Chapter 2
Application Layer

Teacher： Xu Yang

Chapter 2: Outline

2.1 principles of network applications

2.2 Web and HTTP

2.4 electronic mail

- SMTP, POP3

2.5 DNS

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
- …
- …

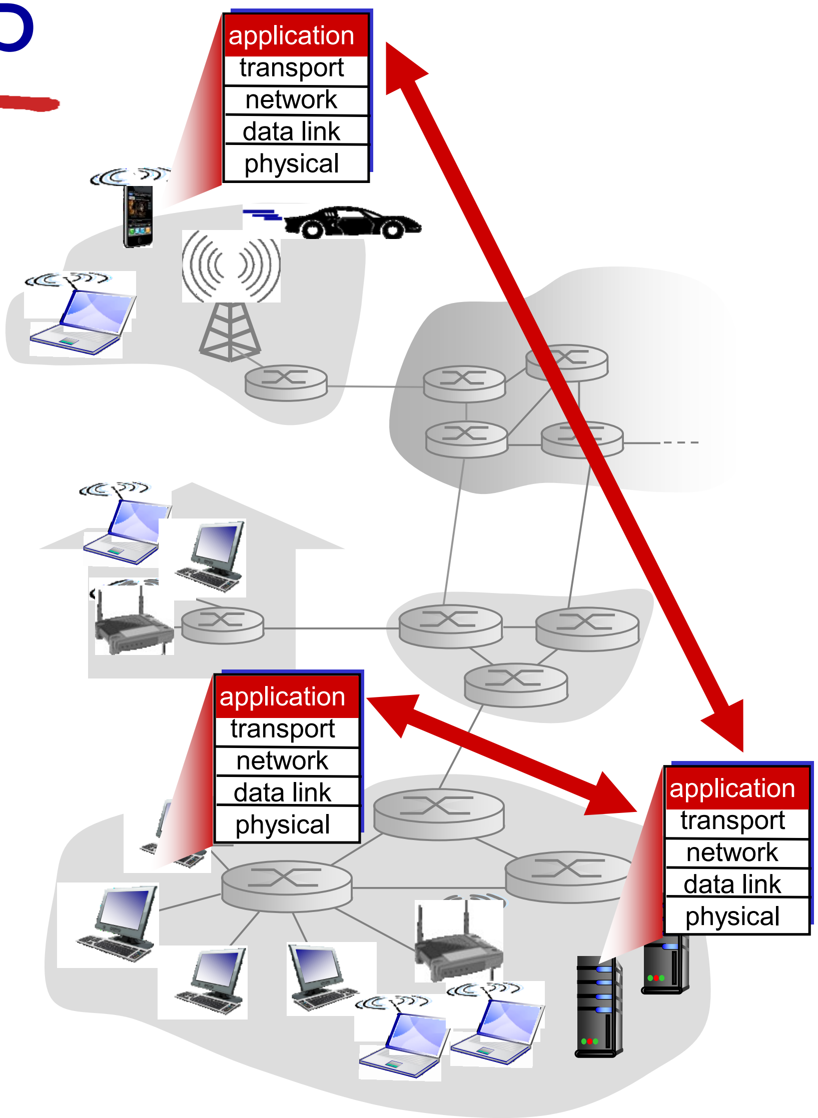
Creating a network app

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



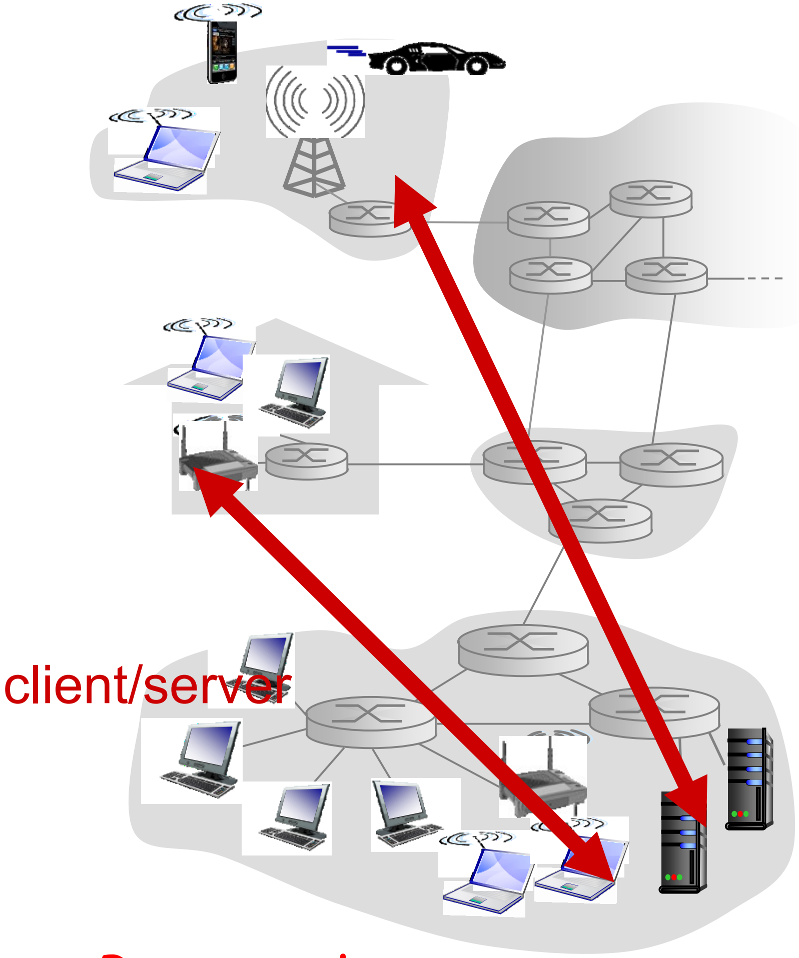
Application architectures

possible structure of applications:

is designed by the application developer and dictates how the application is structured over the various end systems.

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

Client-server architecture



server:

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

clients:

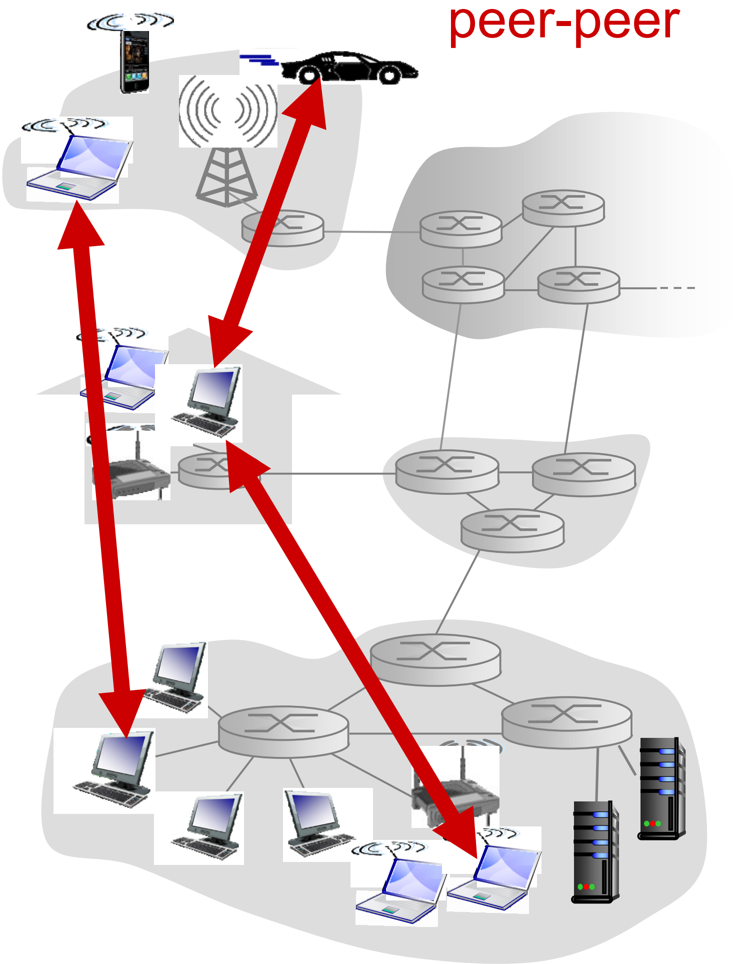
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Pros and cons:

- Infrastructure intensive
- Costly to provide
- Easy to manage and secure

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



Pros and cons:

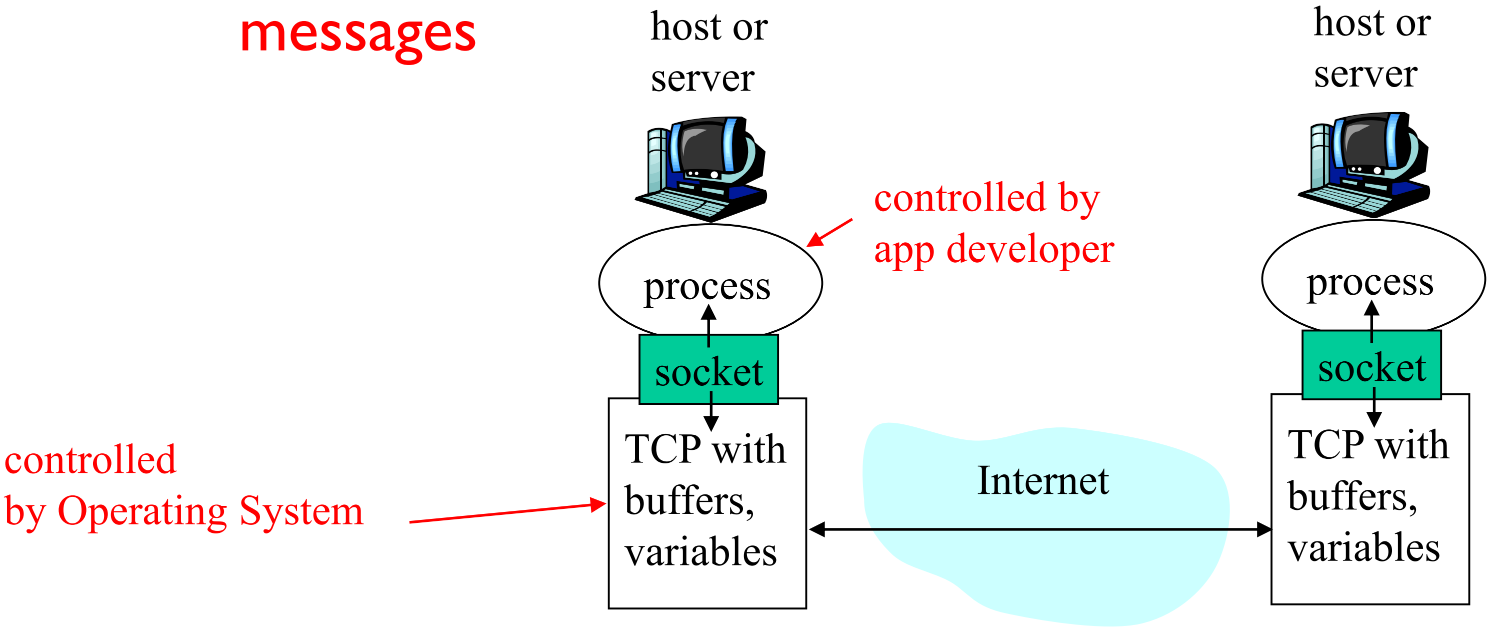
- Highly scalable
- Difficult to manage
- Challenge to secure such as privacy risk, online attacks, etc.

