



University of Colorado **Boulder**

Fundamentals of Data Communications

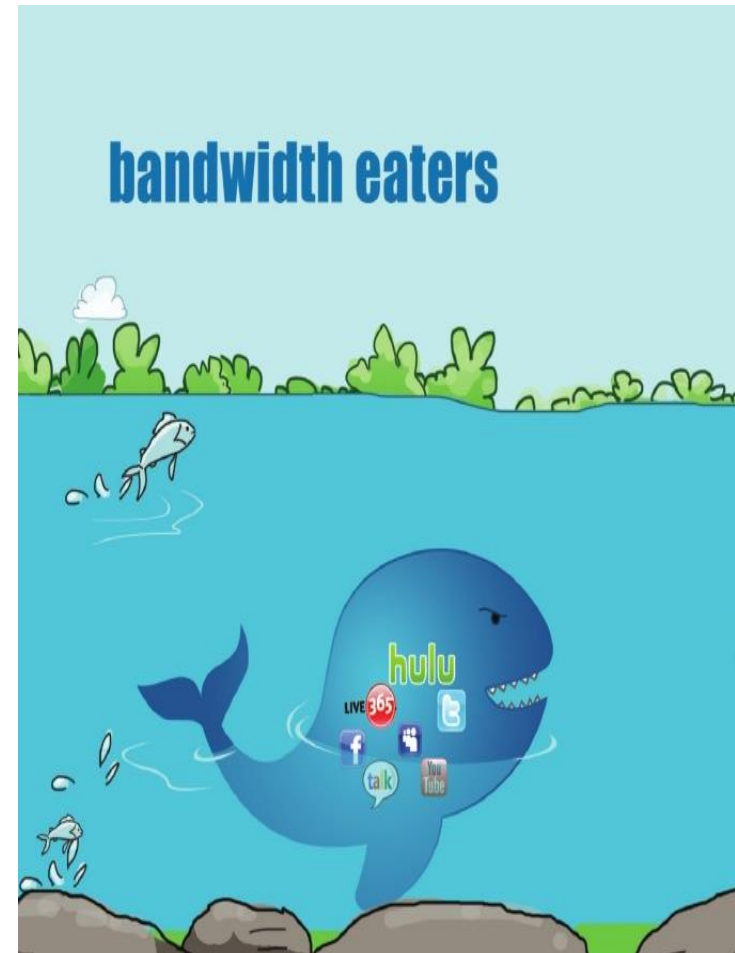
Applications

Levi Perigo, Ph.D.
University of Colorado Boulder
Department of Computer Science
Network Engineering

Review

Impact of User Applications on the Network

- **Interactive applications**
 - Inventory inquiries, database updates.
 - Human-to-machine interaction.
 - Because a human is waiting for a response, response time is important but not critical, unless the wait becomes excessive.
- **Real-time applications**
 - VoIP, video
 - Human-to-human interaction
 - End-to-end latency critical
- **Internet applications**
 - Social media
 - Streaming: audio & video



