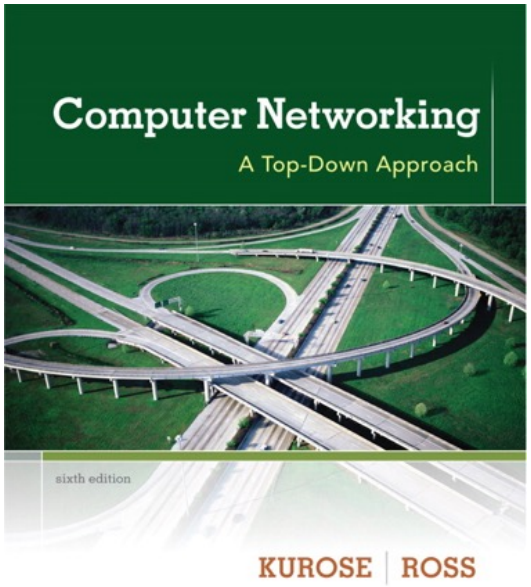
Chapter 2
Application Layer



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

Modified by Prof. Guoliang Xing, Prof. Man Hon Cheung, Prof. Jack Y. B. Lee

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 Socket programming with UDP and TCP

Chapter 2: application layer

our goals:

- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- creating network applications
  - socket API

Some network apps

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

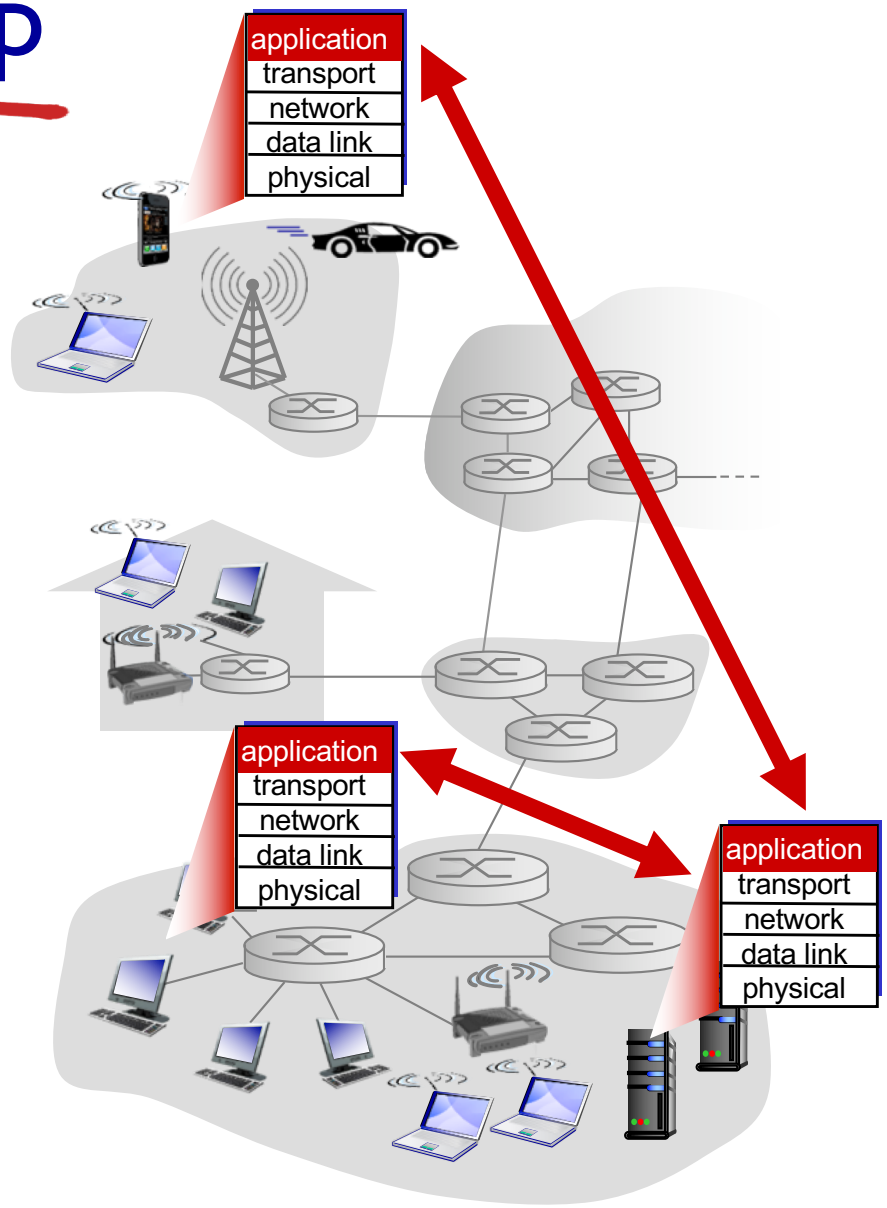
Creating a network app

write programs that:

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation

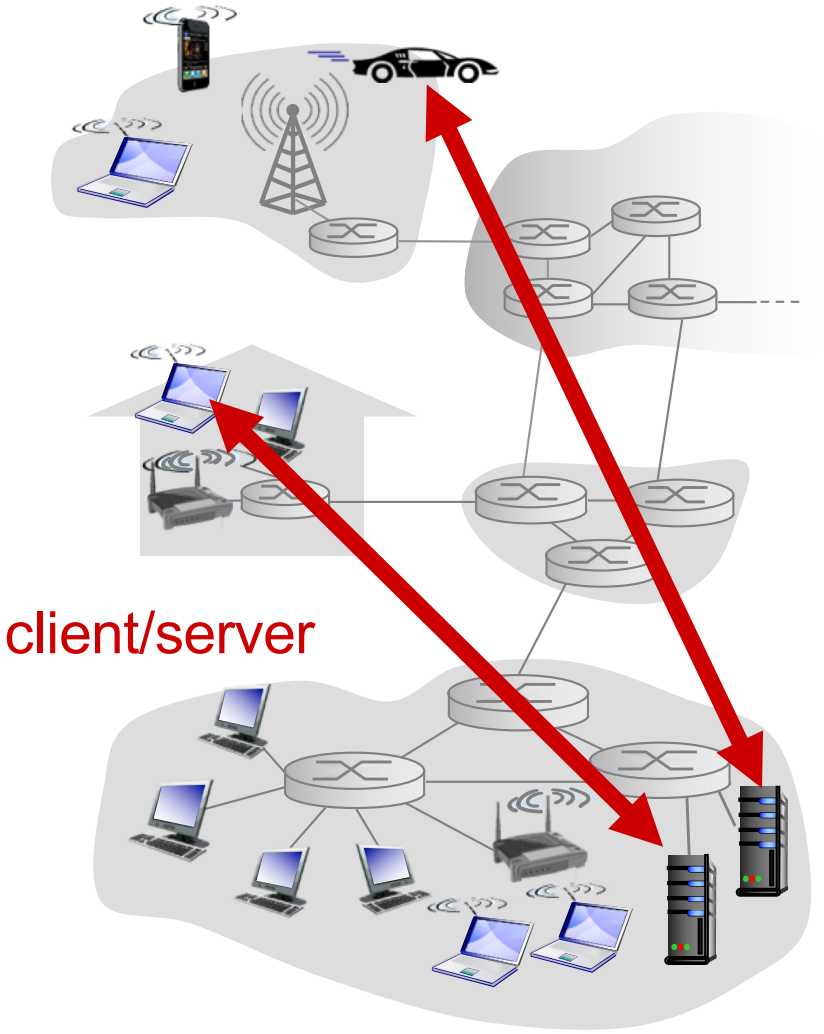


Application architectures

possible structure of applications:

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

Client-server architecture



server:

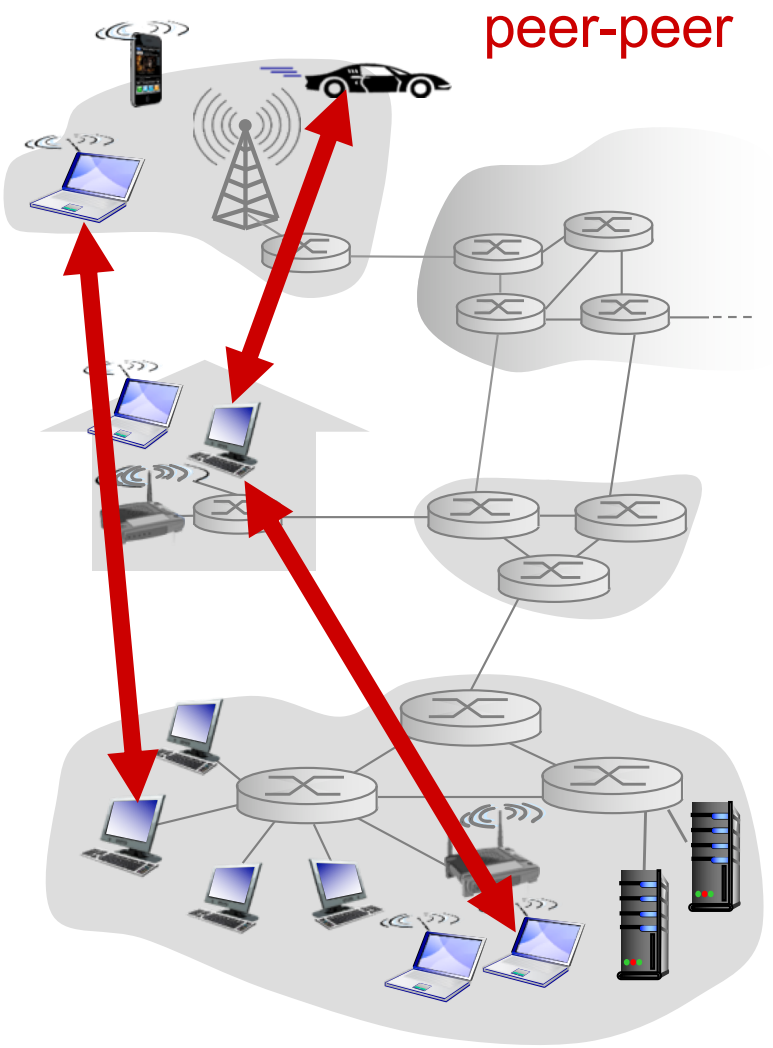
- always-on host
- permanent IP address
- data centers for scaling

clients:

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

P2P architecture

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



Processes communicating

process: program running within a host

Process communications:

- within same host, two processes communicate using inter-process communication defined by operating system
- processes in different hosts communicate by exchanging messages

clients, servers

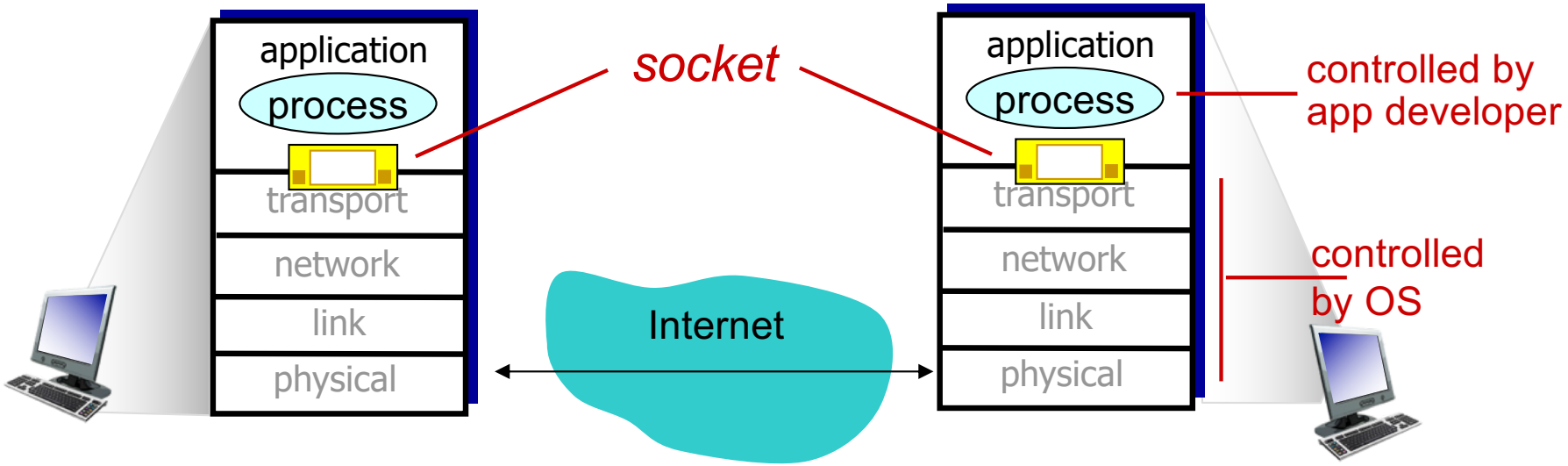
client process: process that initiates communication

server process: process that waits to be contacted

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

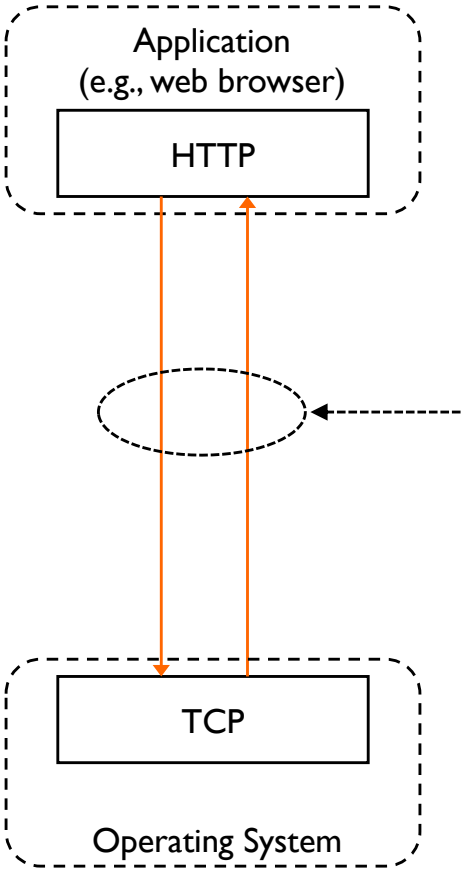
Sockets

- process sends/receives messages to/from the network through a software interface called a socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Sockets

