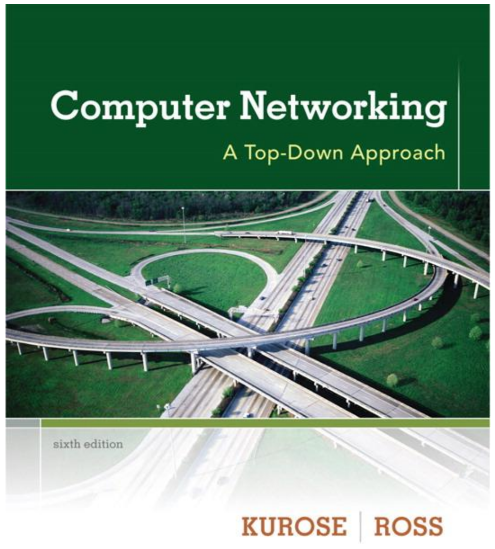
Application Layer



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

Application Layer

- Principles of Network Applications
- Web and HTTP
- FTP
- Electronic mail
  - SMTP, POP3, IMAP
- DNS
- 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
- …
- …

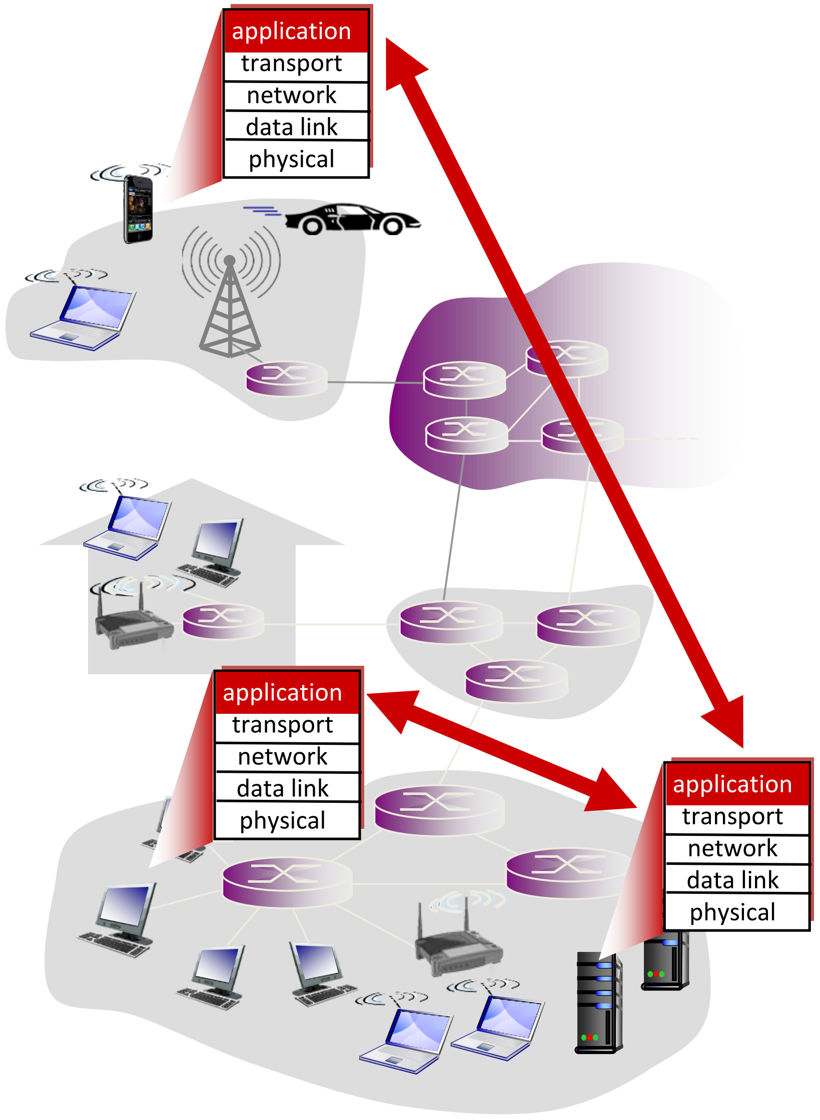
Creating a Network App

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 allow rapid app development, propagation

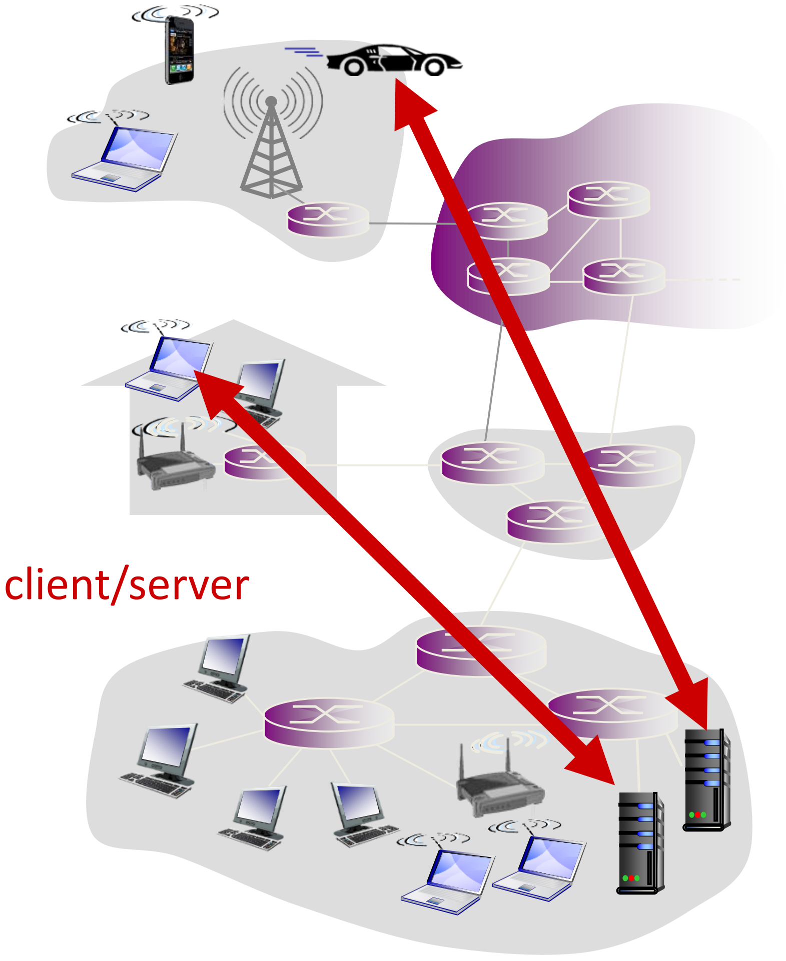


Application Architectures

Possible structure of applications:

- 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

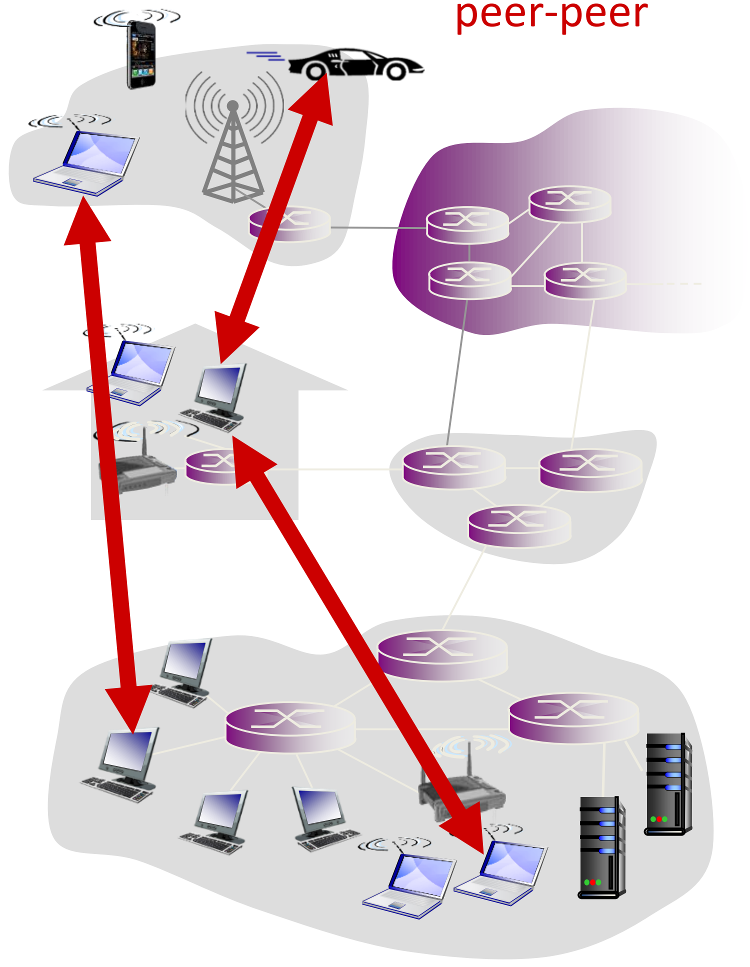
Clients:

- Communicate with server
- May be intermittently connected
- May have dynamic IP addresses
- Do not communicate directly with each other

P2P Architecture

Peer

- *No* always-on server
- Arbitrary end systems directly communicate
- Highly scalable but difficult to manage
- 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



Client-Server
vs.
Peer-to-Peer

Hybrid of Client-Server and P2P

- Skype
  - Voice-over-IP P2P application
  - Centralized Server: finding address of remote party
  - Client-Client Connection: Direct (not through server)
- Instant messaging
  - Chatting between two users is P2P
  - Centralized service: client presence
  - Detection/location
    - User registers its IP address with central server when it comes online
    - User contacts central server to find IP addresses of buddies

Processes Communicating

Process: Program running within a host

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

Clients, servers

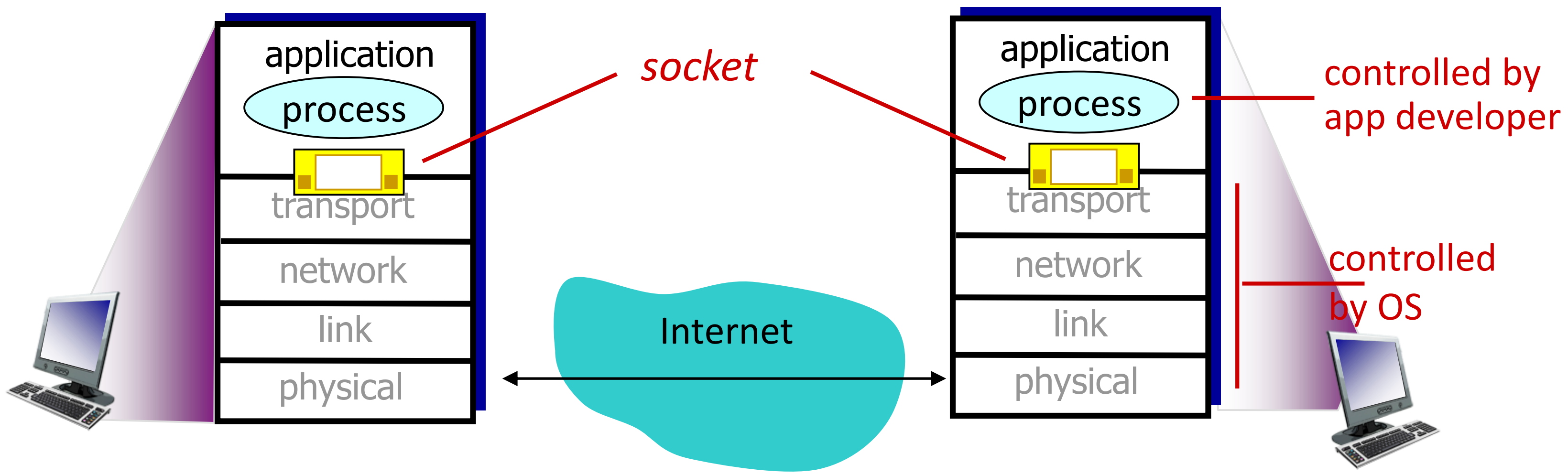
client process: process that initiates communication

server process: process that waits to be contacted

- Aside: Applications with P2P architectures have client processes & server processes

Sockets

- A process sends messages into, and receives messages from, the network through a software interface called a Socket.
- A *process* is analogous to a *house* and its *socket* is analogous to its *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

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

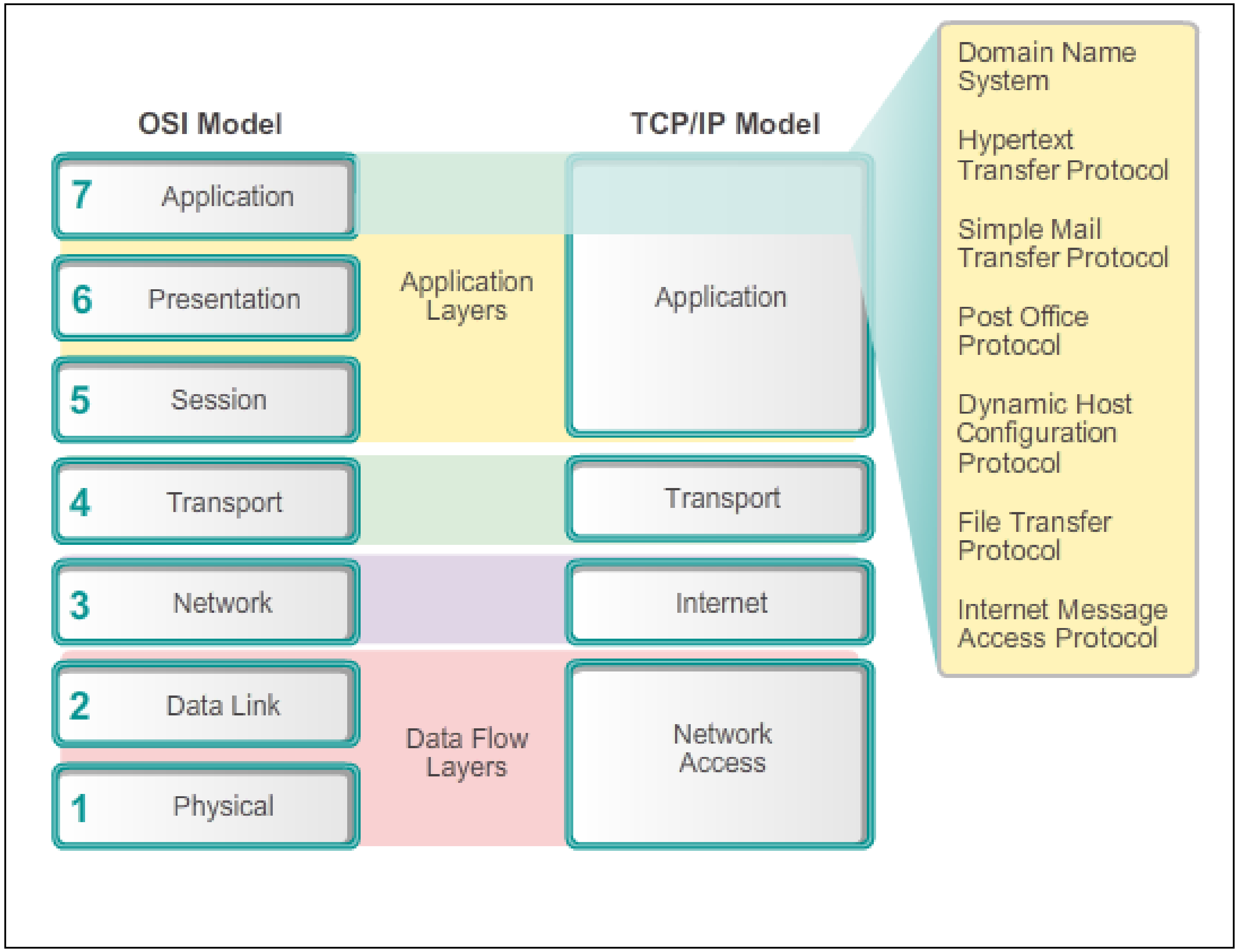
Open Protocols:

- Defined in RFCs
- Allows for interoperability
- e.g. HTTP, SMTP, FTP

Proprietary Protocols:

- e.g. Skype

Application Layer Protocols



Application Layer Protocols

- **Domain Name Service Protocol (DNS)** – used to resolve Internet names to IP addresses
- **Telnet** – a terminal emulation protocol used to provide remote access to servers and networking devices
- **Bootstrap Protocol (BOOTP)** – a precursor to the DHCP protocol, a network protocol used to obtain IP address information during bootup
- **Dynamic Host Control Protocol (DHCP)** – used to assign an IP address, subnet mask, default gateway and DNS server to a host
- **Hypertext Transfer Protocol (HTTP)** – used to transfer files that make up the Web pages of the World Wide Web

