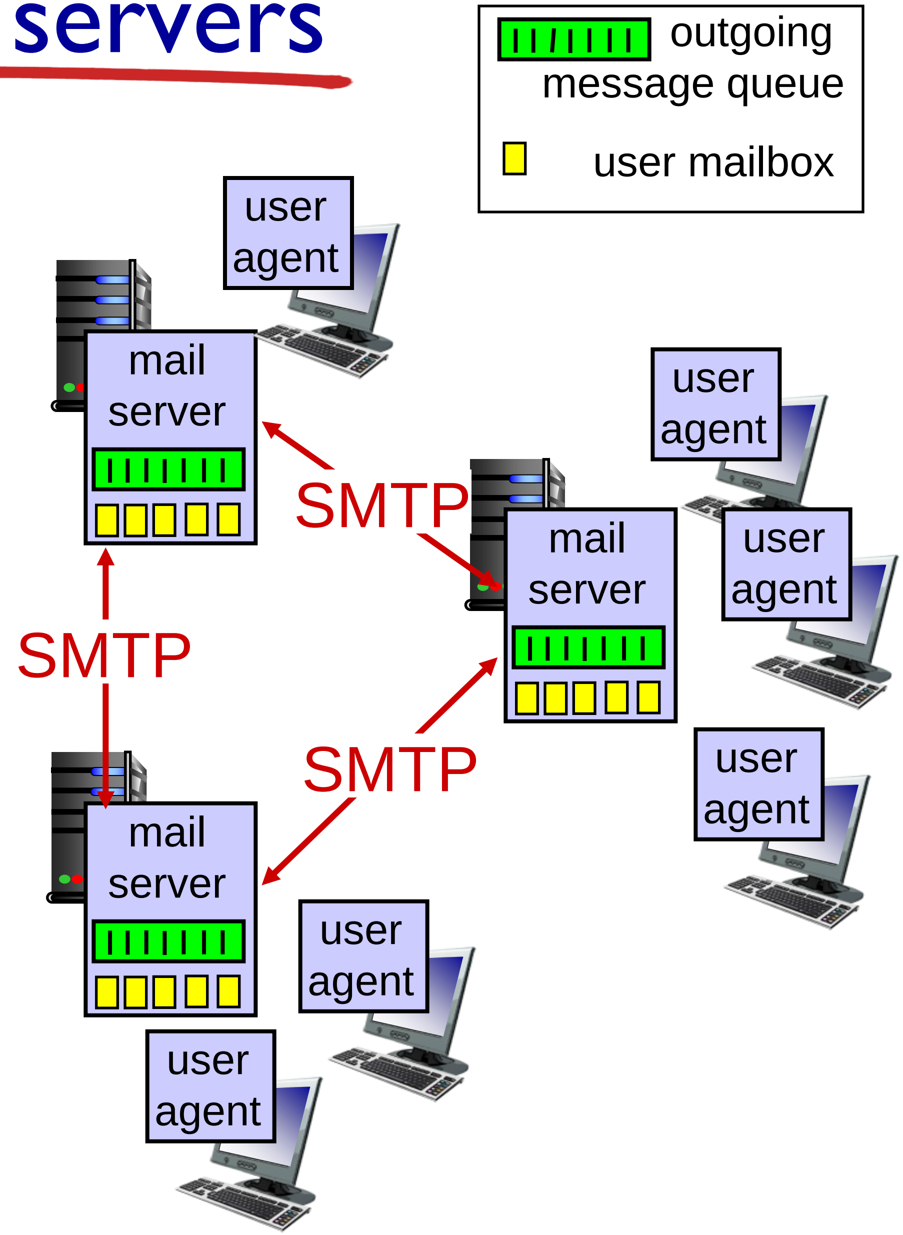
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