How are the Messages/data transmitted between end systems? ----Processes communicating

Host: end system running network application programs

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


host or server
host or server
controlled by app developer
process
process
socket
socket
controlled by Operating System
TCP with buffers, variables
Internet
TCP with buffers, variables

How are the Messages/data transmitted between end systems? ----Processes communicating

Client process: process that initiates communication

Server process: process that waits to be contacted

Applications with P2P architectures have client processes & server processes

- the peer that is downloading the file is labeled as the client,
- and the peer that is uploading the file is labeled as the server.

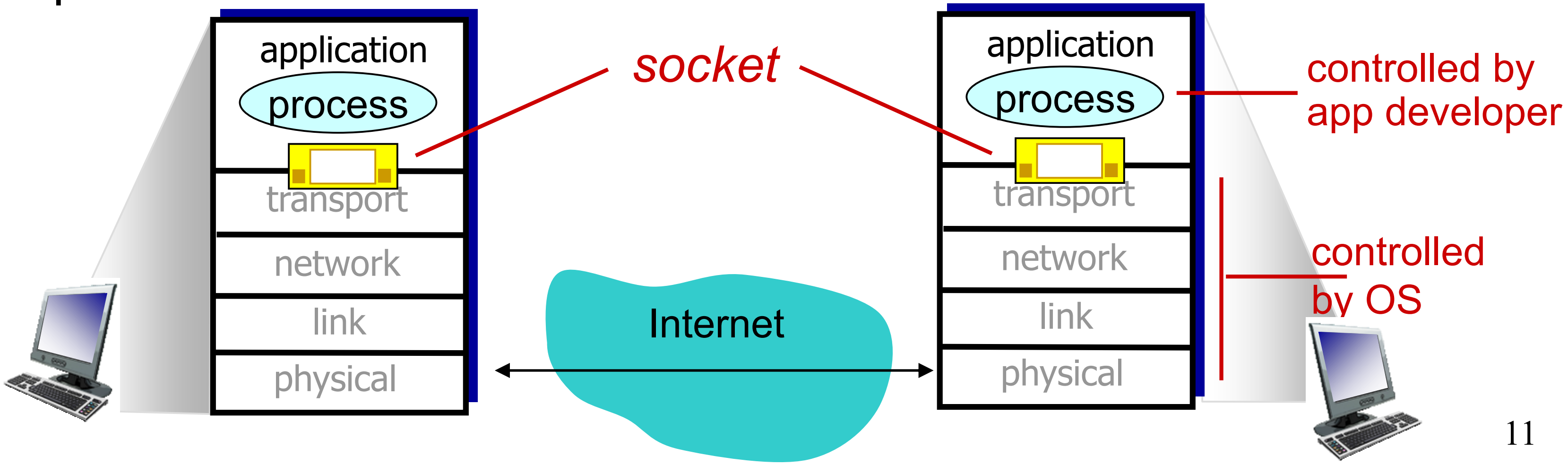
Sockets

Socket: it is a software interface through with a process sends data into, and receives data from the network.

- Socket is a programming interface between the application and the network. Therefore, socket is an Application Programming Interface (API).
- Specifically, socket is the interface between application layer and transport layer.
- It has two sides:
  - on the application-layer side of the socket, the application developer has control of everything;,
  - on the transport-layer side, the only control that the application developer can control is (1) the choice of transport protocol and (2)fix a few transport-layer parameters such as maximum buffer size and maximum segment sizes.

An Analogous Example

- Host analogous to house
  - IP address versus mail address
- process analogous room
- Socket analogous to door
  - Process sends/receives messages to/from its socket
  - sending process pushes message out door
  - sending process relies on transport layer protocol to transmit data to the receiving host and go to the receiving process through its socket.
  - Socket/API can (1) choice of transport protocol; (2) ability to fix a few parameters



Addressing processes

- to receive messages, process must have *identifier*
  - IP address
    - The name or address of the destination host.
    - In Internet, a host is identified by a unique 32-bit IP address (IPV4).
- 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

What transport service does an app need?

Reliable data transfer

- 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 : e-mail, file transfer…

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, 100s of msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100s of msec
text messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport :* to deliver all data sent without error and in the proper order.
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security

UDP service:

- *connectionless,* so there is no handshaking before the two processes start to communicate.
- *unreliable data transfer*
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection 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

Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

Web and HTTP

First, a review…

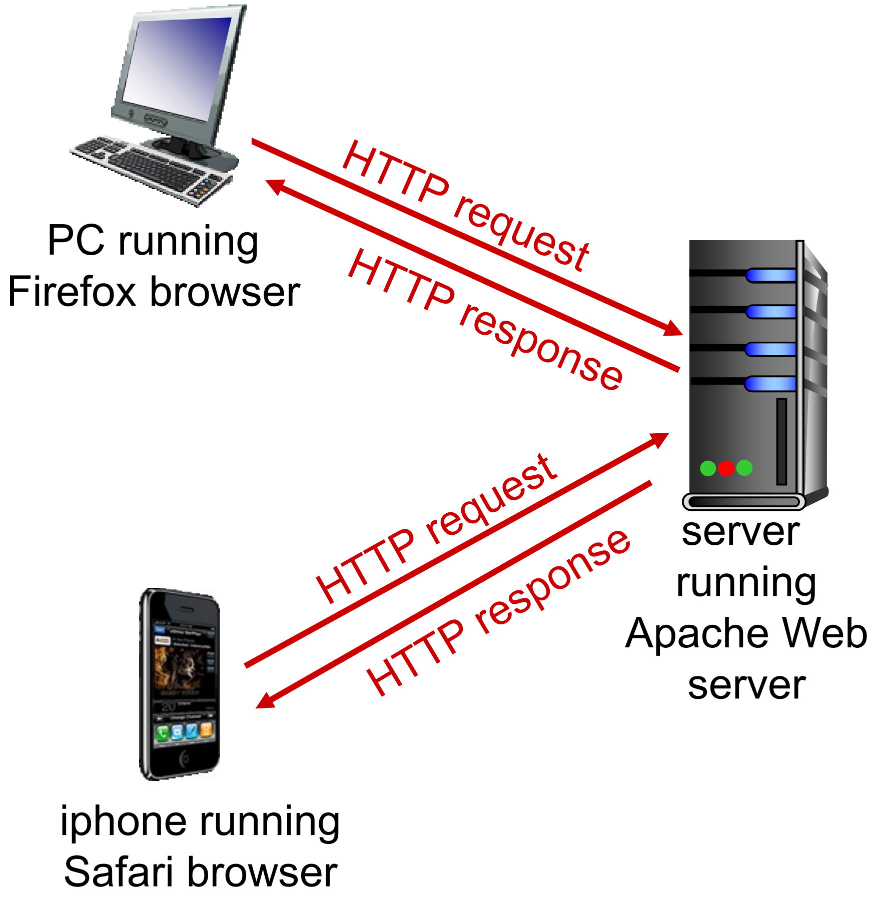
- *web page* consists of *objects*
- An object is simply a file, can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,

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

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


PC running Firefox browser
HTTP request
HTTP response
server running Apache Web server
HTTP request
HTTP response
iphone running Safari browser

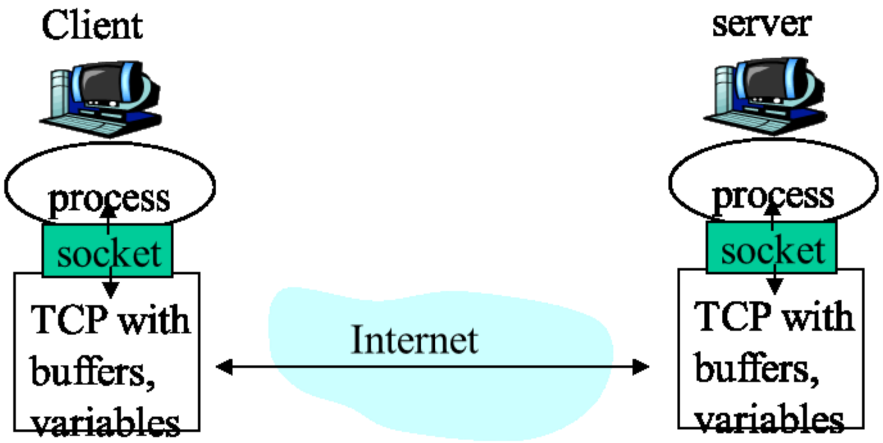
HTTP overview (continued)

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



HTTP connections

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

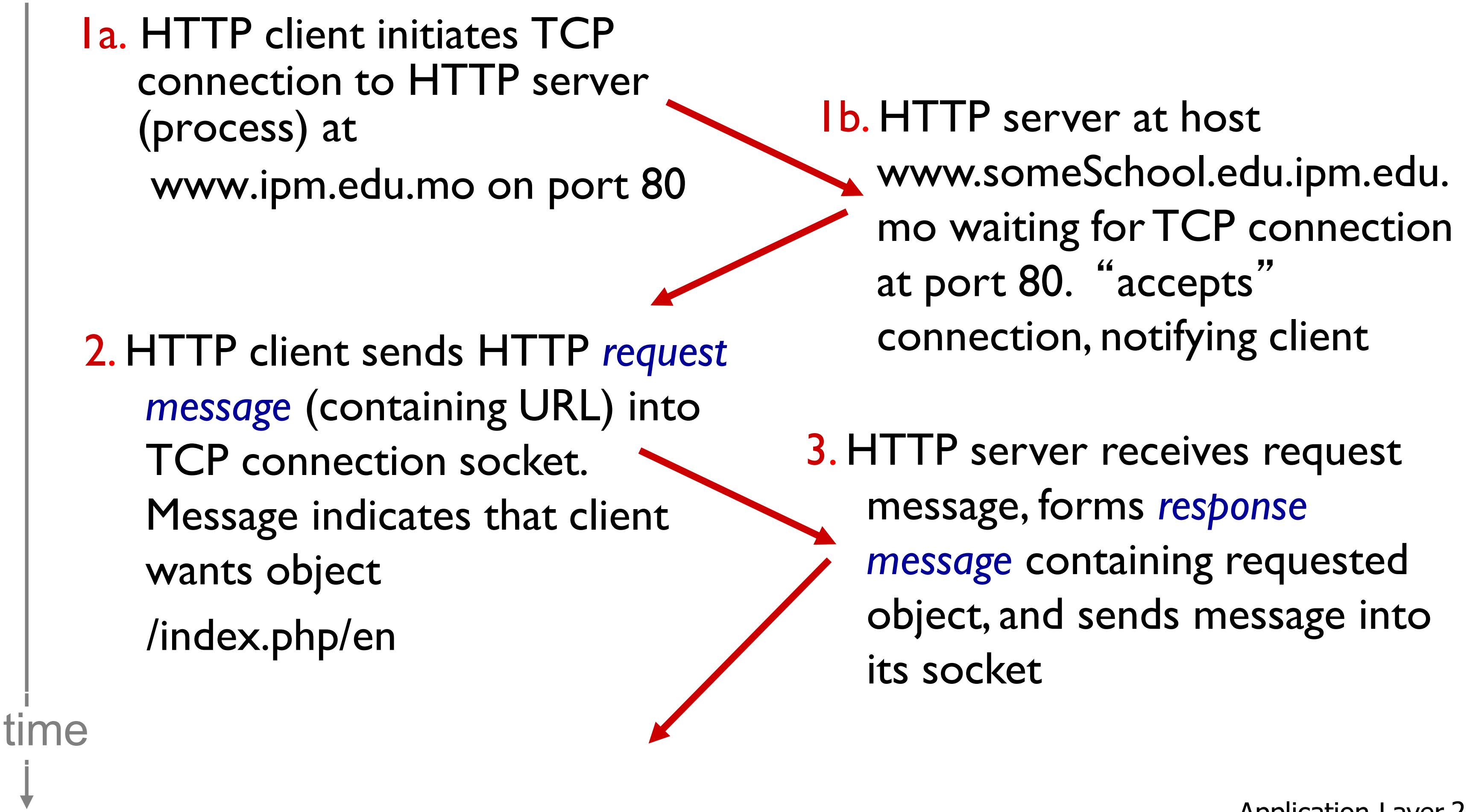
persistent HTTP (default)

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

HTTP uses persistent connections in its default mode, HTTP clients and servers can be configured to use non-persistent connections instead.

