Computer Networks & IOT (18B11CS311)

Even Semester_2022

Application Layer

Topics to be covered under this:

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

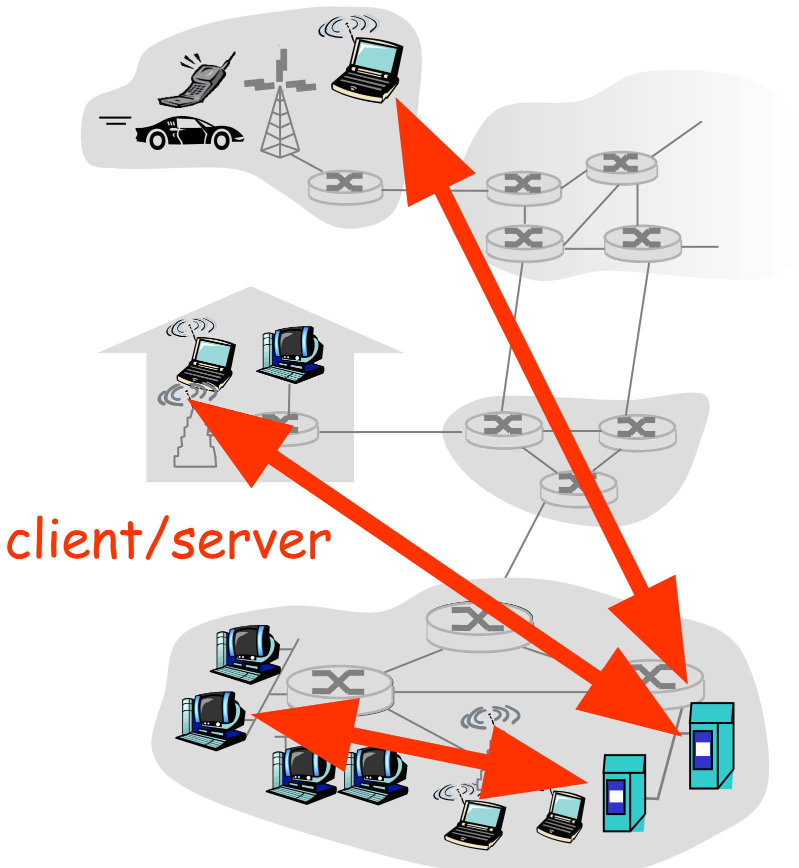
Some network apps

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

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

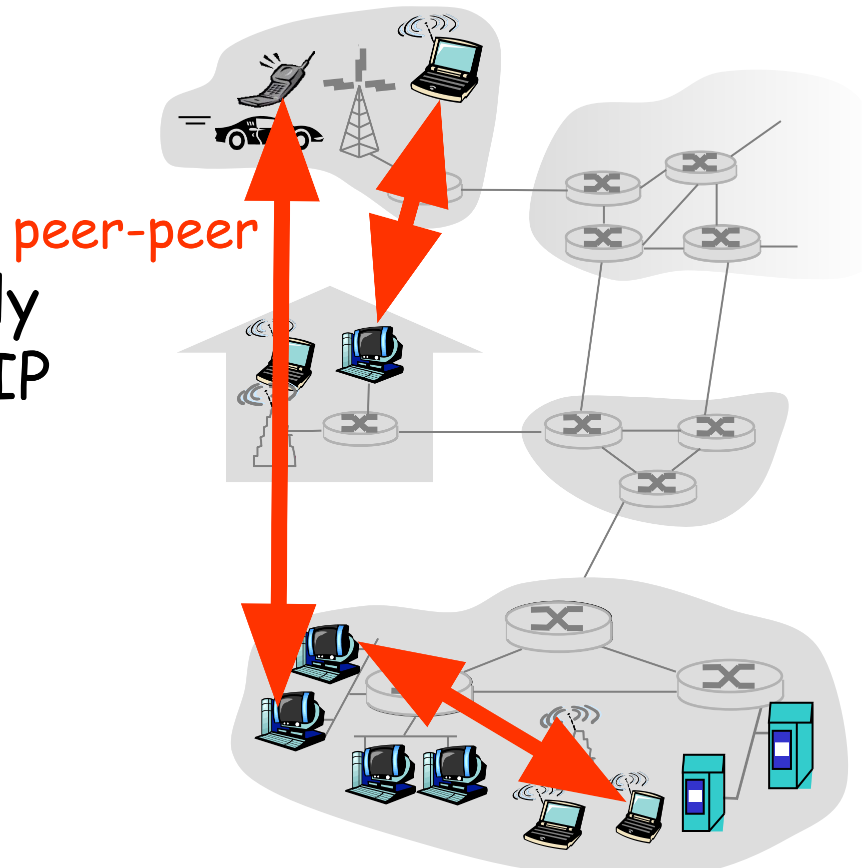
clients:

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

Infrastructure intensive applications: web based email(yahoo mail), internet commerce (amazon),search engine (Google), video sharing (youtube), social networking (Facebook).

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

Traffic intensive applications: File distribution (BitTorent), File sharing (LimeWire), Internet Telephony (Skype).

Hybrid Architecture: Instant Messaging.

Processes communicating

Process: program running within a host.

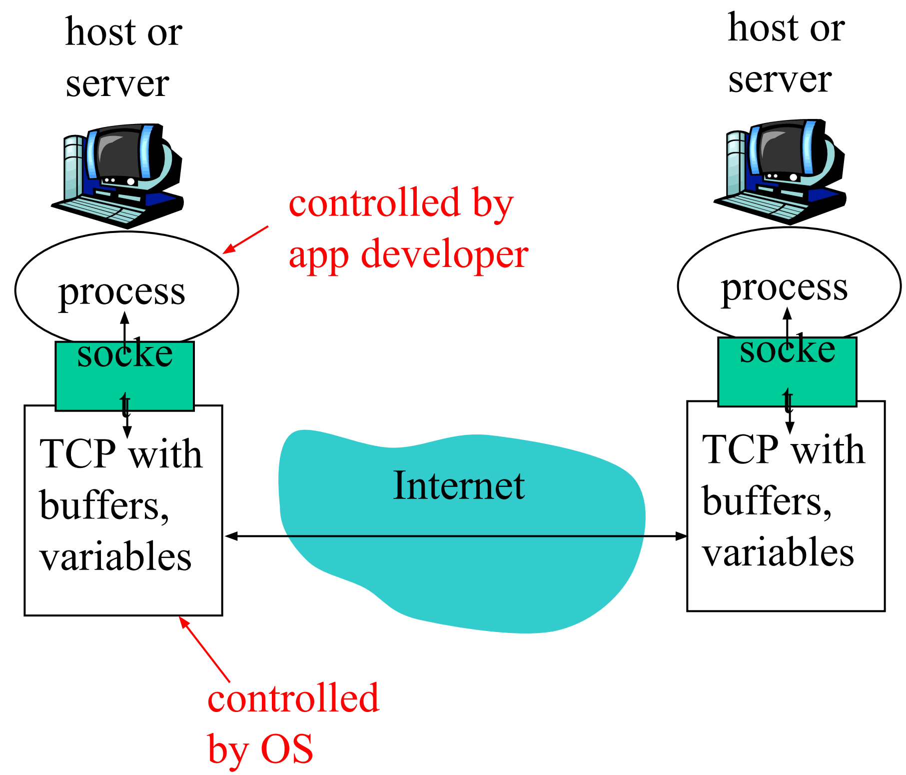
- within same host, two processes communicate using inter-process communication (defined by OS).
- processes in different hosts communicate by exchanging messages

Client process: process that initiates communication

Server process: process that waits to be contacted

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



Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address

Ques: does IP address of host suffice for identifying the process?

Answer:

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

App-layer protocol defines:

how an application processes running on different end systems pass messages to each other

- 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

What transport service does an app need?

Data loss

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

Timing

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

Throughput

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

Security

- Encryption, data integrity, …

Transport service requirements of common apps

Application	Data loss	Throughput	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Internet 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 (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Web and HTTP

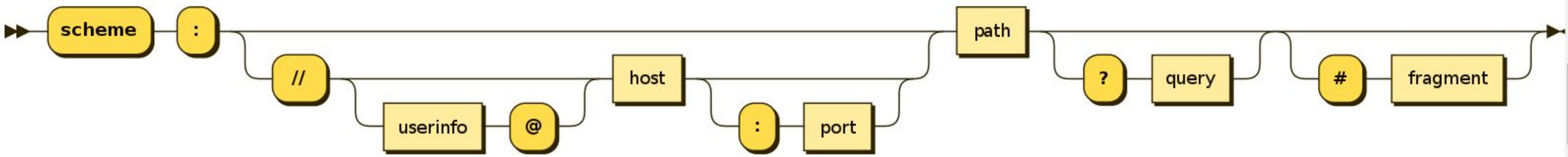
First some jargon

- Web page also called document 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.
- Base HTML references the object in the page with the object's addressable URL
- Example URL:

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

http://www.someschools.edu:80/calender.cgi?month=july#week3

This is represented in a syntax diagram as:



- Examples of popular schemes include `http`, `https`, `ftp`, `mailto`, `file`, `data` and `irc`
- Optional **userinfo** subcomponent that may consist of a user name and an optional password preceded by a colon.
- A **host** subcomponent, consisting of either a registered hostname or an IP address.
- An optional **port** subcomponent preceded by a colon.
- A **path** component, consisting of a sequence of path segments separated by a slash.
- The last part of a *path* is named **pathinfo** and it is optional.

Example:

URI: `"http://www.example.com/questions/3456/my-document"`

where: `"/questions"` is the first part of the *path* (an executable module or program) and `"/3456/my-document"` is the second part of the *path* named *pathinfo*, which is passed to the executable module or program named `"/questions"` to select the requested document.

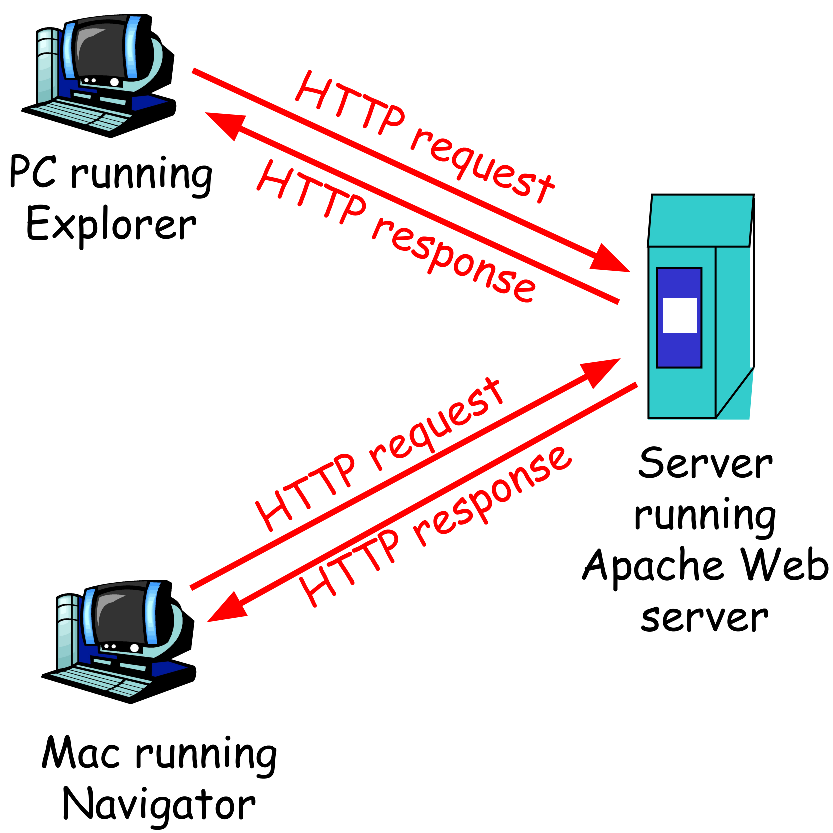
CONT..

- An optional query component preceded by a question mark (?), containing a query string of non-hierarchical data. Its syntax is not well defined, but by convention is most often a sequence of attribute–value pairs separated by a delimiter.
- A Query String is helpful when we want to transfer a value from one page to another
- An optional fragment component preceded by a hash (#). A fragment is an internal page reference, sometimes called a named anchor. It usually appears at the end of a URL and begins with a hash (#) character followed by an identifier. It refers to a section within a web page.

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- ***HTTP*** is an application-layer protocol for transmitting hypermedia documents, such as HTML.
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests



HTTP overview (continued)

Uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests

aside

Protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections

Nonpersistent HTTP

- At most one object is sent over a TCP connection.

Connection: close

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.
- Also known as HTTP Keep-alive or HTTP Connection reuse.