Application Layer Protocols

- **File Transfer Protocol (FTP)** - used for interactive file transfer between systems
- **Trivial File Transfer Protocol (TFTP)** - used for connectionless active file transfer
- **Simple Mail Transfer Protocol (SMTP)** - used for the transfer of mail messages and attachments
- **Post Office Protocol (POP)** - used by email clients to retrieve email from a remote server
- **Internet Message Access Protocol (IMAP)** – another protocol for email retrieval

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, *bandwidth sensitive applications*, (e.g., multimedia) require minimum amount of throughput to be "effective"
- Other apps -"elastic apps"- make use of whatever throughput they get (e.g., email, file transfer)

Security

- Encryption, confidentiality, 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
Internet telephony /video conferencing	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes
instant messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- *Connection-oriented:* setup required between client and server processes
- *Reliable transport* between sending and receiving process
- *Flow control:* sender won't overwhelm receiver
- *Congestion control:* throttle sender when network overloaded
- *Does not provide:* timing, minimum throughput guarantee, security

UDP service:

- *Unreliable data transfer* between sending and receiving process
- *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

HTTP: Hypertext Transfer Protocol
RTP: Real-time Transport Protocol
SIP: Session Initiation Protocol
SMTP: Simple Mail Transfer Protocol
FTP: File Transfer Protocol

Securing TCP

TCP & UDP

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

SSL Secure Sockets Layer

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

SSL is at app layer

- apps use SSL libraries, that "talk" to TCP

SSL socket API

- cleartext passwords sent into socket traverse Internet encrypted

Web and HTTP

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 single *URL*, e.g.,

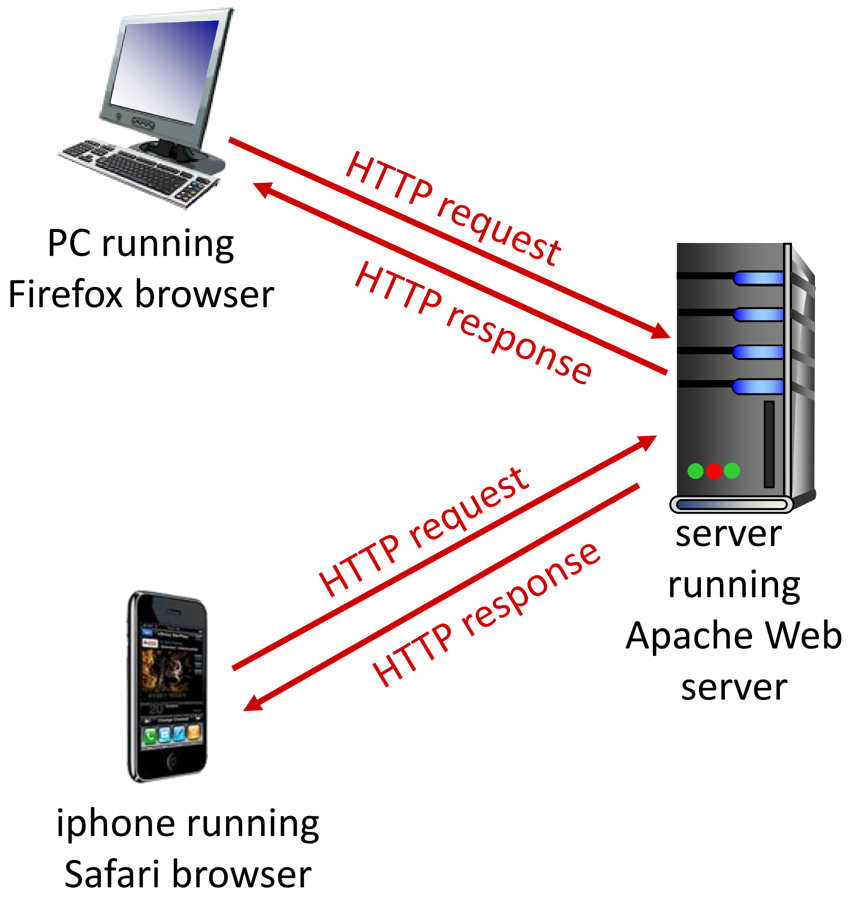
$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

HTTP overview

HTTP: HyperText Transfer Protocol

- Web's application layer protocol
- Client/Server model
  - *Client:* Web browsers (such as Internet Explorer and Firefox) implement the client side of HTTP
    - Browser requests, receives, (using HTTP protocol) and "displays" Web objects
  - *Server:* Web servers, which implement the server side of HTTP, house Web objects, each addressable by a URL.
    - Web server sends (using HTTP protocol) objects in response to requests
    - Popular Web servers include Apache and Microsoft Internet Information Server

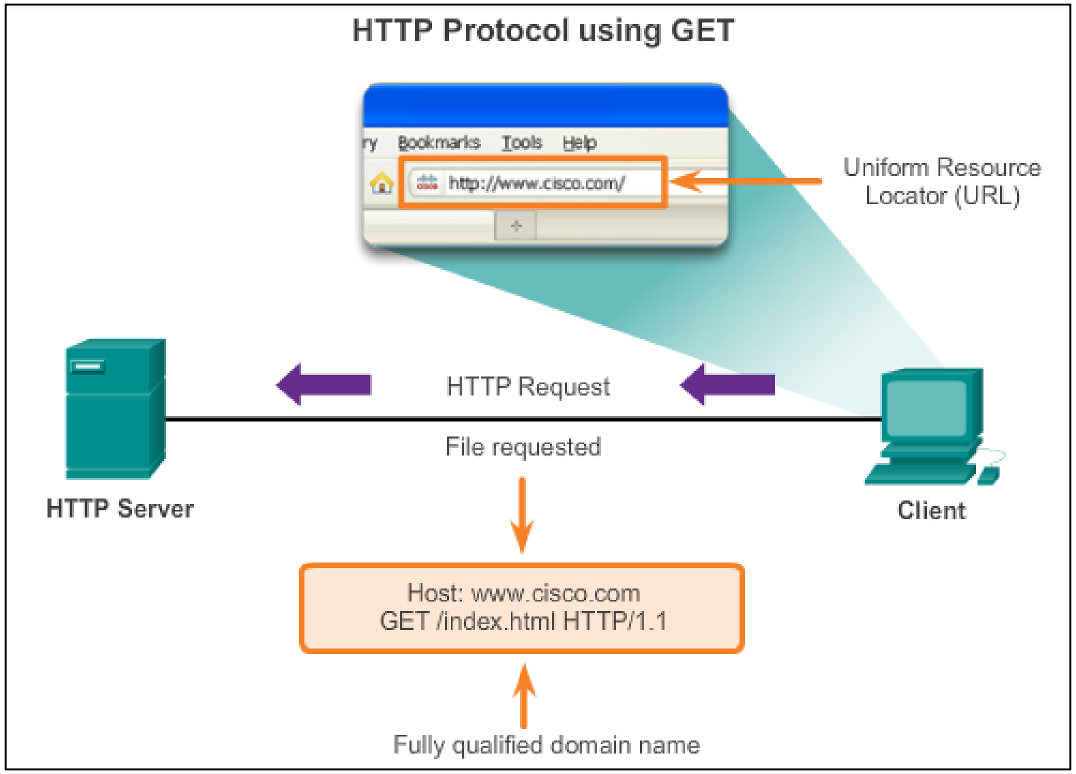
HTTP overview (continued)



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

HTTP request-response behavior

HTTP overview (continued)



HTTP overview (continued)

HTTP Uses TCP as underlying Transport Protocol:

- The HTTP client first initiates a TCP connection with the server.
- Server accepts TCP connection from client.
- Once the connection is established, the browser and the server processes access TCP through their socket interfaces.
- On the client side the socket interface is the door between the client process and the TCP connection.
- On the server side it is the door between the server process and the TCP connection.
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)

HTTP overview (continued)

HTTP is "stateless"

- An HTTP 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 and 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

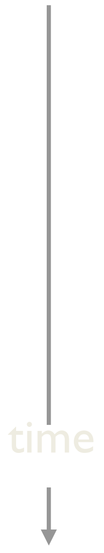
2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