Non-persistent HTTP

suppose user enters URL:
http://www.ipm.edu.mo/index.php/en/



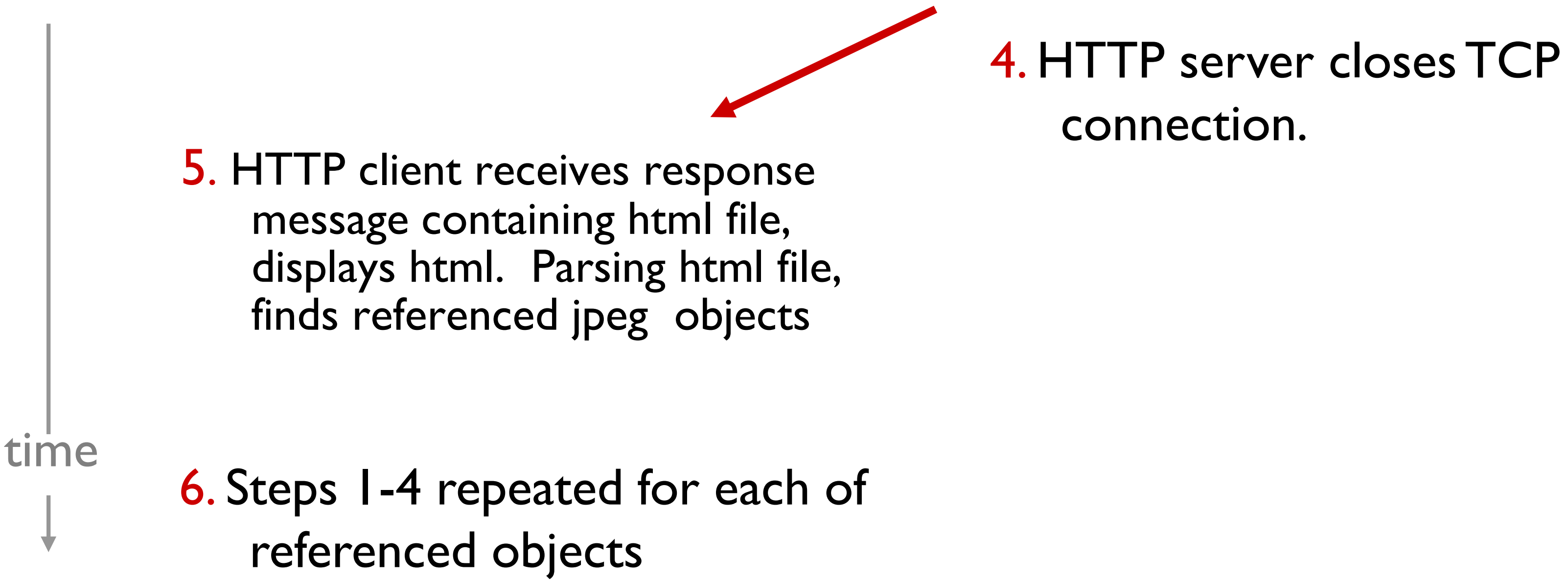
1a. HTTP client initiates TCP connection to HTTP server (process) at
www.ipm.edu.mo on port 80

1b. HTTP server at host www.someSchool.edu.ipm.edu.mo 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
/index.php/en

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

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 referenced jpeg objects

6. Steps 1-4 repeated for each of referenced objects

Note that each TCP connection transports exactly one request message and one response message.

TCP doesn’t actually terminate the connection until it knows for sure that the client has received the response message intact.

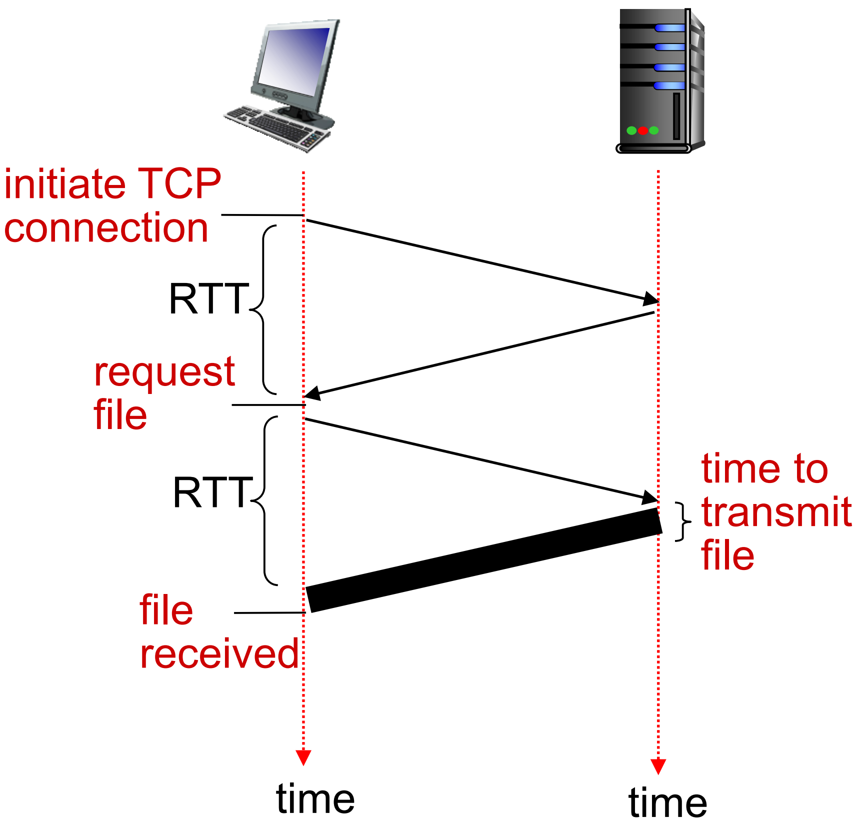
Non-persistent HTTP: response time

RTT (definition) 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



http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/http/http.html

Persistent HTTP

non-persistent HTTP issues:

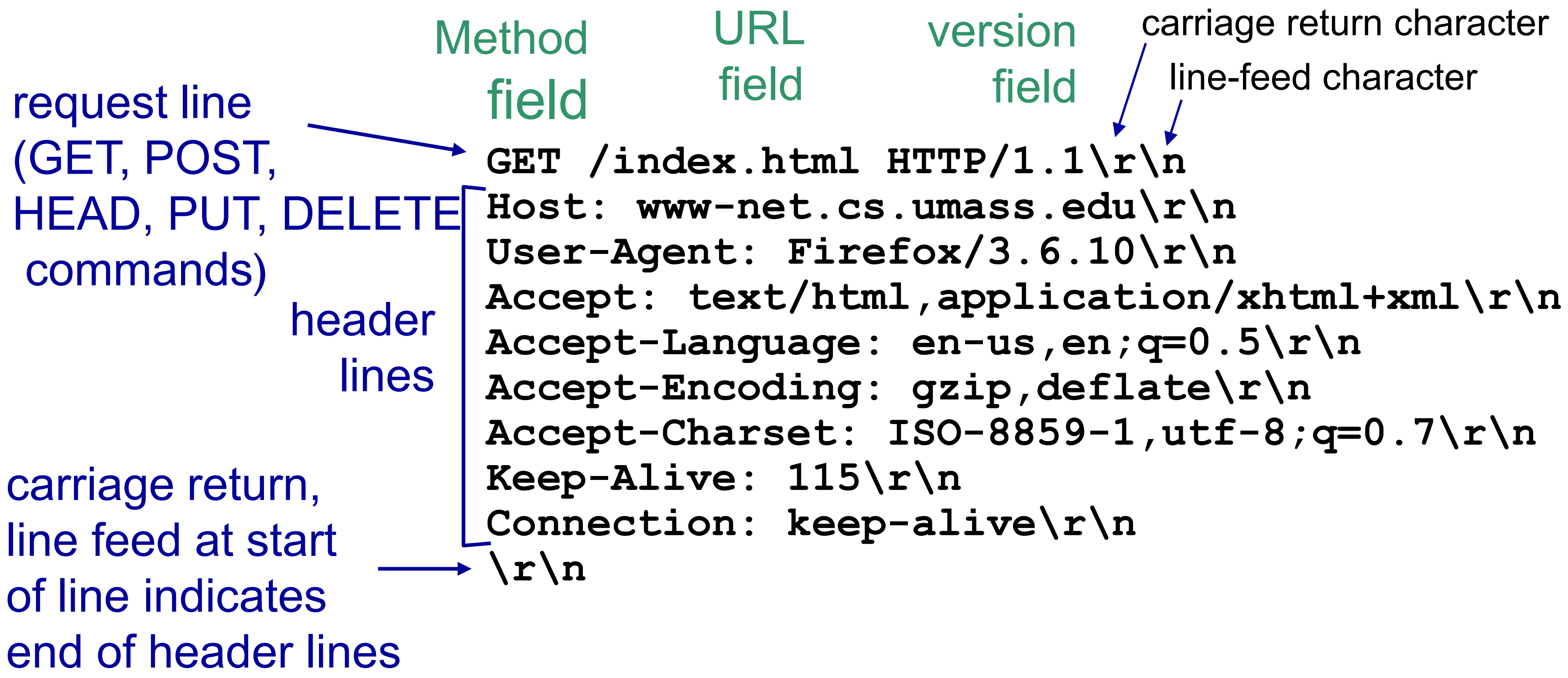
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects
- Most bowsers open 5 to 10 parallel TCP connections, and each of these connections handles one request-response transaction.