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

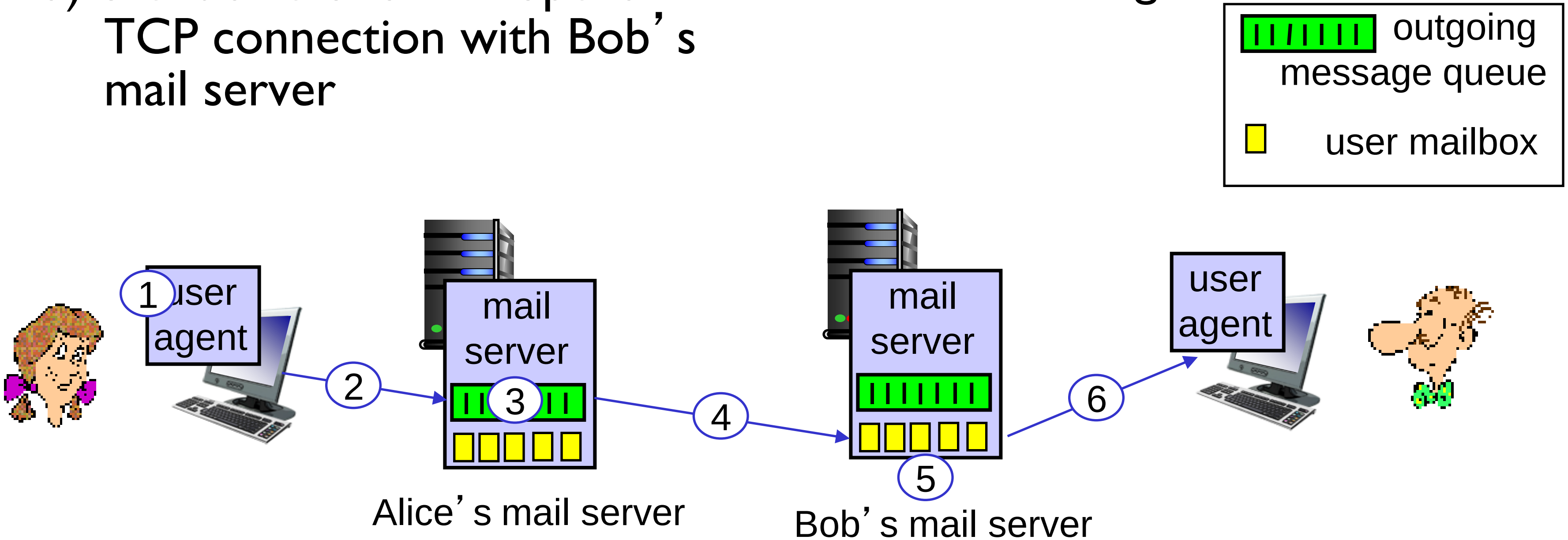


Electronic Mail: SMTP [RFC 2821]

- 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 between Two Mail Servers

```
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
```

SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses `CRLF.CRLF` to determine end of message

comparison with HTTP:

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

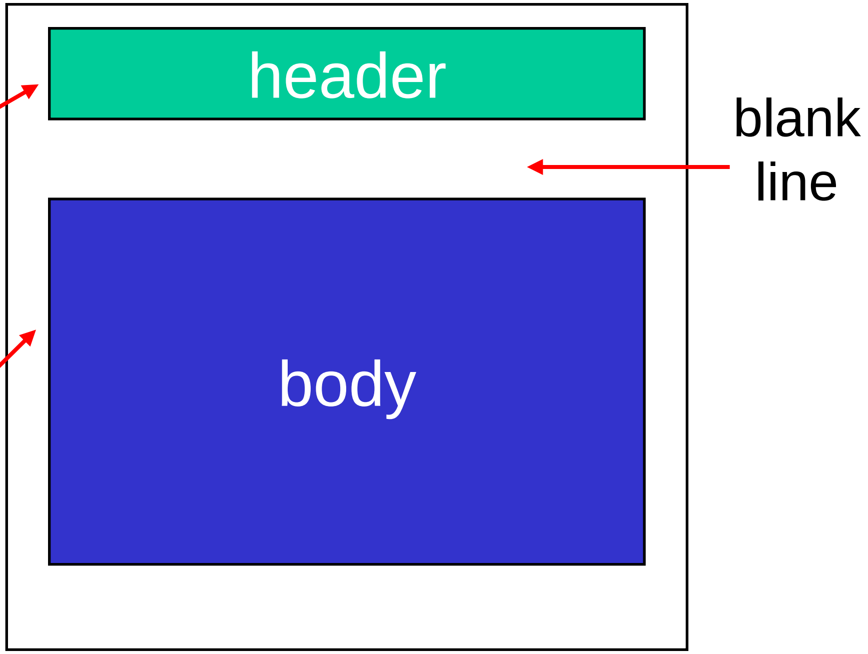
Mail message format

SMTP: protocol for exchanging email msgs

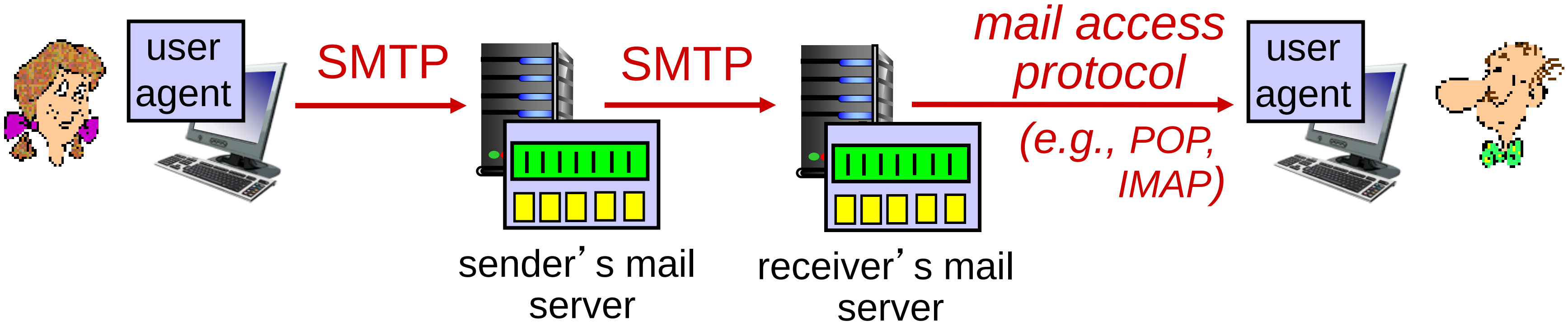
RFC 822: standard for text message format:

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

  different from SMTP 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`**

update phase

```
telnet mailServer 110

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 2 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

2.7 socket programming with UDP and TCP

DNS: domain name system

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

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

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

Domain Name System:

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

DNS: services, structure

DNS services

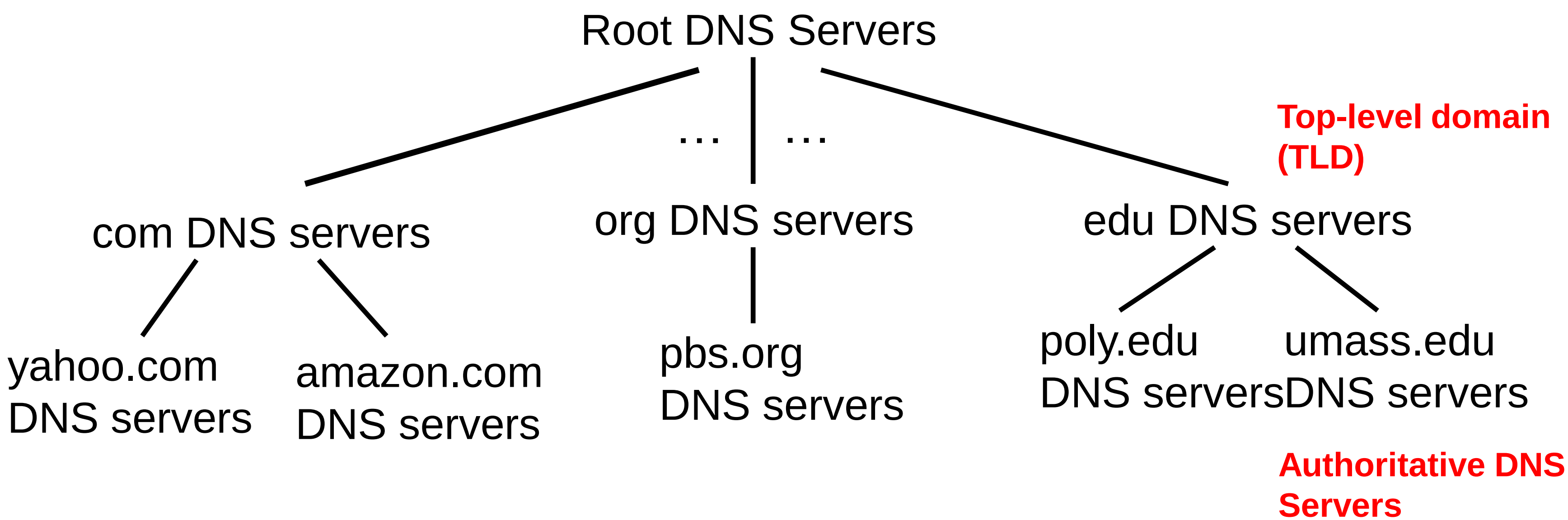
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

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

A: doesn't scale!

DNS: a distributed, hierarchical database

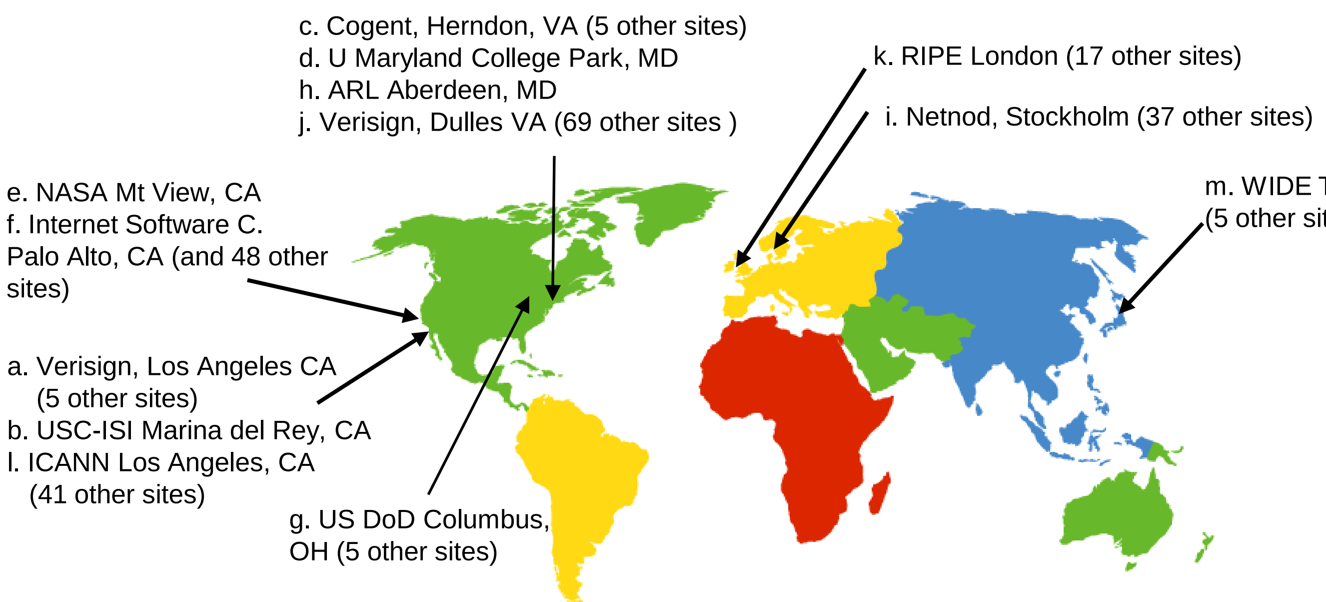