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

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

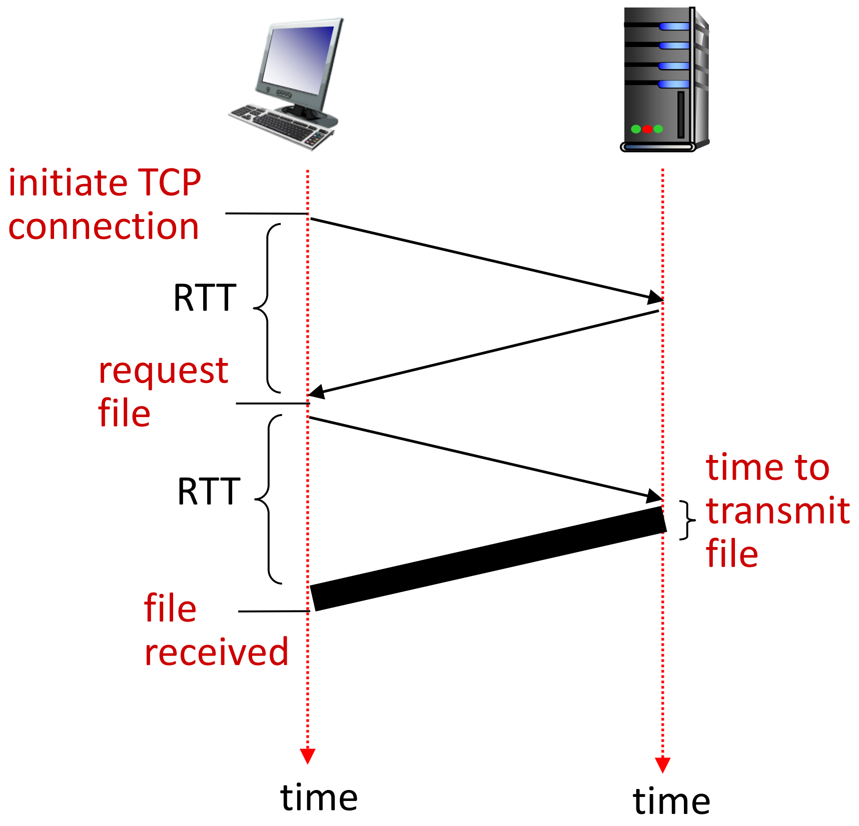
Non-persistent HTTP: response time

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 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

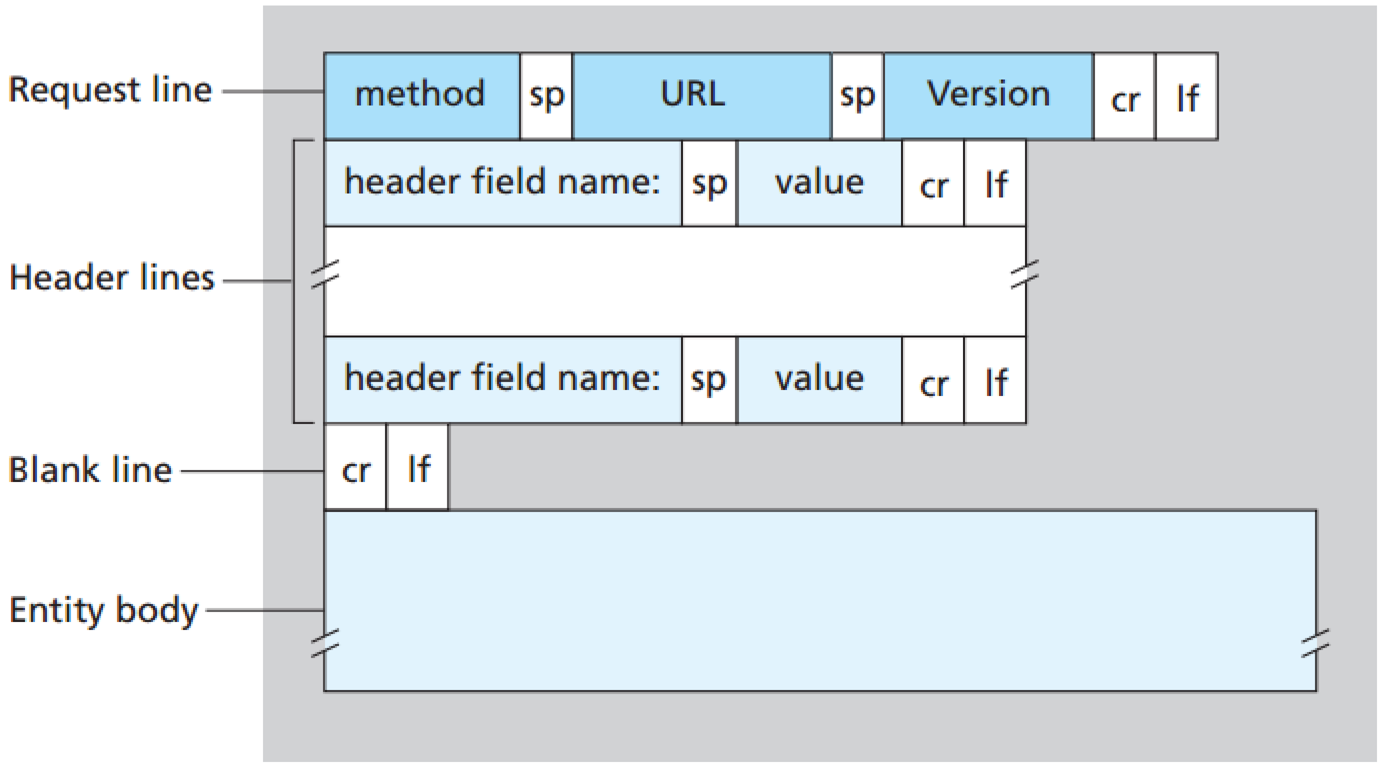
request line (GET, POST, HEAD commands)

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

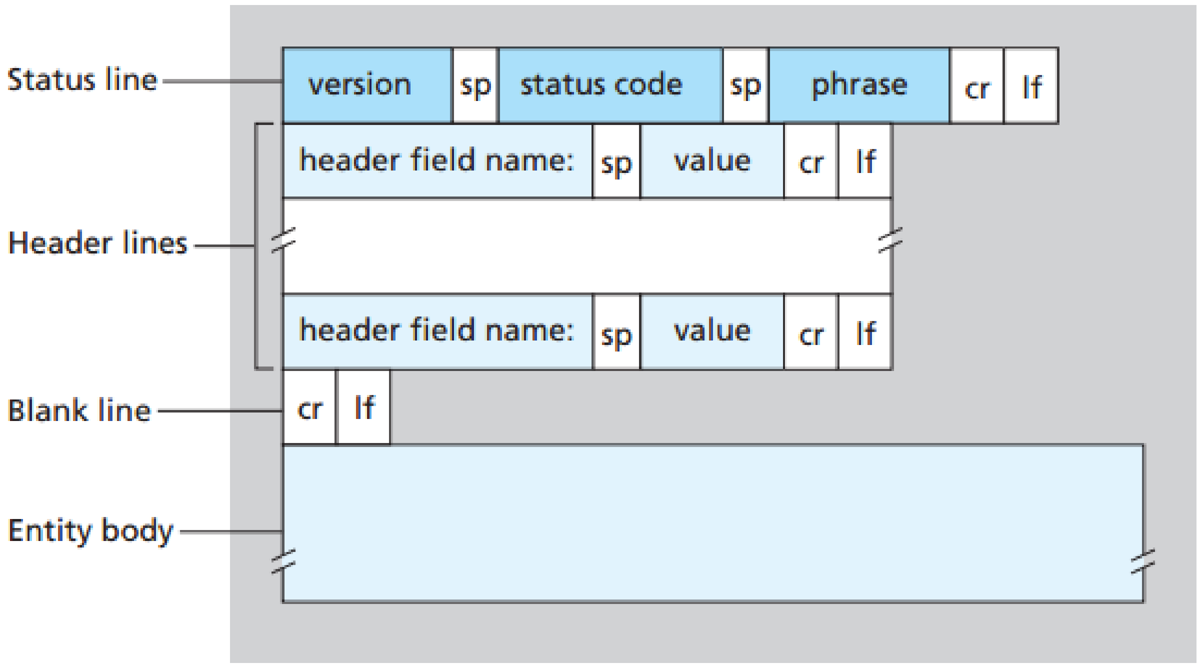
status line (protocol status code status phrase)

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 message: general format



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`

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. anything typed in sent to port 80 at cis.poly.edu

2. type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

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

3. look at response message sent by HTTP server!

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

User-server state: Cookies

Cookies allow sites to keep track of users.