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
- the HTTP server closes a connection when it isn't used for a certain time (a configurable timeout interval).

HTTP request message

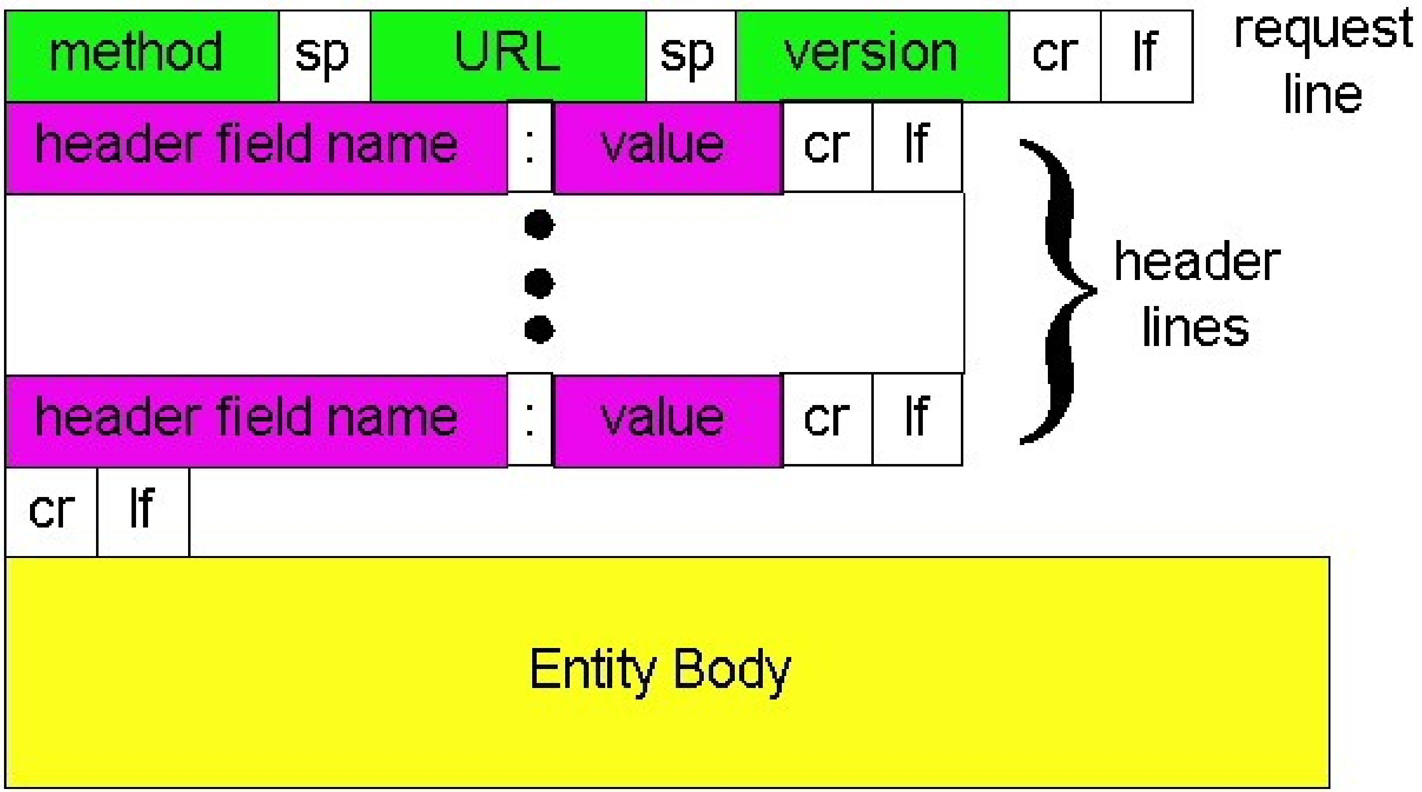
- Two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)



Method field
URL field
HTTP version field
carriage return character
line-feed character
request line (GET, POST, HEAD, PUT, DELETE 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:

- An HTTP client often uses the POST method when the user fills out a form
- input is uploaded to server in entity body

GET method:

- The GET method is used when the browser requests an object, with the requested object identified in the URL field.
- input is uploaded in URL field of request line:

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

Method types

HEAD method

- Is similar to the GET method
- asks server to leave requested object out of response

PUT method

- It allows a user to upload an object to a specific path (directory) on a specific Web server.

DELETE method

- deletes file specified in the URL field

HTTP response message

The Last-Modified: header line indicates the time and date when the object was created or last modified.

The Content-Length: header line indicates the number of bytes in the object being sent.

status line (protocol status code status phrase)

header lines

```
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 ...
```

data, e.g., requested HTML file

The Date: header line indicates the time and date when the HTTP response was created and sent by the server.

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 to keep track of users

four components:

1) cookie header line in the HTTP *response* message

2) cookie header line in next HTTP *request* message

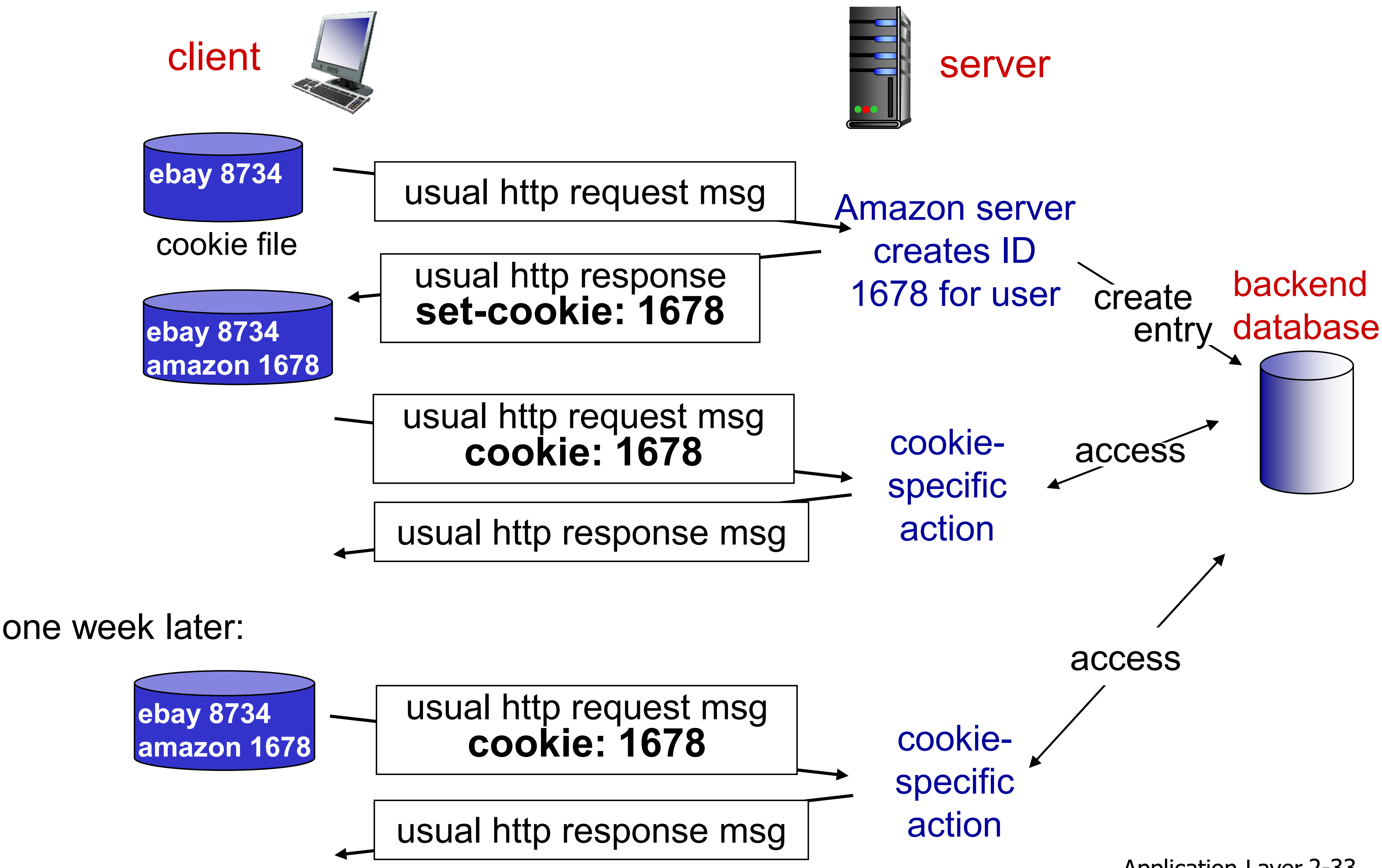
3) cookie file kept on user's host, managed by user's browser

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.)



client

server

ebay 8734

cookie file

usual http request msg

Amazon server creates ID 1678 for user

create entry

backend database

usual http response
set-cookie: 1678

ebay 8734
amazon 1678

usual http request msg
cookie: 1678

cookie-specific action

access

usual http response msg

one week later:

ebay 8734
amazon 1678

usual http request msg
cookie: 1678

cookie-specific action

access

usual http response msg

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":

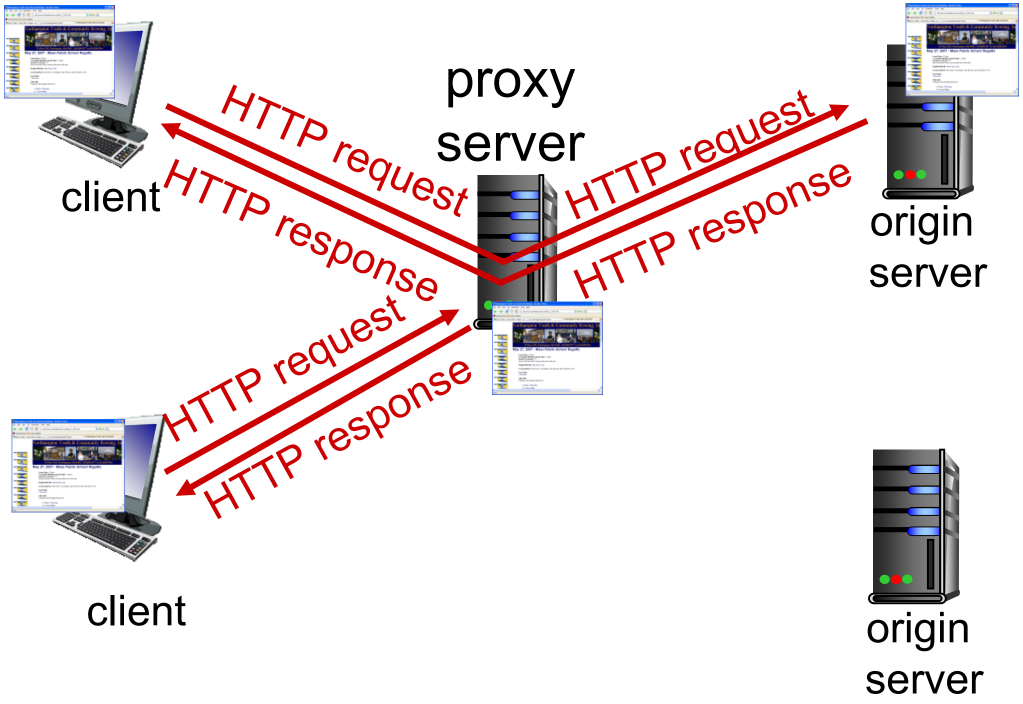
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