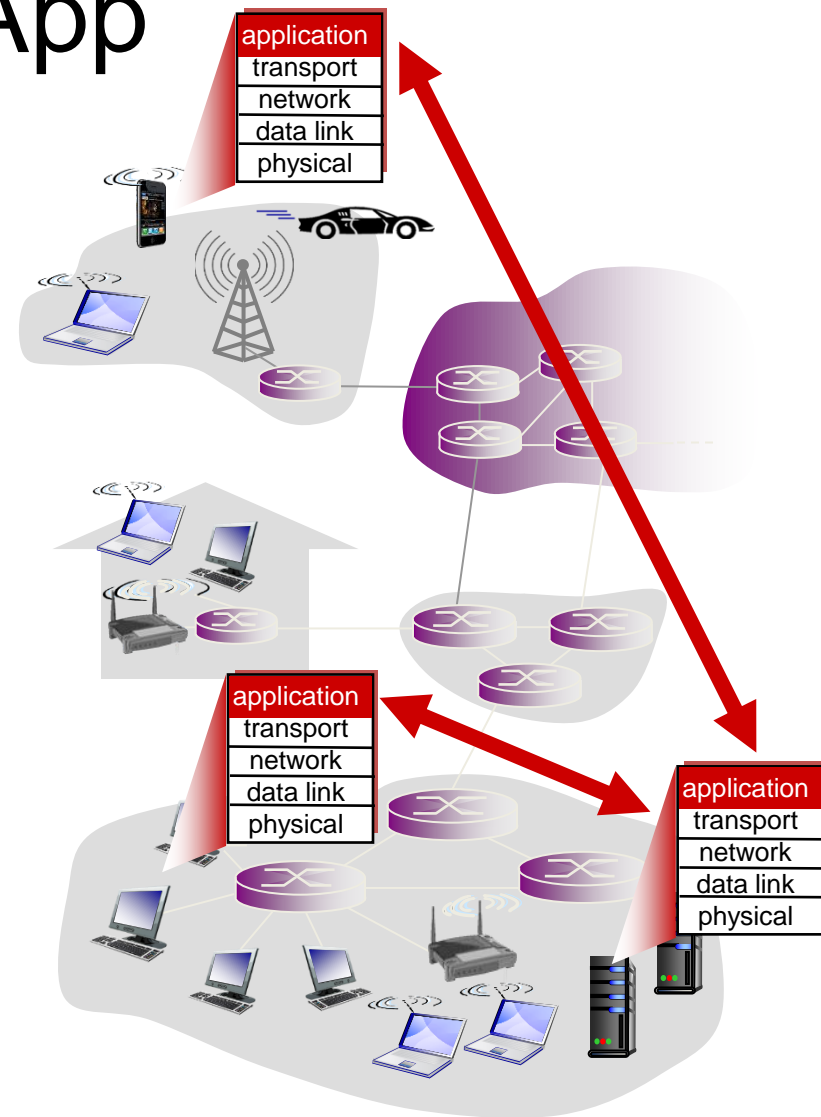
Creating a Network App

Write programs that:

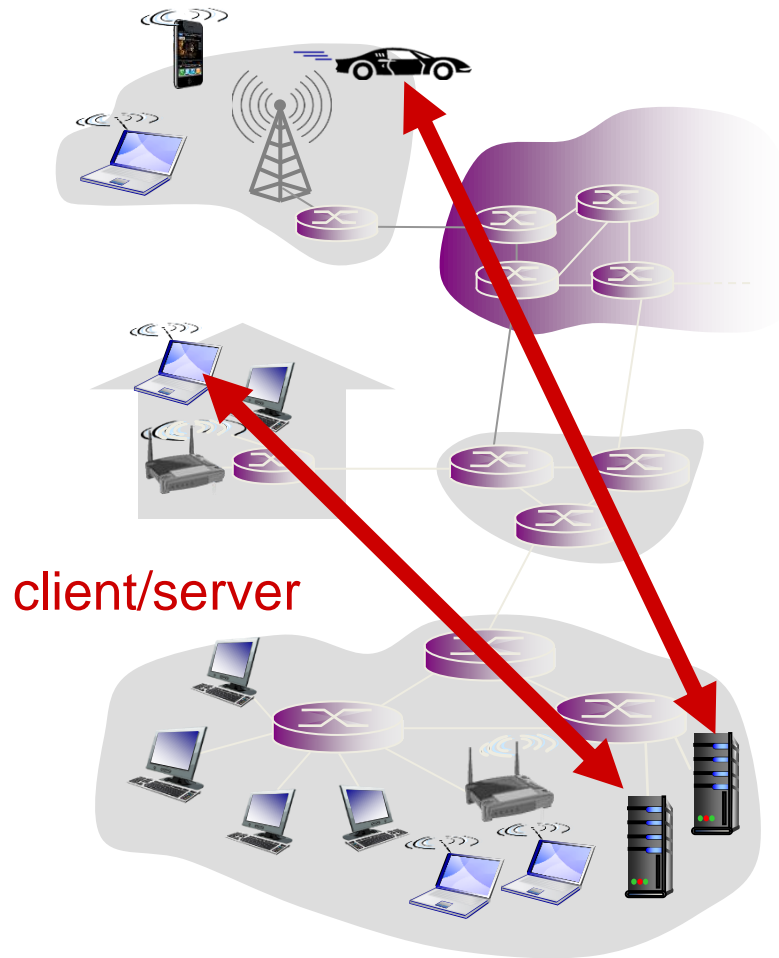
- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software
 - Client/server or p2p

No need to write software for network-core devices

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



Client-server Architecture



Server:

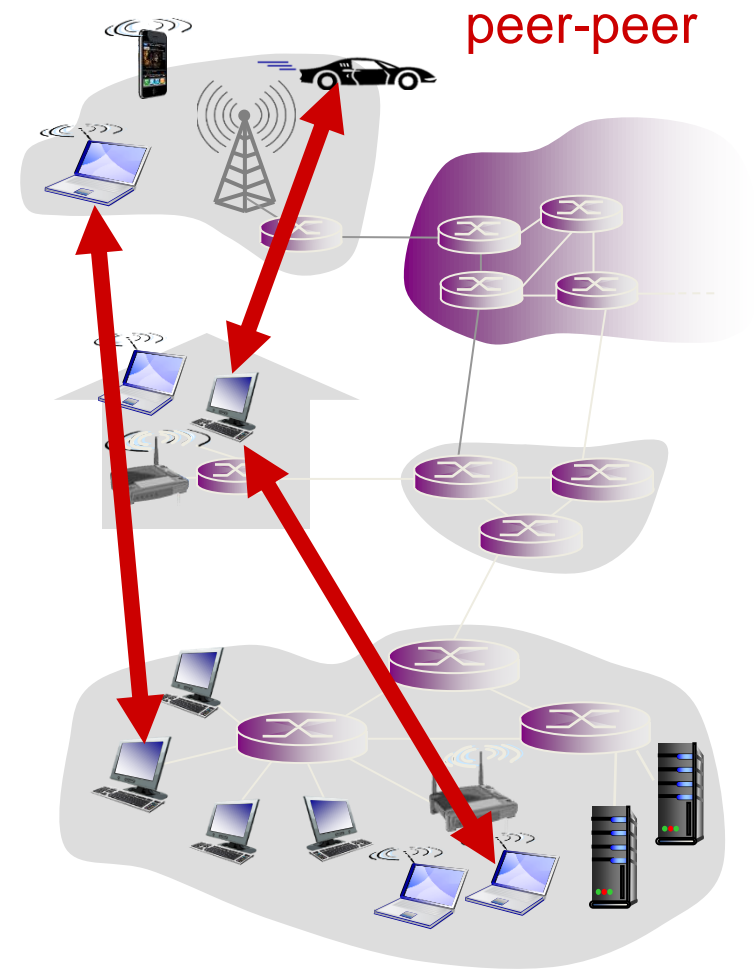
- always-on host
- “permanent” IP address
- data centers for scaling

Clients:

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

P2P architecture

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



Processes Communicating

Process: program running within a host

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

clients, servers

client process: process that initiates communication

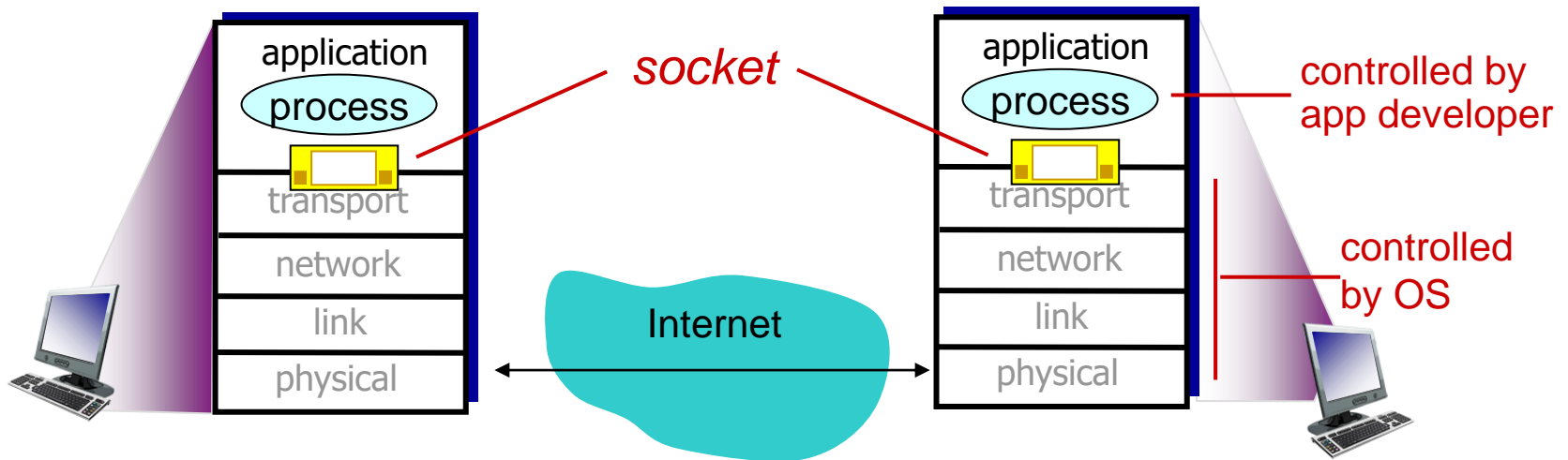
server process: process that waits to be contacted