- Accessing transport services via sockets API:



Network Application Programming Interface (API) (e.g., Sockets)

```
// Prepare address structures //
TCP_PeerAddr          = new sockaddr_in;
memset (TCP_PeerAddr, 0, sizeof(struct sockaddr_in));
TCP_PeerAddr->sin_family                      = AF_INET;
TCP_PeerAddr->sin_port                        = htons(0x4747);
TCP_PeerAddr->sin_addr.s_addr                 = inet_addr("137.189.97.120");

// Create a socket and accept for incoming connection //
SOCKET Sockfd = socket(AF_INET, SOCK_STREAM, 0);
int retval = connect(Sockfd, (struct sockaddr *)TCP_PeerAddr,
                     sizeof(struct sockaddr_in));
if (retval == SOCKET_ERROR)
   // connect error
else // (retval == 0)
   // connect successful ...
```

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* no, *many* processes can be running on same host

- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80
- more shortly…

App-layer protocol defines

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

Two main types of app-layer protocols:

1. open protocols:
   - defined in RFCs
   - allows for interoperability
   - e.g., HTTP, SMTP
2. proprietary protocols:
   - Not defined in RFCs or elsewhere
   - A single development team has complete control over what goes in the code
   - e.g., Skype

What transport service does an app need?

data integrity

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

timing

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

throughput

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

security

- encryption, data integrity, …

Transport service requirements: common apps

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

Internet transport protocols services

TCP service:

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

UDP service:

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

Securing TCP

Problem of TCP & UDP: no encryption

- cleartext passwords sent into socket traverse Internet in cleartext

Solution: Secure Sockets Layer (SSL)

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

SSL is at app layer

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

SSL socket API

- cleartext passwords sent into socket traverse Internet encrypted
- See Chapter 7

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

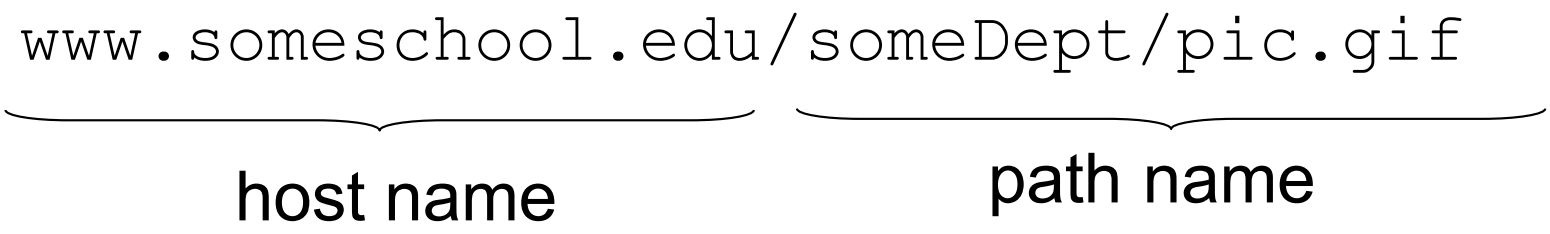
2.6 P2P applications

2.7 Socket programming with UDP and TCP

Web and HTTP

First, a review…

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



HTTP overview

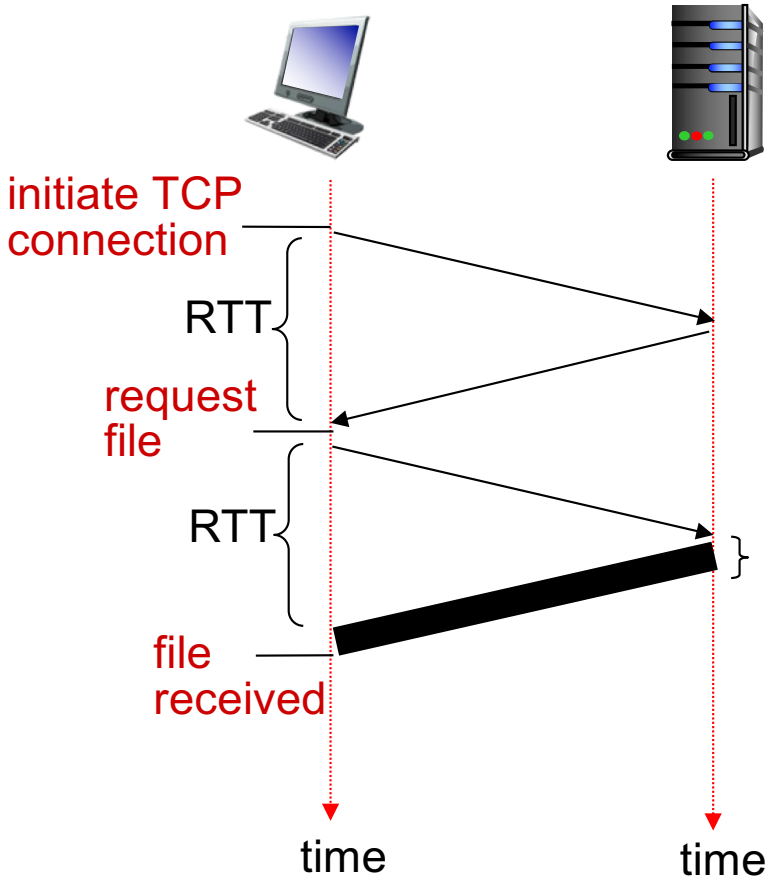
HTTP: hypertext transfer protocol

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

HTTP overview (continued)

uses TCP:

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



HTTP overview (continued)

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

Question: should each request/response pair be sent over a separate or the same TCP connection?

non-persistent HTTP

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

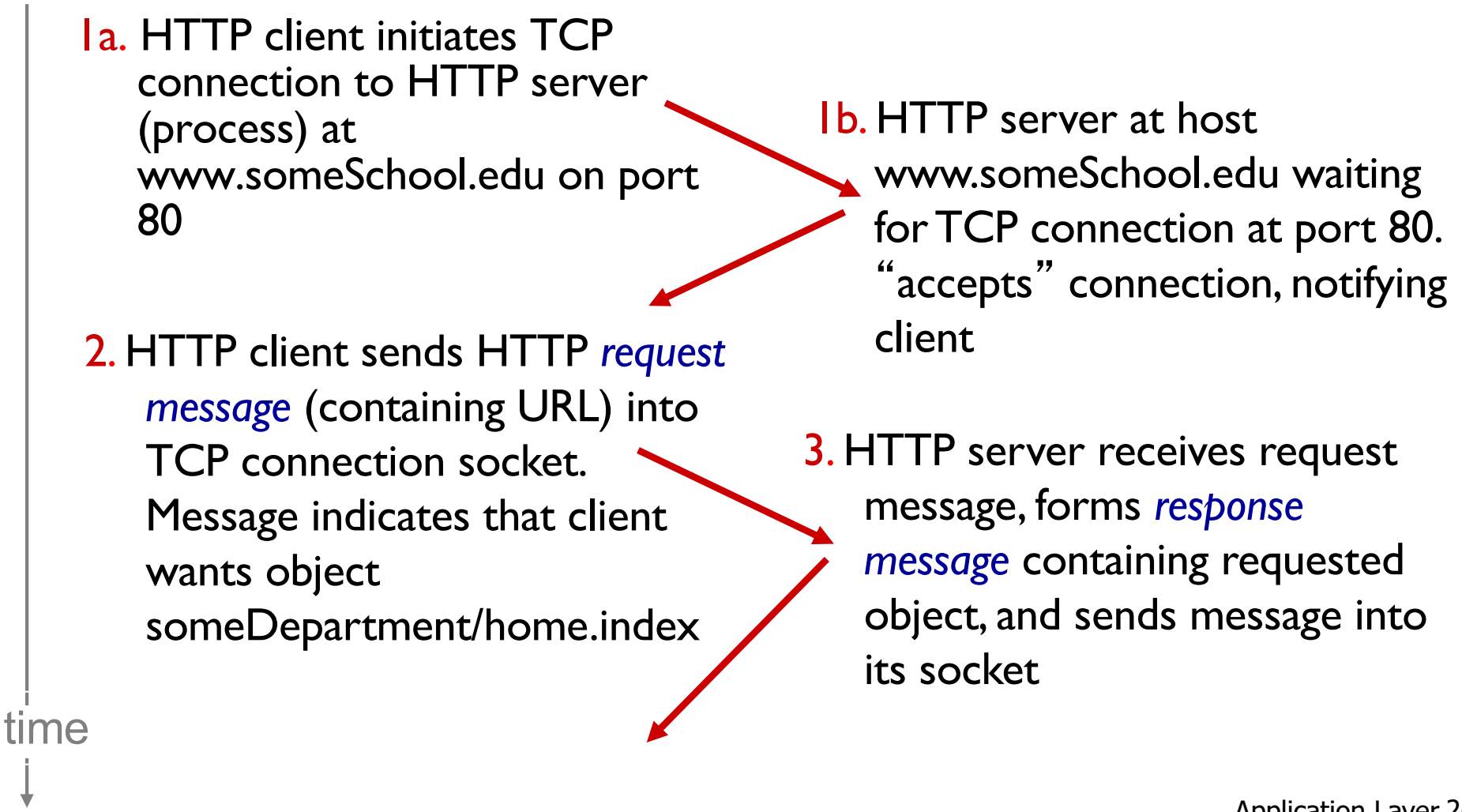
persistent HTTP

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

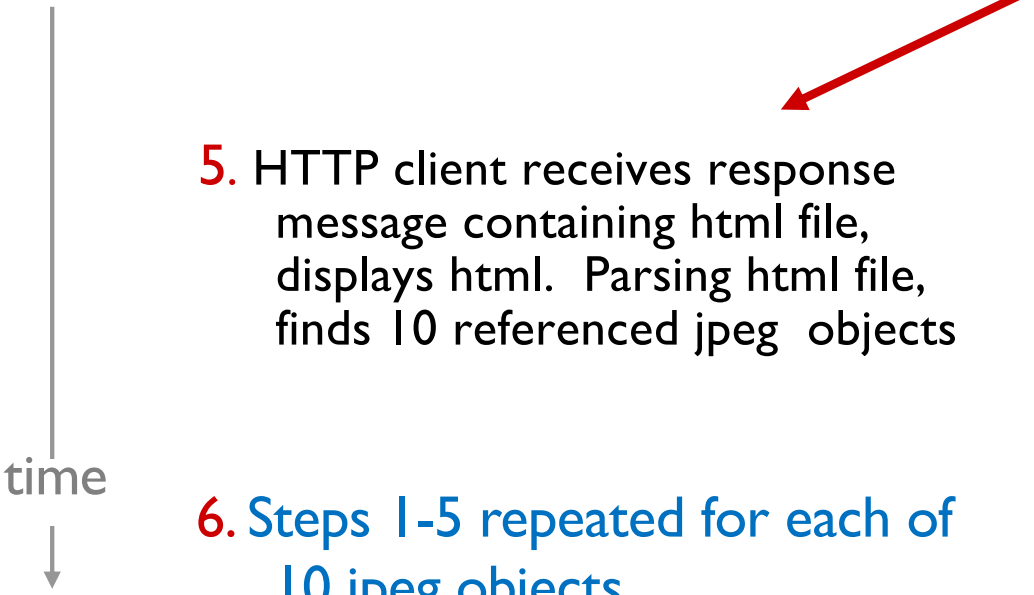
Non-persistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)