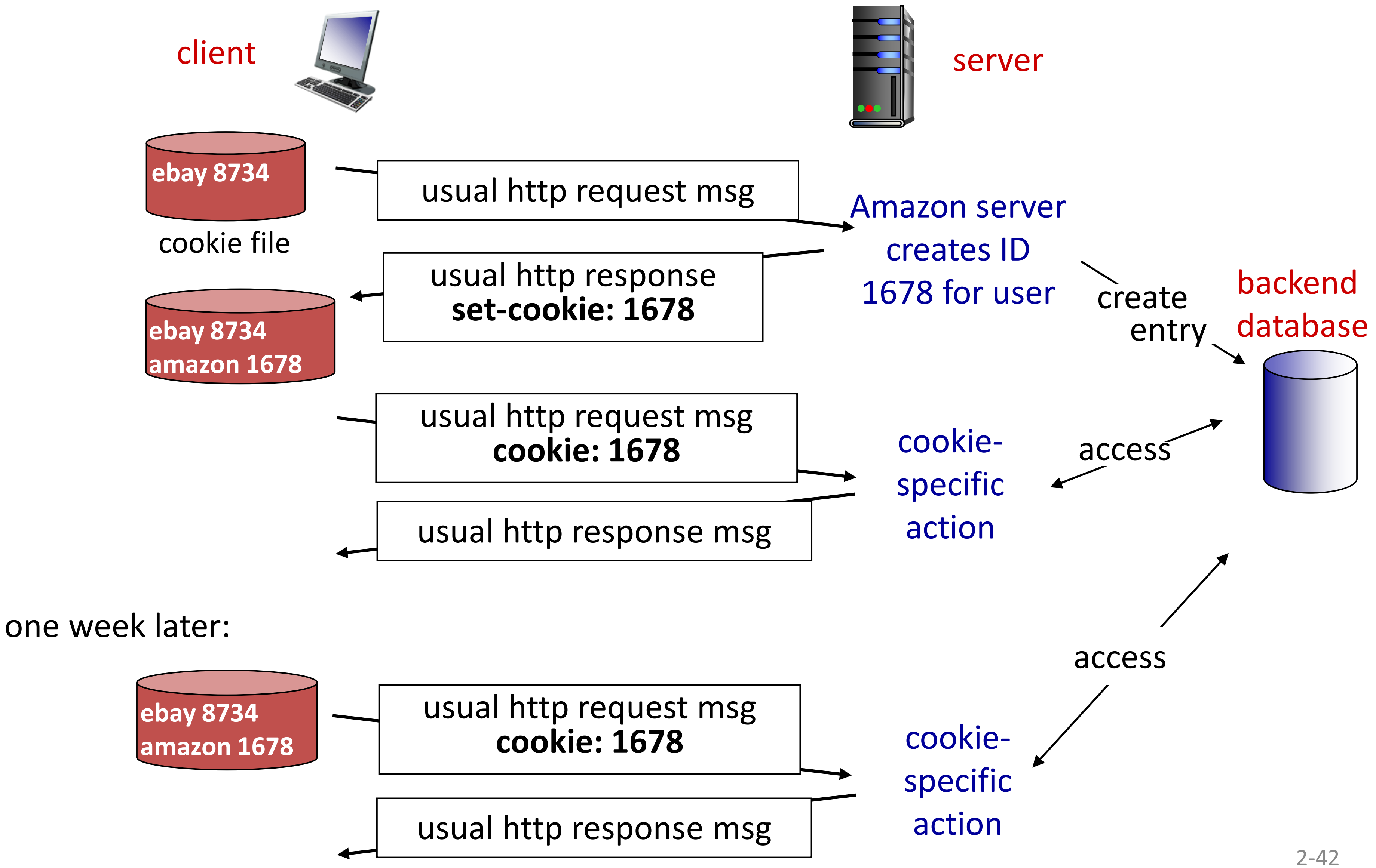
Four components of Cookie technology:

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 back-end database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

Cookies are mainly used for:

- Session management
  - Logins, shopping carts, game scores, or anything else the server should remember
- Personalization
  - User preferences, themes, and other settings
- Tracking
  - Recording and analyzing user behaviour

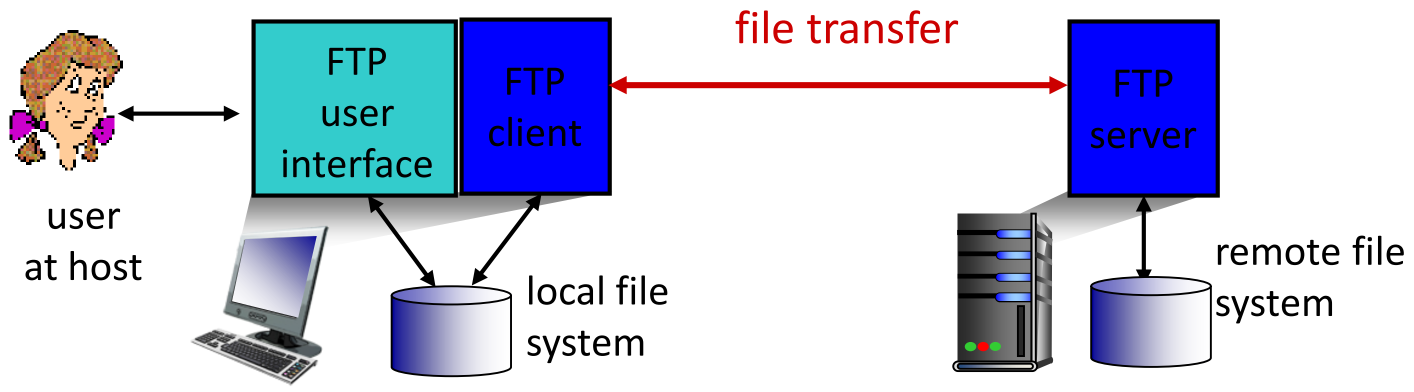
How to keep "state":

- Protocol endpoints: maintain state at sender/receiver over multiple transactions
- Cookies: http messages carry state

Cookies and privacy:

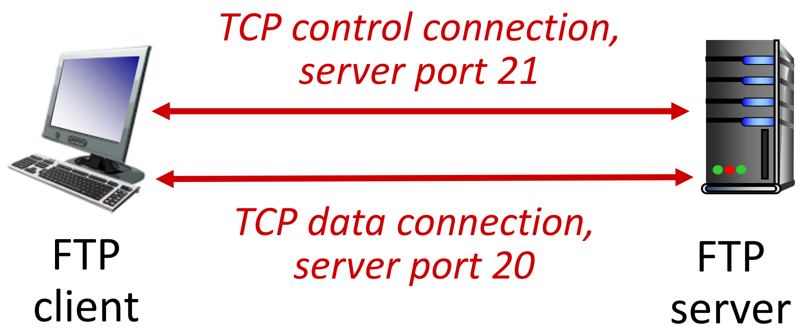
- Cookies permit sites to learn a lot about you
- You may supply name and e-mail to sites

FTP: the file transfer protocol [RFC 959]



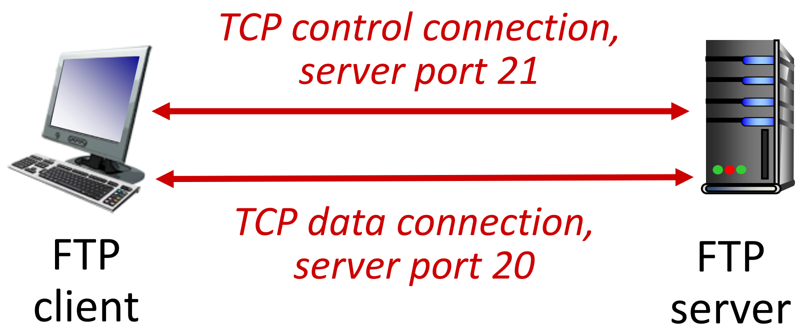
- Transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

FTP: separate control, data connections


TCP control connection, server port 21
TCP data connection, server port 20
FTP client
FTP server

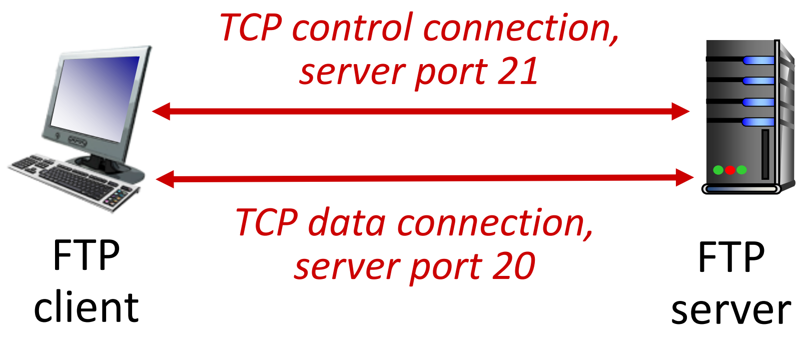
- FTP uses two parallel TCP connections to transfer a file: a *control connection* and a *data connection*
- The *control connection* is used for sending control information between the two hosts—information such as user identification, password, commands to change remote directory, and commands to "put" and "get" files.
- The *data connection* is used to actually send a file