Web caches (proxy server)

Web cache: is also called as *a proxy server*, it is a network entity that satisfies HTTP requests on behalf of an origin Web 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:

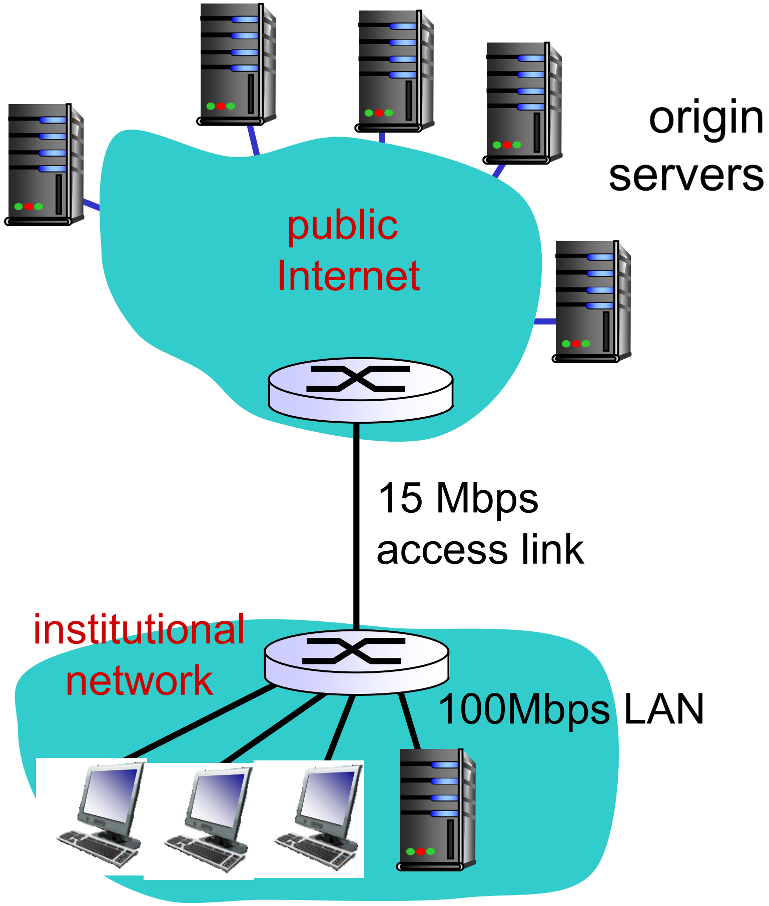
- avg object size: 1M bits
- avg request rate from browsers to origin servers:15/sec
- RTT from first router to any origin server: 2 sec
- access link rate: 15Mbps

consequences:

- LAN traffic intensity: 0.15
- access link traffic intensity = 1

problem!

- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + millseconds



the traffic intensity on the access link (from the Internet router to institution router) is

(15 requests/sec) (1 Mbits/request) /(15 Mbps) = 1

The traffic intensity on the LAN

(15 requests/sec) (1 Mbits/request) /(100 Mbps) = 0.15

Caching example: fatter access link

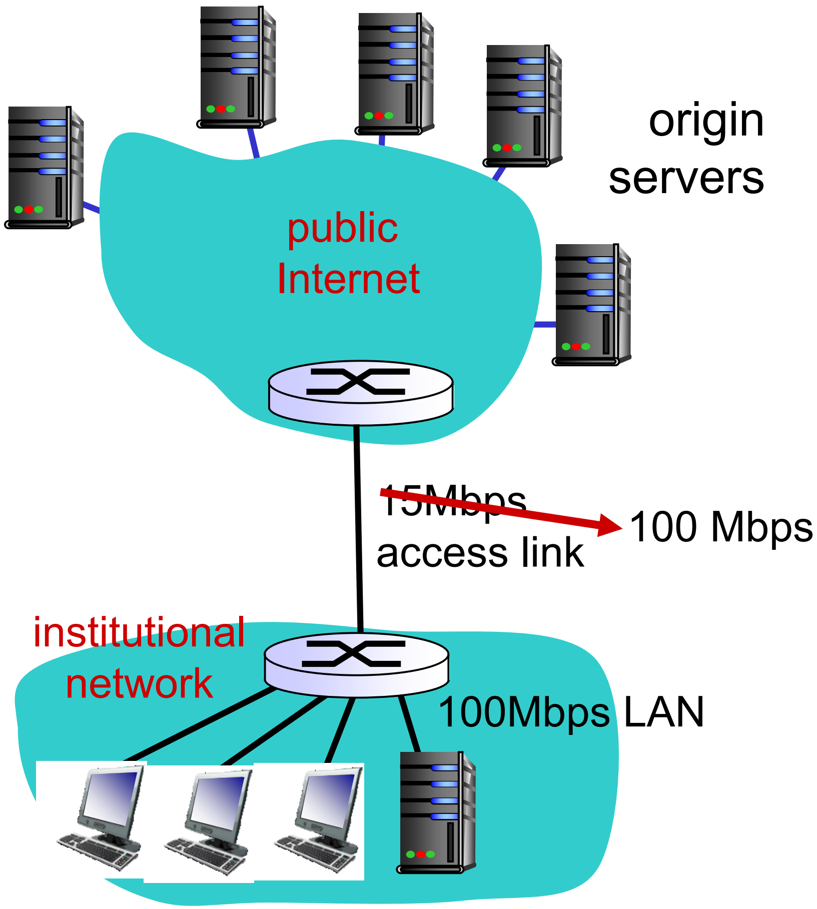
assumptions:

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

consequences:

- LAN traffic intensity: 0.15
- access link traffic intensity = 1 → 0.15
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + msecs

  minutes → msecs

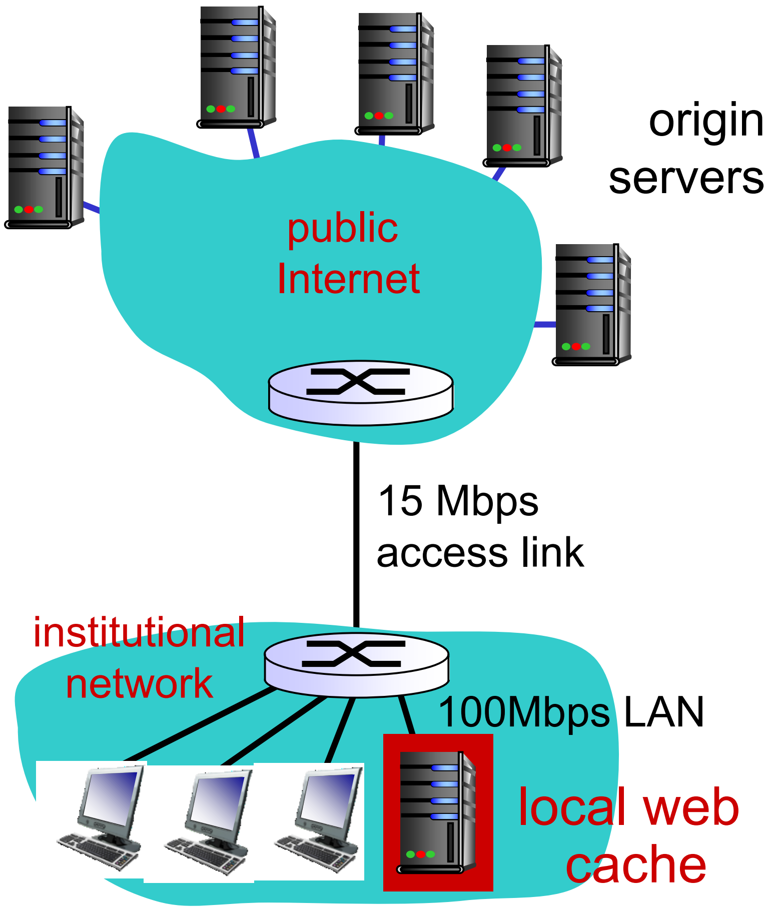


Cost: increased access link speed (not cheap!)

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
  - Traffic intensity = 9/15 = .6
- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.0) + 0.4 (~msecs)
  - = ~ 1.2 secs
  - less than with 100 Mbps link (and cheaper too!)



Summary

- Principle of network application
- Application architectures
  - client-server
  - P2P
  - hybrid
- application service requirements:
  - packet loss, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - Connectionless, unreliable: UDP
- Specific application and protocol
  - Web and HTTP
  - Cookie, Web cache

Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.4 electronic mail

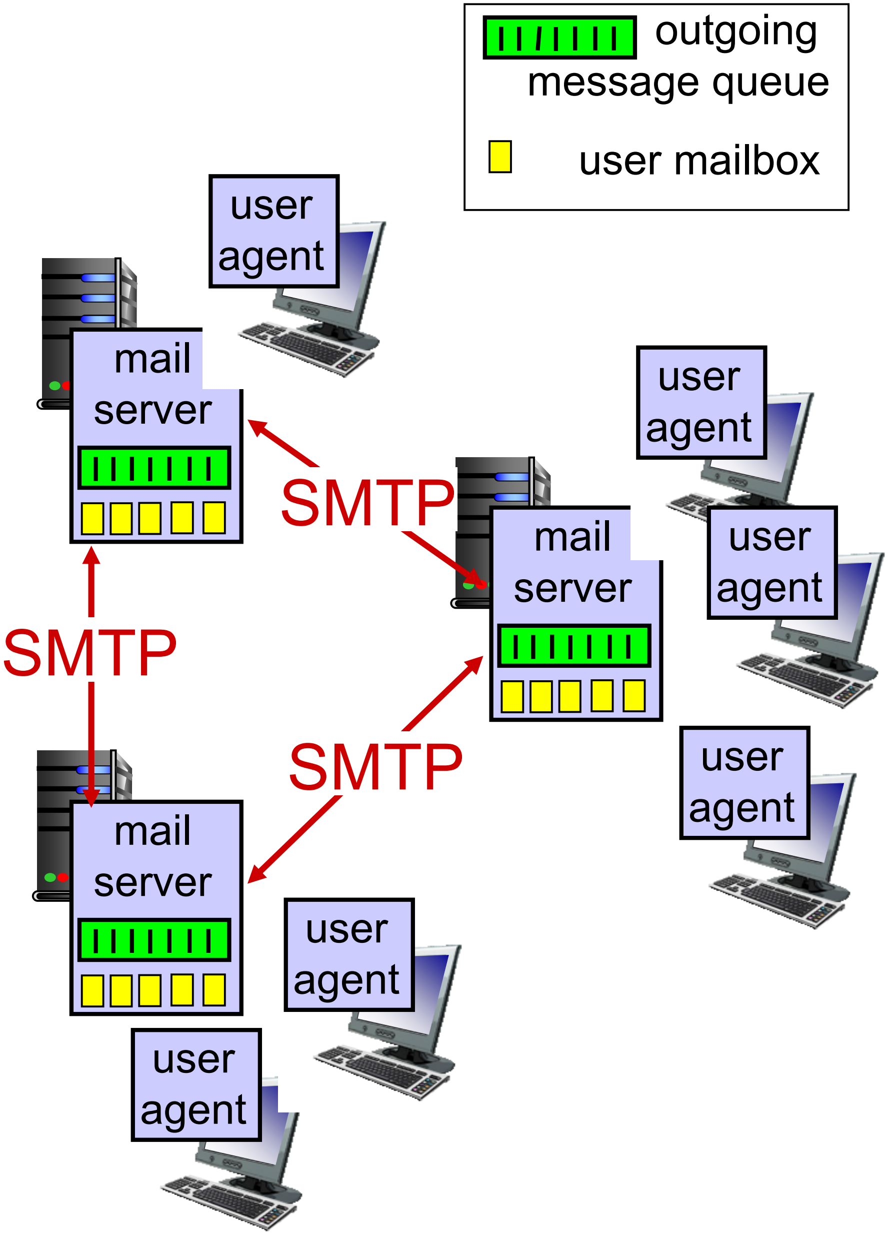
- SMTP, POP3, IMAP

2.5 DNS

Application Layer: Electronic Mail

- One of the oldest and the most important applications in the Internet.
- E-mail is an asynchronous communication medium.
- E-mail is fast, inexpensive, and easy to distribute

Electronic mail



Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP (Simple Mail Transfer Protocol)

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 :

A protocol used to send and receive email messages between mail servers.