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

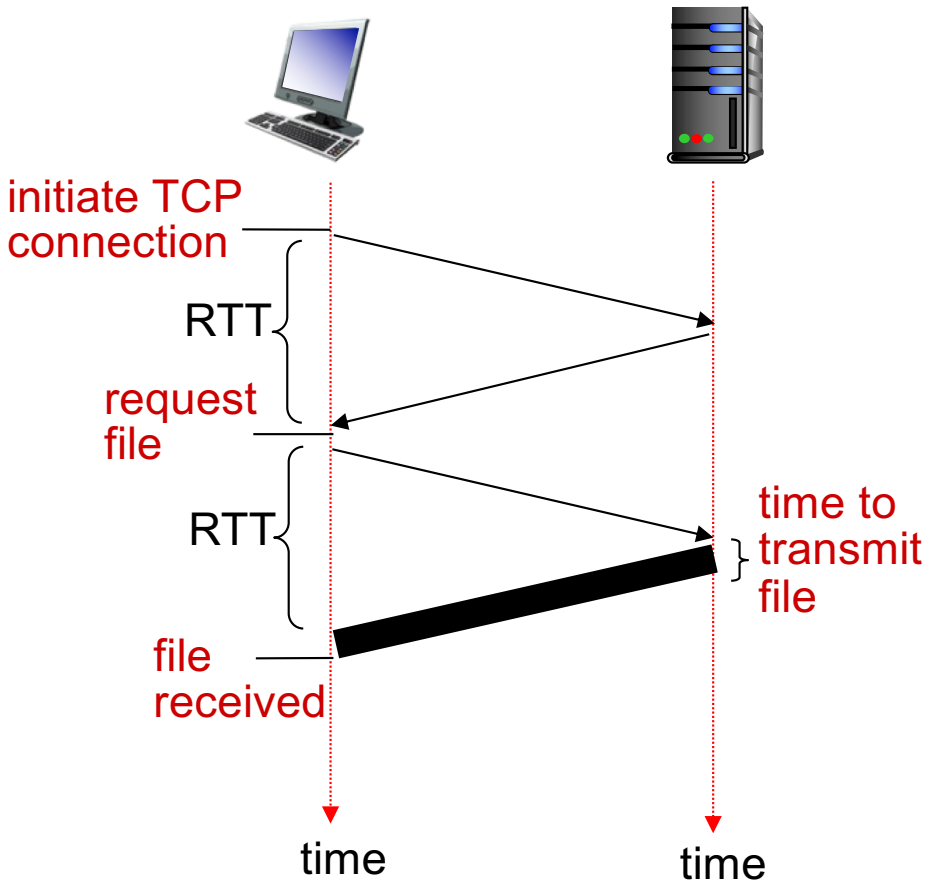
Non-persistent HTTP: response time

Round-trip time (RTT): 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



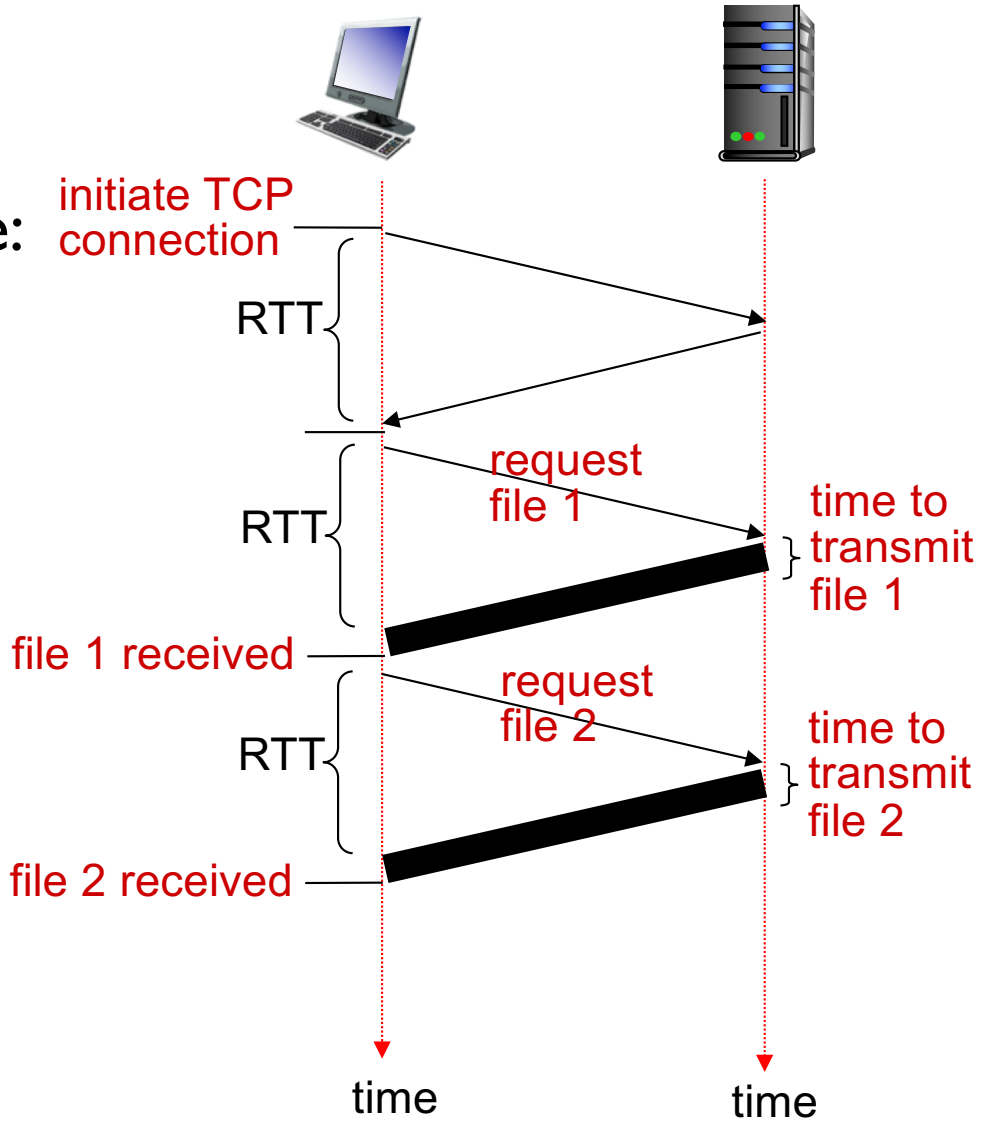
Persistent HTTP: response time

First request is same as non-persistent HTTP

Subsequent HTTP response time:

- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- persistent HTTP response time =

  RTT+ file transmission time



Non-Persistent & 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

Concurrent HTTP

- What?
  - Not part of the HTTP specification
  - Implemented by almost all web browsers
  - After the initial HTTP connection which retrieves the HTML body, initiate multiple (4~6) HTTP sessions (per domain) to retrieve multiple objects (e.g., images) in parallel.
- Why?
  - Reduce delay in waiting for previous HTTP response to complete transfer (aka head-of-line blocking)
  - Potentially overlaps the server-side processing time of multiple HTTP requests
  - Effectively multiplies the TCP congestion window size by the number of TCP connections, resulting in higher aggregate throughput
- Shortcomings:
  - Use up more resources at server
  - Fairness to competing data flows

User-server state: cookies

Q: As HTTP server is stateless, how can a Web site identify users?

A: Cookies, defined in [RFC 6265], allow sites to keep track of users.

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 (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":

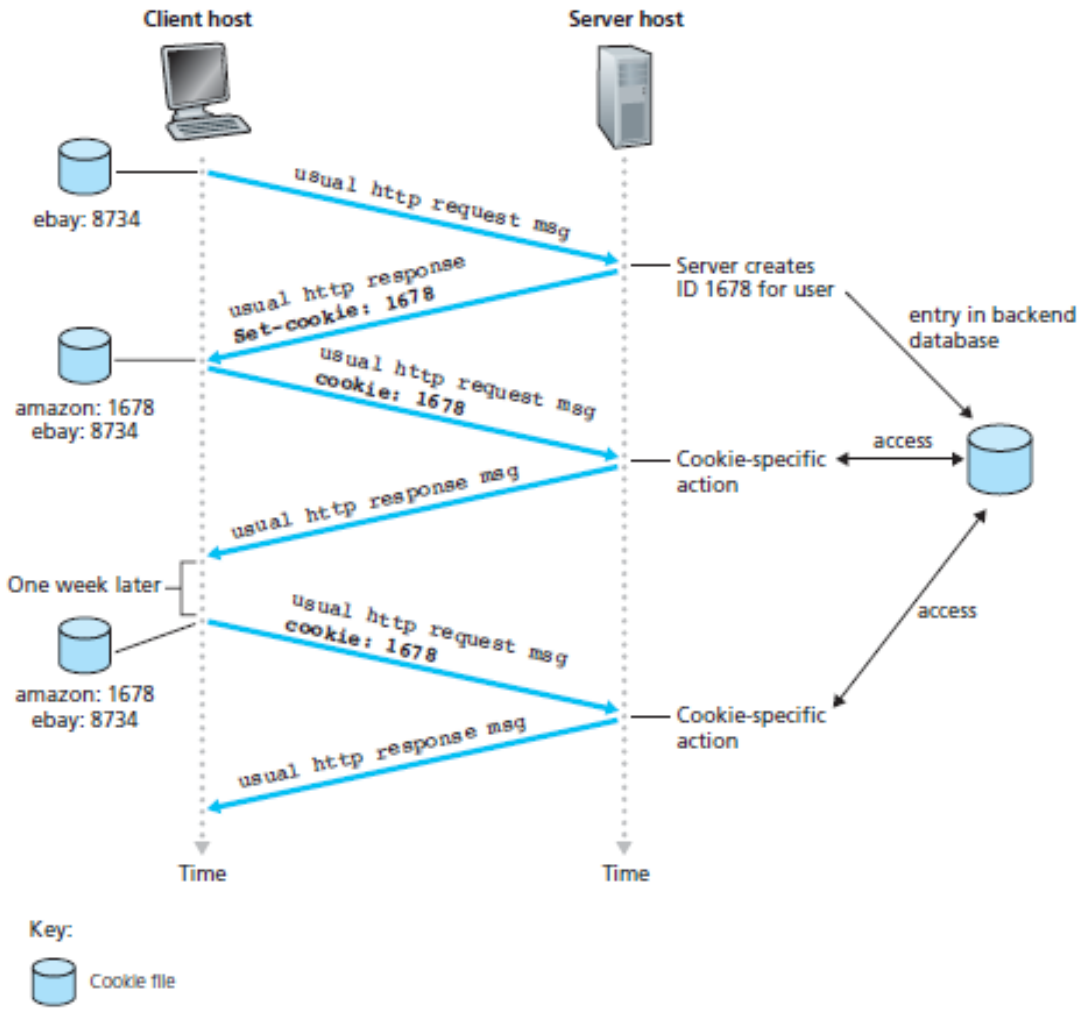
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

User-server state: cookies

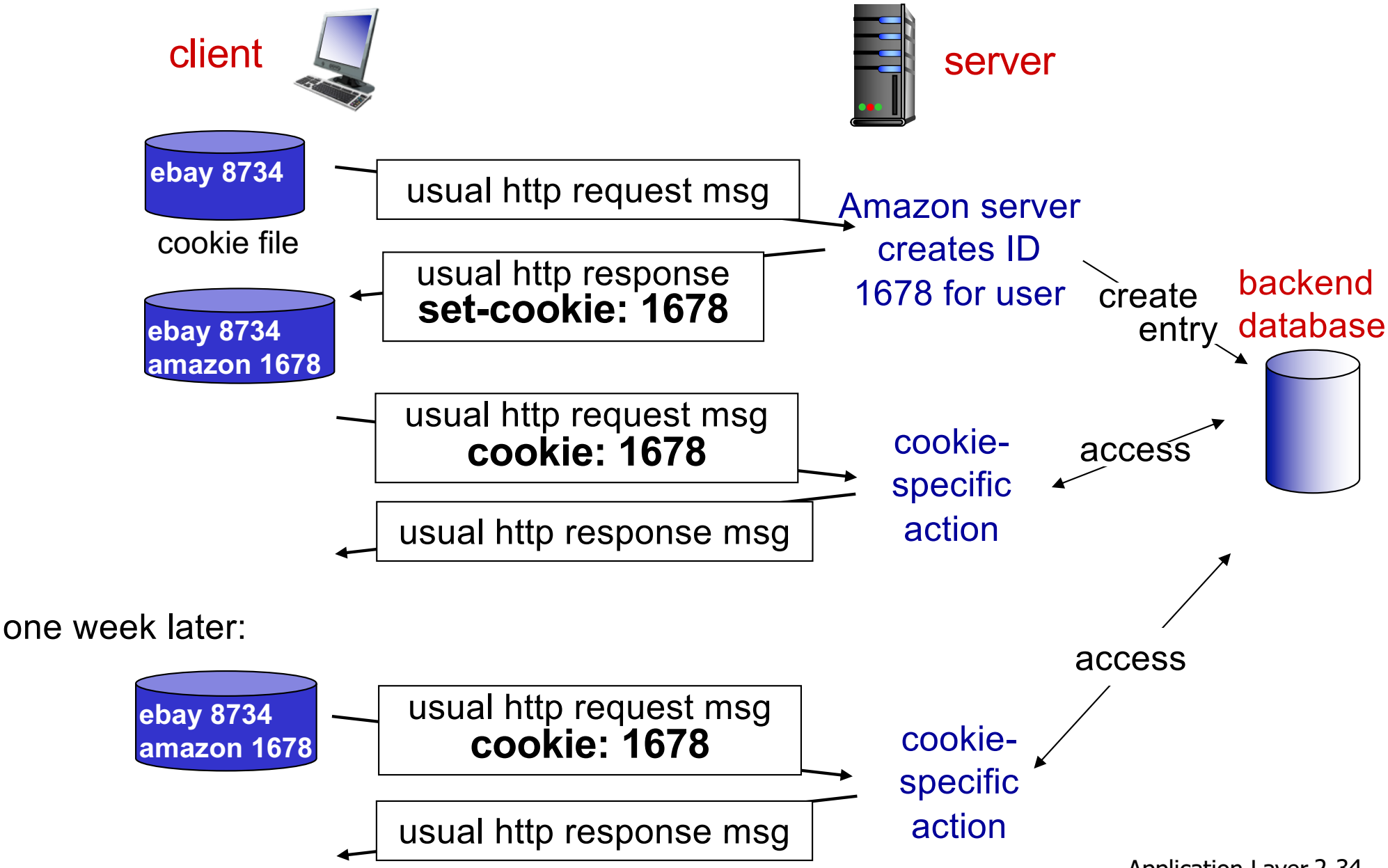
many Web sites use cookies

four components:

1) cookie header line of HTTP *response* message
2) cookie header line in next HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) backend database at Web site



Cookies: keeping "state" (cont.)



Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

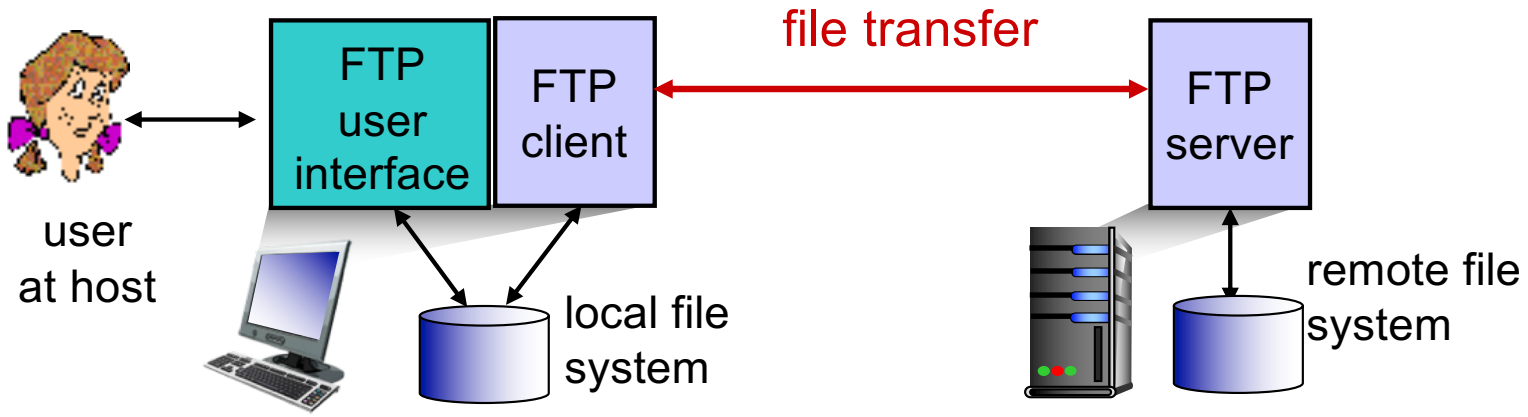
2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 Socket programming with UDP and TCP

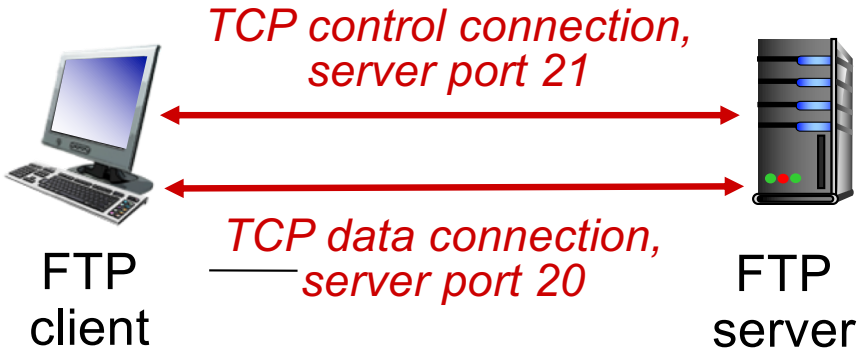
FTP: the file transfer protocol



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

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, using TCP
- Control connection (port 21):
  - authentication
  - client browses remote directory, sends commands
- Data connection (port 20):
  - when server receives file transfer command, *server* opens 2nd TCP data connection (for file) *to* client
  - after transferring one file, server closes data connection



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

FTP commands, responses

sample commands:

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

sample return codes

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

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 Socket programming with UDP and TCP

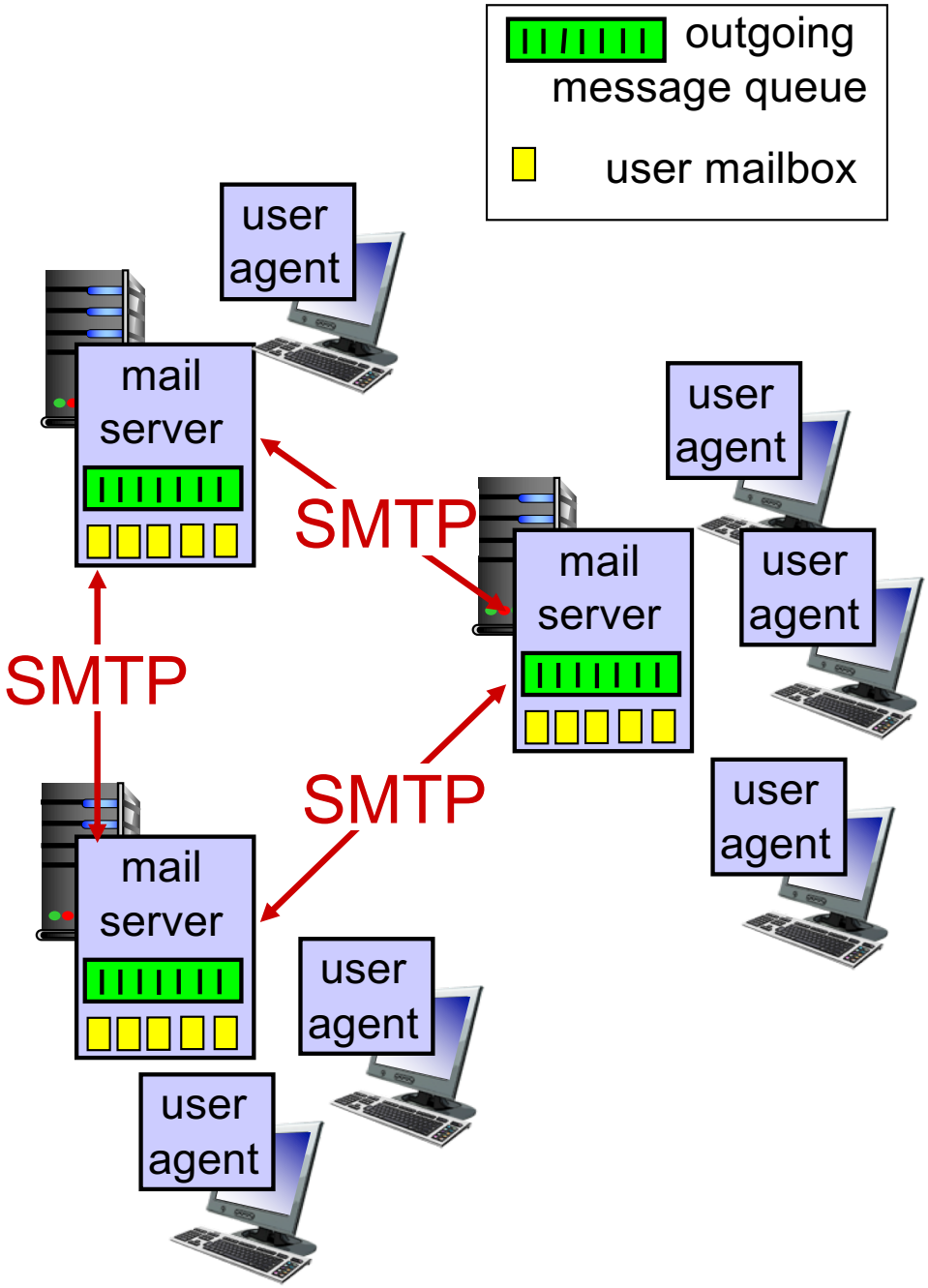
Electronic mail

Three major components:

1. user agents
2. mail servers
3. simple mail transfer protocol: SMTP

User Agent

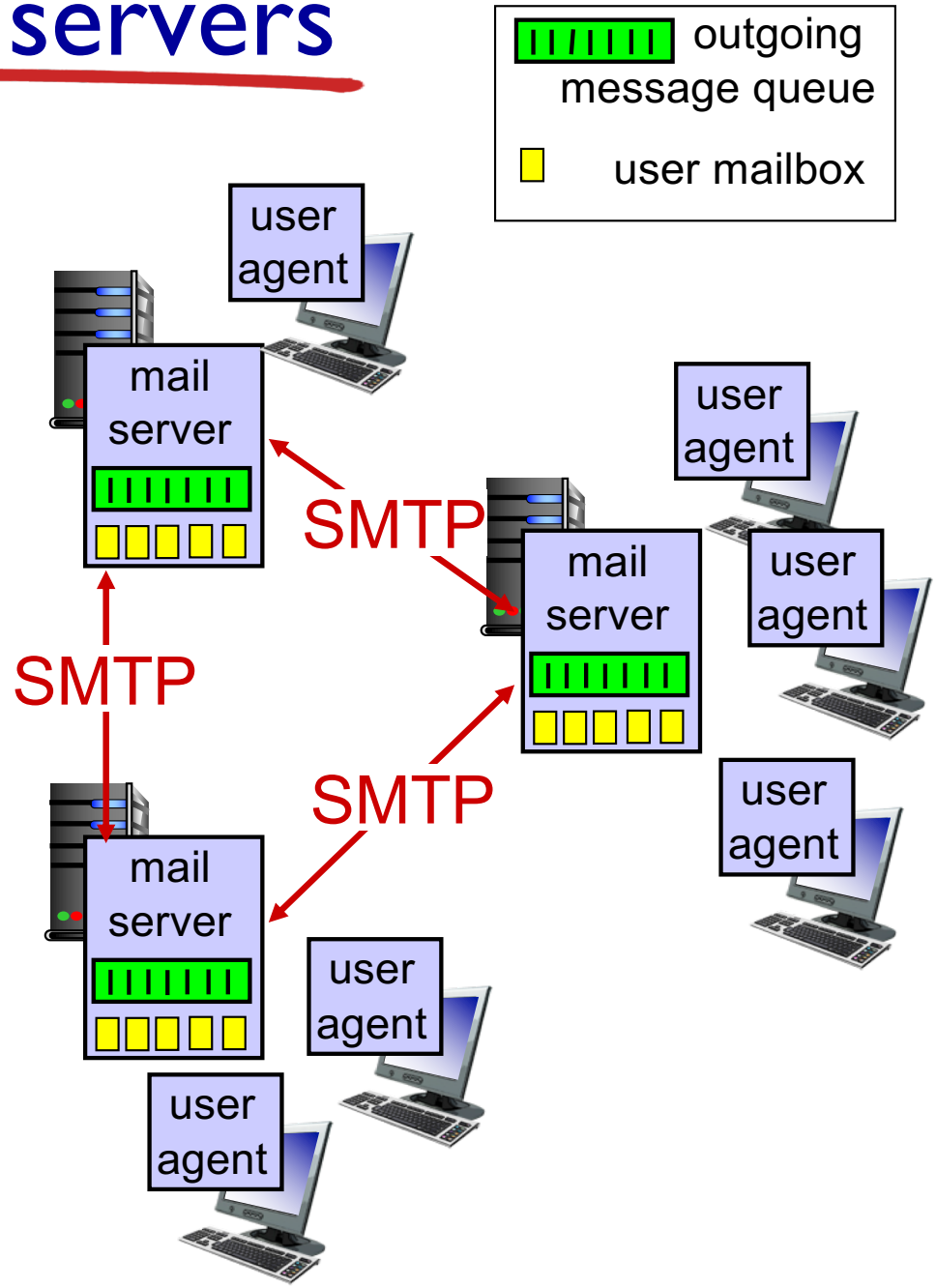
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client



Electronic mail: mail servers

mail servers:

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



Electronic Mail: SMTP [RFC 2821]

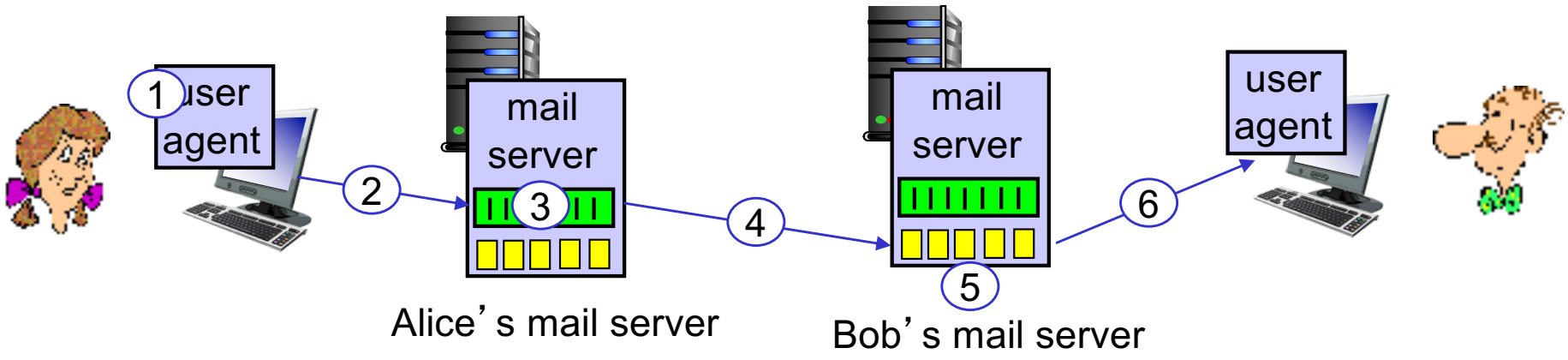
- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP, FTP)
  - commands: ASCII text
  - response: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