Nonpersistent HTTP

Suppose user enters URL

`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.

time

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

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

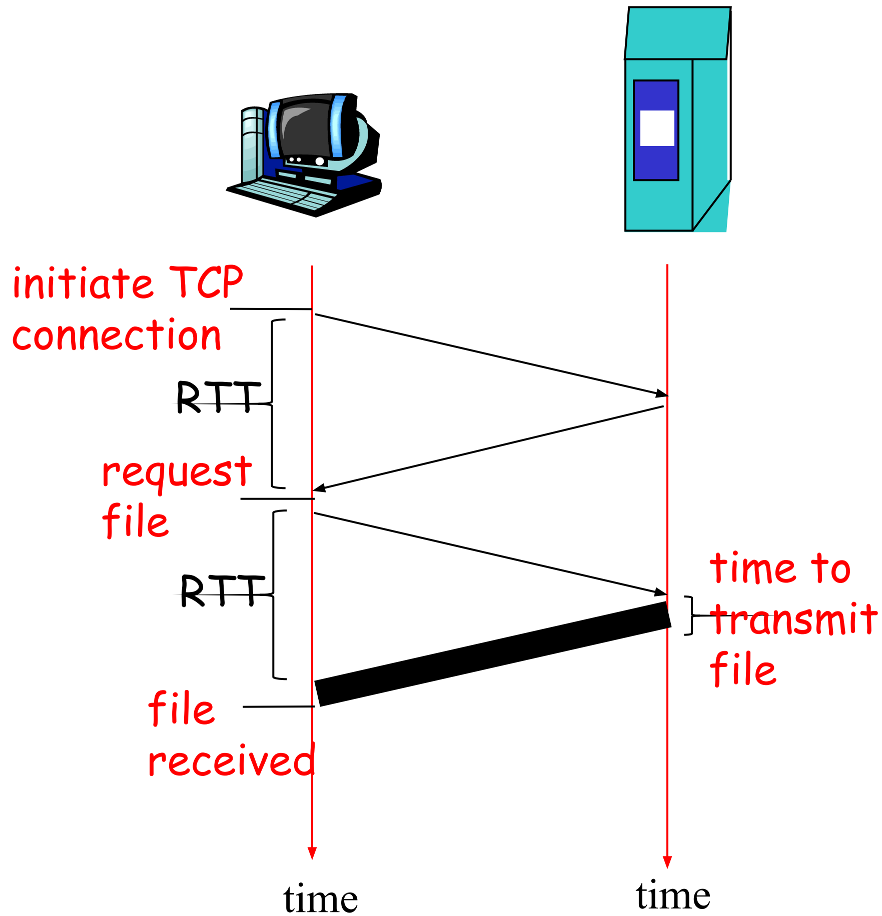
Non-Persistent HTTP: Response time

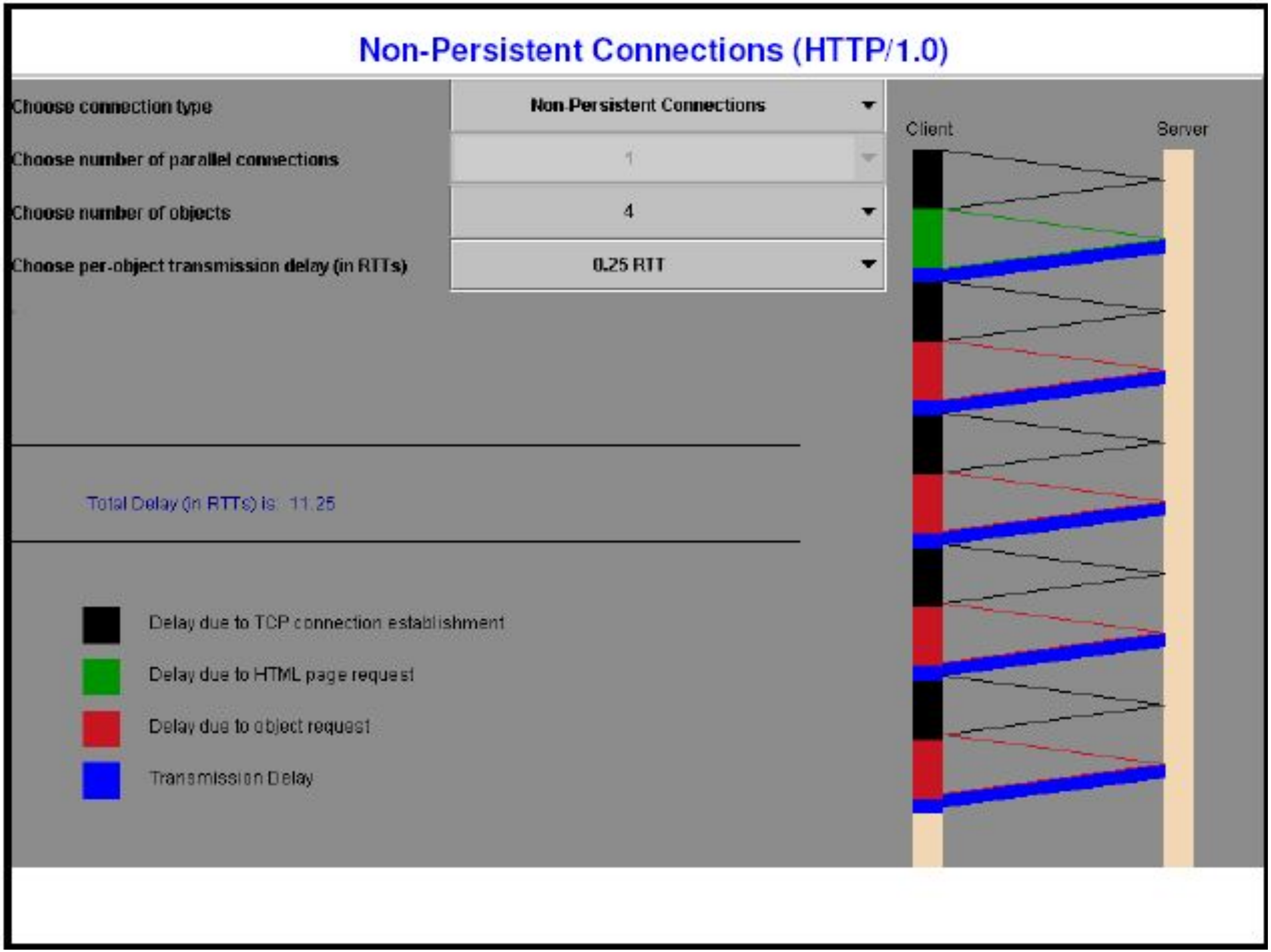
Definition of RTT: time for a small packet to travel from client to server and back.

Response time:

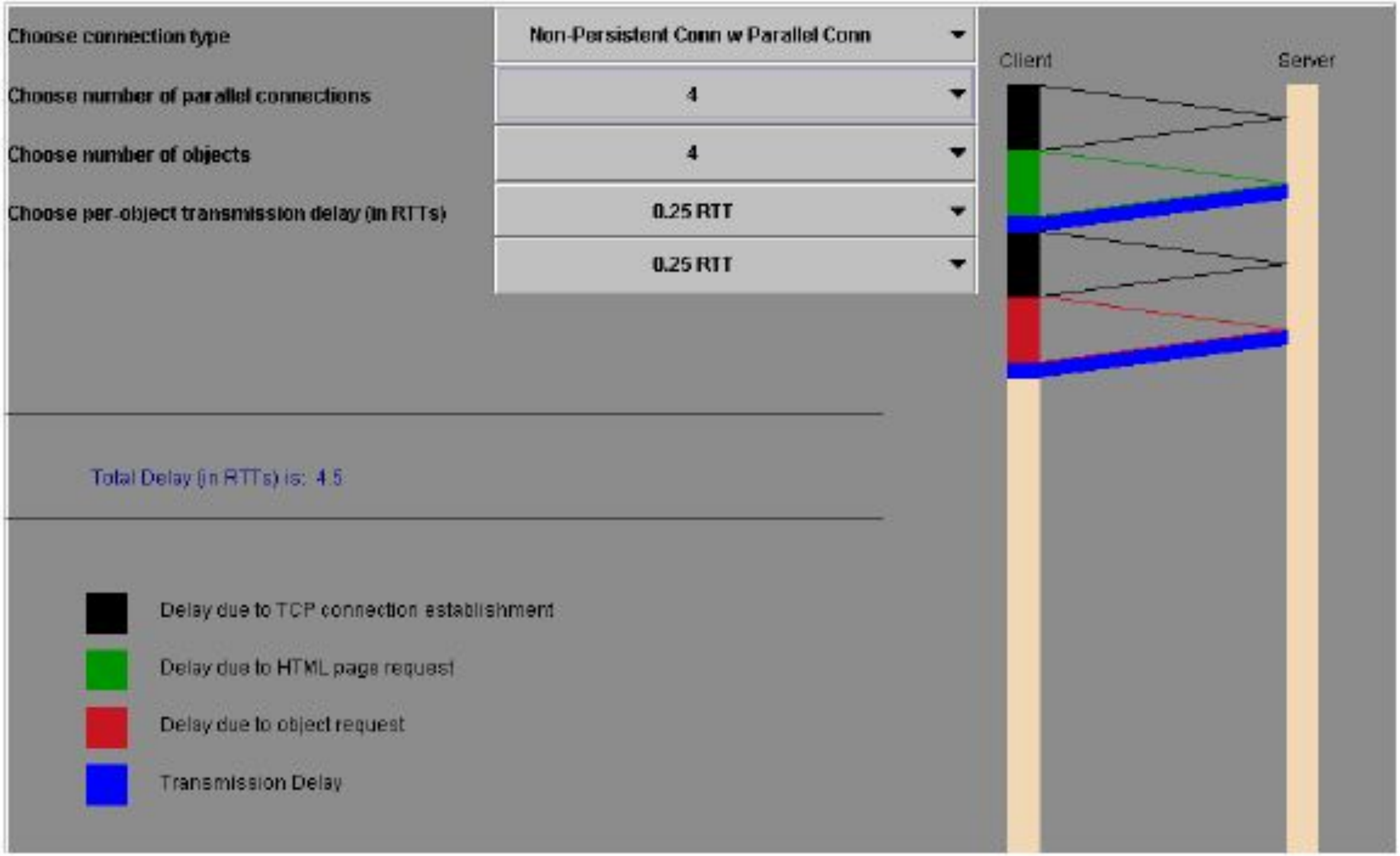
- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

total = 2RTT+transmit time




Non-Persistent Connections (HTTP/1.0)
Choose connection type
Non-Persistent Connections
Choose number of parallel connections
1
Choose number of objects
4
Choose per-object transmission delay (in RTTs)
0.25 RTT
Total Delay (in RTTs) is: 11.25
Delay due to TCP connection establishment
Delay due to HTML page request
Delay due to object request
Transmission Delay
Client
Server

Non-Persistent with Parallel Sessions



Nonpersistent HTTP issues:

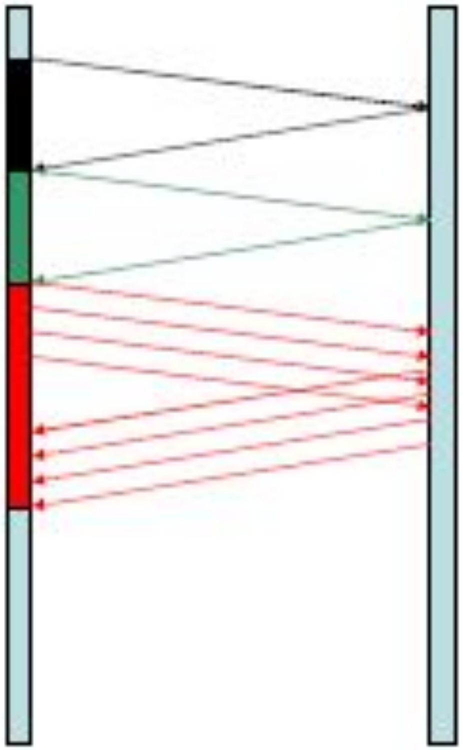
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

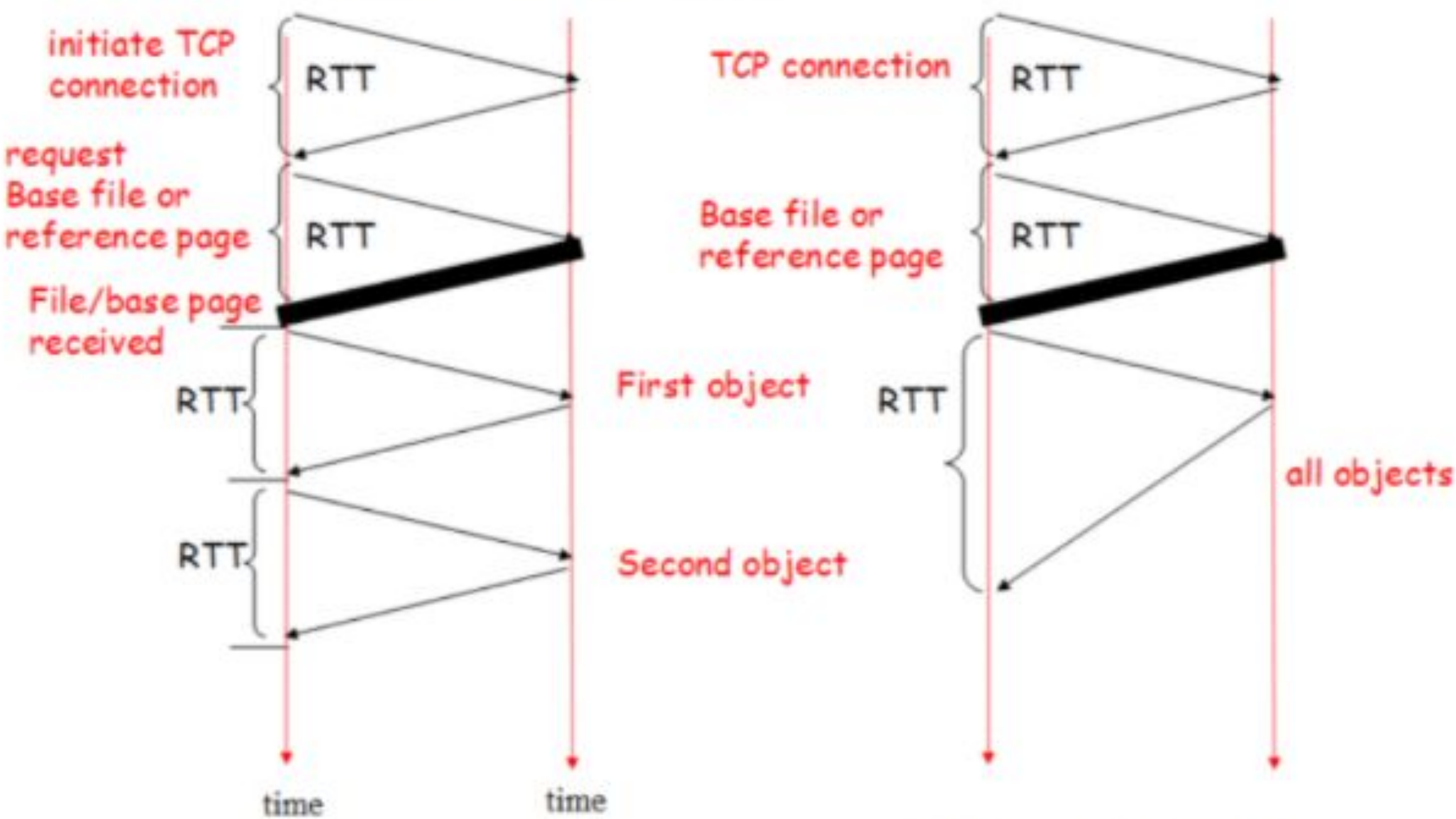
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

* With Pipelining and without pipelining

Persistent with Pipelining



Persistent & Pipelined/non-pipelined connections



- Persistent without pipelining
- Persistent with pipelining

Question

Suppose the HTML file references three very small objects on the same server. Neglecting transmission times how much time elapses with

a) Non persistent HTTP?
b) Persistent HTTP without pipelining?
c) Persistent HTTP with pipelining?