client wants IP for www.amazon.com; 1st approx:

- client queries 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

Oct - 2006

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

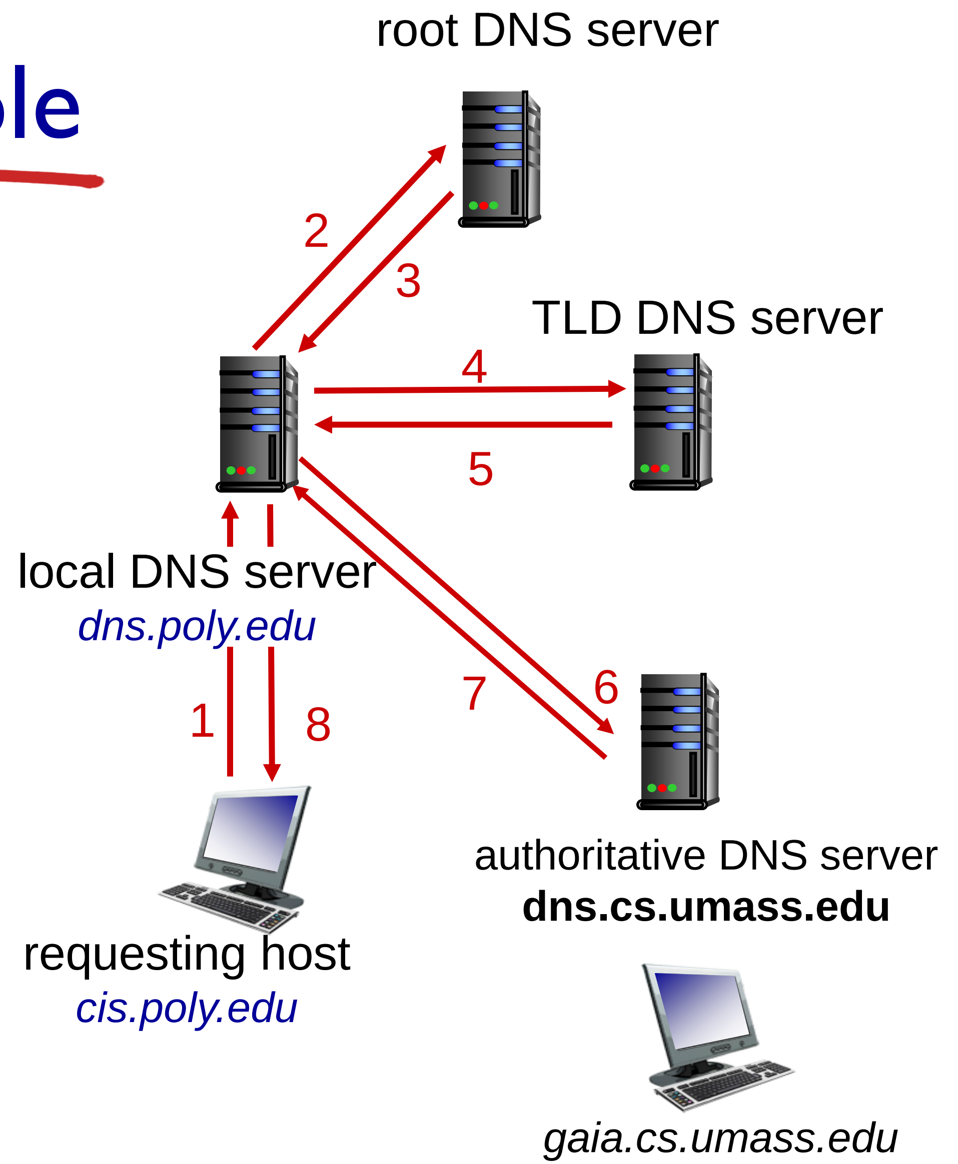
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts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

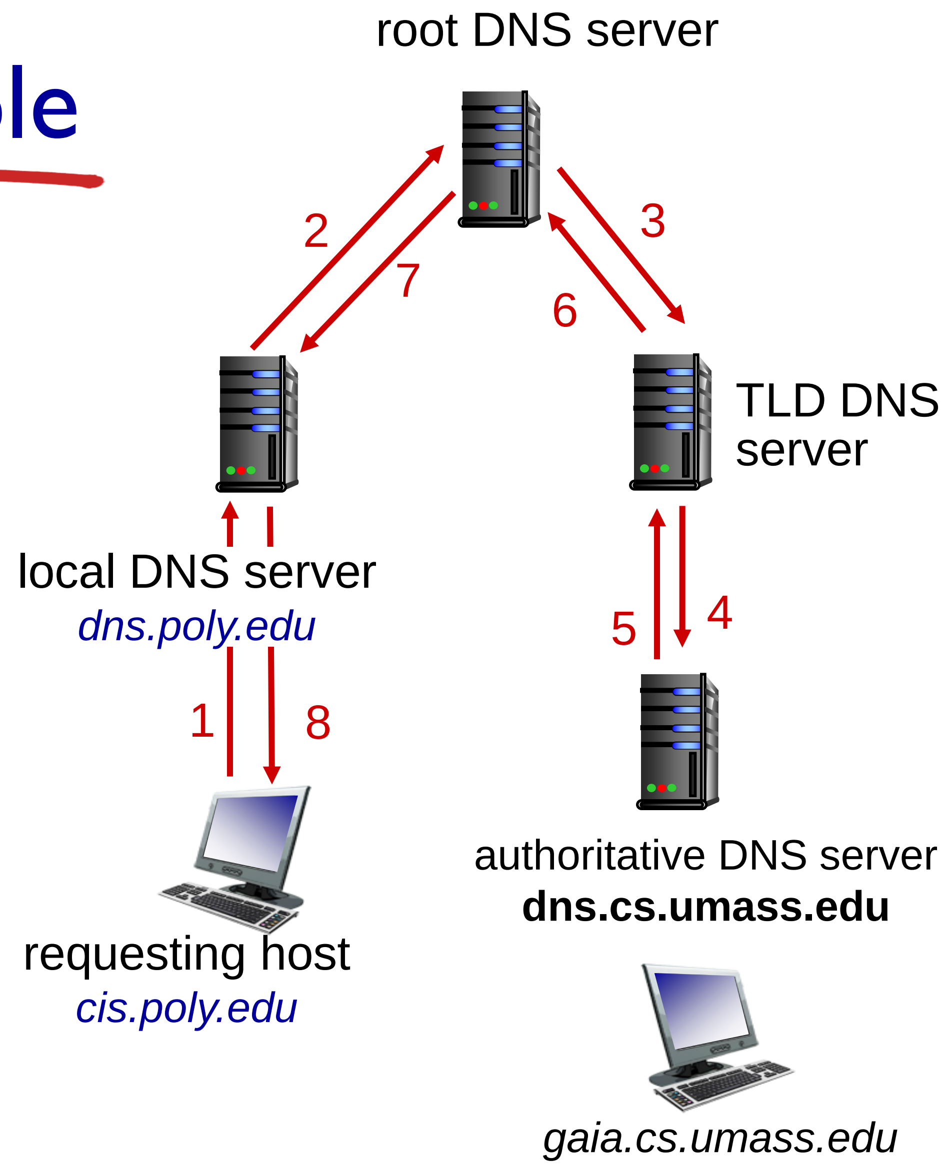
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

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



DNS records

DNS: distributed db storing resource records (RR)

RR format: `(name, value, type, ttl)`

type=A

- `name` is hostname
- `value` is IP address
- Ex: (relay1.bar.foo.com, 145.37.93.126, A)

type=NS

- `name` is domain (e.g., foo.com)
- `value` is hostname of authoritative name server for this domain
- Ex: (foo.com, dns.foo.com, NS)

type=CNAME