Sample SMTP interaction

```
S: 220 hamburger.edu (server's host name)
C: HELO crepes.fr    (client's host name)
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr> (beginning of message)
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: .                                  (end of message)
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

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



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.

comparison with HTTP:

- HTTP: pull protocol (pull the information from the server)
  SMTP: push protocol (sending mail server pushes the file to the receiving mail server)
- Both have ASCII command/response interaction, status codes

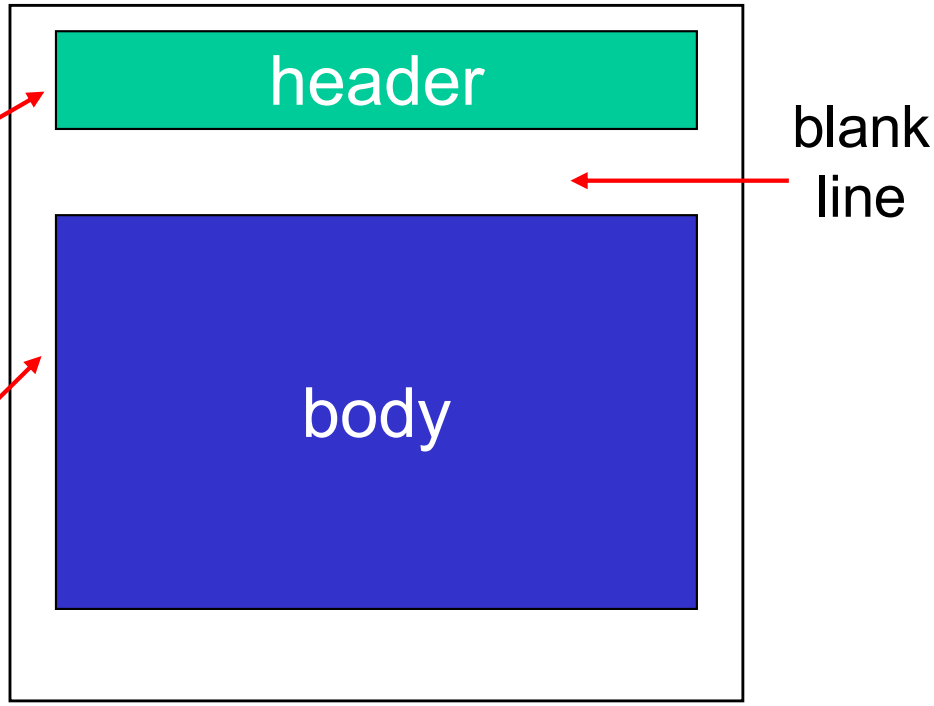
Mail message format

SMTP: protocol for exchanging email msgs

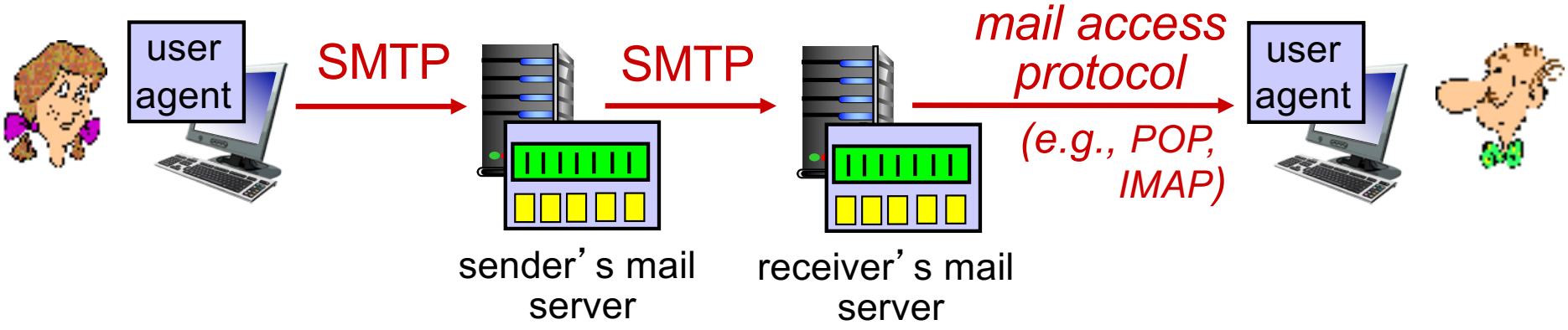
RFC 822: standard for text message format:

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

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



Mail access protocols



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

POP3 protocol

- POP3 is an extremely simple mail access protocol
  - its functionality is rather limited.
- POP3 begins when the user agent (the client) opens a TCP connection to the mail server (the server) on port 110.
- With the TCP connection established, POP3 progresses through three phases:
  - Authorization
  - Transaction
  - Update

POP3 protocol

authorization phase

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

transaction phase, client:

- **`list`**: list the size of each of the stored messages
- **`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 2 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

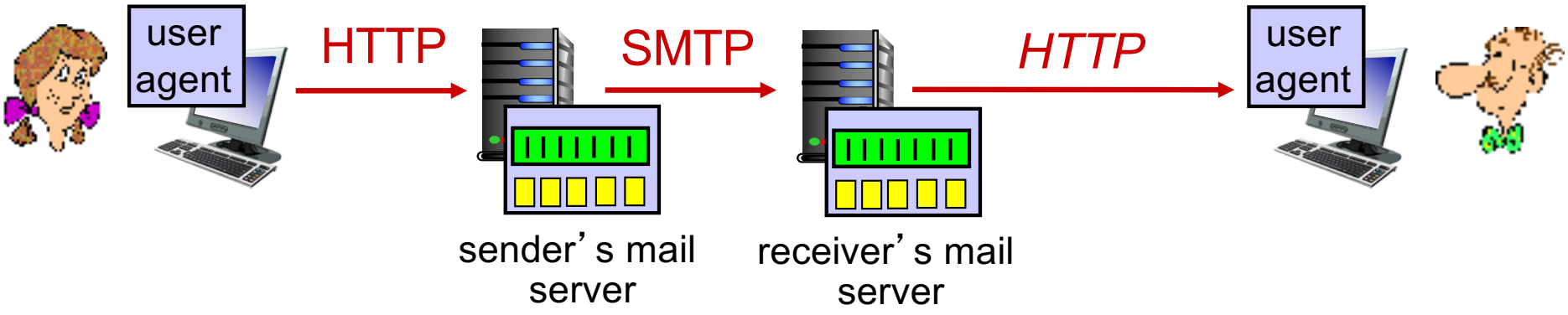
POP3 protocol

- Update phase: Remove messages from mailbox in this example
- Two Modes:
  - "download and delete": the user agent deletes the messages on the mail server after downloading them
  - "download and keep": the user agent leaves the messages on the mail server after downloading them
- POP3 is stateless across sessions.

IMAP protocol

- 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

Web-Based E-mail



- User agent: Web browser
- User communicates with remote mailbox via HTTP:
  - Send e-mail: e-mail message is sent from browser to mail server over HTTP (not SMTP)
  - Mail server message exchange: Still using SMTP
  - Access message in mailbox: e-mail message is sent from mail server to browser using the HTTP (not POP3 or IMAP)

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 Socket programming with UDP and TCP

DNS: domain name system

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

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

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

Domain Name System (DNS):

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

DNS: services

- hostname to IP address translation
  - Hostname: mnemonic identifier for a host
    - E.g., www.yahoo.com
- host aliasing
  - E.g., canonical hostname: relay1.west-coast.enterprise.com
  - Two alias: enterprise.com and www.enterprise.com (more mnemonic)
- mail server aliasing
  - E.g., E-mail address: bob@hotmail.com
  - Alias hostname: hotmail.com
  - Canonical hostname: relay1.west-coast.hotmail.com
  - Note: a company's mail server and Web server can have identical (aliased) hostnames, e.g., enterprise.com.
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

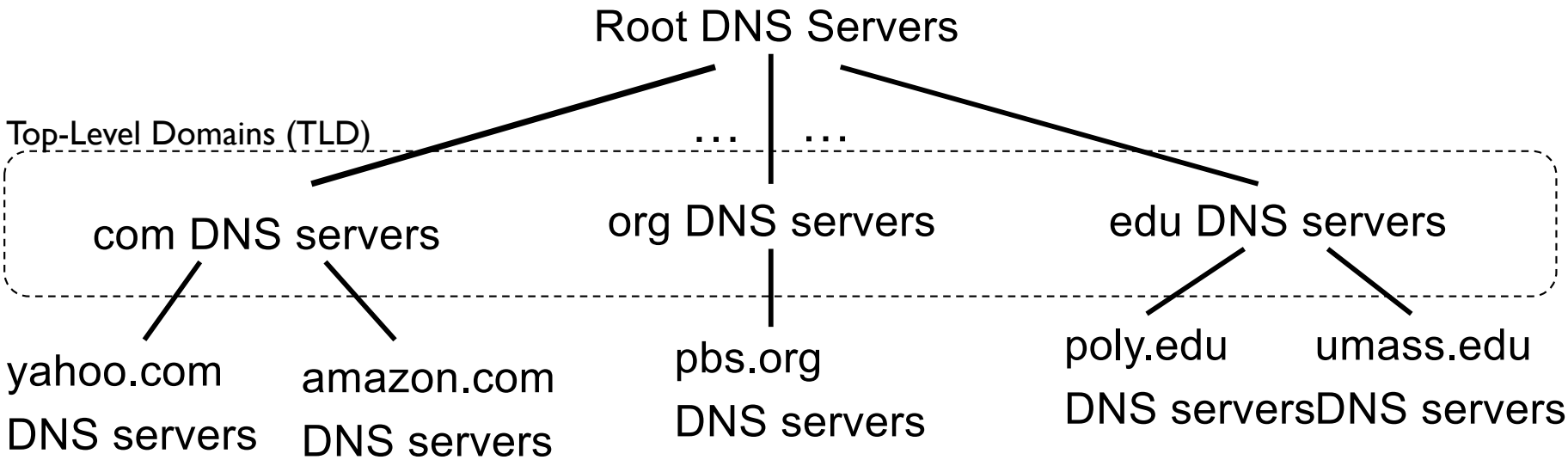
DNS structure

why not centralize DNS?

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

A: doesn't scale!

DNS: a distributed, hierarchical database



Principle

- parent server has IP-name records for children

To resolve IP for www.amazon.com; 1st approx:

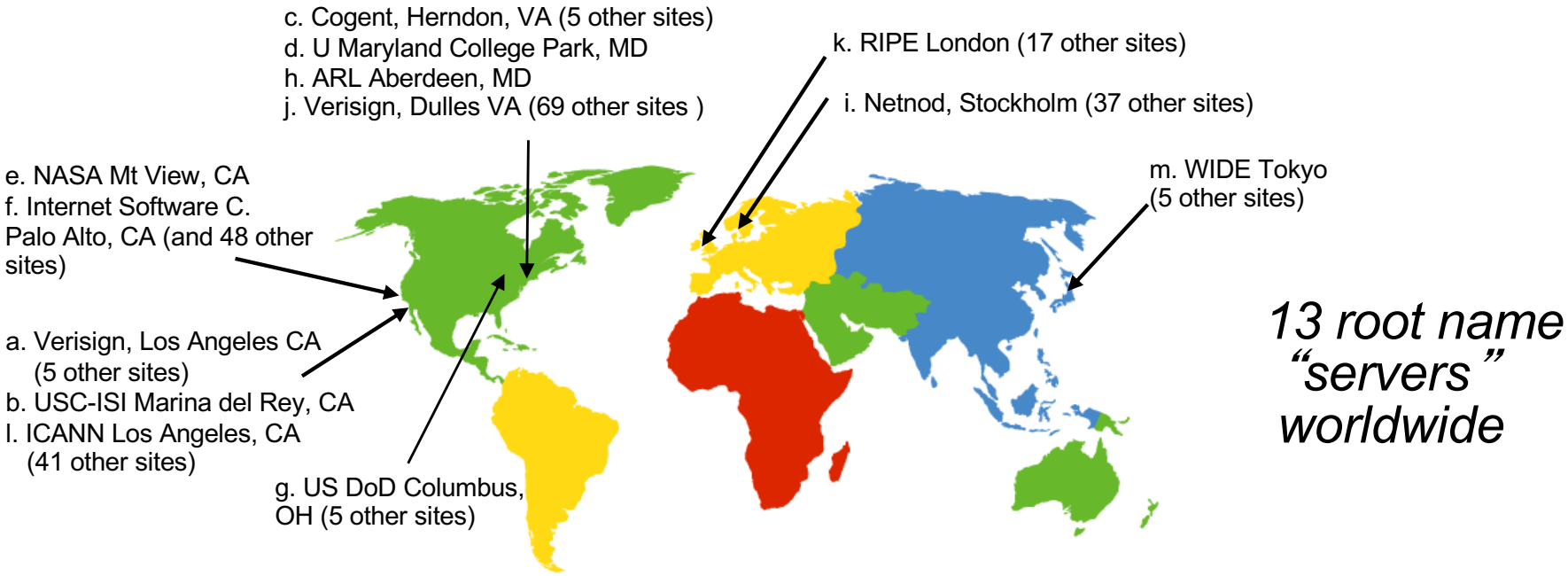
- queries root server to find .com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known

  → gets mapping

  → returns mapping to local name server



13 root name "servers" worldwide

TLD, authoritative servers

Top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- further delegates query to authoritative DNS servers

Authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts (e.g., *.cuhk.edu.hk), known as a DNS zone
- can be maintained by organization or service provider
- can further sub-divide into smaller DNS zones each with their own authoritative DNS server (e.g., *.ie.cuhk.edu.hk)

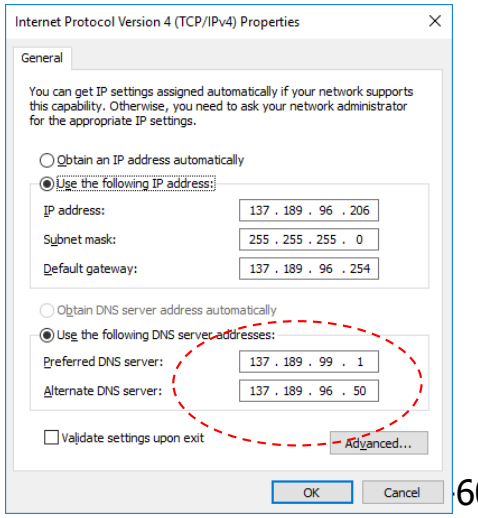
Local DNS name server

- 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