FTP: separate control, data connections


TCP control connection, server port 21
TCP data connection, server port 20
FTP client
FTP server

- FTP client contacts FTP server at port 21, using TCP
- FTP client authorized over control connection
- FTP 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 at Port 20
- After transferring one file, server closes data connection
- Server opens another TCP data connection to transfer another file

FTP: separate control, data connections


TCP control connection, server port 21
TCP data connection, server port 20
FTP client
FTP server

- FTP control connection: *"out of band"*
- HTTP control connection: *"in band"* (sends request and response header lines into the same TCP connection that carries the transferred file itself)
- FTP server maintains "state": current directory, earlier authentication

FTP commands, responses

- Some of the more common commands are given below:
  - **USER *username***
  - **PASS *password***
  - **LIST** returns list of file in current directory
  - **RETR filename** retrieves (gets) file
  - **STOR filename** stores (puts) file onto remote host

FTP commands, responses

- Some typical replies, along with their possible messages, are as follows:
  - **`331 Username OK, password required`**
  - **`125 data connection already open; transfer starting`**
  - **`425 Can’t open data connection`**
  - **`452 Error writing file`**

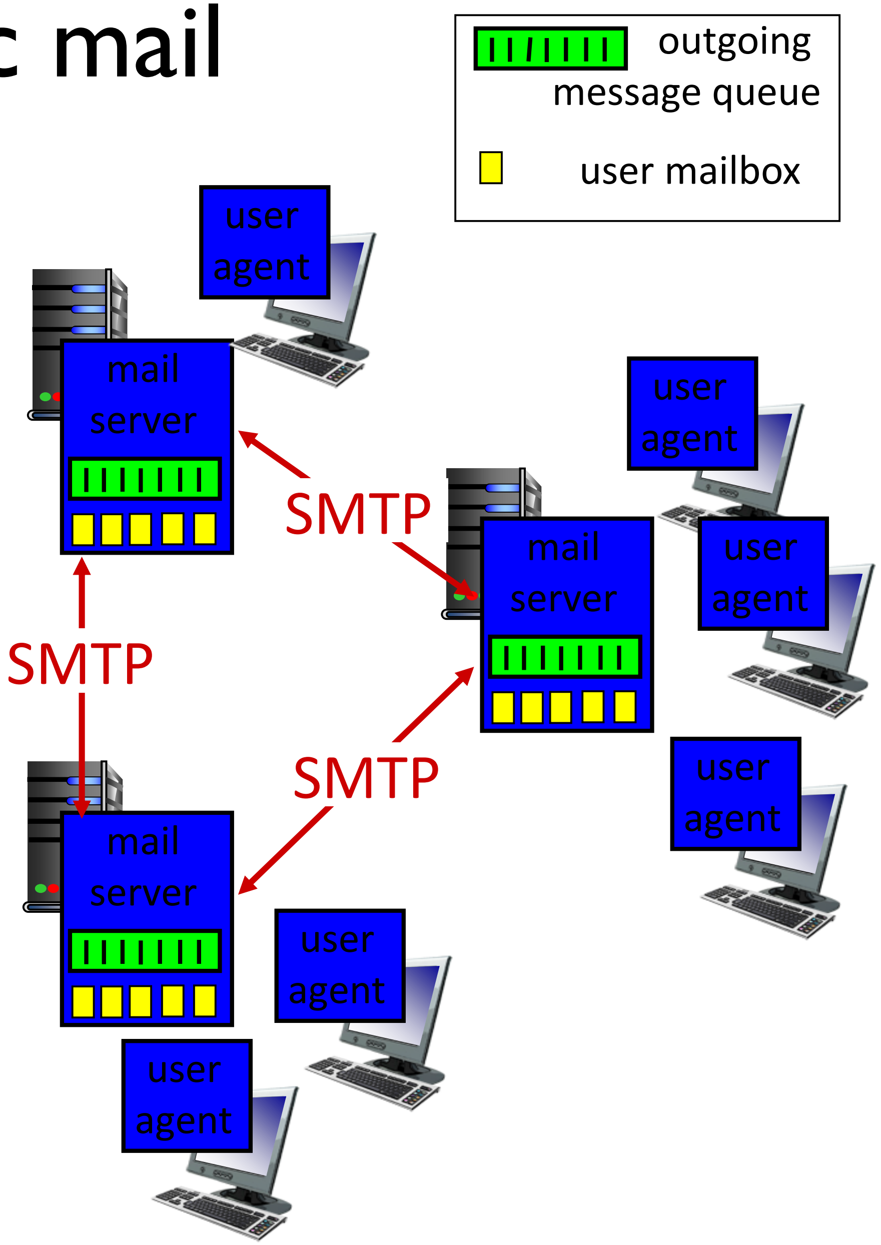
Electronic mail

Three major components:

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

User Agent

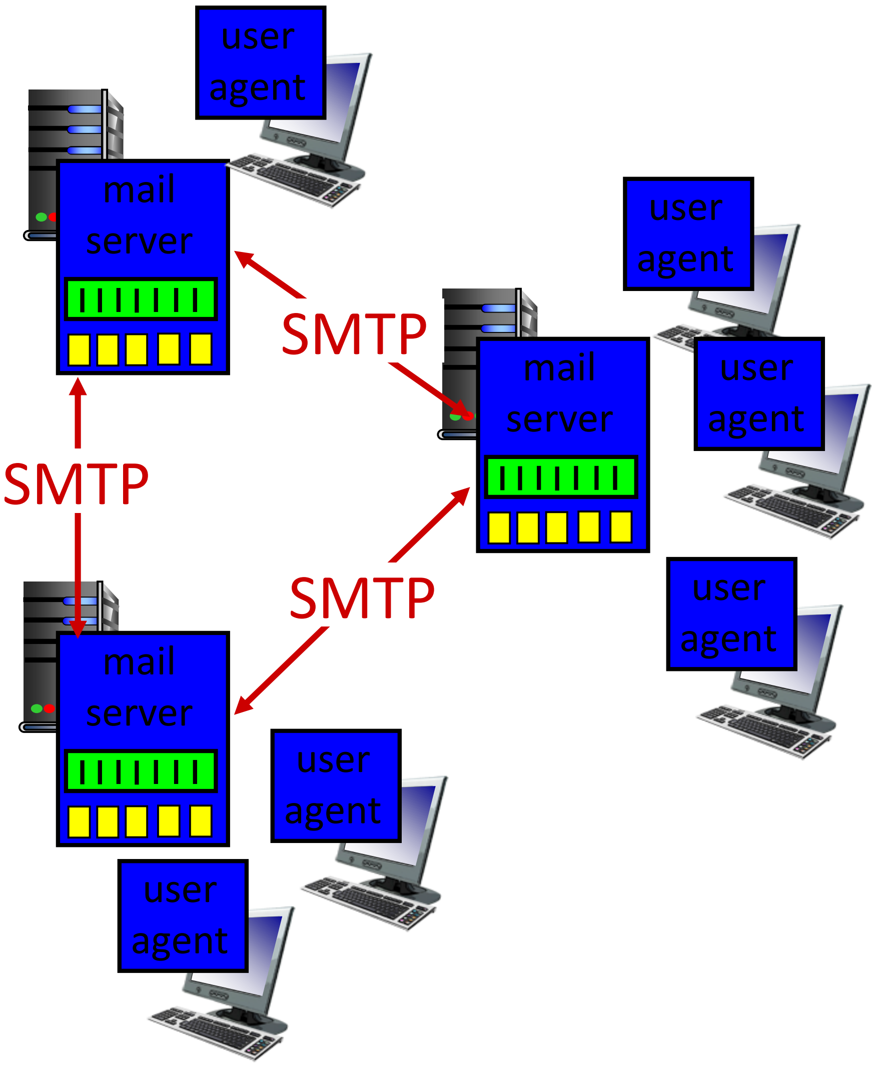
- a.k.a. "mail reader"
- allows users to read, reply to, forward, save, and compose messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