- client: sending mail server
- "server": receiving mail server
- SMTP uses TCP as its underlying transport protocol to provide the reliable data transfer service.

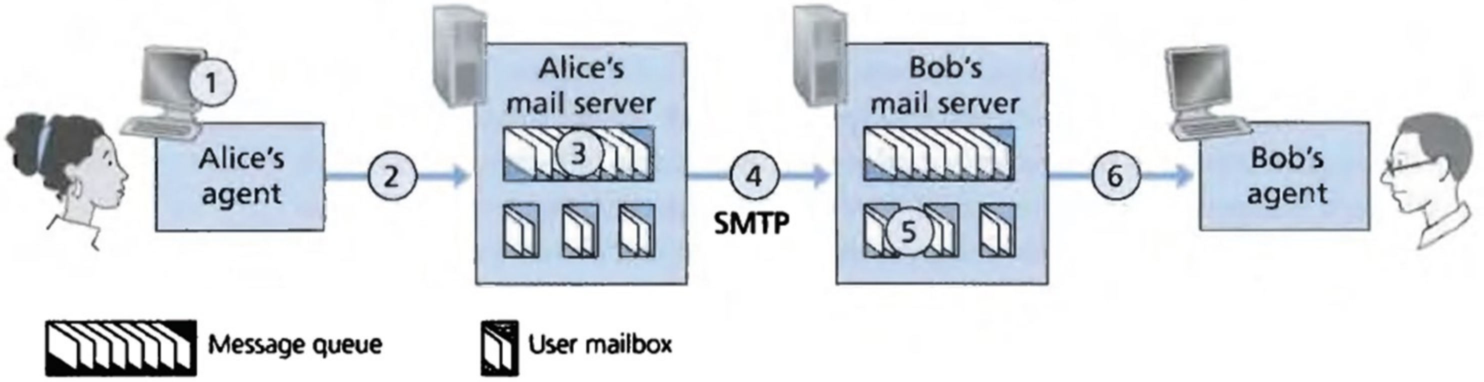


Figure 2.17 ♦ Alice sends a message to Bob

Electronic Mail: SMTP [RFC 2821]

- Client's SMTP mail server establishes a TCP connection to the recipients SMTP server using 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 ASCII

Scenario: Alice sends message to Bob

1) Alice uses UA to compose message "to" `bob@someschool.edu`

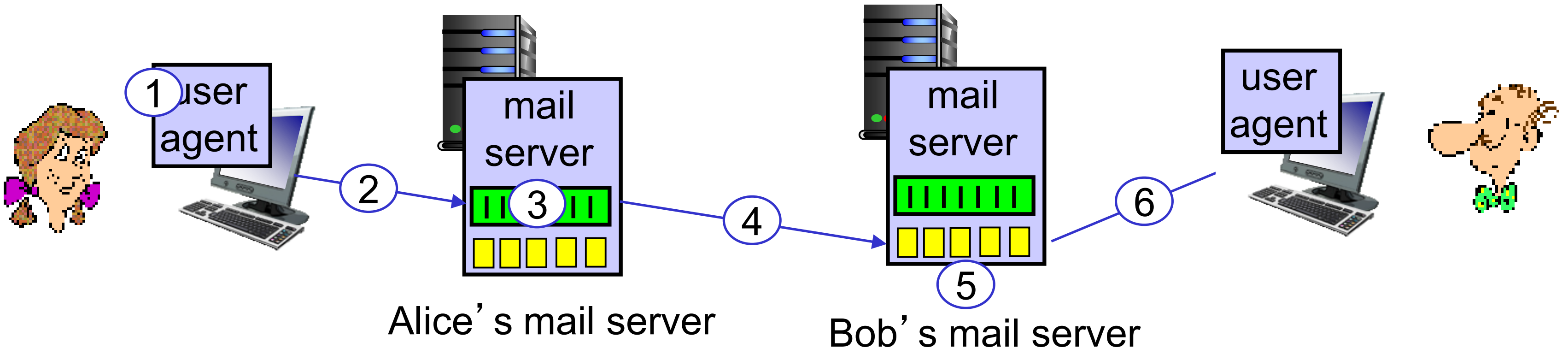
2) Alice's UA sends message to her mail server; message placed in message queue