```
C:\>ipconfig /all

Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) Ethernet Connection I217-LM
   Physical Address. . . . . . . . . : 64-00-6A-5F-70-A3
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   IPv4 Address. . . . . . . . . . . : 137.189.96.206(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 137.189.96.254
   DHCPv6 IAID . . . . . . . . . . . : 56885354
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-1D-96-3D-F9-64-00-6A-5F-
70-A3
   DNS Servers . . . . . . . . . . . : 137.189.99.1
                                       137.189.96.50
```

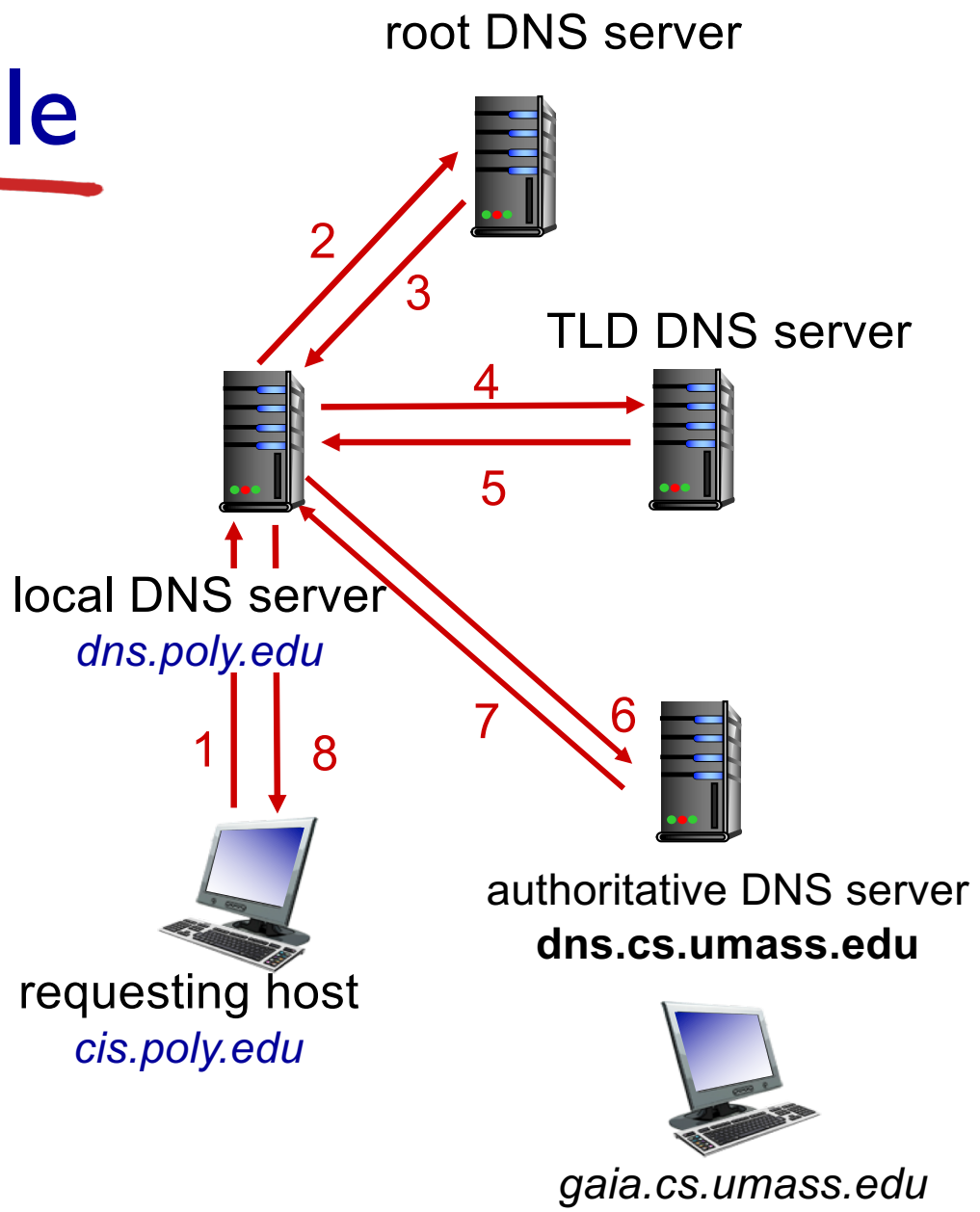


DNS name resolution example

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

Recursive query:

- puts burden of name resolution on contacted name server
- E.g., (1,8)



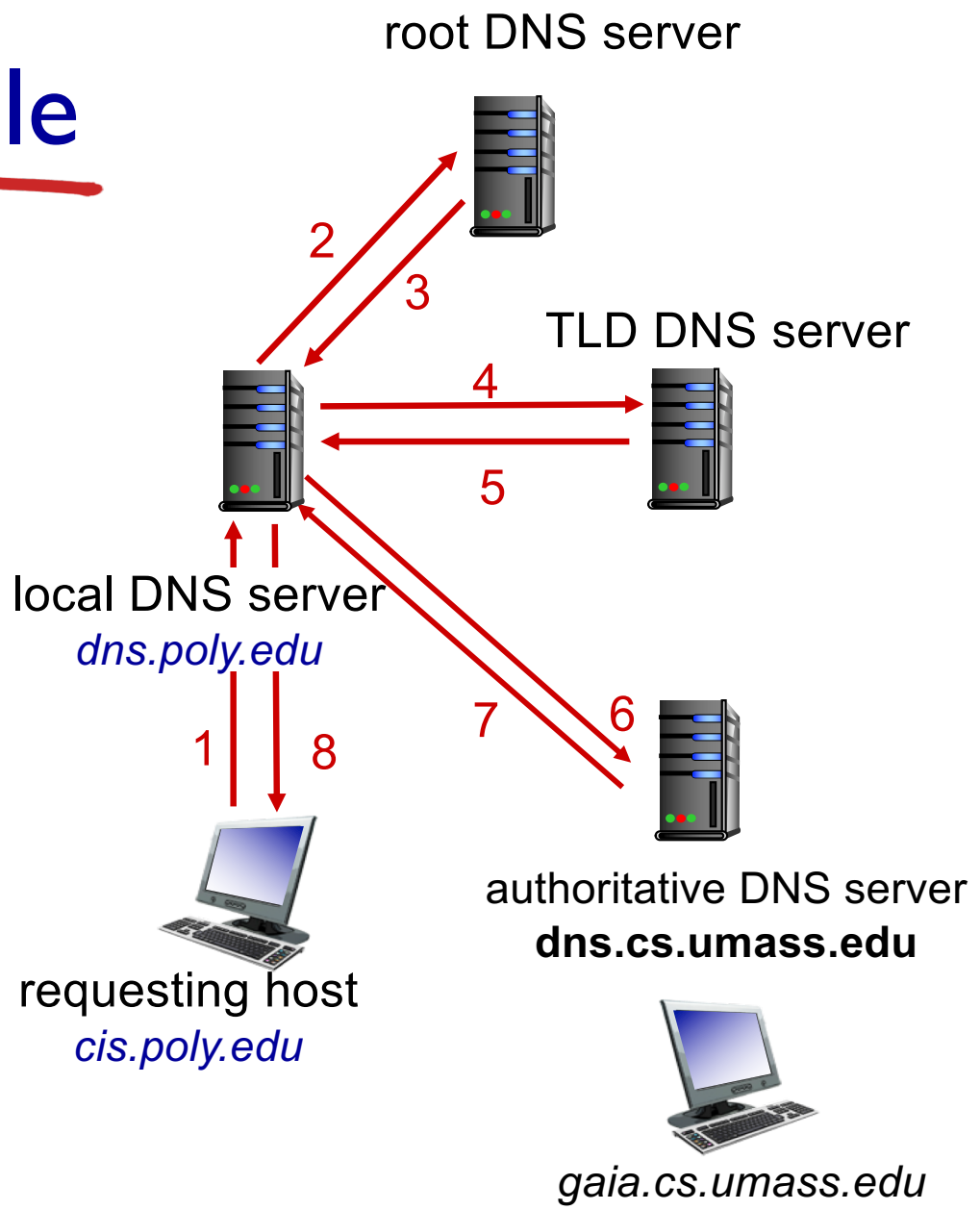
DNS name resolution example

Iterative query:

- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"
- E.g., (2,3), (4,5), (6,7)

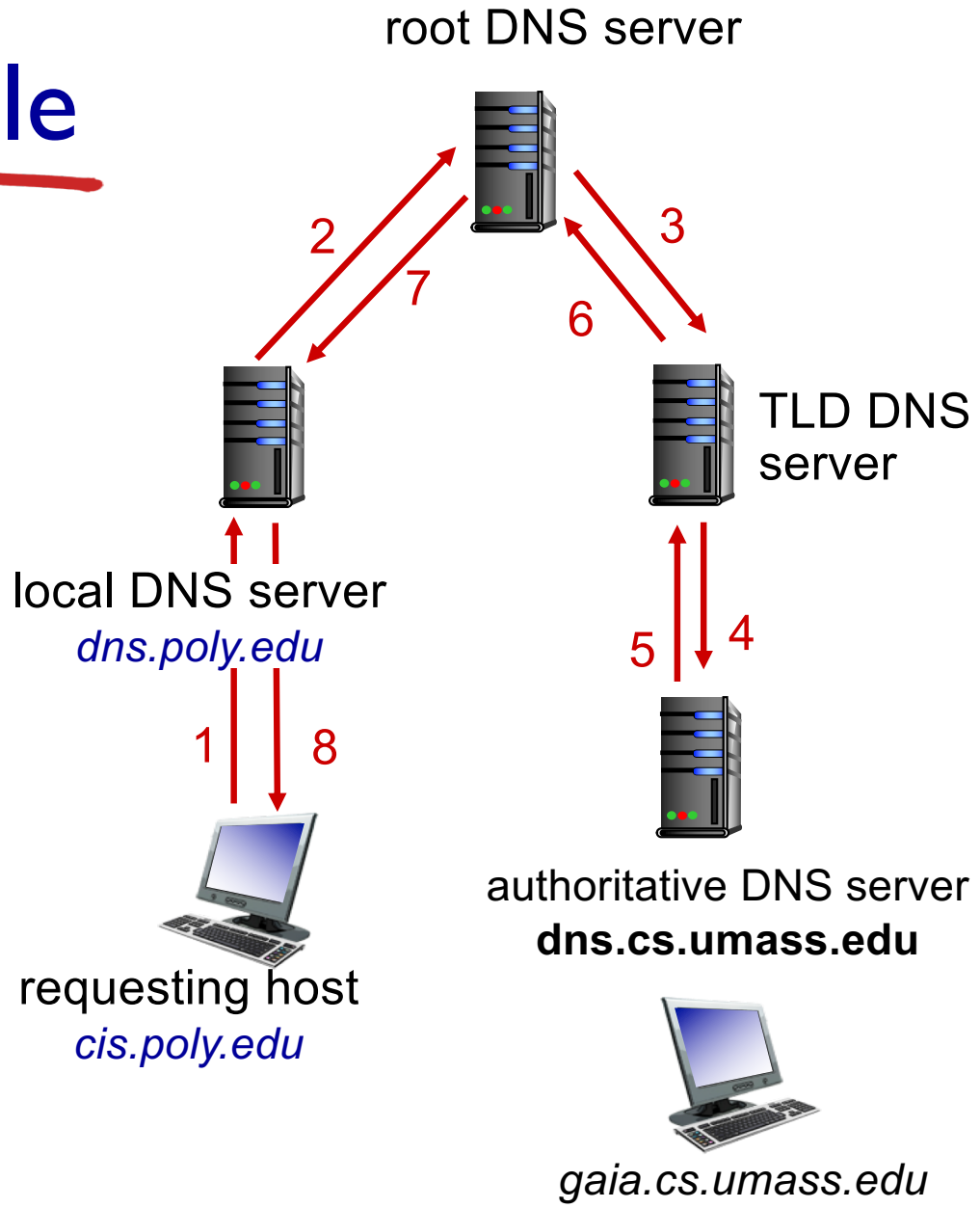
Typical query pattern:

- query from the requesting host to the local DNS server is recursive
- remaining queries are iterative.



DNS name resolution example

- Another possible query pattern: All the queries are recursive
- heavy load at upper levels of hierarchy



DNS caching

- once (any) name server learns mapping, it *caches* it
  - Adv: improve delay performance and to reduce the number of DNS messages ricocheting around
  - Query for same hostname: DNS server can provide desired IP address, even if it is not authoritative
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - cache entries removed after time to live (TTL)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard
  - RFC 2136

DNS records

DNS: distributed database storing resource records (RR)

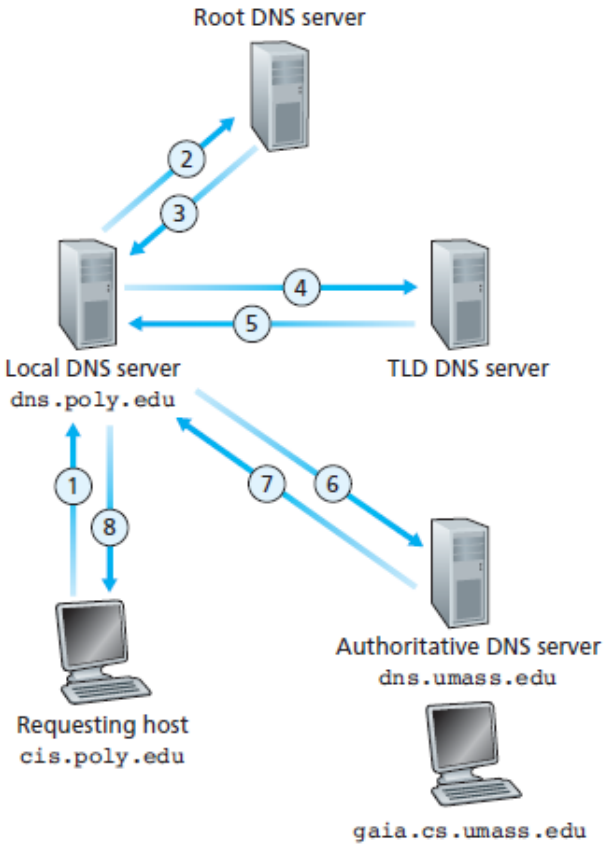
RR format: `(name, value, type, ttl)`

type=A

- **`name`** is hostname (e.g., relay1.bar.foo.com)
- **`value`** is IP address

type=NS

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



DNS records (cont'd)