HTTP request message

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

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

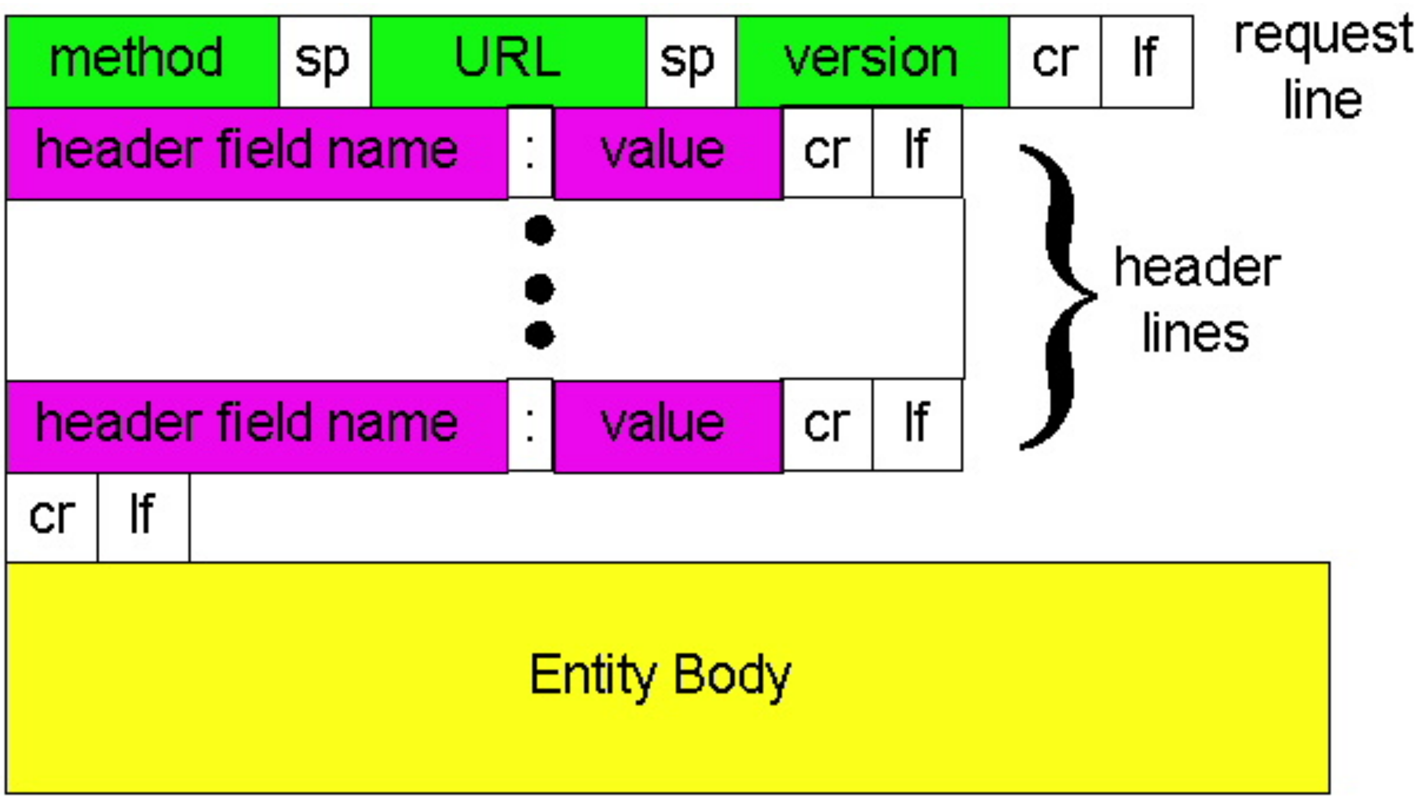
(extra carriage return, line feed)

request line (GET, POST, HEAD commands)

header lines

Carriage return, line feed indicates end of message

HTTP request message: general format



HTTP request Method types

HTTP/1.0

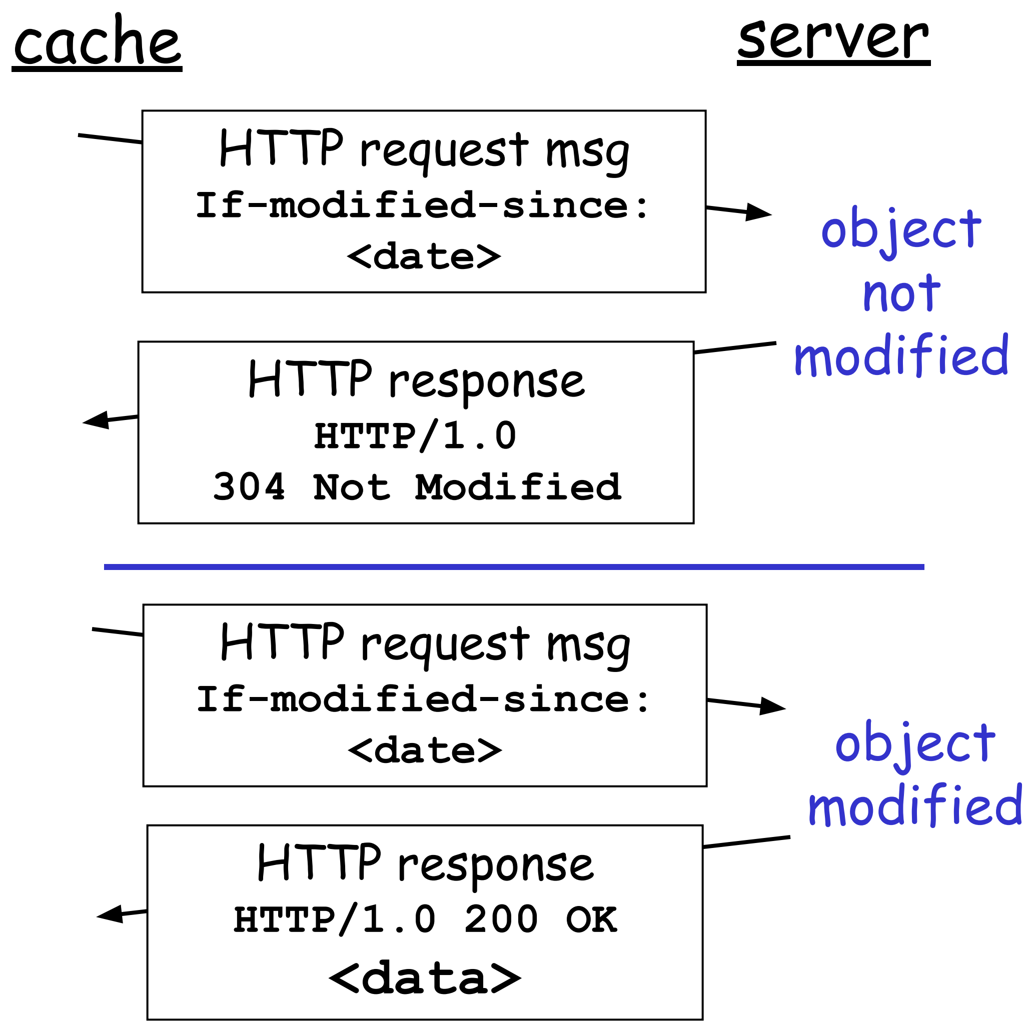
1. GET
2. POST
3. HEAD

HTTP/1.1

- GET, POST, HEAD
- PUT
  - uploads an object used in conjunction with web publishing tools.
  - uploads an object to a specific path (directory)in URL field on a specific web server.
- DELETE
  - deletes file specified in the URL field

Conditional GET

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



HTTP request Header field lines

These fields are used to pass additional information about the request or about the client or to add conditions to the request.

Common HTTP request headers:-

1.Host
2. Content Length
3. Content-type
4. Authentication
5. Use Agent
6. Accept language
8. Cookie

HTTP response message

status line (protocol status code status phrase)

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

header lines

data, e.g., requested HTML file

HTTP response status codes

1xx :- indicates informational message only

2xx :- indicates that all is well with the request

3xx :- indicates some form of redirection i.e. redirects the client to another URL

4xx :- indicates an error and server is blaming the browser for doing something wrong

5xx :- indicates an error on the servers part.

HTTP response status codes

In first line in server->client response message.

A few sample codes:

100 CONTINUE partial request received, continue until reject

101 SWITCHING PROTOCOLS Server switches protocol

200 OK

- request succeeded, response is included in the content

200 No response

- request succeeded, but no response is provided.

301 Moved Permanently

- Indicates that URL of the requested resource has changed.

302 Found

- It functions like 301 response except that the move is temporary

303 see other

- It indicates that the resource has temporarily moved and it is obtained from new URL via GET request only.

400 Bad Request

- request message not understood by server due to bad syntax

401 Unauthorized

- Indicates that the requested resource is in a protected medium.

403 Forbidden

- Indicates that client is not allowed to access the requested resource for some reasons, other than valid HTTP login.

404 Not Found

- requested document not found on this server

408 Request time out

500 Internal server error

- Indicates that something happened on the server that caused the transaction to fail.

503 Service unavailable

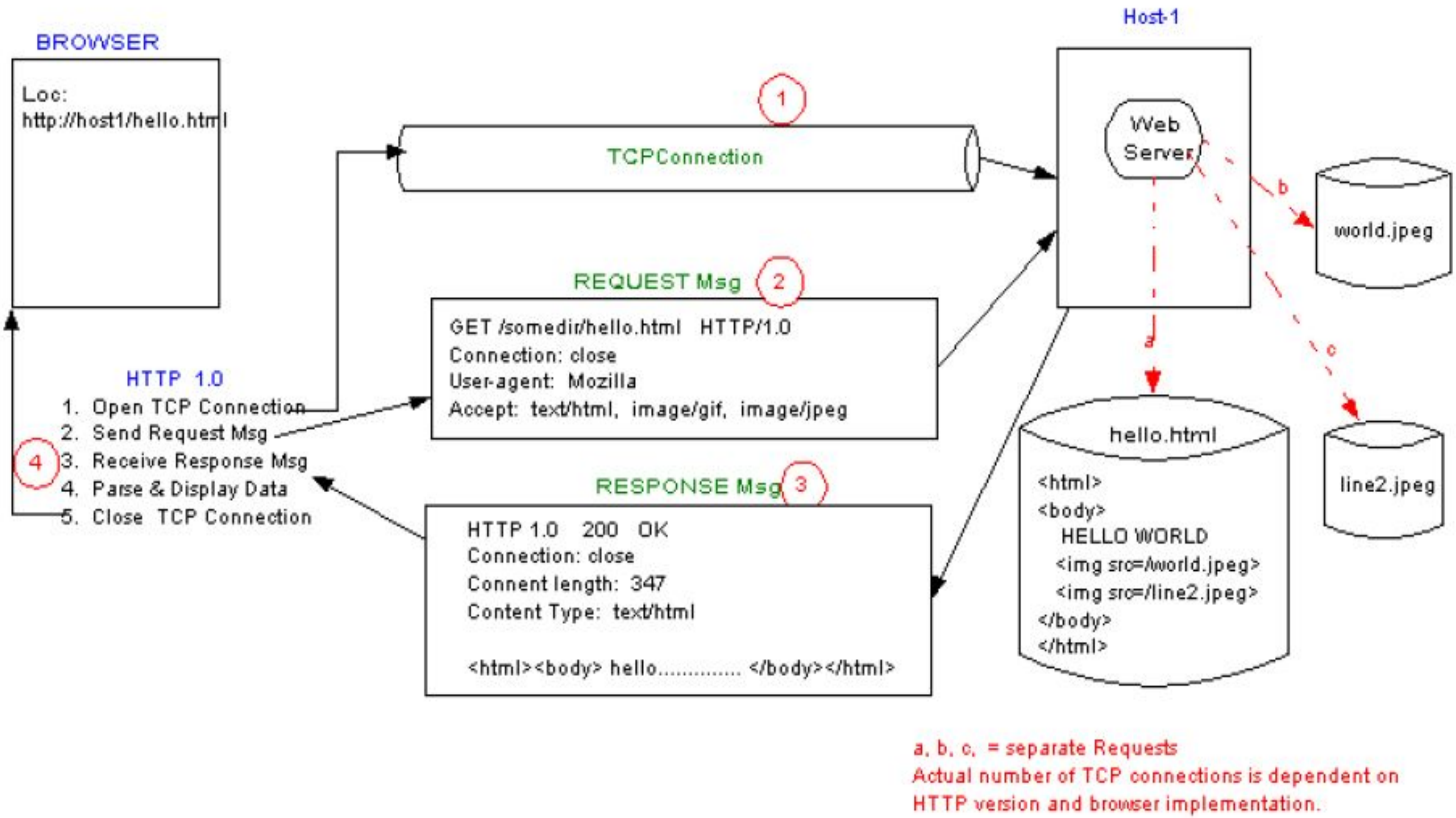
- Indicates that the server is unable to respond to the request due to a high volume of traffic

505 HTTP Version Not Supported

HTTP response server Headers

1. Content length
2. Content type
3. Date
4. Last Modified
5. Location
6. Server
7. Set cookie

HTTP Conceptual Architechure



State True or False?

a) Suppose a user requests a Web page that consists of some text and two images. For this page the client will send one request message and recieve three response messages?

b) True or false. Two distinct Web pages (e.g., www.mit.edu/research.html and www.mit.edu/students.html) can be sent over the same persistent connection?

c) With non-persistent connections between browser and origin server, it is possible for a single TCP segment to carry two distinct HTTP request messages?

d) The `Date:` header in the HTTP response message indicates when the object in the response was last modified?