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



Sockets

- **Process sends/receives messages to/from its **socket****
- **Socket analogous to door**
 - sending process shoves message out door
 - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Addressing Processes

- To receive messages, process must have *identifier*
- Host device has unique 32-bit IP address
- Q: Does IP address of host on which process runs suffice for identifying the process?
 - A: No, *many* processes can be running on same host
- *Identifier* includes both **IP address** and **port numbers** associated with process on host.
- Example port numbers:
 - HTTP server: 80
 - mail server: 25
- To send HTTP message to cs.colorado.edu web server:
 - **IP address**: 128.119.245.12
 - **port number**: 80



App-layer Protocol Defines:

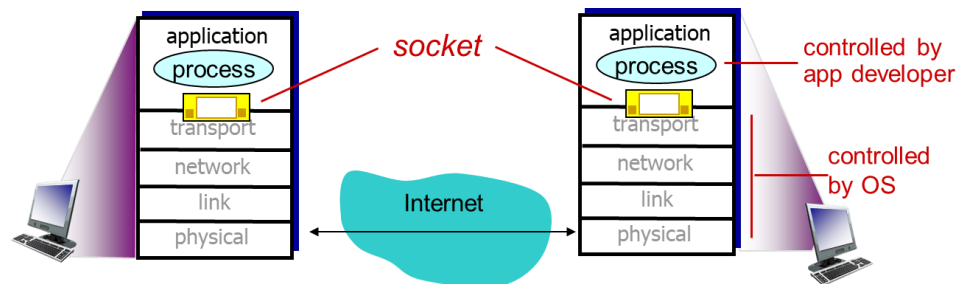
- **Types of messages exchanged**
 - e.g., request, response
- **Message syntax:**
 - what fields in messages & how fields are delineated
- **Message semantics**
 - meaning of information in fields
- **Rules** for when and how processes send & respond to messages

Open protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

Proprietary protocols:

- e.g., Skype



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., VoIP, 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



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



Securing TCP

- **TCP & UDP**

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

- **SSL**

- 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*



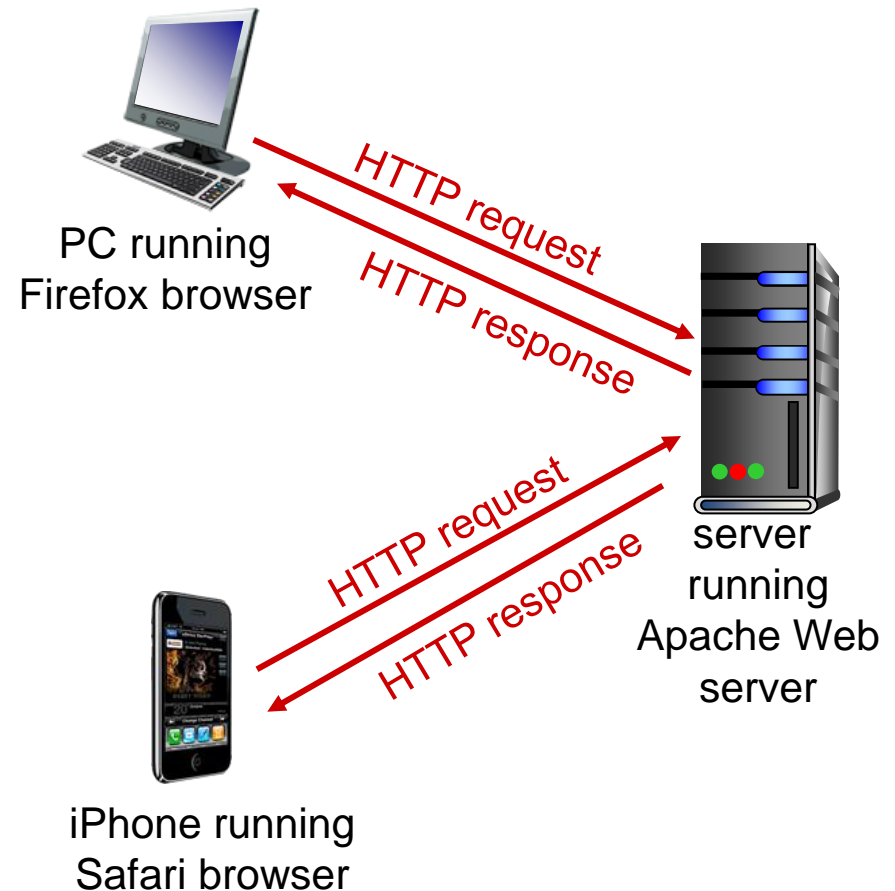
Hypertext Transfer Protocol (HTTP)