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

type=CNAME

- **`name`** is alias name
- **`value`** is canonical (the real) name
- E.g., (foo.com, relay1.bar.foo.com, CNAME, ttl)

type=MX

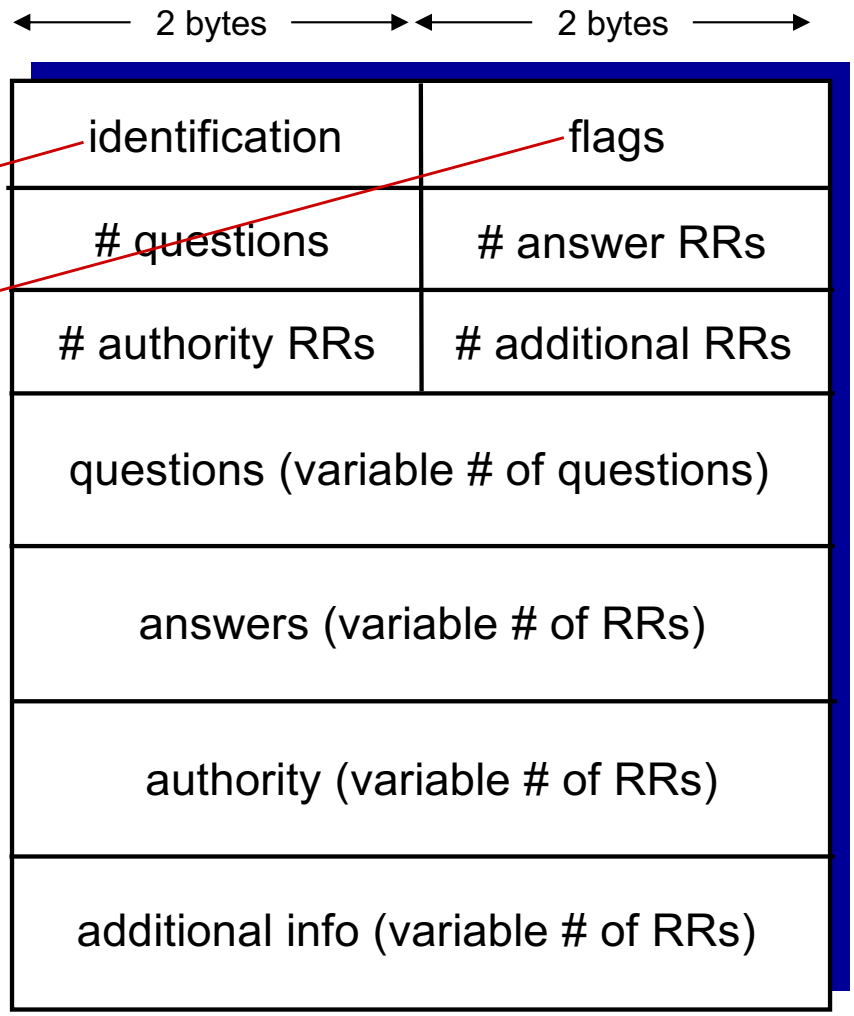
- **`value`** is name of mail server associated with **`name`**
- E.g., (foo.com, mail.bar.foo.com, MX, ttl)

DNS protocol, messages

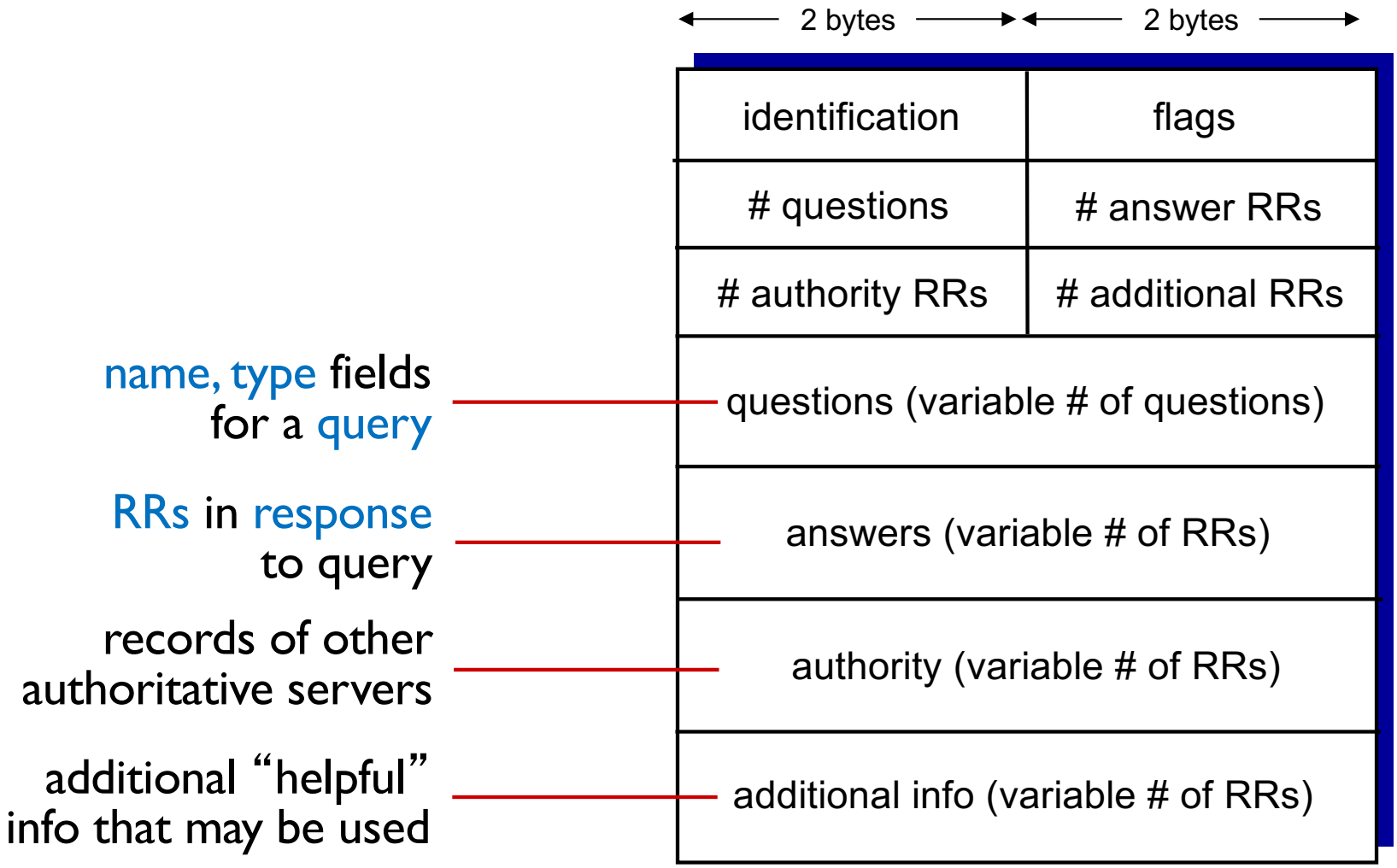
- *query* and *reply* messages, both with same *message format*

msg header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired by client (if DNS server does not have the record)
  - recursion supported
  - reply is authoritative



DNS protocol, messages



Inserting records into DNS

- example: new startup "Network Utopia"
- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative 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 RRs at your authoritative server
  - type A record for Web server www.networkuptopia.com

Attacking DNS

DDoS attacks

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

Redirect attacks

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

Exploit DNS for DDoS

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

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

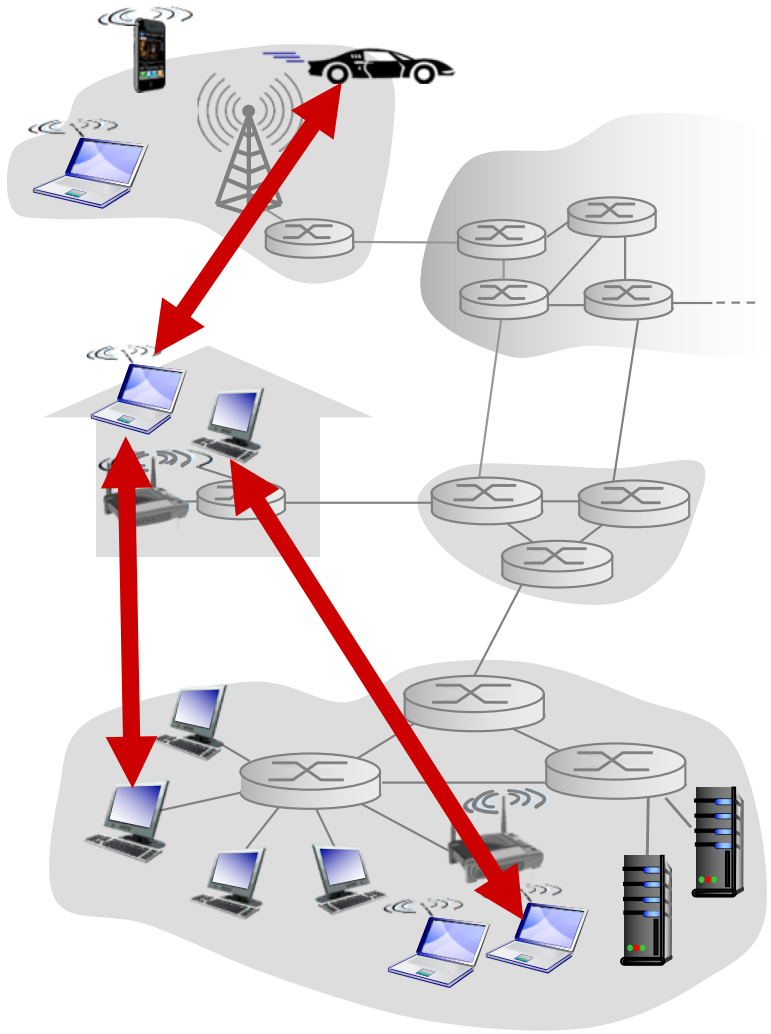
2.7 Socket programming with UDP and TCP

Pure P2P architecture

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

examples:

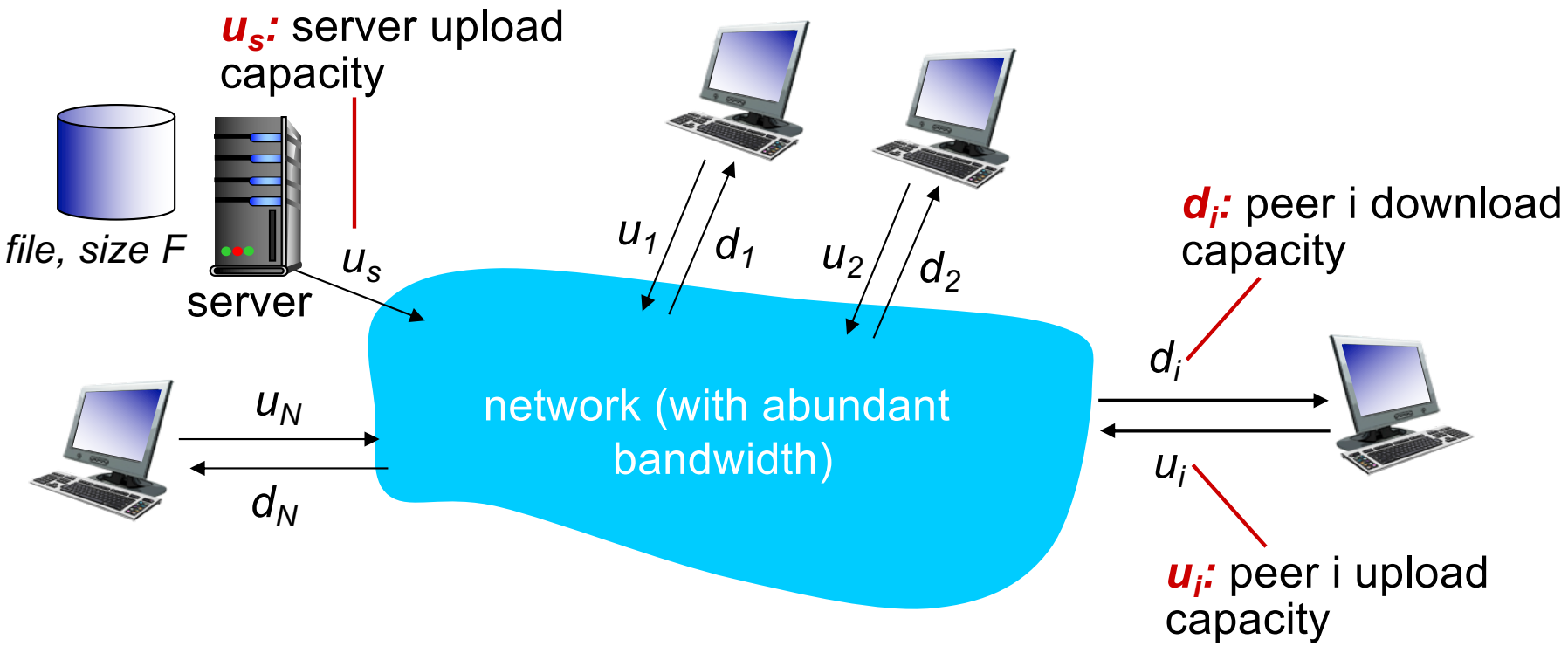
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



File distribution: client-server vs P2P

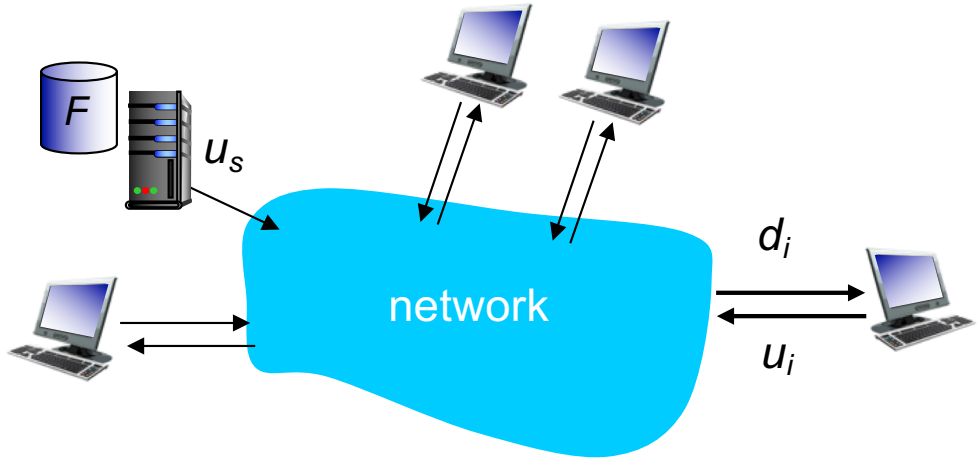
Question: how much time to distribute file (size *F*) from one server to *N* peers?

- peer upload/download capacity is limited resource



File distribution time: client-server

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



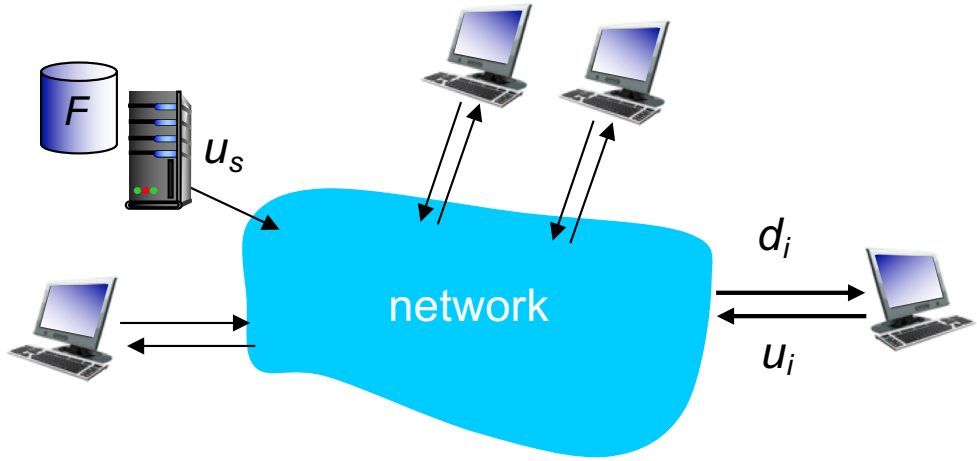
time to distribute F to N clients using client-server approach

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

increases linearly in N

File distribution time: P2P