Ques:

GET /cs453/index.html HTTP/1.1<cr><lf>Host: gaia.cs.umass.edu<cr><lf>User-Agent: Mozilla/5.0 (Windows;U; Windows NT 5.1; en-US; rv:1.7.2) Gecko/20040804 Netscape/7.2 (ax) <cr><lf>Accept:ext/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?

Ques:

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, indicat- ing 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- odified: Sat, 10 Dec2005 18:27:46 GMT<cr><lf>ETag: "526c3-f22- 88a4c80"<cr><lf>Accept- Ranges: bytes<cr><lf>Content-Length: 3874<cr><lf> Keep-Alive: timeout=max=100<cr><lf>Connection: Keep-live<cr><lf>Content-Type: text/html; charset= ISO-8859- <cr><lf><cr><lf><!doctype html public "-//w3c//dtd html 4.0 transitional//en"><lf><html><lf> <head><lf> <meta http-equiv="Content- ype" 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?

User-server state: cookies

Many major Web sites use cookies

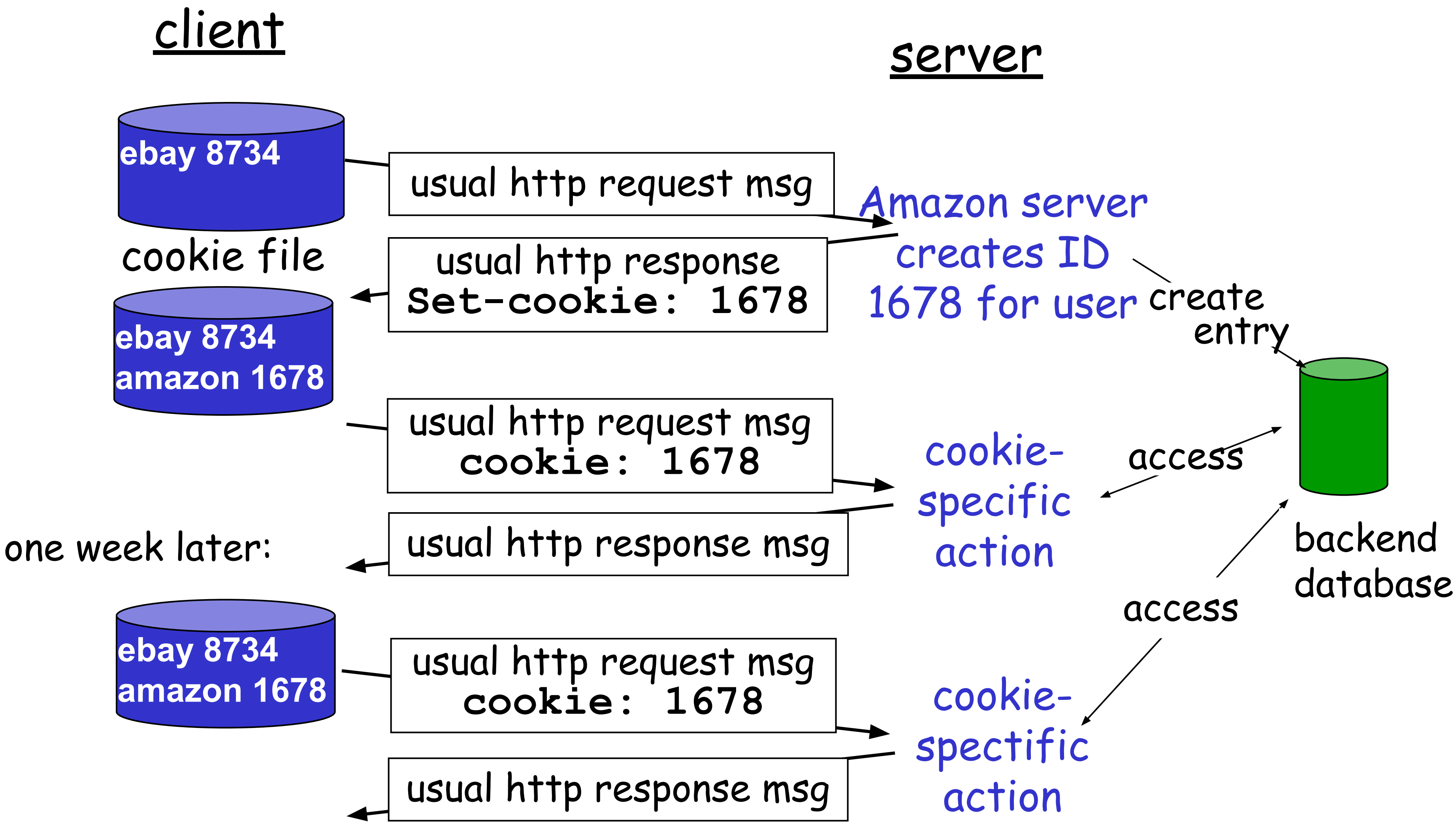
Four components:

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 always from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping “state” (cont.)



Cookies (continued)

What cookies can bring:

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

aside

Cookies and privacy:

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

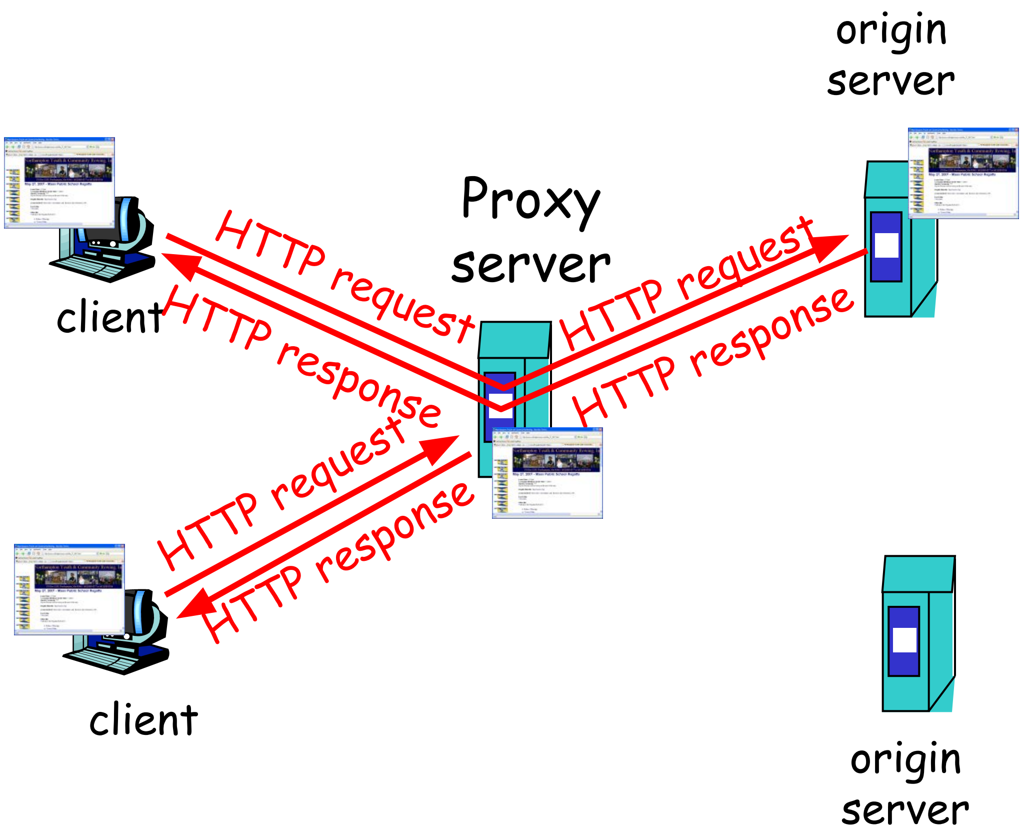
How to keep "state":

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

Web caches (proxy server)

Goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



More about Web caching

- cache acts as both client and server
- typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?

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

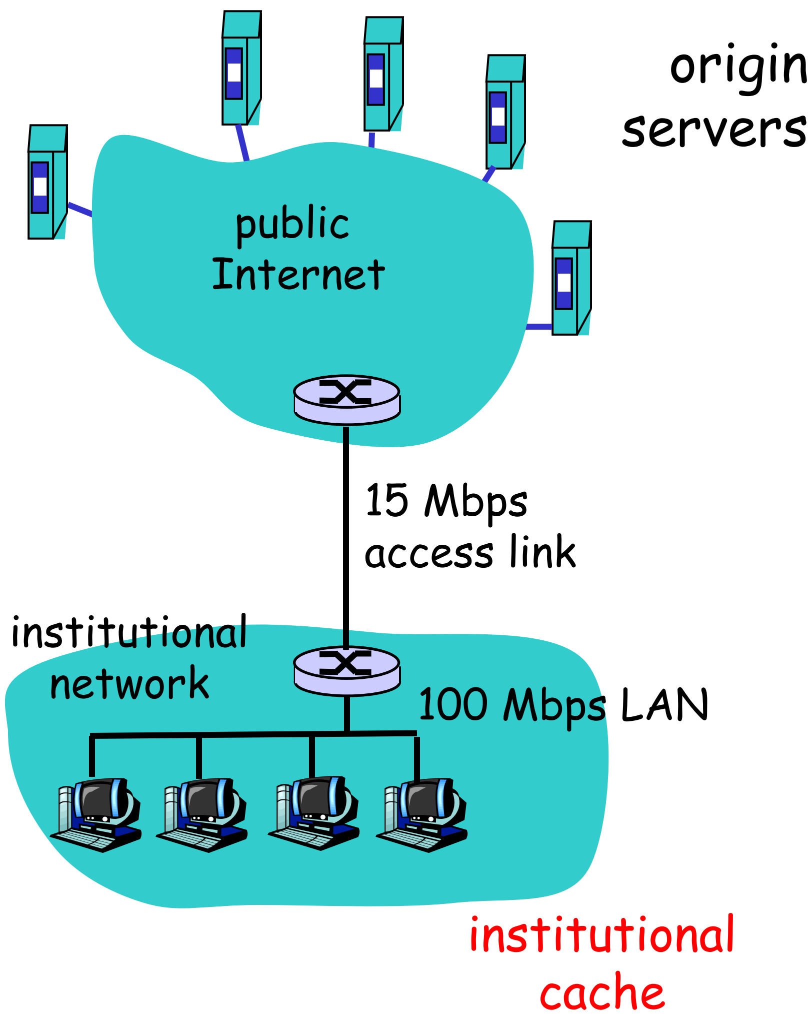
Caching example

Assumptions

- average object size = 1M bits
- avg. request rate from institution's browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