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

The following transcript begins as soon as the TCP connection is established.

```
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

  If the sending mail server has several messages to send to the same receiving mail server, it can send all of the messages over the same TCP connection.
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses `CRLF.CRLF` to determine end of message (where CR and LF stand for carriage return and line feed, respectively)

Comparison of HTTP and SMTP

Common characteristics

- Both are client-and-server Model
- Both use the reliable data transfer service of TCP
- Use persistent connection

Difference

- HTTP: pull protocol-someone loads information on a Web server and users use HTTP to pull the information from the server.
- SMTP: push protocol-the sending mail server pushes the file to the receiving mail server.
- SMTP has 7-bit ASCII restriction. HTTP does not have this kind of restriction.
- HTTP: each object encapsulated in its own response message
- SMTP: place all of message's objects into one message

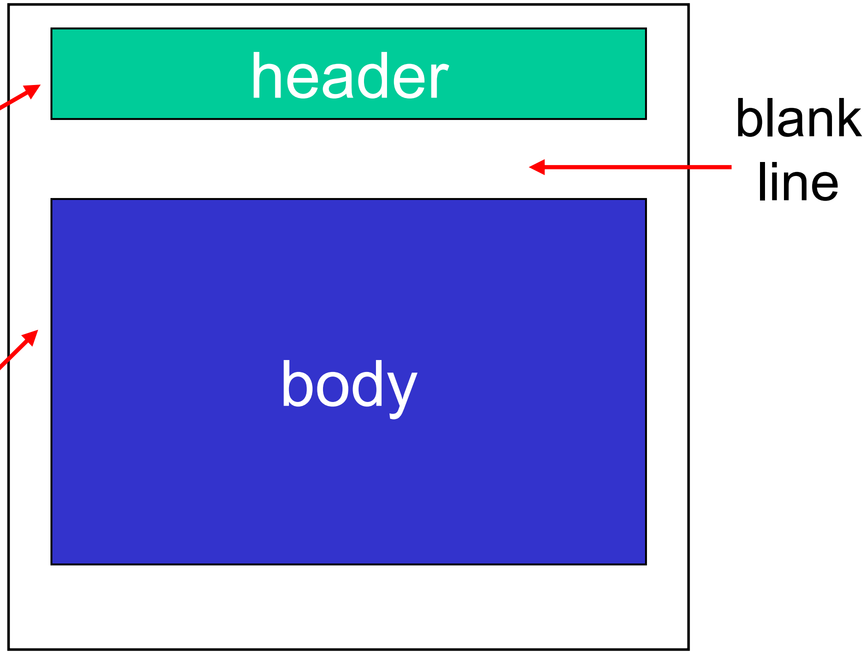
Mail message format

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



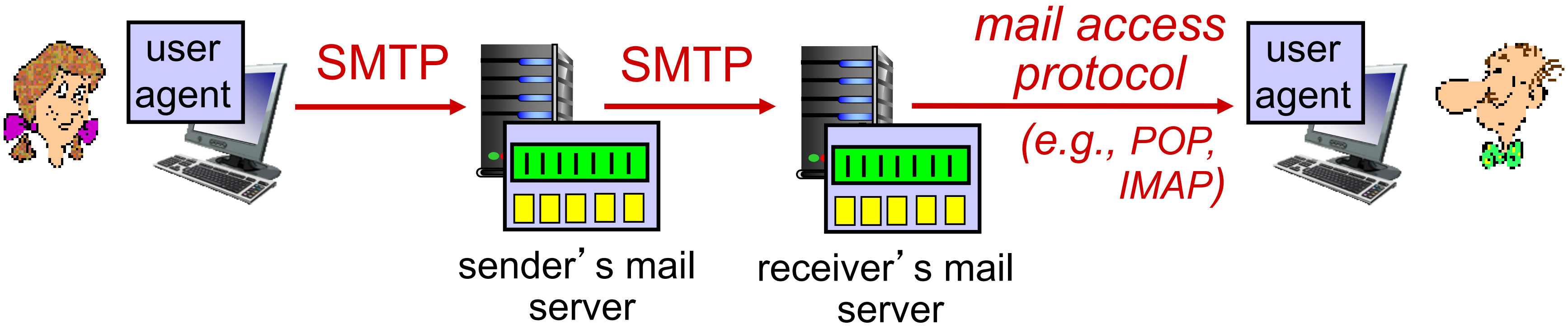
A typical message header looks like this:

From: alice@crepes.fr

To: bob@hamburger.edu

Subject: Searching for the meaning of life.

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: Web-based Email, such as gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

Port: 110

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)

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

Summary

- Three main component of E-mail system
  - Users agents, mail server, and SMTP
- SMTP and the comparison with HTTP.
- SMTP mail message format
- Mail access protocols
  - PoP, IMAP, HTTP

Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

DNS: Domain Name System

Internet hosts: two ways to identify a host

- hostname, e.g., www.yahoo.com
  - Easy to be remembered, human beings prefer
  - Provide little information about the location of this host
  - Difficult to process by router
- IP address (IPV4: 4 bytes (i.e.,32 bit) such as 121.7.106.83)
  - Fixed length, hierarchical structure,
  - Easy to process by router, routers prefer
  - Difficult to be remembered

The transformation between hostname and IP address

- Applications running at a host need to translate a hostname to its IP address (identifier is IP address + port number)

DNS: Domain Name System

DNS is posed to achieve this function

- Before a network application (e.g., web browser) sends a message to its receiving host, it should know the IP address of the receiving host.
- The application invokes the client side of DNS, specifying the hostname that needs to be translated.
- Then, DNS client sends a query message to a DNS server.
- After a time delay, DNS client receives a DNS reply message including the IP address for this hostname.
- The IP address is then passed to the invoking application.
- Once the browser receives the IP address from DNS, it can initiate a TCP connection to the HTTP server process (IP address + port 80)

DNS (Domain Name System)

- DNS provides
  - Translation between hostname to IP address
  - Host aliasing: translation from alias name to canonical name.
    - Two types of hostnames
      - Canonical name: formal hostname, e.g., relay1.west-cost.enterprise.com
      - Alias name: easy to remember, e.g., enterprise.com
  - Mail server aliasing: translation from alias name to canonical name of a mail server
    - e.g., mail server is mailhost1.ipm.edu.mo, and E-mail address is xuyang@ipm.edu.mo

DNS (Domain Name System)

- Load distribution
  - Replicated web servers for busy sites: a set of IP addresses for one canonical name
    - Busy sites are replicated over multiple web servers.
    - Each replicated server run on a different end system and have a different IP address.
    - But all replicated servers are associated with one canonical name.
  - DNS rotation distributes the traffic among the replicated servers.
    - When clients make a DNS query for a busy site, the server responds with the entire set of IP addresses, but rotates the ordering of the addresses within each reply

How to Implement DNS (Domain Name System)

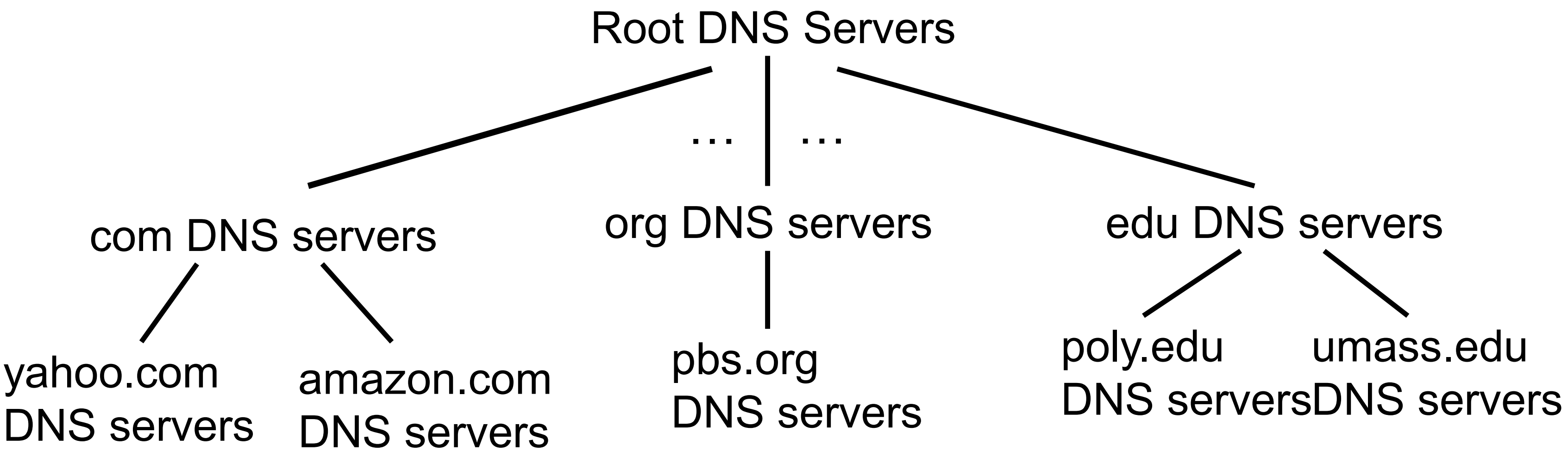
Centralized DNS? One DNS server contains all the mappings between the hostnames and IP addresses

- single point of failure
- traffic volume, less scalability
- distant centralized database---a single DNS server cannot be “close to” all end systems, which will leads to long delay for some end systems.
- Maintenance: this centralized database will be huge and has to be updated frequently to account for every new host.

A: doesn’t scale!

Distributed DNS: DNS consists of a large number of DNS servers distributed around the globe and specify how the DNS server and querying hosts communicate.

DNS: a distributed, hierarchical database