HTTP Overview

HTTP: hypertext transfer protocol

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



HTTP Overview (continued)

- *Uses TCP:*

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

HTTP is “stateless”

- server maintains no information about past client requests

aside

protocols that maintain “state” are complex!

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



HTTP Connections

Non-persistent HTTP

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

Persistent HTTP

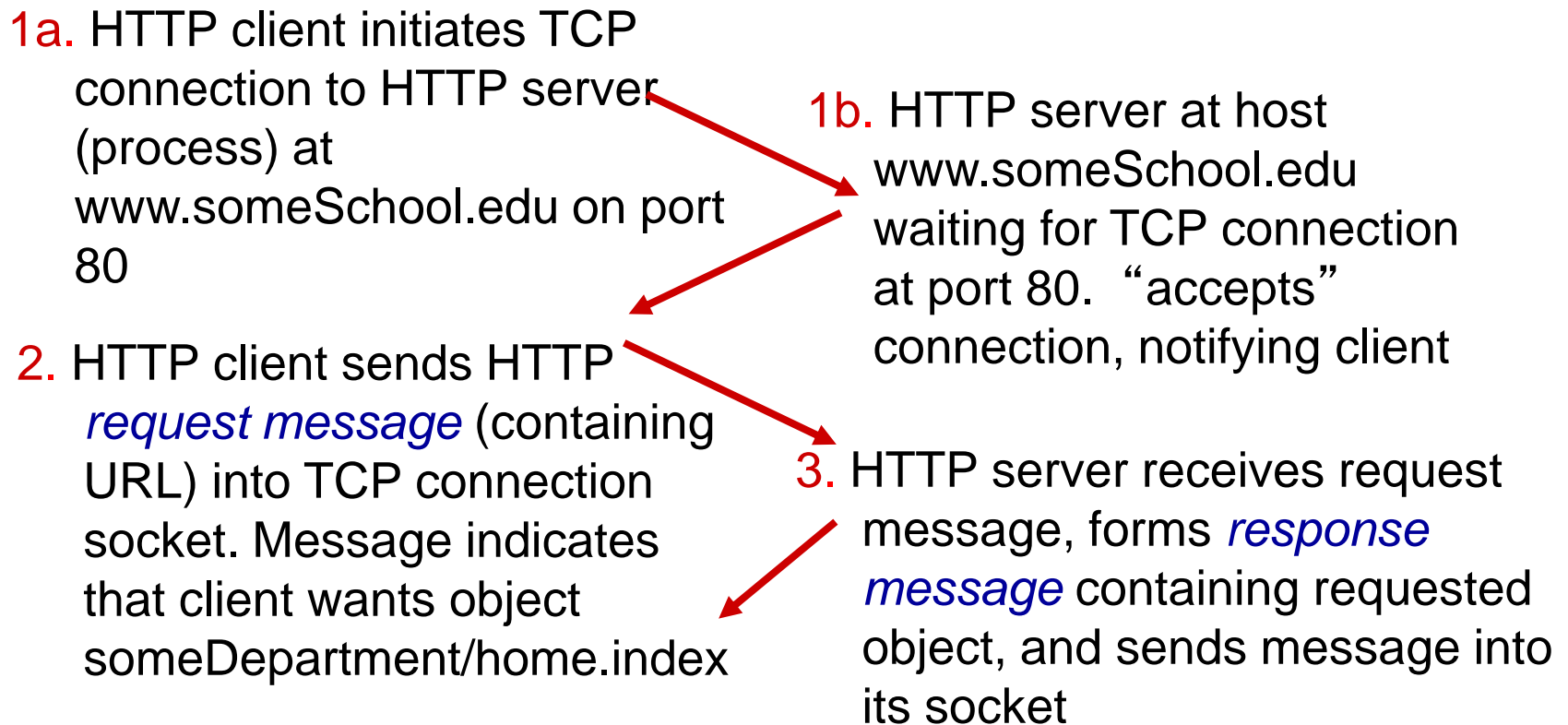
- multiple objects can be sent over single TCP connection between client, server