Consequences

- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + milliseconds



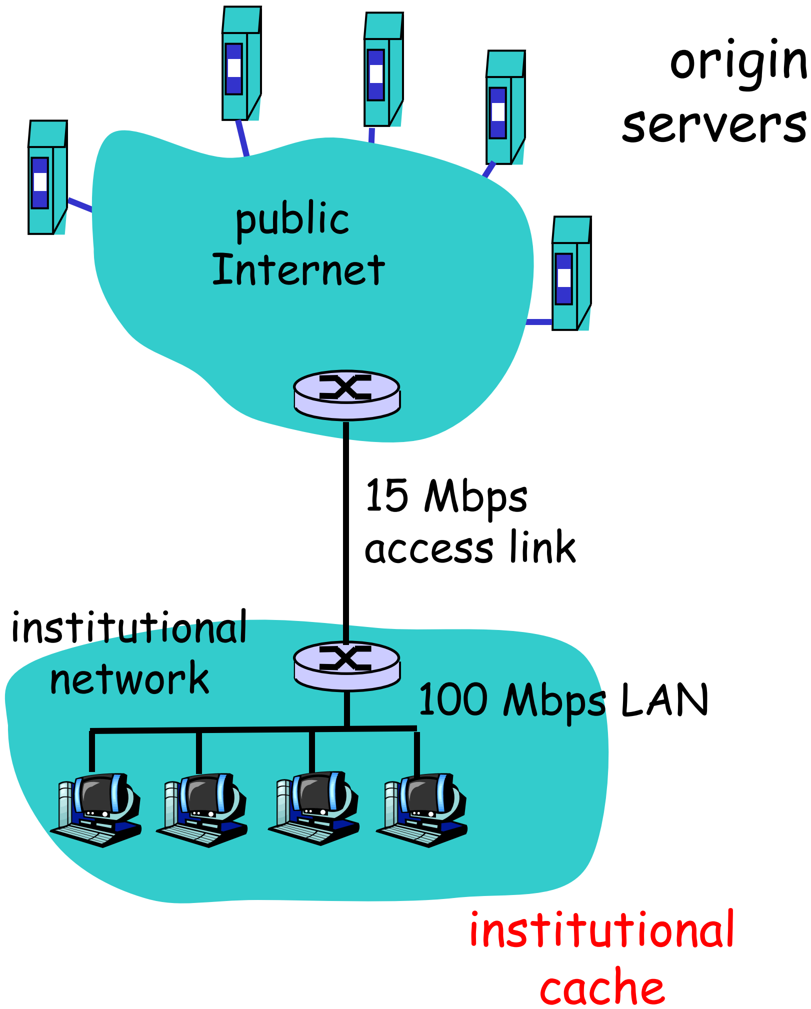
Caching example (cont)

possible solution

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

consequence

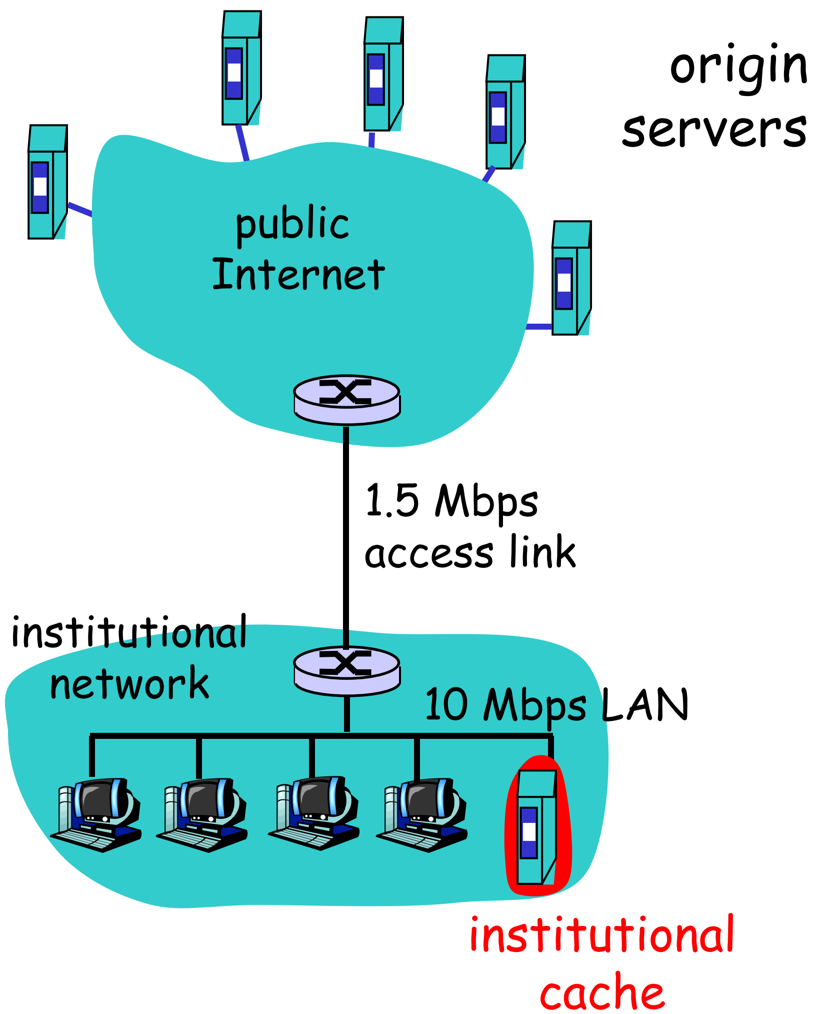
- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay
  = 2 sec + msecs + msecs
- often a costly upgrade



Caching example (cont)

possible solution: install cache
- suppose hit rate is 0.4

consequence
- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay = .6*(2.01) secs + .4*.01seconds < 1.4 secs



Using a Web browser, you visit the web site for www.hamburger.com. The base HTML page for the main page www.hamburger.com is 30,000 bits. Once the base HTML page is fetched, it contains URL references for the following embedded images:

http://www.hamburger.com/burger banner.jpg 15,000 bits
http://www.hamburger.com/lettuce.jpg 5,000 bits
http://www.hamburger.com/mmm bacon.jpg 10,000 bits
http://www.hamburger.com/veggie.jpg 10,000 bits
http://www.hamburger.com/disclaimer.txt 5,000 bits
http://www.hamburger.com/royale with cheese.jpg 35,000 bits

Your Web browser uses the HTTP protocol to download the base page and the embedded objects. Make the following assumptions:

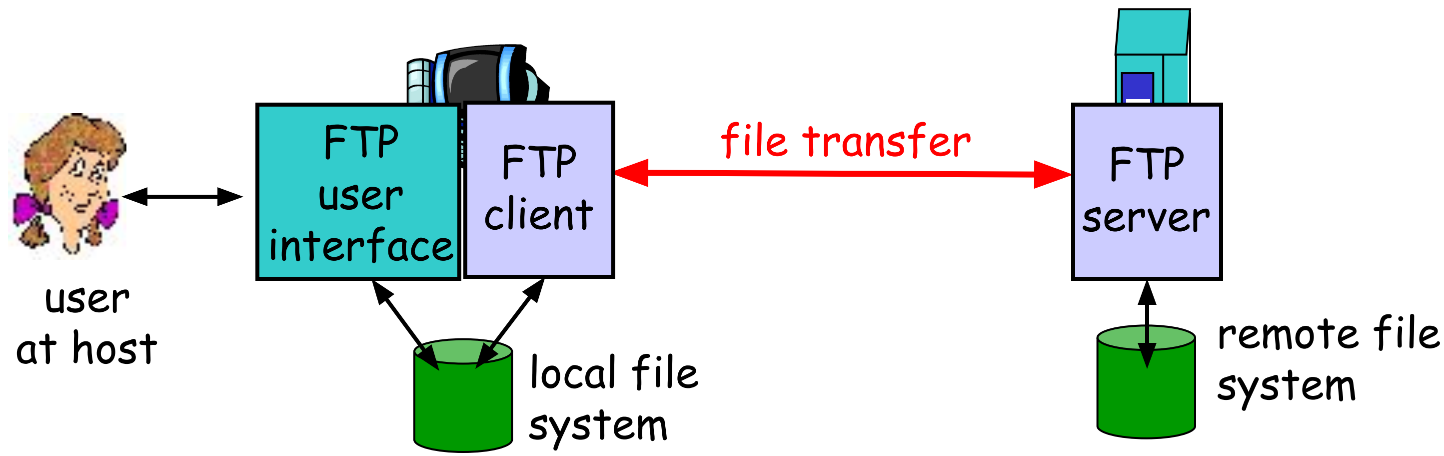
- At most 10,000 bits of data fits into a single packet. You can ignore the overhead of any headers or framing.
- You must first download the entire base page before you can start fetching the embedded images.
- HTTP requests are 1,000 bits in size.
- Any new connection to a machine requires a connection-establishment handshake.

- For this problem, you do not need to worry about closing connections, and you can ignore the delay introduced in acknowledging the final data packet sent by the server to your browser.
- All senders use windows of 20,000 bits.
- No packets are lost.

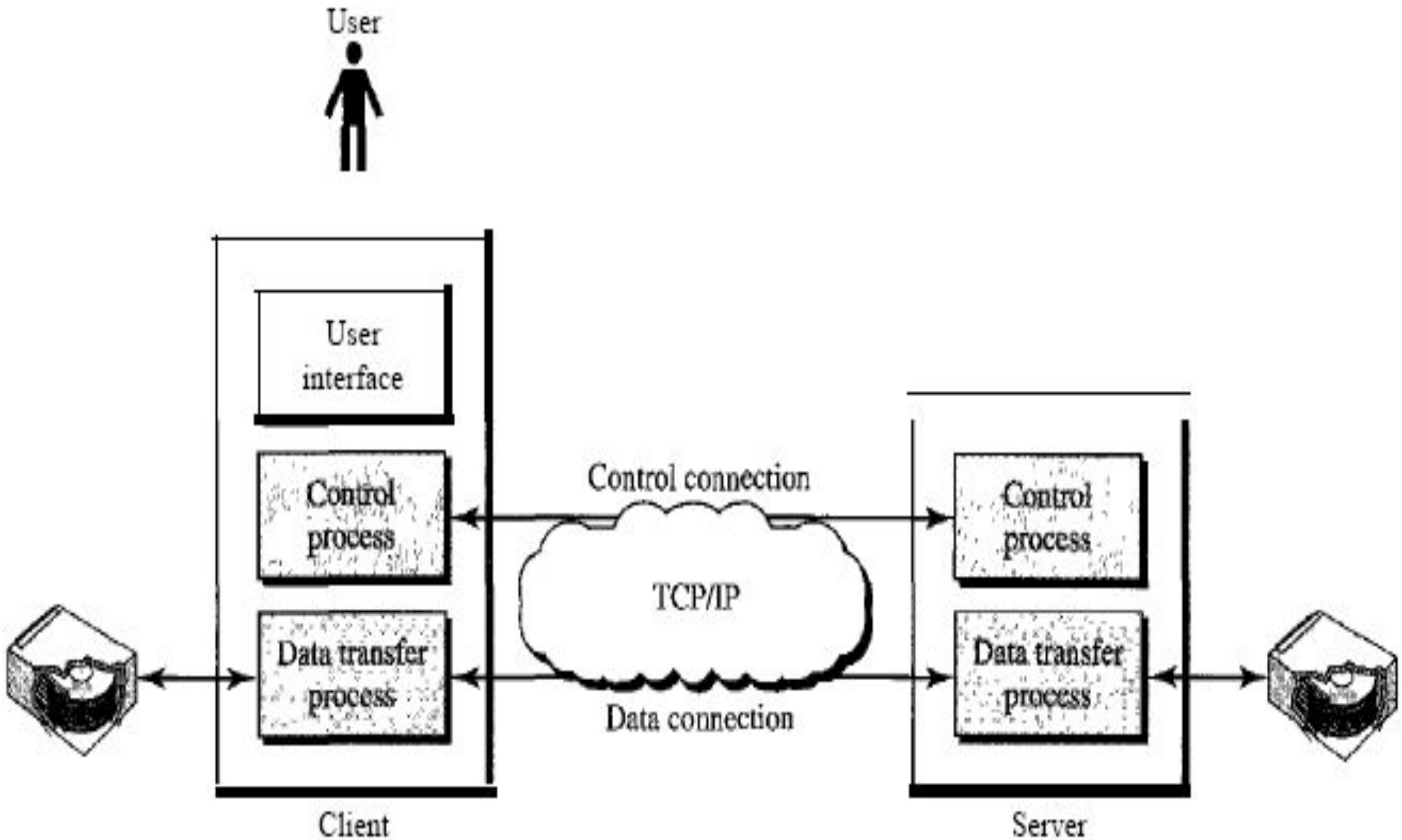
a. For the initial transfer of the home page, how many RTTs are required, and what occurs during each of them?
b. How quickly (in terms of RTTs) can your browser download the base page for www.hamburger.com and all embedded objects if the browser uses:
    i. One connection per item, with up to 4 concurrent connections.
    ii. A single persistent, non-pipelined connection.

FTP

FTP: the file transfer protocol

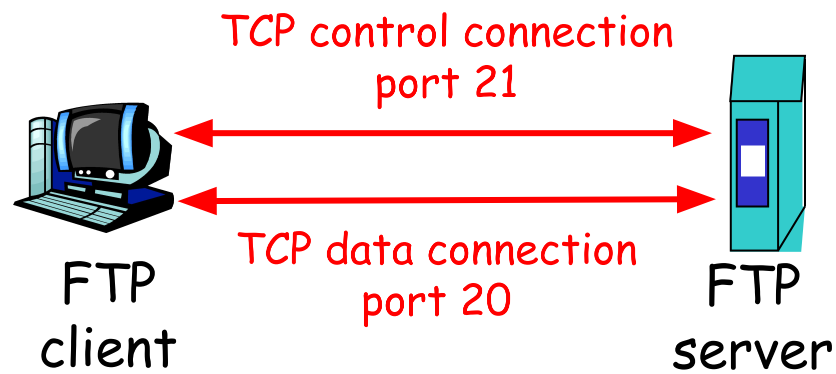


- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp server: port 21
- FTP use two parallel TCP connections: Control and data connection


User
User interface
Control process
Data transfer process
Client
Control connection
TCP/IP
Data connection
Control process
Data transfer process
Server

FTP: separate control, data connections

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



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

- In active mode, the client establishes the command channel (from client port X to server port 21) but the server establishes the data channel (from server port 20 to client port Y, where Y has been supplied by the client).
- In passive mode, the client establishes both channels. In that case, the server tells the client which port should be used for the data channel.