- `name` is alias name for some "canonical" (the real) name
- `value` is canonical name
- Ex: (foo.com, relay1.bar.foo.com, CNAME)

type=MX

- `value` is name of mailserver associated with `name`
- Ex: (foo.com, mail.bar.foo.com, MX)

DNS protocol, messages

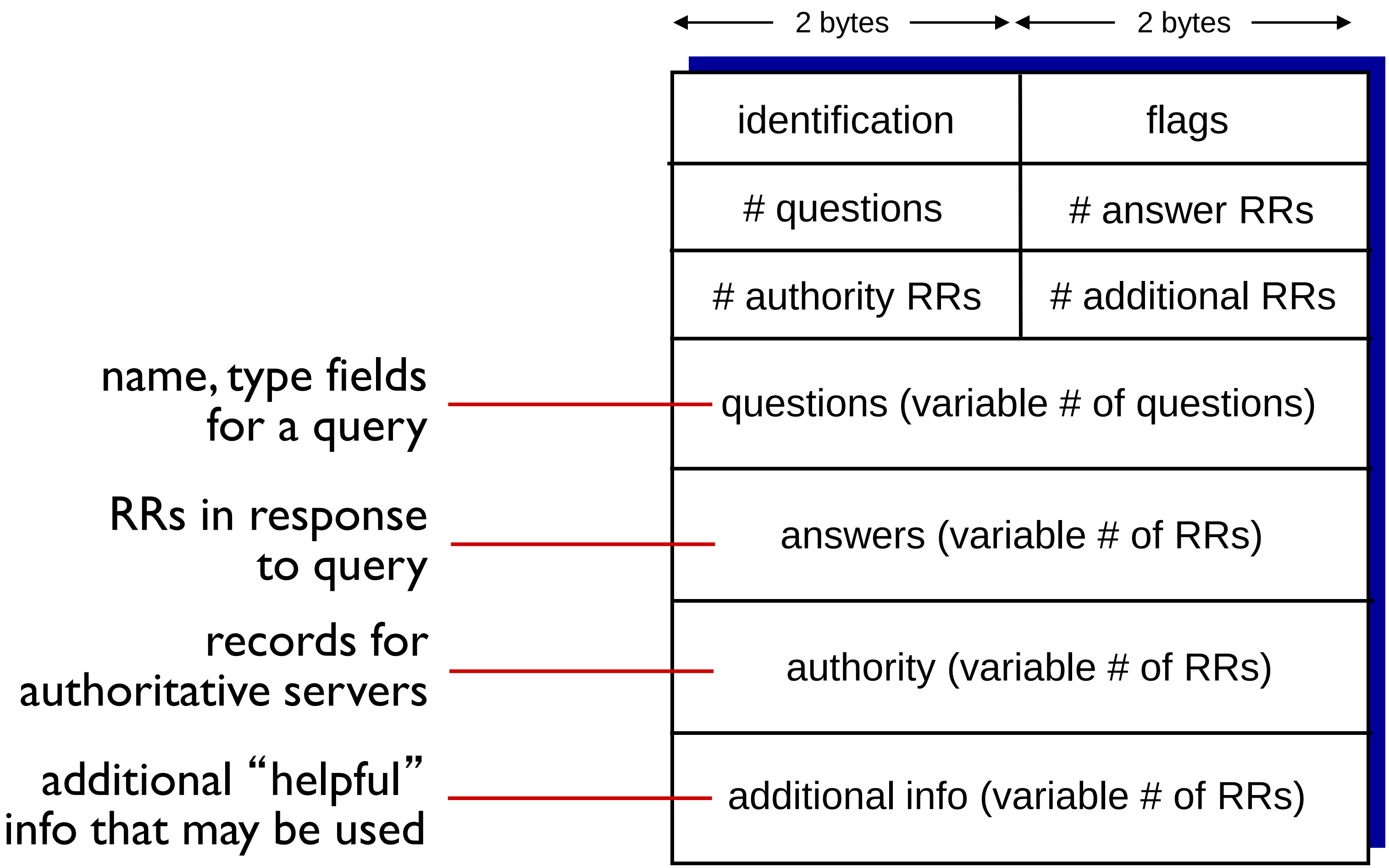
- *query* and *reply* messages, both with same *message format*

msg header

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

← 2 bytes →	← 2 bytes →
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

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

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

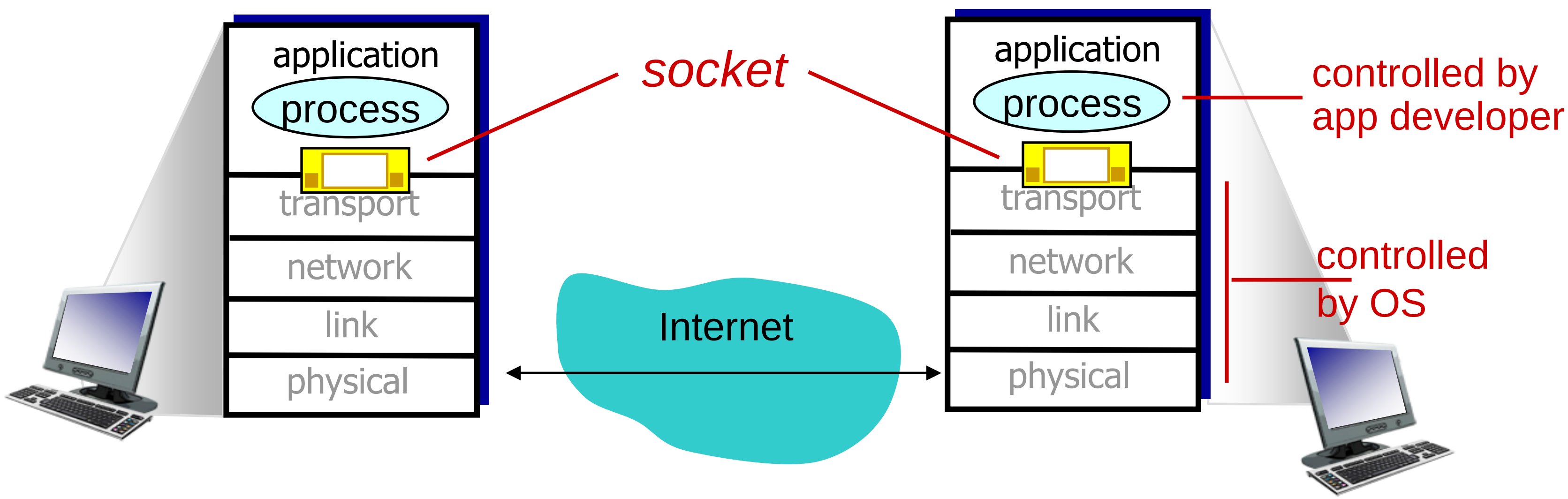
2.5 DNS

2.7 socket programming with UDP and TCP

Socket programming

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



Socket programming

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

Application Example:

1. Client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

Socket programming *with UDP*

UDP: no “connection” between client & server

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- rcvr extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes (“datagrams”) between client and server

Client/server socket interaction: UDP

server (running on serverIP)

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

Create datagram with server IP and
port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

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:

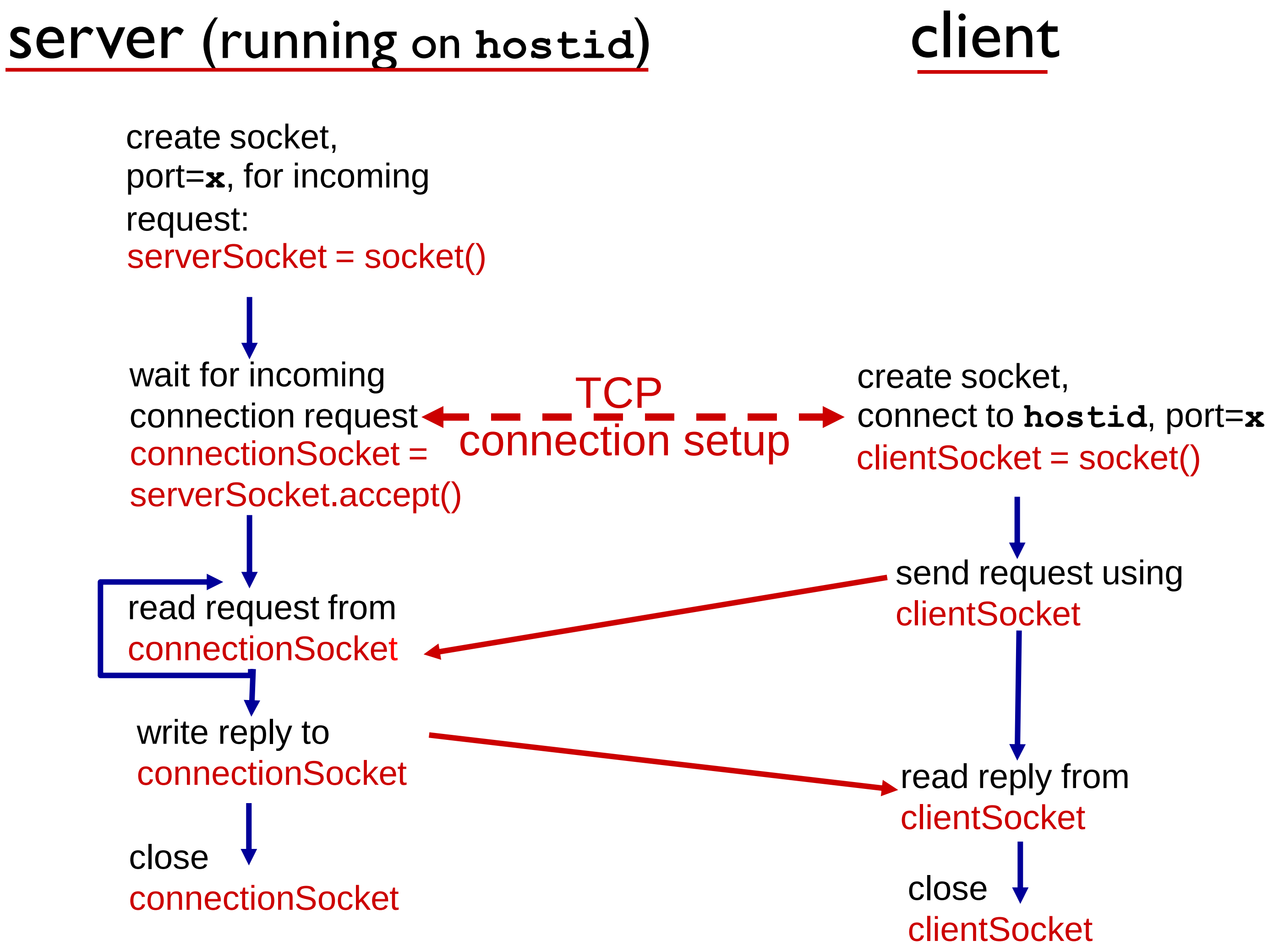
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint:

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

Client/server socket interaction: TCP



TCP – Client and Server Programs in Java

TCPClient.java

Here is the code for the client side of the application:

```
import java.io.*;
import java.net.*;
class TCPClient {
   public static void main(String argv[]) throws Exception
   {
      String sentence;
      String modifiedSentence;
      BufferedReader inFromUser = new BufferedReader(
               new InputStreamReader(System.in));
      Socket clientSocket = new Socket("hostname", 6789);
      DataOutputStream outToServer = new DataOutputStream(
               clientSocket.getOutputStream());
      BufferedReader inFromServer =
         new BufferedReader(new InputStreamReader(
               clientSocket.getInputStream()));
      sentence = inFromUser.readLine();
      outToServer.writeBytes(sentence + '\n');
      modifiedSentence = inFromServer.readLine();
      System.out.println("FROM SERVER: " +
                     modifiedSentence);
      clientSocket.close();
   }
}
```

TCP – Client and Server Programs in Java

TCPServer.java

Now let's take a look at the server program.

```
import java.io.*;
import java.net.*;
class TCPServer {

    public static void main(String argv[]) throws Exception
        {
            String clientSentence;
            String capitalizedSentence;
            ServerSocket welcomeSocket = new ServerSocket
               (6789);
            while(true) {
               Socket connectionSocket = welcomeSocket.
                  accept();
               BufferedReader inFromClient =
                  new BufferedReader(new InputStreamReader(
                     connectionSocket.getInputStream()));
               DataOutputStream outToClient =
                  new DataOutputStream(
                     connectionSocket.getOutputStream());
               clientSentence = inFromClient.readLine();
               capitalizedSentence =
                     clientSentence.toUpperCase() + '\n';
               outToClient.writeBytes(capitalizedSentence);
            }
        }
 }
```

GTU Questions

Problem - 1