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

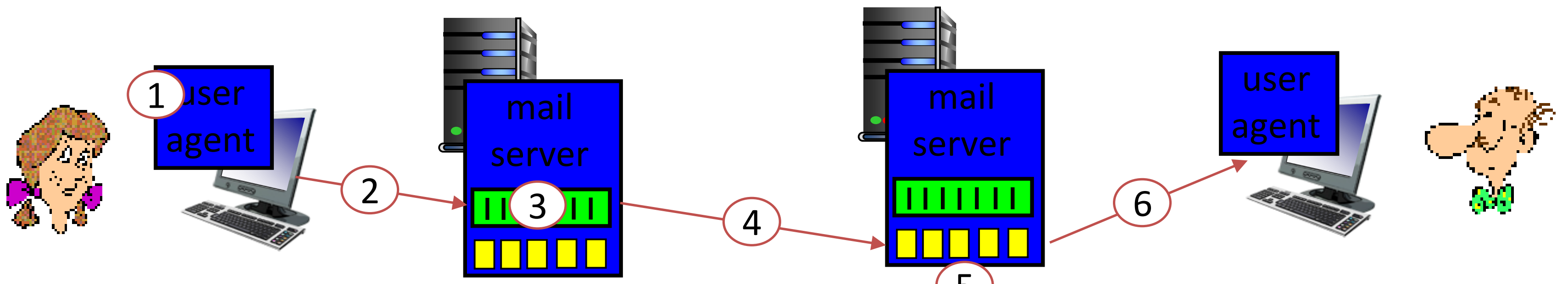


Electronic Mail: SMTP [RFC 2821]

- Uses TCP to reliably transfer email message from client to server, port 25
- Direct transfer: sending server to receiving server
- SMTP has two sides:
  - a client side, which executes on the sender's mail server
  - a server side, which executes on the recipient's mail server
- Both the client and server sides of SMTP run on every mail server.

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



Alice's mail server

Bob's mail server

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

Comparison with HTTP:

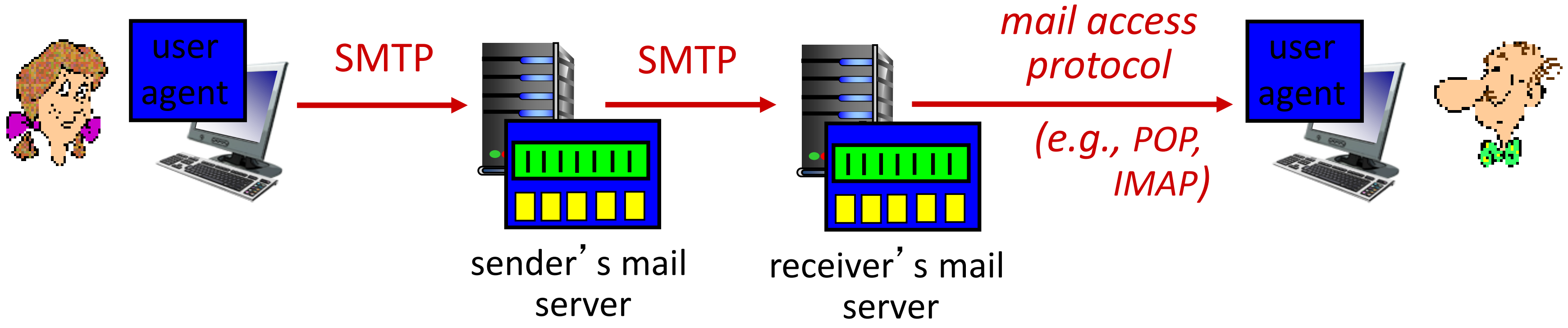
- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

Pull/Push

- HTTP is mainly a **pull protocol**—someone loads information on a Web server and users use HTTP to pull the information from the server at their convenience.
- In particular, the TCP connection is initiated by the machine that wants to receive the file.

- On the other hand, SMTP is primarily a **push protocol**—the sending mail server pushes the file to the receiving mail server.
- In particular, the TCP connection is initiated by the machine that wants to send the file.

Mail access protocols



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

POP3 protocol

authorization phase

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

transaction phase, client:

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

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

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

POP3 (more) and IMAP

more about POP3

- POP3 begins when the user agent (the client) opens a TCP connection to the mail server (the server) on port 110
- 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

DNS: Domain Name System

People: many identifiers:

- NID, name, passport #

Internet hosts, routers:

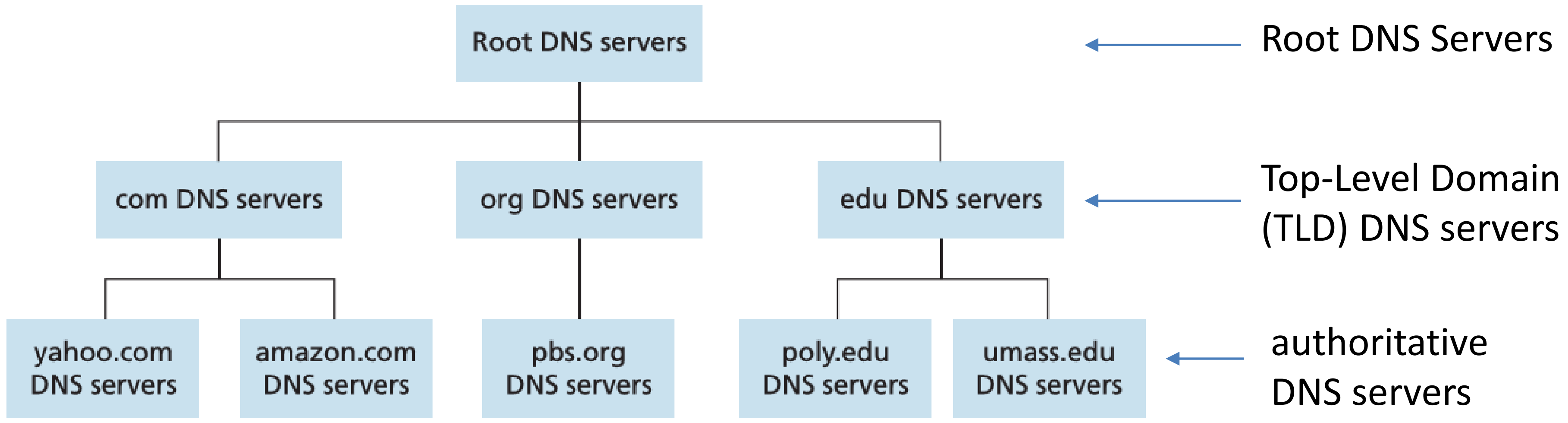
- IP address (32 bit) - used for addressing datagrams
- "hostname", e.g., www.yahoo.com - used by humans

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

Domain Name System:

- *Distributed database* implemented in hierarchy of many *name servers*
- *Application-layer protocol:* hosts, name servers communicate to *resolve* hostnames (address/name translation)
- DNS protocol runs over UDP and uses port 53

DNS: a Distributed, Hierarchical database

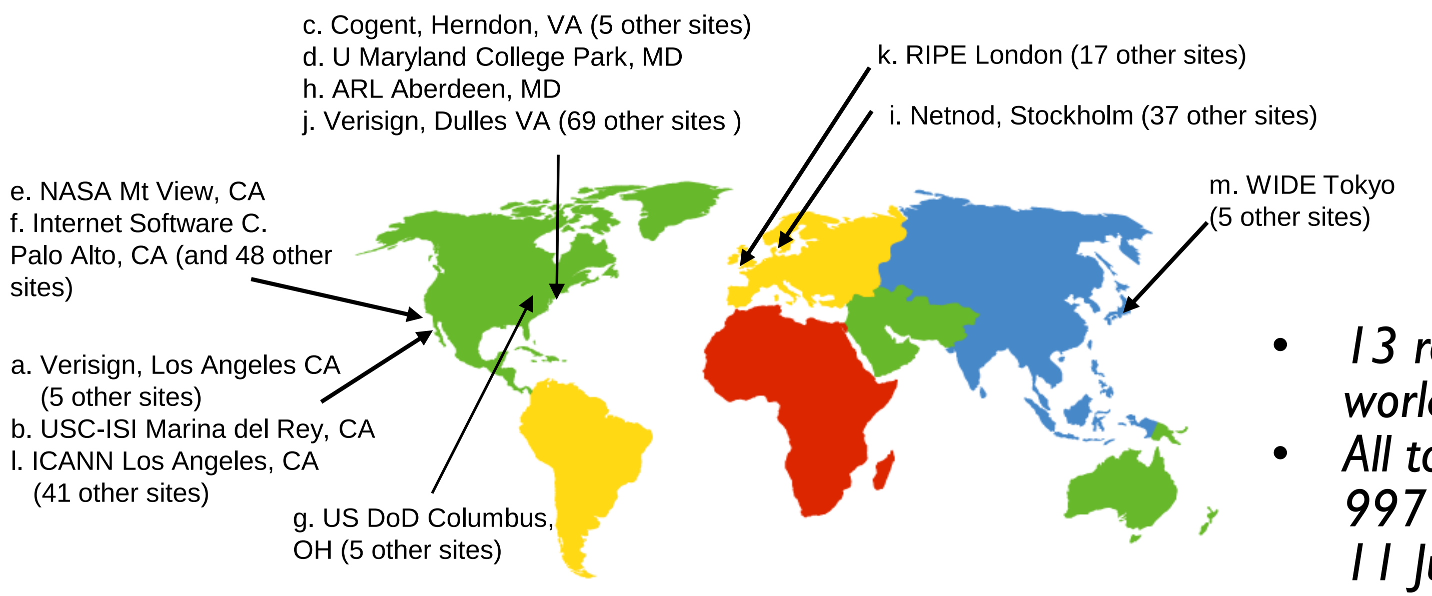


Client wants IP for www.amazon.com; 1st approx:

- 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



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)
m. WIDE Tokyo (5 other sites)
e. NASA Mt View, CA
f. Internet Software C. Palo Alto, CA (and 48 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*
- *All together, there are 997 root servers as of 11 July 2019*

TLD and Authoritative servers

Top-Level Domain (TLD) DNS Servers:

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

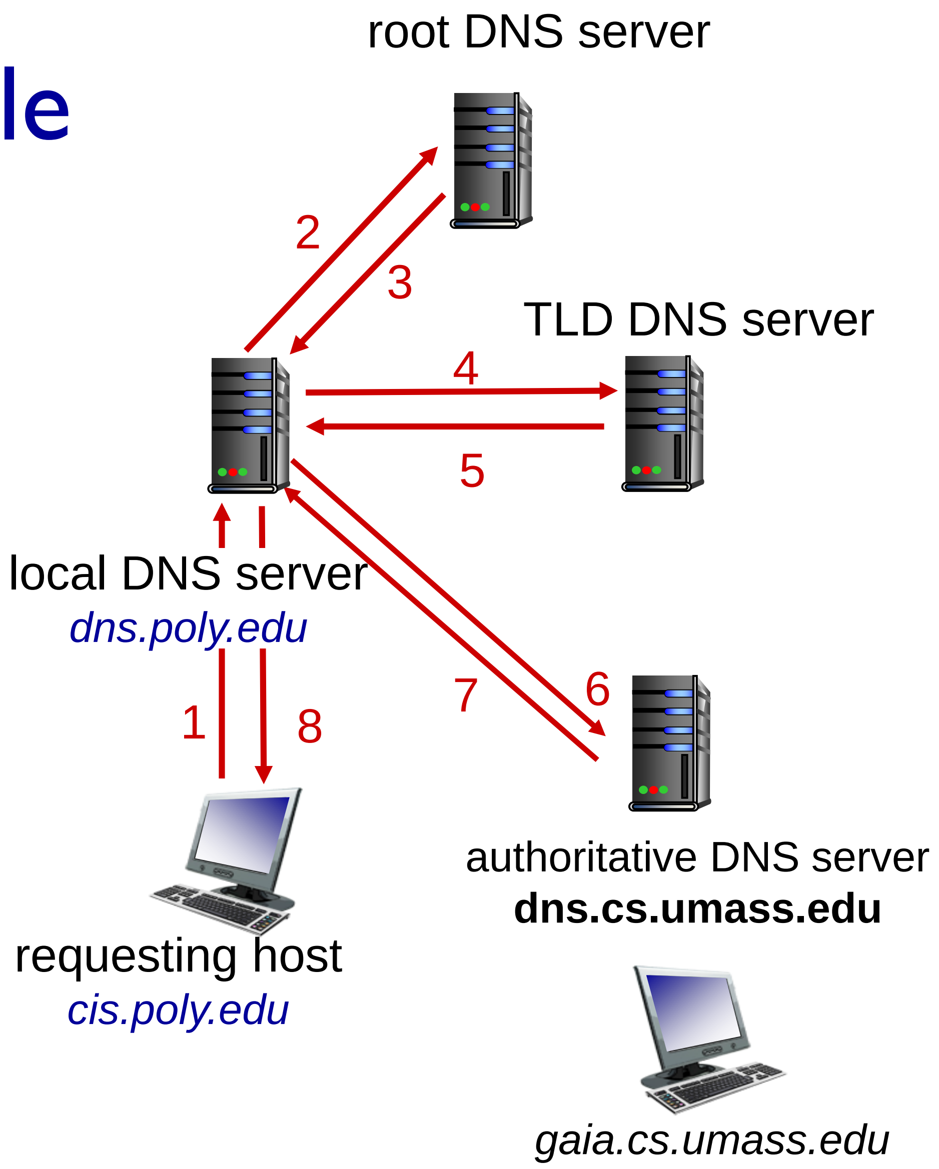
- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one
  - also called “default name server”
- When host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts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

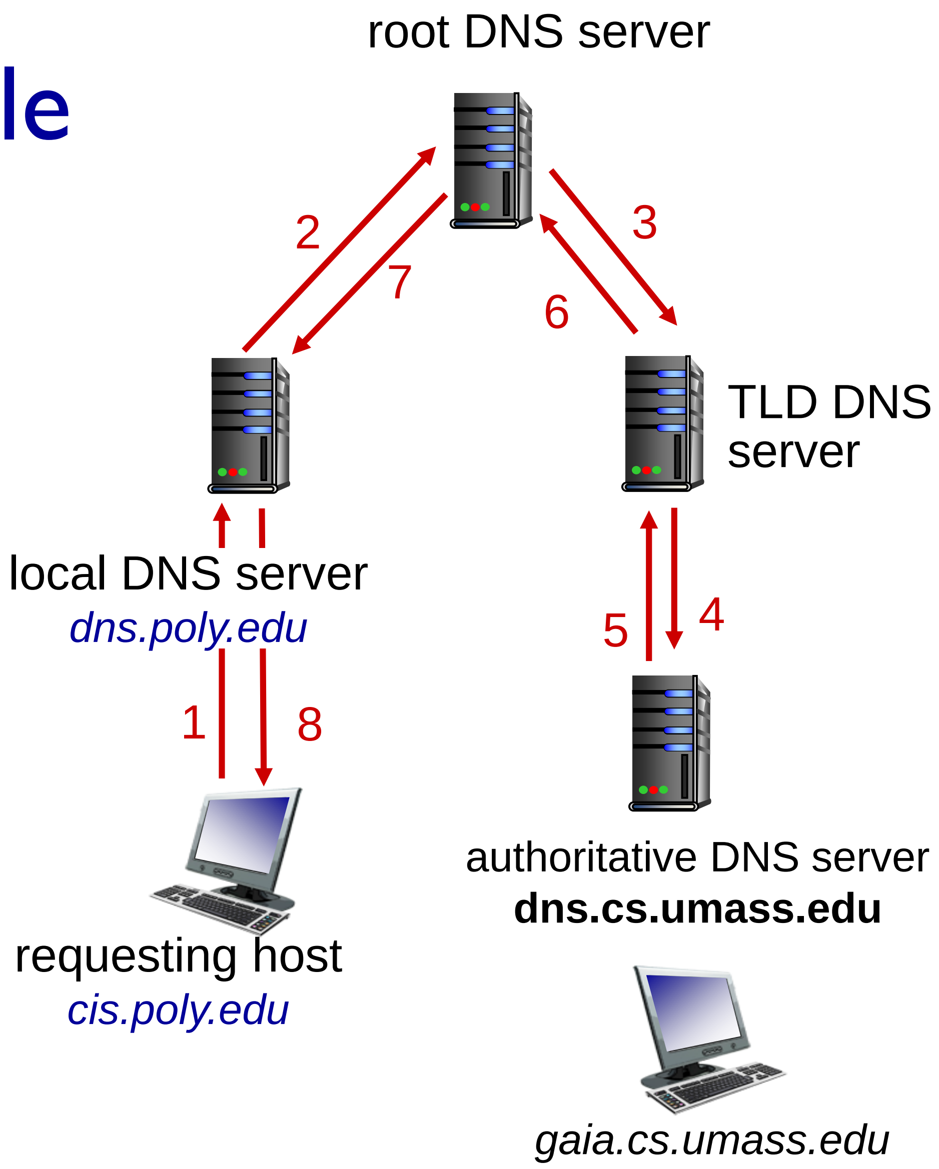
- Contacted server replies with name of server to contact



DNS name resolution example

Recursive Query:

- puts burden of name resolution on contacted name server


root DNS server
2
7
3
6
TLD DNS server
local DNS server
dns.poly.edu
1
8
5
4
authoritative DNS server
dns.cs.umass.edu
requesting host
cis.poly.edu
gaia.cs.umass.edu