Examples:

- FileZilla
- Cyberduck
- WinSCP
- Transmit

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**

```
$ ftp voyager.deanza.tbda.edu
Connected to voyager.deanza.tbda.edu.
220 (vsFTPd 1.2.1)
530 Please login with USER and PASS.
Name (voyager.deanza.tbda.edu:forouzan): forouzan
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls reports
227 Entering Passive Mode (153,18,17,11,238,169)
150 Here comes the directory listing.



226 Directory send OK.
ftp>quit
221 Goodbye.
```

1. FTP server listens for connection on port number
   a) 20
   b) 21
   c) 22
   d) 23
2. In FTP protocol, client contacts server using ____ as the transport protocol.
   a) transmission control protocol
   b) user datagram protocol
   c) datagram congestion control protocol
   d) stream control transmission protocol
3. In which mode FTP, the client initiates both the control and data connections.
   a) active mode
   b) passive mode
   c) both (a) and (b)
   d) none of the mentioned

Electronic Mail
SMTP, POP3, IMAP

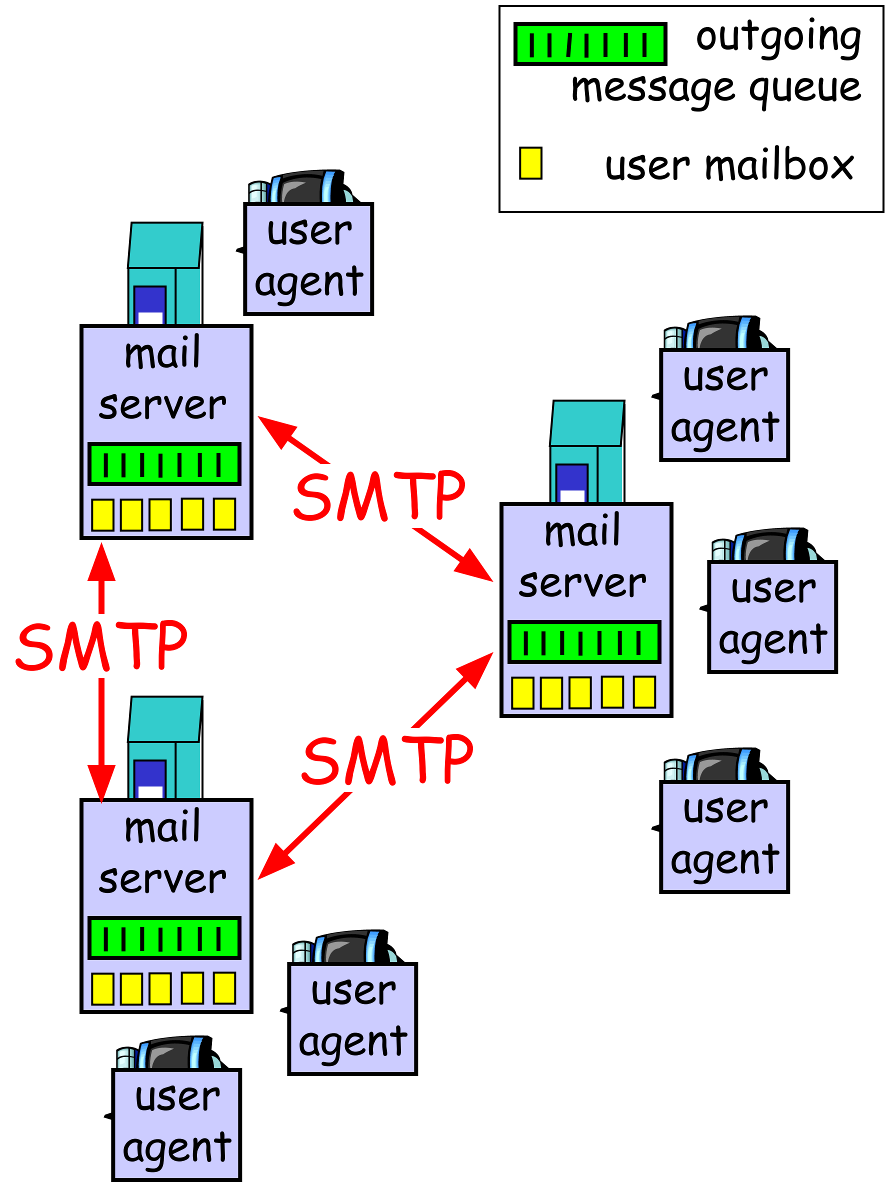
Electronic Mail

Three major components:

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

User Agent

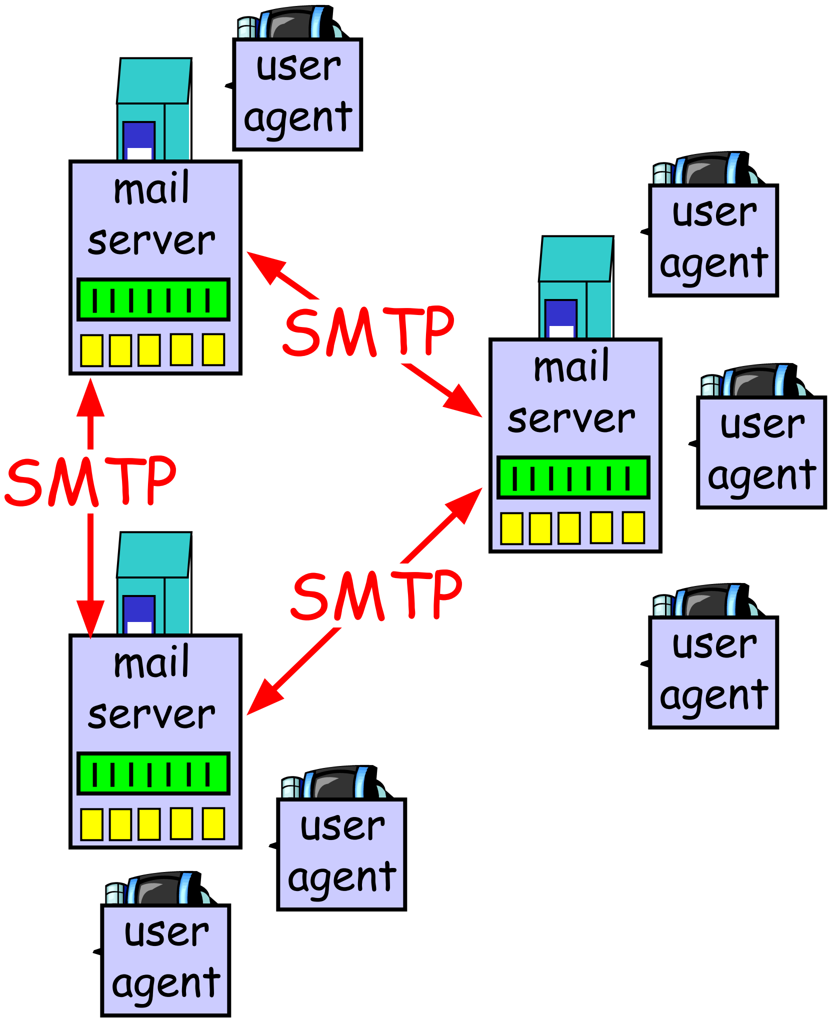
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Mozilla Thunderbird
- outgoing, incoming messages stored on server



Electronic Mail: mail servers

Mail Servers

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

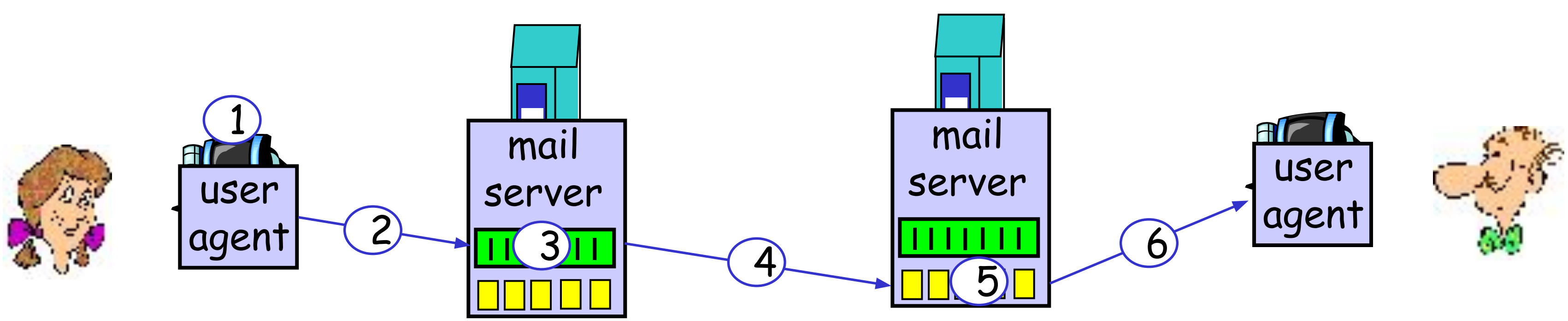


Electronic Mail: SMTP

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

Scenario: Alice sends message to Bob

1) Alice uses UA to compose message and "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: 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
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

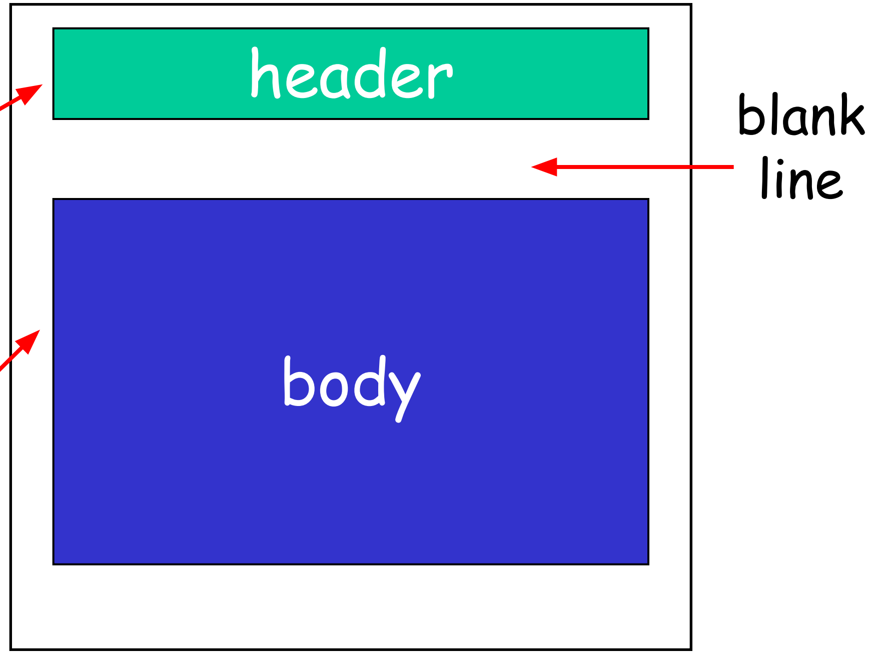
Mail message format

SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

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

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



Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type

MIME version

method used to encode data

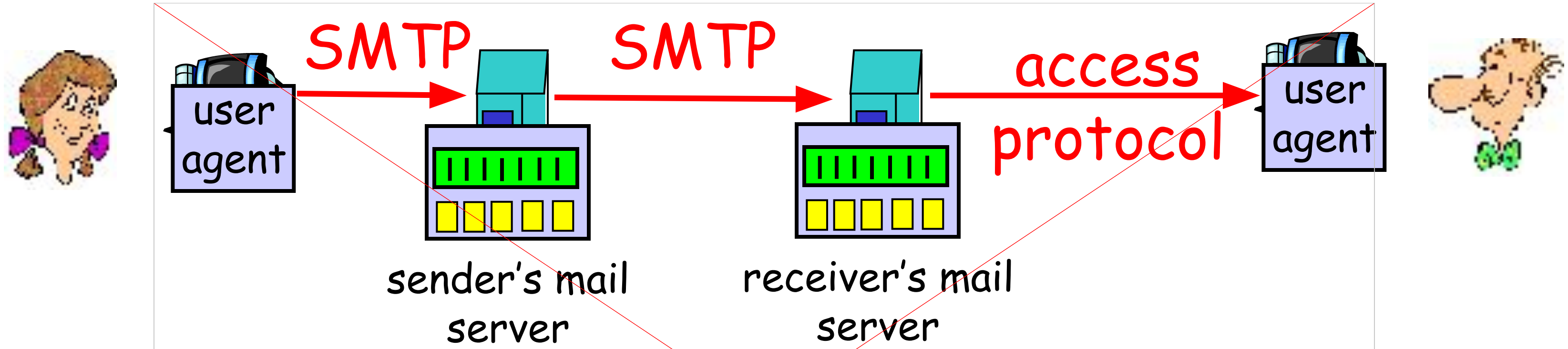
multimedia data type, subtype, parameter declaration