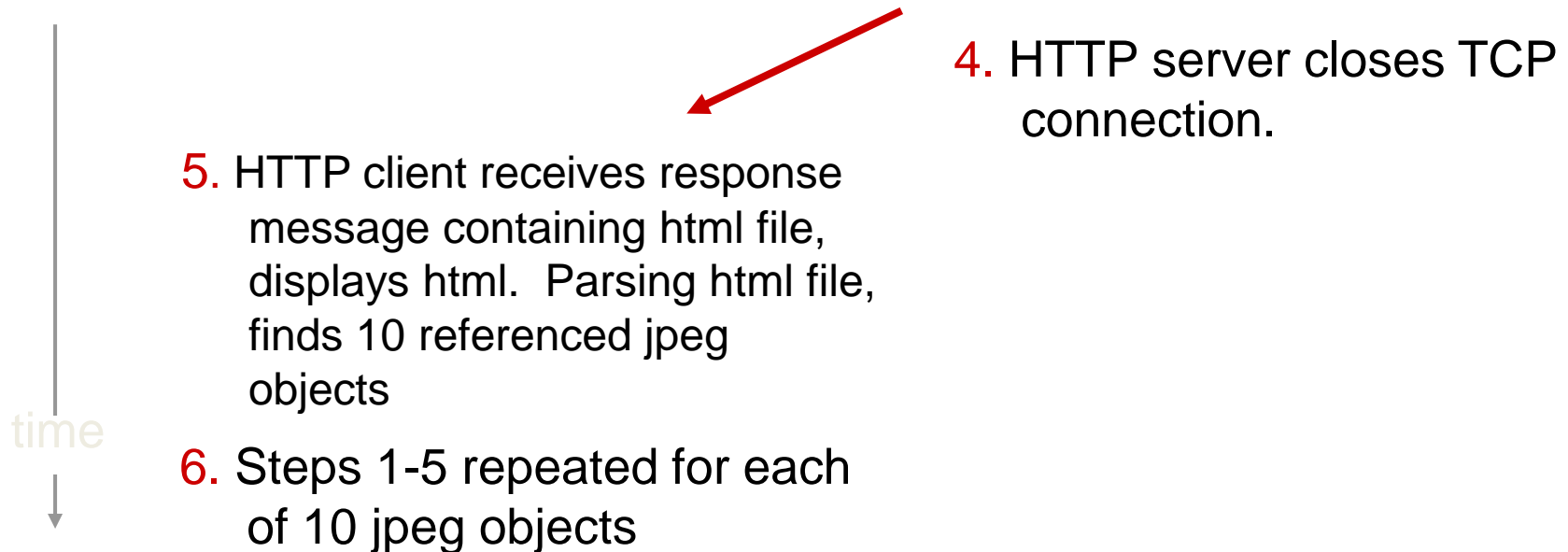
Non-persistent HTTP

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

(contains text,
references to 10
jpeg images)



Non-persistent HTTP (cont.)



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 general types of HTTP messages: *request*, *response*
- HTTP request message:

- ASCII (human-readable format)

request line
(GET, POST,
HEAD commands)

header
lines

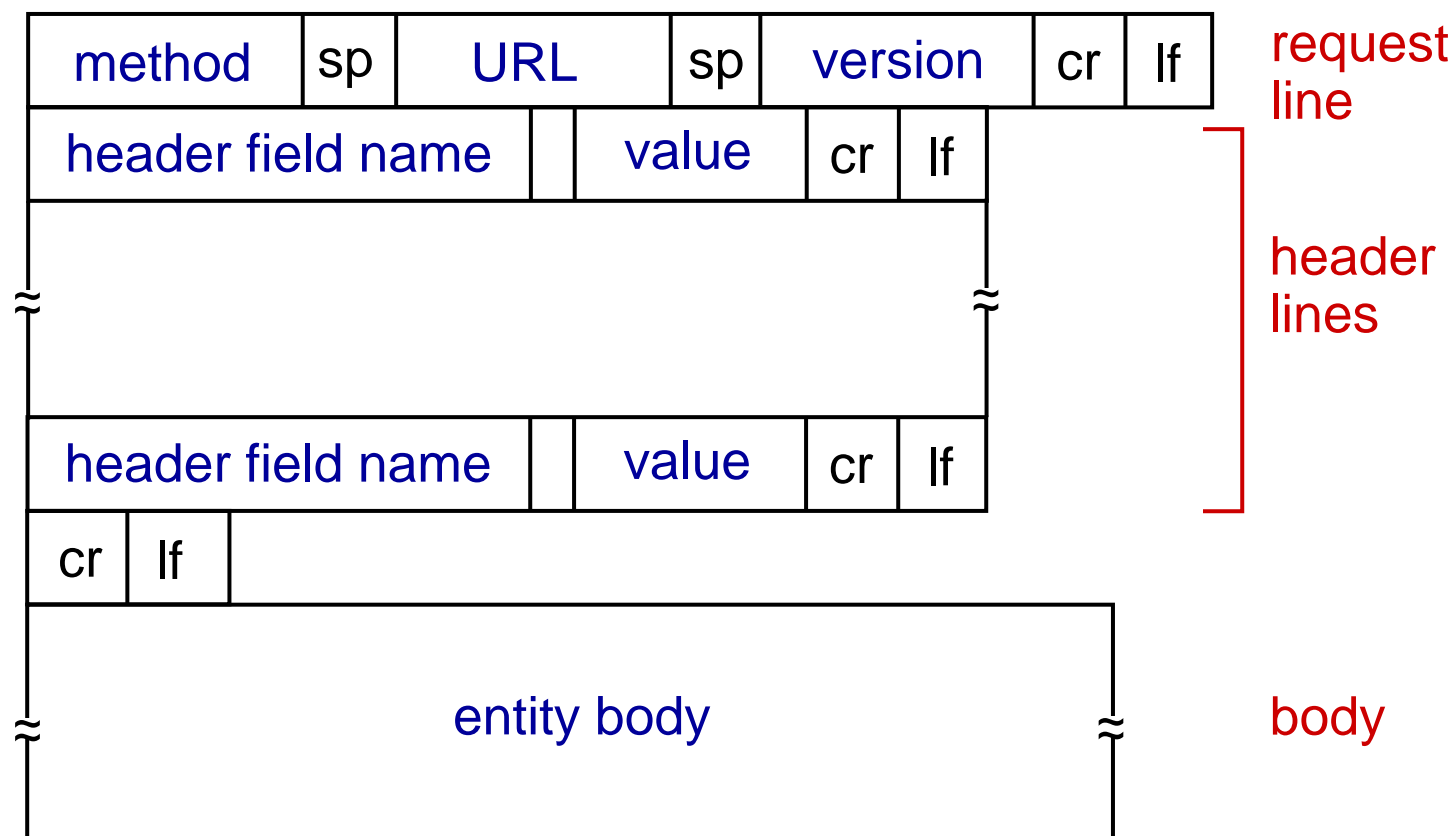
carriage return,
line feed at start
of line indicates
end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.colorado.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
```

carriage return character
line-feed character



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`



HTTP Response Message

status line

(protocol

status code

status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
      GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
      1\r\n
\r\n
data data data data data ...
```



HTTP Response - Status Codes

- Status code appears in first line in server-to-client response message.
- Some sample codes:

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

- requested object moved, new location specified later in this msg (Location:)

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported



User-Server State: cookies

Many Web sites use cookies

Four components:

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

Example:

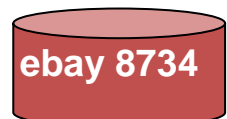
- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
 - unique ID
 - entry in backend database for ID

Cookies: keeping “state” (cont.)