```
GET /cs453/index.html HTTP/1.1<cr><lf>Host: gai
a.cs.umass.edu<cr><lf>User-Agent: Mozilla/5.0 (
Windows;U; Windows NT 5.1; en-US; rv:1.7.2) Gec
ko/20040804 Netscape/7.2 (ax) <cr><lf>Accept:ex
t/xml, application/xml, application/xhtml+xml, text
/html;q=0.9, text/plain;q=0.8,image/png,*/*;q=0.5
<cr><lf>Accept-Language: en-us,en;q=0.5<cr><lf>Accept-
Encoding: zip,deflate<cr><lf>Accept-Charset: ISO
-8859-1,utf-8;q=0.7,*;q=0.7<cr><lf>Keep-Alive: 300<cr>
<lf>Connection:keep-alive<cr><lf><cr><lf>
```

a. What is the URL of the document requested by the browser?

b. What version of HTTP is the browser running?

c. Does the browser request a non-persistent or a persistent connection?

d. What is the IP address of the host on which the browser is running?

e. What type of browser initiates this message? Why is the browser type needed in an HTTP request message?

Problem – 1 (Solution)

a) The document request was http://gaia.cs.umass.edu/cs453/index.html. The Host : field indicates the server's name and /cs453/index.html indicates the file name.

b) The browser is running HTTP version 1.1, as indicated just before the first <cr><lf> pair.

c) The browser is requesting a persistent connection, as indicated by the Connection: keep-alive.

d) This is a trick question. This information is not contained in an HTTP message anywhere. So there is no way to tell this from looking at the exchange of HTTP messages alone. One would need information from the IP datagrams (that carried the TCP segment that carried the HTTP GET request) to answer this question.

e) Mozilla/5.0. The browser type information is needed by the server to send different versions of the same object to different types of browsers.

Problem - 2

(c) Consider the following HTTP message and answer the following questions: 07

GET /home.asp HTTP/1.1
Host: gtu.ac.in
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cookie: OGPC=5061921-11:5061952-13:5061985-24:5061983-27:5061968-13:5062004-7:5062009-6:5062022-12:;
SID=DQAAALgBAAA3RAje9UlLOOSuH0G91uzL5JOJNUYU2aVOml6jEWVTCo9-
User-Agent: Chrome/49.0.2623.110
X-Client-Data: CIS2yQEIprbJAQjDtskBCP2VygE
Connection:keep-alive

i. From which browser URL is requested?
ii. Does browser request a non-persistent or a persistent connection?
iii. Which is the (complete) URL of the document requested by the user?
iv. Which HTML method is used to retrieve the requested URL?

Problem - 3

P5. The text below shows the reply sent from the server in response to the HTTP GET message in the question above. Answer the following questions, indicating where in the message below you find the answer.

```
HTTP/1.1 200 OK<cr><lf>Date: Tue, 07 Mar 2008
12:39:45GMT<cr><lf>Server: Apache/2.0.52 (Fedora)
<cr><lf>Last-Modified: Sat, 10 Dec2005 18:27:46
GMT<cr><lf>ETag: “526c3-f22-a88a4c80”<cr><lf>Accept-
Ranges: bytes<cr><lf>Content-Length: 3874<cr><lf>
Keep-Alive: timeout=max=100<cr><lf>Connection:
Keep-Alive<cr><lf>Content-Type: text/html; charset=
ISO-8859-1<cr><lf><cr><lf><!doctype html public “-
//w3c//dtd html 4.0 transitional//en”><lf><html><lf>
<head><lf> <meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1"><lf> <meta
name="GENERATOR" content="Mozilla/4.79 [en] (Windows NT
5.0; U) Netscape]"><lf> <title>CMPSCI 453 / 591 /
NTU-ST550A Spring 2005 homepage</title><lf></head><lf>
<much more document text following here (not shown)>
```

a. Was the server able to successfully find the document or not? What time was the document reply provided?

b. When was the document last modified?

c. How many bytes are there in the document being returned?

d. What are the first 5 bytes of the document being returned? Did the server agree to a persistent connection?

Problem – 3 (Solution)

a) The status code of 200 and the phrase OK indicate that the server was able to locate the document successfully. The reply was provided on Tuesday, 07 Mar 2008 12:39:45 Greenwich Mean Time.

b) The document index.html was last modified on Saturday 10 Dec 2005 18:27:46 GMT.

c) There are 3874 bytes in the document being returned.

d) The first five bytes of the returned document are : <!doc. The server agreed to a persistent connection, as indicated by the Connection: Keep-Alive field

(b) Why distributed database design is more preferred over centralized design to implement DNS in the Internet? Justify. Also explain the way of DNS servers to handle the recursive DNS query using suitable diagram. **08**

(b) Explain the working of electronic mail protocols SMTP, IMAP and POP3 in brief with suitable diagram. **08**

(a) What is HTTP? Differentiate its persistent and non-persistent types with request-response behavior of HTTP. **06**

(b) Explain the concept of Cookies and its components with suitable example. **08**

OR

(b) Explain the high-level view of Internet e-mail system and its major components. **08**

(b) Explain HTTP GET and HTTP POST method in detail. **04**

(a) What is proxy server? What are the benefits of caching proxy server? **03**

(c) Write a note on "Dynamic Host Configuration Protocol". **07**

Briefly explain the working of SMTP.

How DNS is useful in Internet?

(b) Explain the movement of files between local and remote systems using FTP. **04**

What is HTTP? Explain with respect to persistent and non-persistent connections. **07**

Discuss the DNS services in detail. **07**