encoded data

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
```

Mail access protocols



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

POP3 protocol[110 port]

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

Two Modes of POP3

1. "download and delete" mode.
2. "Download-and-keep": copies of messages on different clients

POP3 is stateless across sessions

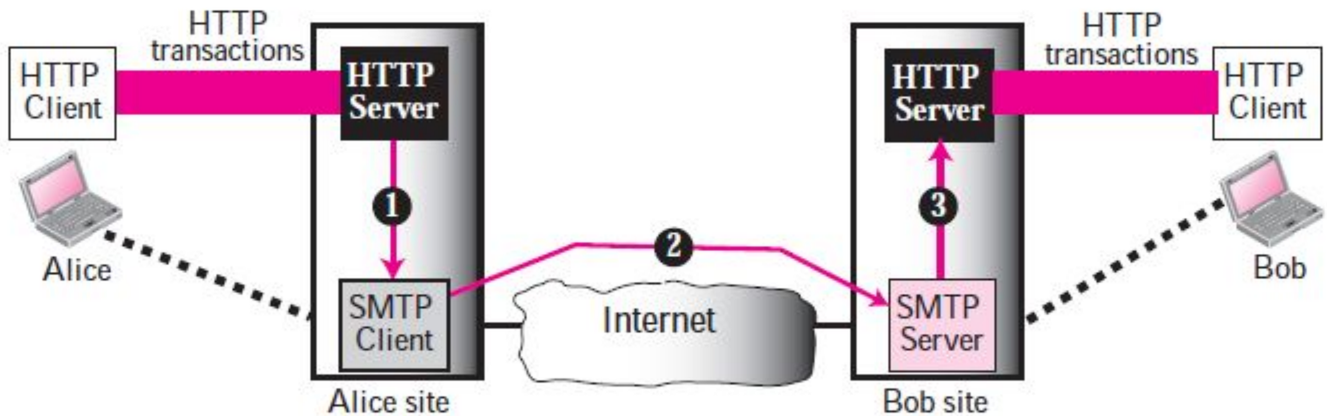
IMAP

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

Example

- GUI User agents: Mozilla Thunderbird, Microsoft outlook,Apple Mail
- Web Based email: Hotmail,yahoo,google

WEB-BASED MAIL



- User agent: Web Browser
- User communicates with its remote mailbox via HTTP
- EMAIL message is sent from bob mail server to bob's browser using HTTP protocol rather than POP or IMAP protocol when bob wants to access message in his mailbox.
- Sending mail from browser to mail server over HTTP not SMTP
- Mail server to mail server still uses send/recieves of mails using SMTP

DNS

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 addresses and name ?

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 services

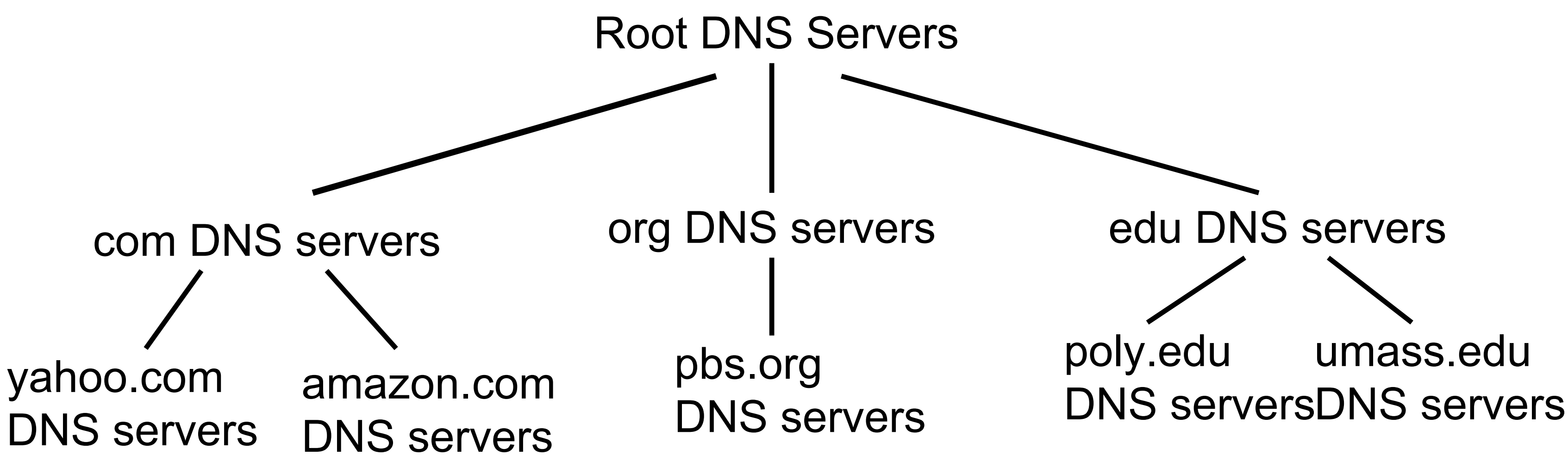
- hostname to IP address translation
- host aliasing
  - Canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

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

doesn't *scale!*

Distributed, Hierarchical Database

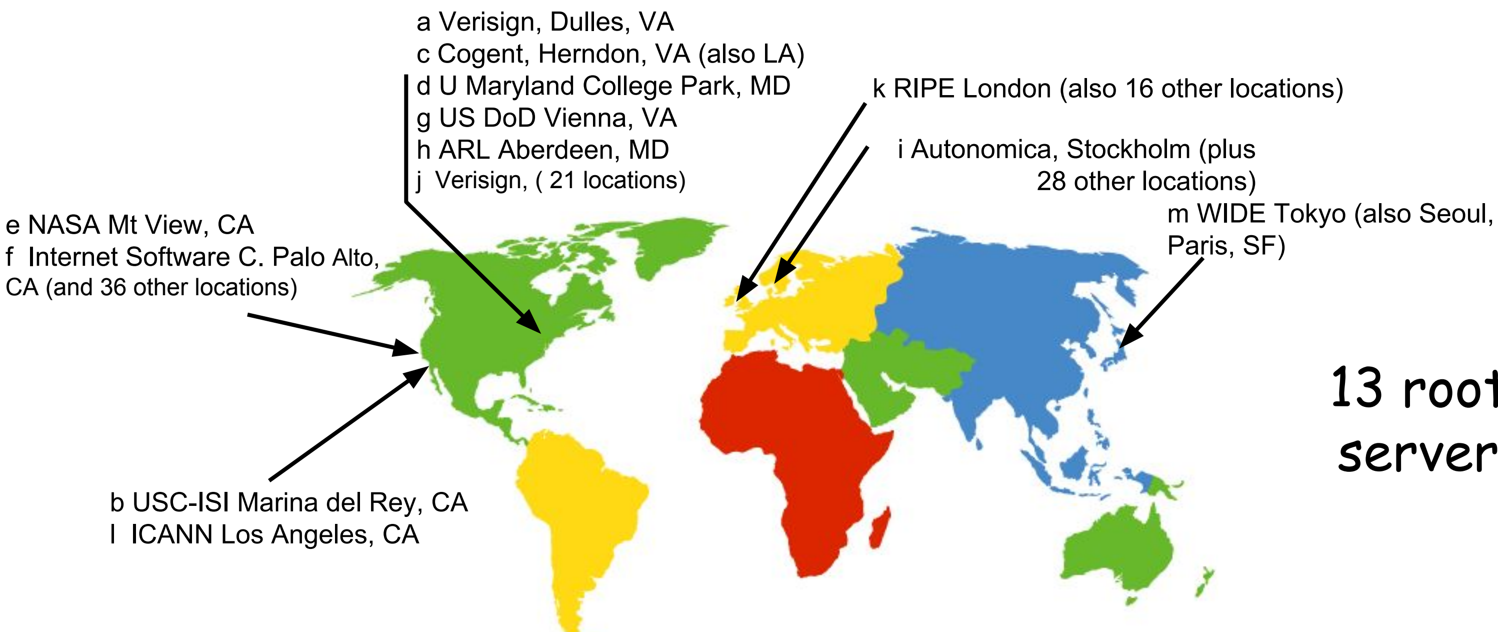


Client wants IP for www.amazon.com; 1st approx:

- 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

1. 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

2. Top-level domain (TLD) servers:

- responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
- Network Solutions maintains servers for com TLD
- Generic domains, country domains and inverse domains

3. 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

4. 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

The domain name is a component of a Uniform Resource Locator (URL) used to access web sites

- **URL:** `http://www.example.net/index.html`
- **Top-level domain name:** `net`
- **Second-level domain name:** `example.net`
- **Host name:** `www.example.net`

Domain Name Syntax

- **URL**
- **Host name**
- **domain name:** Any name registered in the DNS is a domain name.
- **A fully qualified domain name (FQDN)** is a domain name that is completely specified in the hierarchy of the DNS, having no omitted parts.
- ***partially-qualified domain name (PQDN)*** only specifies a portion of a domain name. It is a *relative* name that has meaning only within a particular context; the partial name must be interpreted within that context to fully identify the node.
- **Top level Domain**
- **Second level Domain**
- **Labels:** A domain name consists of one or more parts, technically called *labels*, that are conventionally concatenated, and delimited by dots
- Each label may contain up to 63 characters.
- Processing is done for sequence of domain labels from right to left, going top to bottom within the tree.

- **Tree:** The DNS hierarchy can be visualized as a tree where each node in the tree corresponds to a domain and the tree corresponds to the hosts being named.
- **Zones:** partition of the hierarchy into sub trees called zones.
- **Zone files:** server makes a database called zone file and keeps all the information for every node under that domain.
- Primary server
- Secondary server
- Zone transfer

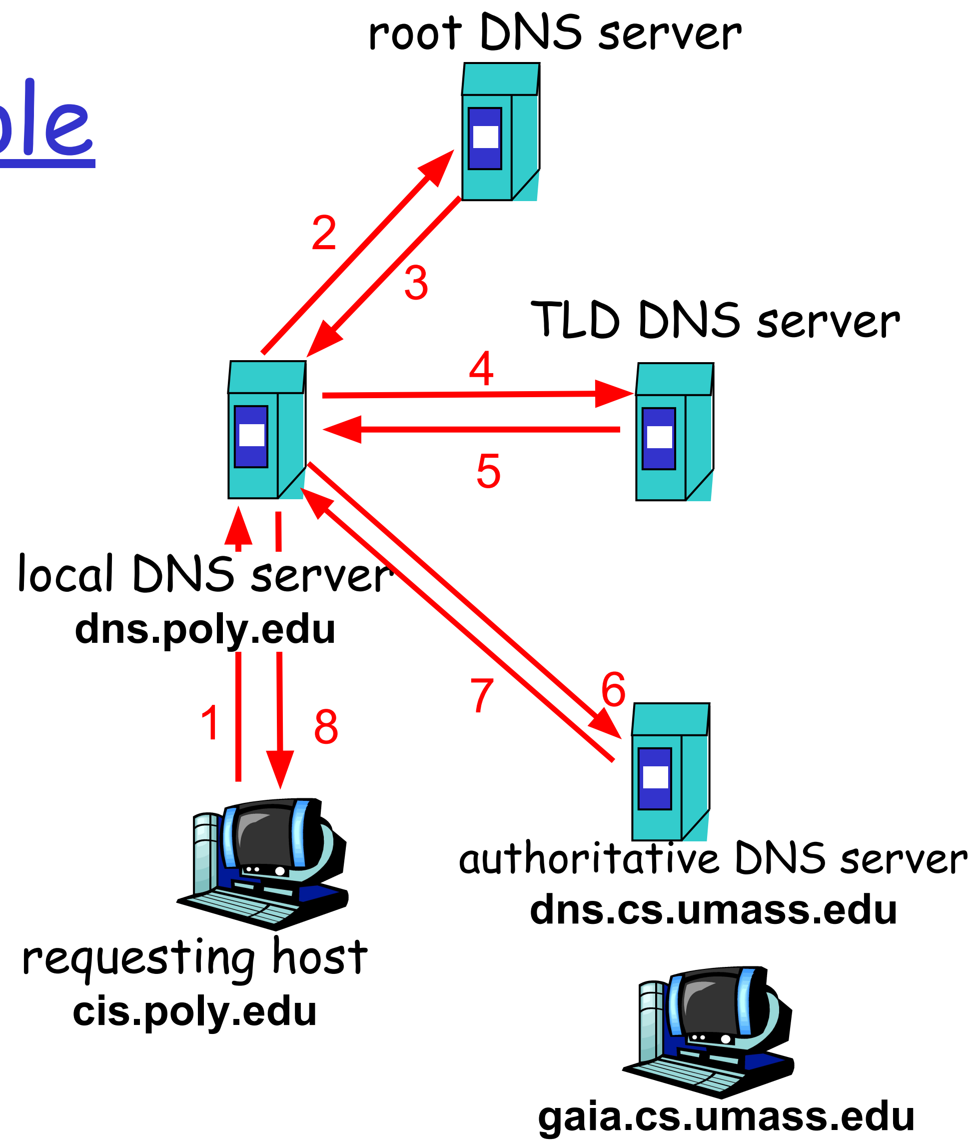
- **DNS uses TCP for Zone Transfer over Port: 53**
- **DNS uses UDP for DNS queries over Port:53**
- Mapping names to addresses
- Mapping adresses to names

DNS name resolution example

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

iterated query:

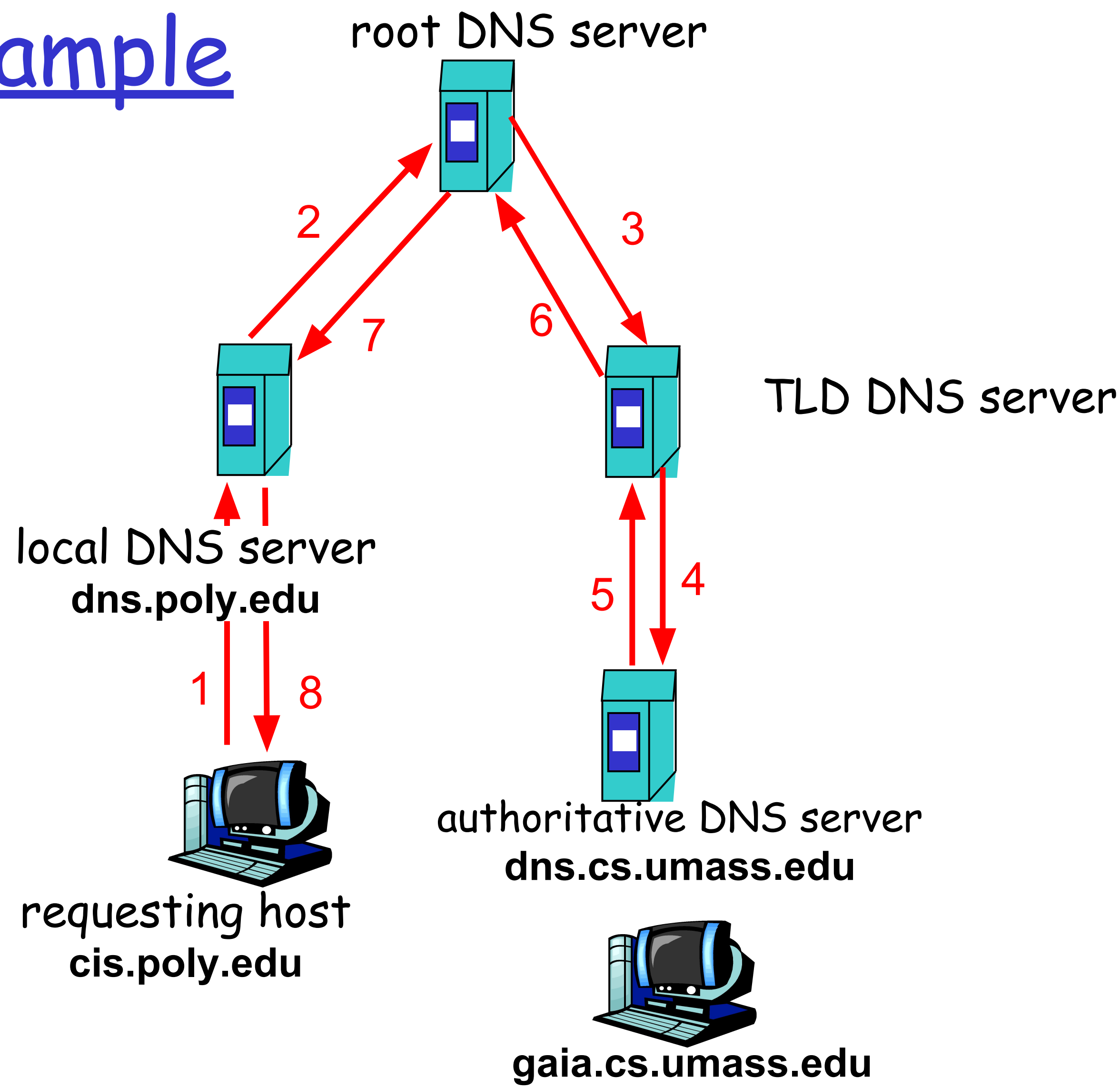
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

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



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)`

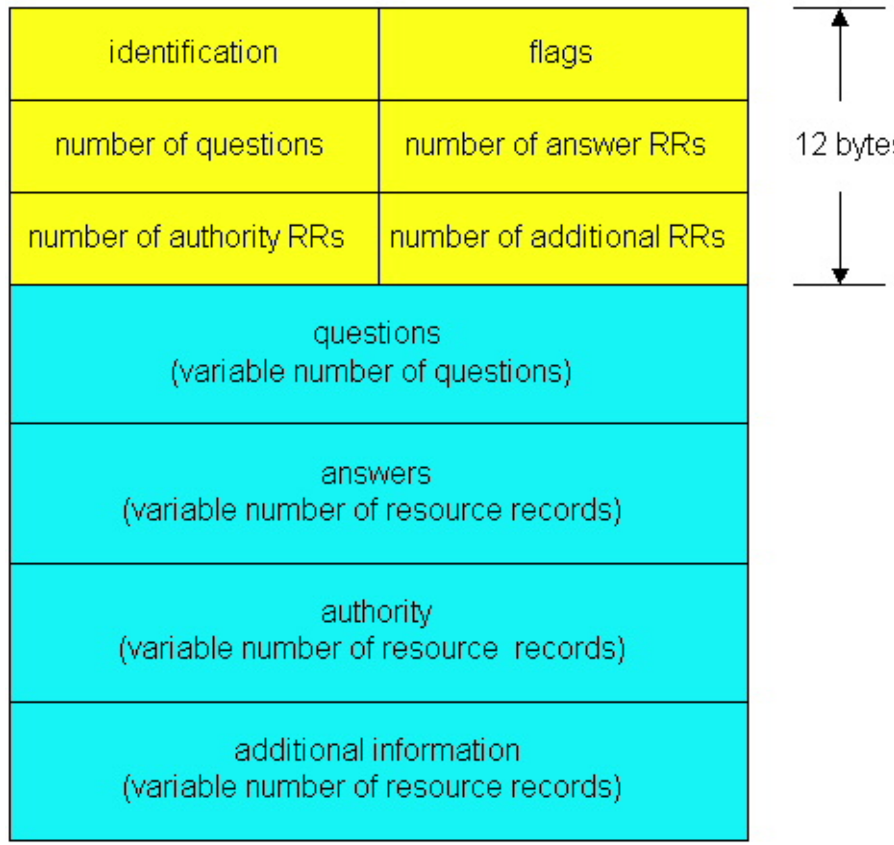
- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is hostname of authoritative name server for this domain
- Type=CNAME
  - **name** is alias name for some "canonical" (the real) name
    `www.ibm.com` is really `servereast.backup2.ibm.com`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

message header

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



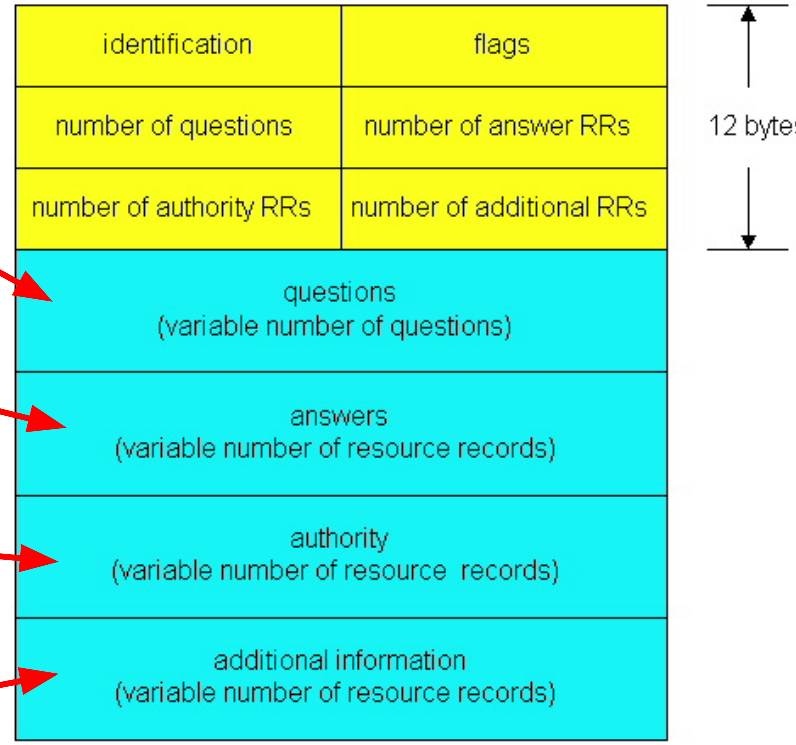
DNS protocol, messages

Name, type fields
for a query

RRs in response
to query

records for
authoritative servers

additional "helpful"
info that may be used