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



time to distribute F to N clients using P2P approach

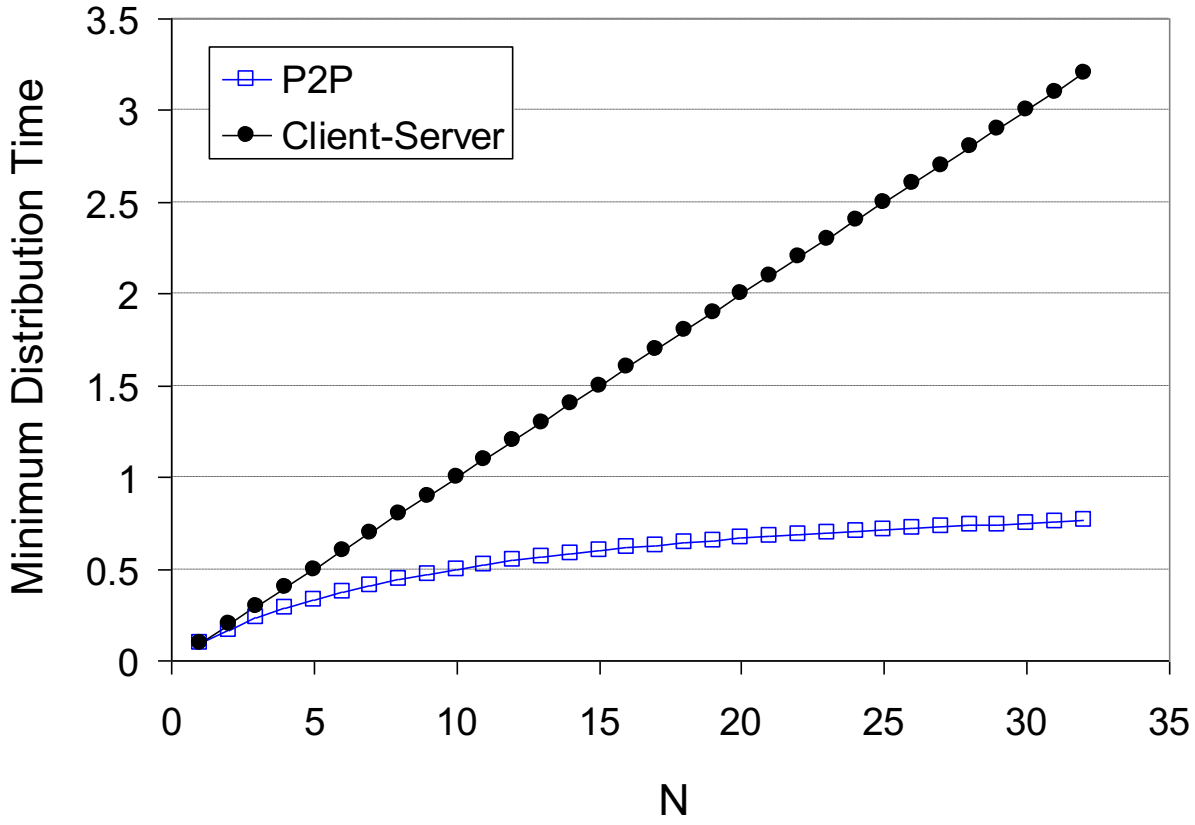
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

increases linearly in *N* …

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

Client-server vs. P2P: example

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



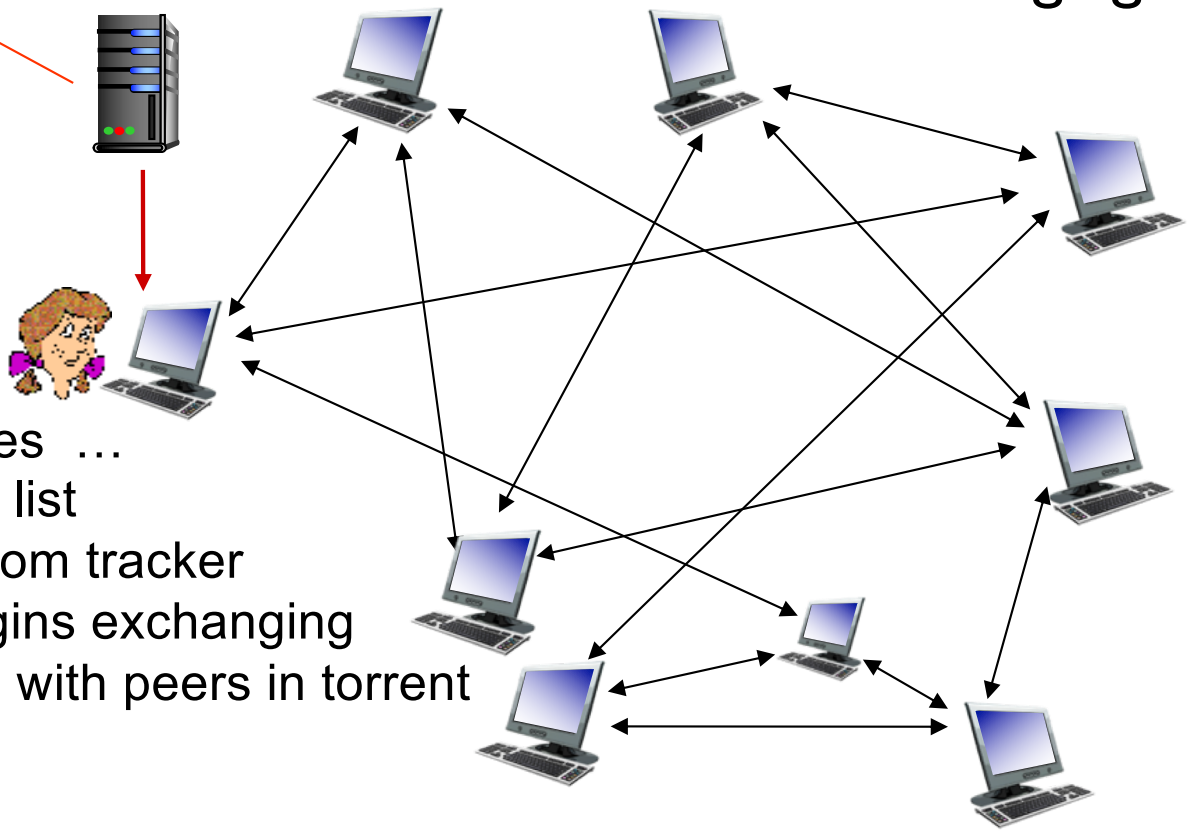
- Scalability: Minimal distribution time of P2P architecture is always less than that of the client-server architecture.
- Reason: Peers serve as redistributors as well as consumers of bits.

P2P file distribution: BitTorrent

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

tracker: tracks peers participating in torrent

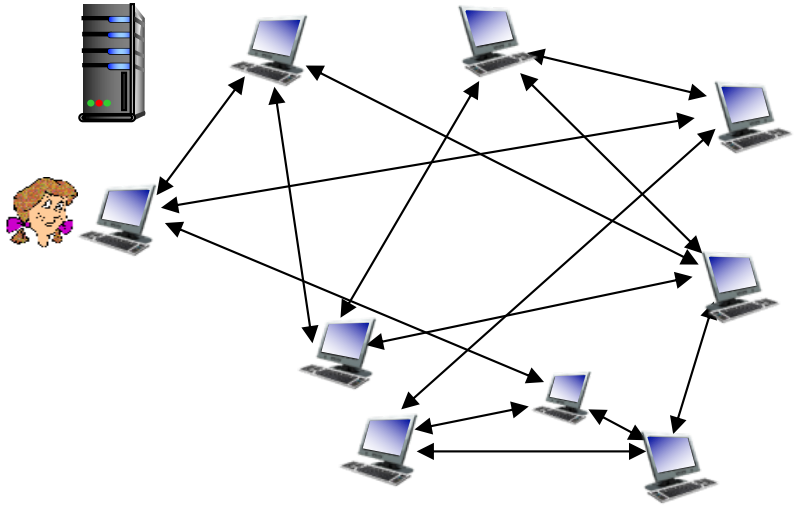
torrent: group of peers exchanging chunks of a file



Alice arrives …
… obtains list
of peers from tracker
… and begins exchanging
file chunks with peers in torrent

P2P file distribution: BitTorrent

- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")



- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
  - once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

BitTorrent: requesting, sending file chunks

requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

=> rarest chunks get more quickly redistributed

=> (roughly) equalize the numbers of copies of each chunk in the torrent

sending chunks: tit-for-tat

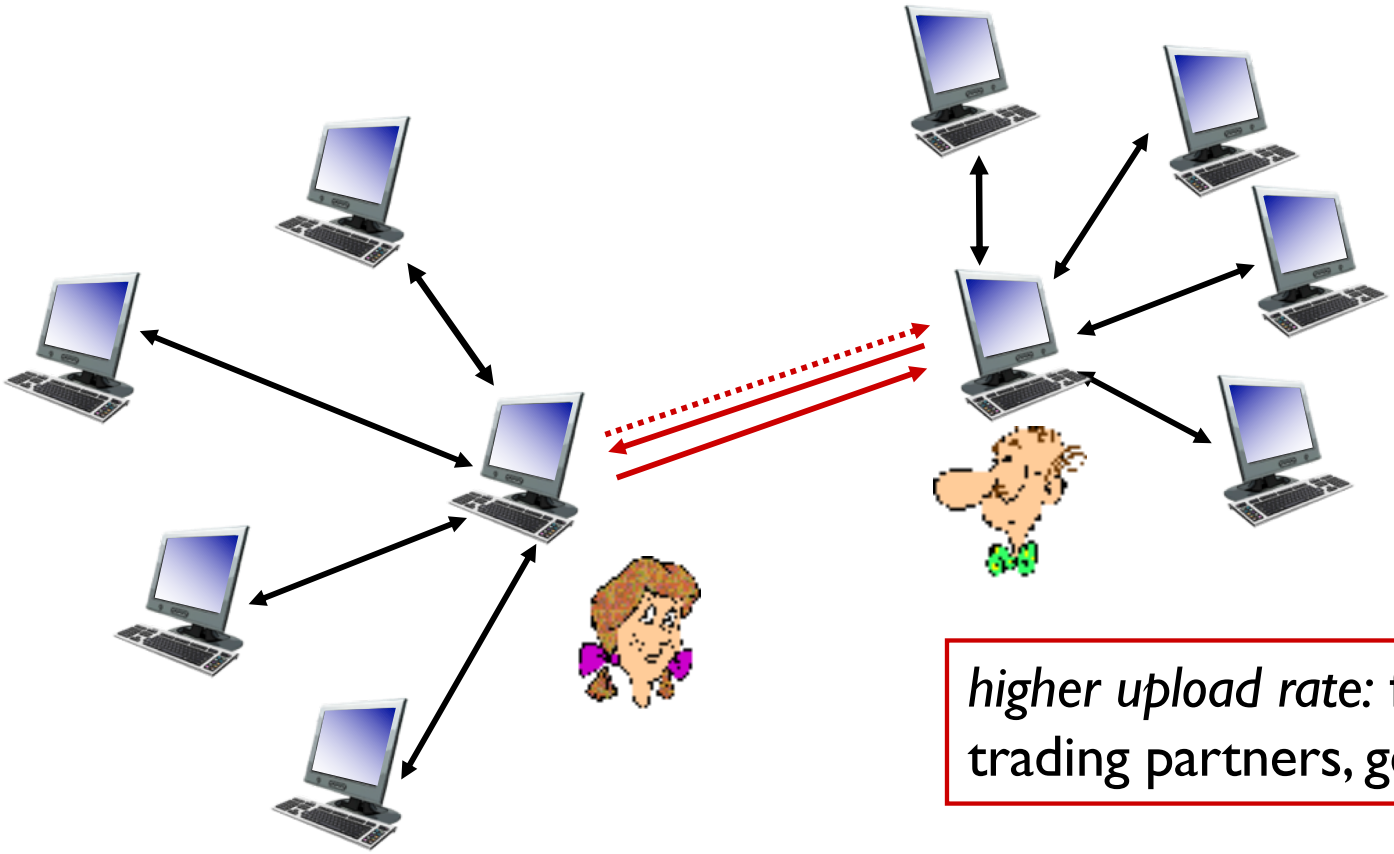
- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

BitTorrent: tit-for-tat

(1) Alice "optimistically unchokes" Bob

(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers



higher upload rate: find better trading partners, get file faster !

Chapter 2: summary

our study of network apps now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent
- socket programming: TCP, UDP sockets

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:

- control vs. data msgs
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable msg transfer
- "complexity at network edge"