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

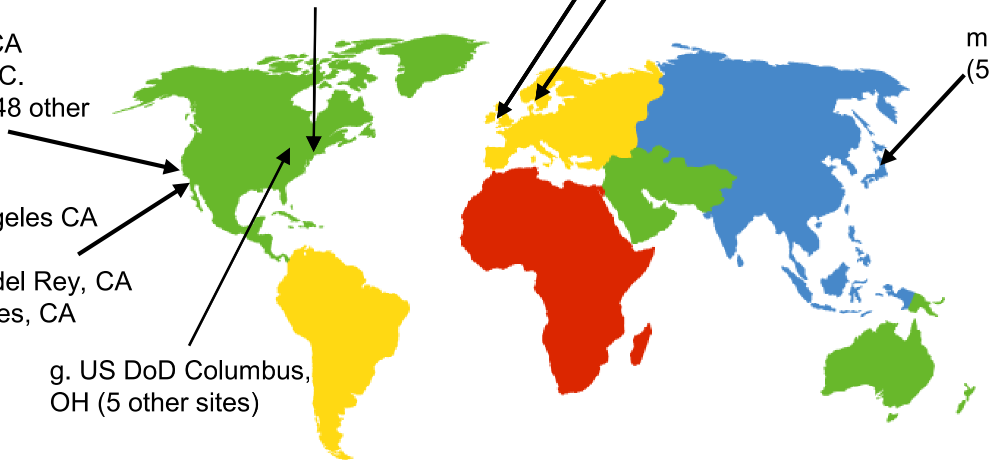
- client queries root server to find com DNS server
- client queries .com DNS server (TLD server) to get amazon.com DNS server
- client queries amazon.com DNS server (authoritative server)to get IP address for www.amazon.com

DNS (Domain Name System)

- The DNS is
  - A distributed database implemented in hierarchy of DNS servers. (scalability and reliability)
  - An application-layer protocol that allows hosts to query the distributed database.
  - An application-layer protocol that uses UDP as its underlying transport layer protocol and use Port 53.
  - DNS is commonly employed by other application-layer protocols—including HTTP, SMTP, and FTP—to translate user-supplied hostnames to IP addresses.

DNS: root name servers

- In the Internet, there are 13 root DNS servers labeled A throughput M in the following figures, most of which are located in North America. (2012)
- Root DNS servers typically do not contain hostname to IP mappings; they contain mappings for locating top-level domain (TLD) servers.



c. Cogent, Herndon, VA (5 other sites)
d. U Maryland College Park, MD
h. ARL Aberdeen, MD
j. Verisign, Dulles VA (69 other sites)

k. RIPE London (17 other sites)

i. Netnod, Stockholm (37 other sites)

e. NASA Mt View, CA
f. Internet Software C.
Palo Alto, CA (and 48 other sites)

m. WIDE Tokyo
(5 other sites)

a. Verisign, Los Angeles CA
(5 other sites)
b. USC-ISI Marina del Rey, CA
l. ICANN Los Angeles, CA
(41 other sites)

g. US DoD Columbus, OH (5 other sites)

13 root name "servers" worldwide

TLD DNS Servers

- Top-level domain (TLD) servers are responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  - Company Network Solutions maintains the TLD servers for com top-level domain.
  - Company Educause maintains the TLD servers for the edu top-level domain.
- TLD servers typically do not contain hostname to IP mappings; they contain mappings for locating authoritative servers.

Authoritative DNS Servers

- Authoritative DNS servers: provides mapping from hostname to IP address for organization's servers (e.g., Web server, mail server).
  - Store the mapping from hostname to IP address.
  - Each organization can implement its own authoritative DNS server to hold these records, it also can have these records stored in an authoritative DNS server of some service provider.
  - Each organization with publicly accessible hosts (e.g., Web servers or mail servers) on the Internet must have records that map the name of these hosts to IP addresses.

Local DNS Server

- Each ISP (residential ISP, company, university, etc) has a local DNS server. When a host connects to Internet through a ISP, the ISP provides the host with the IP addresses of its local DNS server.
- A local DNS server does not strictly belong to the hierarchy of servers.
- A host's local DNS server is typically "close to" the host.
  - For an institutional ISP, at the same LAN
  - For a residential ISP, no more than a few routers.

Local DNS Server

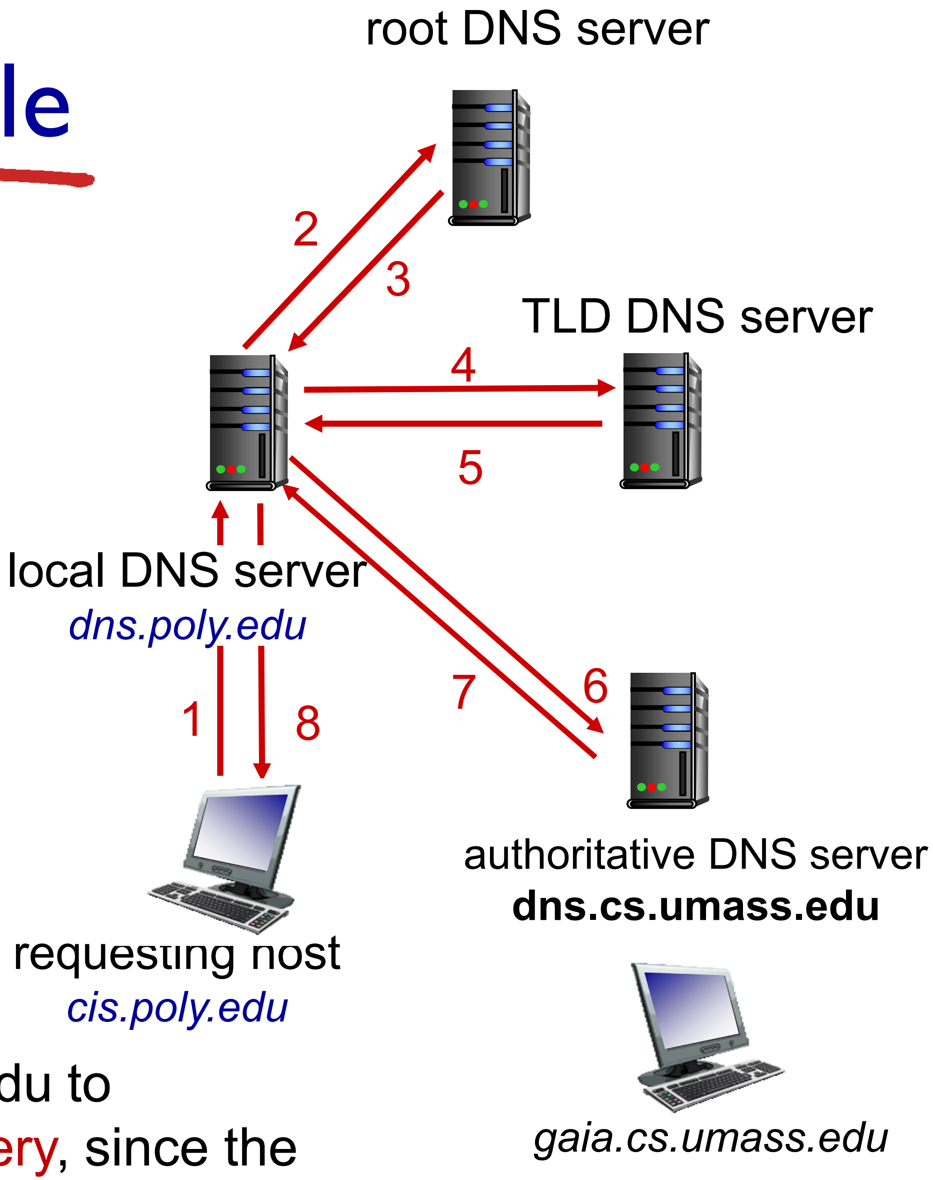
- A local DNS server is also called a default name server. When a host makes a DNS query, query is sent to its local DNS server
  - The local DNS server acts as a proxy, forwards query into hierarchy of servers.
  - Reduces lookup latency for commonly searched hostnames
- The IP address of your local DNS server can be easily determined by using the command ipconfig /all in Windows OS

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
- 8 DNS messages were sent: 4 query messages and 4 reply messages.

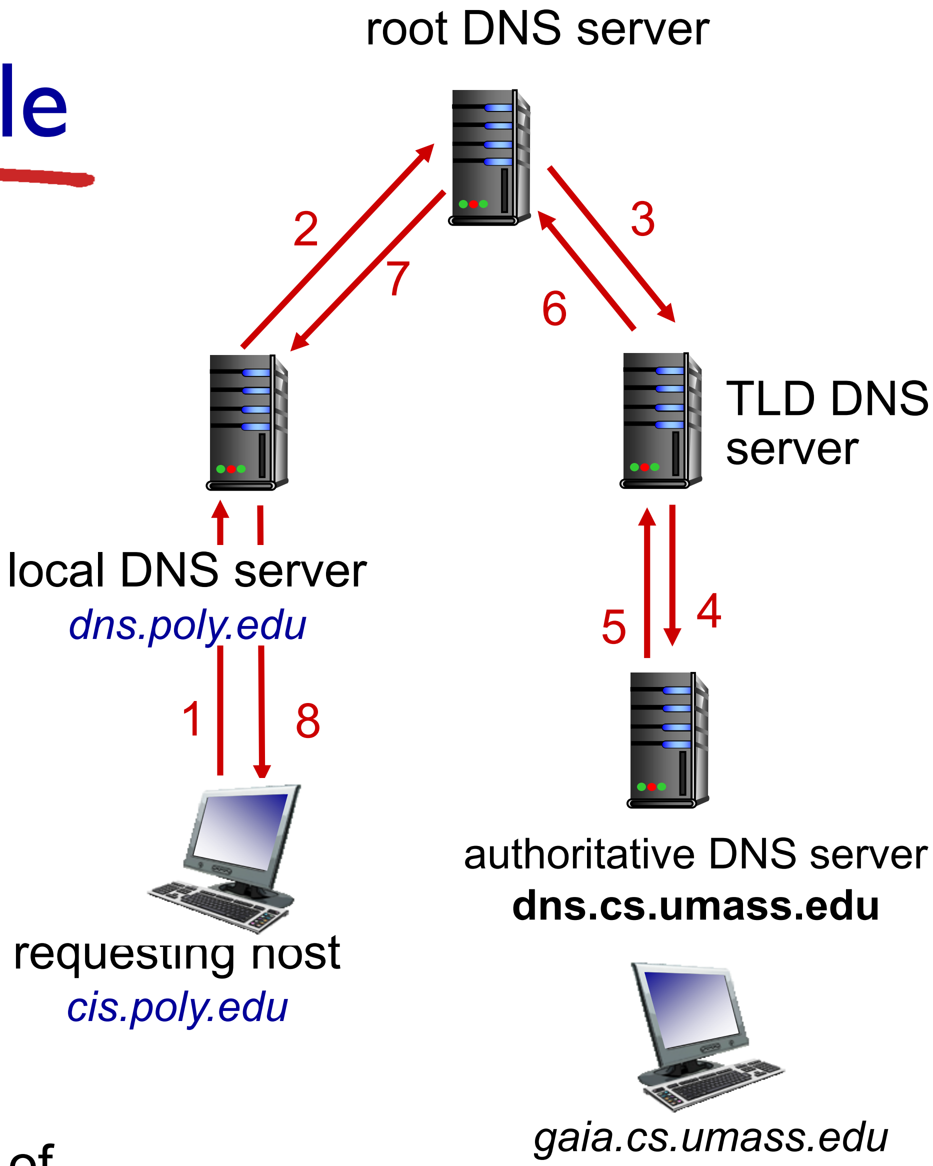


The query sent from cis.poly.edu to dns.poly.edu is a recursive query, since the query asks dns.poly.edu to obtain the mapping on its behalf.

DNS name resolution example

recursive query:

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



a DNS query chain for which all of the queries are recursive.

DNS: caching, updating records

- Caching:(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
- Caching is to improve delay performance by reducing the number of DNS messages.

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`**

Ttl: time to live.

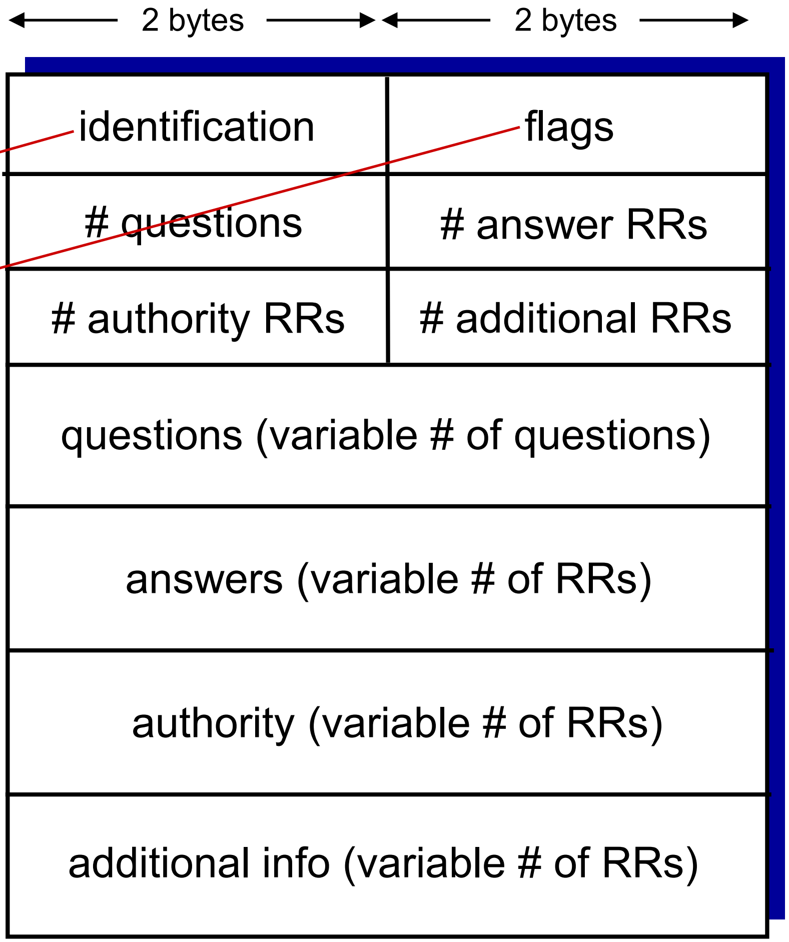
- **Determines when a RR should be removed from a cache**

DNS protocol, messages

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

msg header

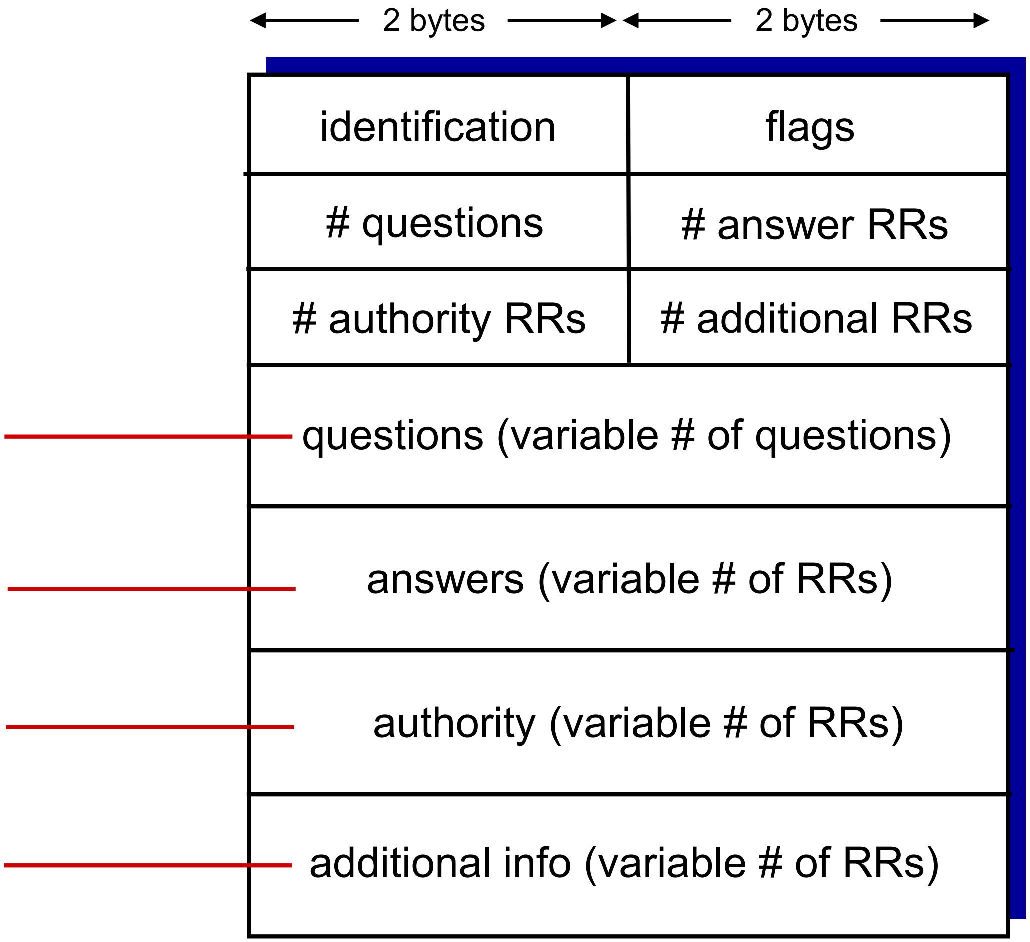
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



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)	

The first 12 bytes is the header section

DNS protocol, messages



← 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)	

name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used

Chapter 2: summary

our study of network apps now complete!

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

- specific protocols:
  - HTTP
  - SMTP, POP, IMAP
  - DNS

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"