```
N:\>nslookup -type=A google.com
Server:  UnKnown
Address:  172.16.68.10

Non-authoritative answer:
Name:    google.com
Address:  142.250.194.78


N:\>nslookup -type=NS google.com
Server:  UnKnown
Address:  172.16.68.10

Non-authoritative answer:
google.com      nameserver = ns3.google.com
google.com      nameserver = ns4.google.com
google.com      nameserver = ns2.google.com
google.com      nameserver = ns1.google.com

ns2.google.com  internet address = 216.239.34.10
ns2.google.com  AAAA IPv6 address = 2001:4860:4802:34::a
ns1.google.com  internet address = 216.239.32.10
ns1.google.com  AAAA IPv6 address = 2001:4860:4802:32::a
```

```
N:\>nslookup -type=MX google.com
Server:  UnKnown
Address:  172.16.68.10

Non-authoritative answer:
google.com      MX preference = 50, mail exchanger = alt4.aspmx.l.google.com
google.com      MX preference = 10, mail exchanger = aspmx.l.google.com
google.com      MX preference = 40, mail exchanger = alt3.aspmx.l.google.com
google.com      MX preference = 20, mail exchanger = alt1.aspmx.l.google.com
google.com      MX preference = 30, mail exchanger = alt2.aspmx.l.google.com

alt4.aspmx.l.google.com internet address = 142.250.115.26
alt4.aspmx.l.google.com AAAA IPv6 address = 2607:f8b0:4023:1004::1b
aspmx.l.google.com      internet address = 74.125.200.27
aspmx.l.google.com      AAAA IPv6 address = 2404:6800:4003:c04::1b
alt3.aspmx.l.google.com internet address = 142.250.141.27
alt3.aspmx.l.google.com AAAA IPv6 address = 2607:f8b0:4023:c0b::1b
alt1.aspmx.l.google.com internet address = 173.194.202.27
alt1.aspmx.l.google.com AAAA IPv6 address = 2607:f8b0:400e:c00::1b
alt2.aspmx.l.google.com internet address = 142.250.142.27
alt2.aspmx.l.google.com AAAA IPv6 address = 2607:f8b0:4023:1c01::1a
```

```
N:\>nslookup -type=CNAME google.com
Server:  UnKnown
Address:  172.16.68.10

google.com
        primary name server = ns1.google.com
        responsible mail addr = dns-admin.google.com
        serial  = 432150313
        refresh = 900 (15 mins)
        retry   = 900 (15 mins)
        expire  = 1800 (30 mins)
        default TTL = 60 (1 min)
```

Inserting records into DNS

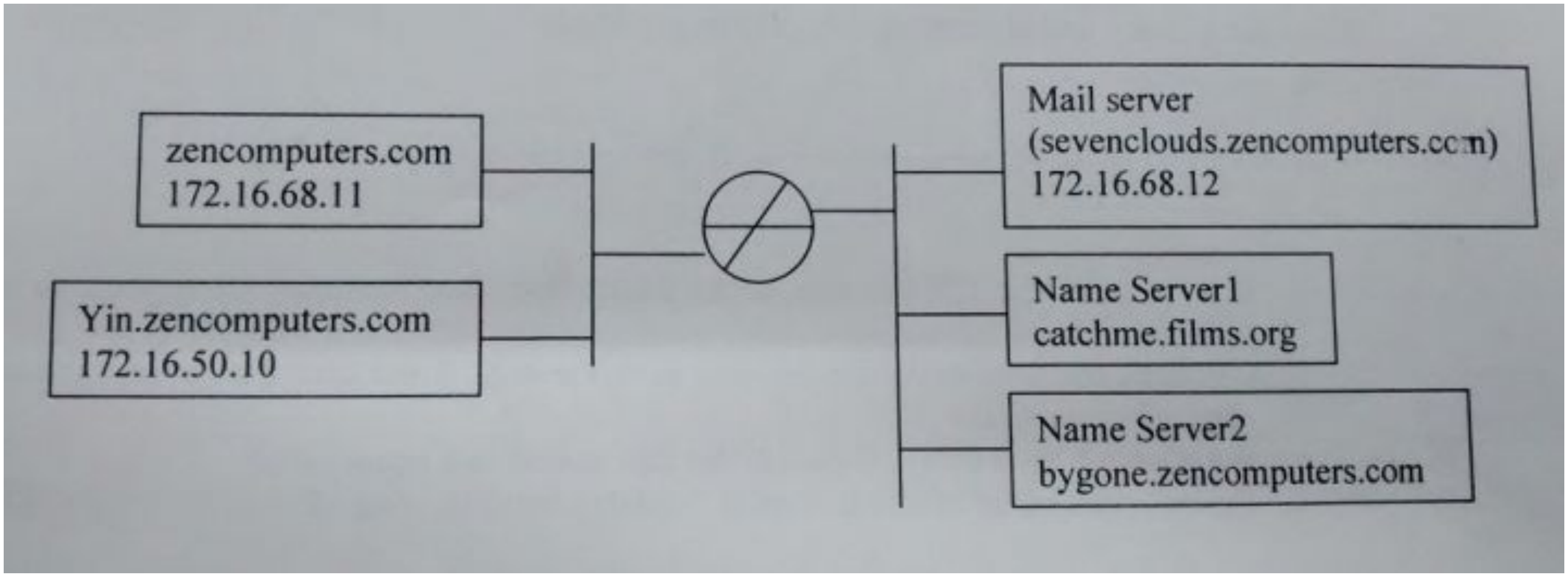
- example: new startup "Network Utopia"
- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts two RRs into com TLD server:

```
(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?

Questions..

Question:



zencomputers.com
172.16.68.11
Yin.zencomputers.com
172.16.50.10
Mail server
(sevenclouds.zencomputers.com)
172.16.68.12
Name Server1
catchme.films.org
Name Server2
bygone.zencomputers.com

Question: What would be the type for the resource record (RR) that contains the hostname of the mail server?

Question: Suppose within your Web browser you click on a link to obtain a Web page. The Web page you are getting is at server S0. Its size is very small, but it contains four objects Stored at servers S1, S2, S3 and S4. Each object has size of 1000bits. Assume that each TCP connection established from your host has throughput of 10000bits/sec. Let RTTi denote the RTT between the local host and the server Si. How much time elapses from when the client clicks on the link until the client receives all the objects?

Note: In this question you should assume (i) that the IP addresses for all URLs are cached in your local host so no DNS lookup is required and (ii) your browser is NOT using persistent connections and also is Not using parallel connections.

Question : Is the user-agent (e.g., mail-reader) of the receiver Of an email message uses SMTP to download the message from The receiver's mailbox. If yes/no then how it is done.

Question.: In SMTP, show the connection establishment phase from aaa@xxx.com to bbb@yyy.com.

Question: In SMTP, show the message transfer phase from aaa@xxx.com to bbb@yyy.com.

The message is "Good morning my friend."

Question: In SMTP, show the connection termination phase from aaa@xxx.com to bbb@yyy.com.

Question: Suppose within your web browser you click on a link to obtain a Web page. The IP address for the associated URL is not cached in your local host, so a DNS lookup is necessary to obtain the IP address. Suppose that *n DNS servers are visited before your host received the IP* address from DNS; the successive visits incur an RTT of *RTT1, RTT2, …, RTTn. Further Suppose* that the Web page associated with the link contains exactly one object, consisting of a small amount of HTML text. Let *RTT0 denote the RTT between the local host and the Server* containing the object. Assuming zero transmission time of the object, how much time elapses between when the client clicks on the link until it receives the object?