client



server



cookie file

usual http request msg

Amazon server
creates ID
1678 for user

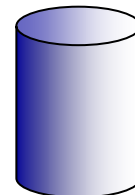
usual http response
set-cookie: 1678



ebay 8734
amazon 1678

create
entry
'1678'

backend
database



usual http request msg
cookie: 1678

cookie-
specific
action

Access
'1678'

usual http response msg

Access
'1678'

cookie-
specific
action

one week later:



ebay 8734
amazon 1678

usual http request msg
cookie: 1678

usual http response msg



University of Colorado
Boulder

File Edit View Go Capture Analyze Statistics Help

Enter **http** + Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
1091	75.2589961	65.55.131.124	192.168.1.112	HTTP	HTTP/1.1 200 OK (text/html)
1184	76.564417	192.168.1.112	200.94.118.253	HTTP	GET /view/login.live.com?xtn_source=extension&xtn_medium=firefox HTTP/1.1
1111	76.781739	200.94.118.253	192.168.1.112	HTTP	HTTP/1.0 200 OK (text/html)
1135	77.276673	192.168.1.112	199.7.58.72	OCSP	Request
1139	77.309658	199.7.58.72	192.168.1.112	OCSP	Response
1164	84.290326	192.168.1.112	207.68.173.76	HTTP	POST /www.signup1.0 HTTP/1.1 (application/x-www-form-urlencoded)
1172	84.433753	207.68.173.76	192.168.1.112	HTTP	HTTP/1.1 302 Found (text/html)
1175	84.438886	192.168.1.112	207.68.173.76	HTTP	GET / HTTP/1.1
1206	84.739394	207.68.173.76	192.168.1.112	HTTP	HTTP/1.1 200 OK (text/html)
1217	85.148678	192.168.1.112	207.68.173.76	HTTP	GET /rss/cctopics.aspx HTTP/1.1
1221	85.197829	192.168.1.112	65.55.131.124	HTTP	GET /cgi-bin/mymn/mymn.html?1242174874422 HTTP/1.1
1223	85.293151	207.68.173.76	192.168.1.112	HTTP/XML	HTTP/1.1 200 OK
1227	85.325899	65.55.131.124	192.168.1.112	HTTP	HTTP/1.1 200 OK (text/html)
1244	85.757436	192.168.1.112	64.4.33.7	HTTP	GET /cgi-bin/mymn/mymn.js?mc=0a248a31c13e53a5c45fcca7db97ac6b26688857875nocache=0
1258	86.185452	64.4.33.7	192.168.1.112	HTTP	HTTP/1.1 200 OK (application/x-javascript)
1272	181.814658	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1

www1242174874422

Host: www.msn.com/r/n

User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.1b4) Gecko/20090507 Firefox/3.5b4/r/n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8/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,*;q=0.7/r/n

Keep-Alive: 300/r/n

Connection: keep-alive/r/n

[truncated] Cookie: MUI=0+366UID=015159c5a7784e88333e54884123fd; MUID=0033E1D56E7B408CB5791A92EE42843; mh=MSFT; CULTURE=EN-US; Flight00cupid=68; Flight00cupid=68

Content-Type: application/x-www-form-urlencoded/r/n

> Content-Length: 1386/r/n

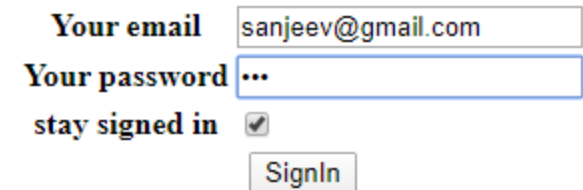
> Line-based text data: application/x-www-form-urlencoded

HTTP Cookie [http.cookie], 290 bytes Packets: 1264 Displayed: 131 Marked: 0 Dropped: 0 Profile: Default



Cookies

- Remembers stateful information (such as login) for stateless HTTP protocol



Your email

Your password

stay signed in ☒

- Three main purposes
 - Session management
 - ***Logins, shopping carts, game scores, etc.***
 - Personalization
 - ***User preferences, themes, other settings***
 - Tracking
 - ***Recording and analyzing user behavior***



Cookie Tracking



- Websites can use cookies in questionable ways:
 - Privacy and safety
- Share long strings of information about what sites you've visited, and what you've done there
 - Data can be transmitted to other sites/parties without knowledge
 - ***Advertisers – allows them to build basic personal profiles about you, and serve relevant ads you're likely to buy***
 - What about smart homes?
- Don't contain name or email address



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



Digital Footprint



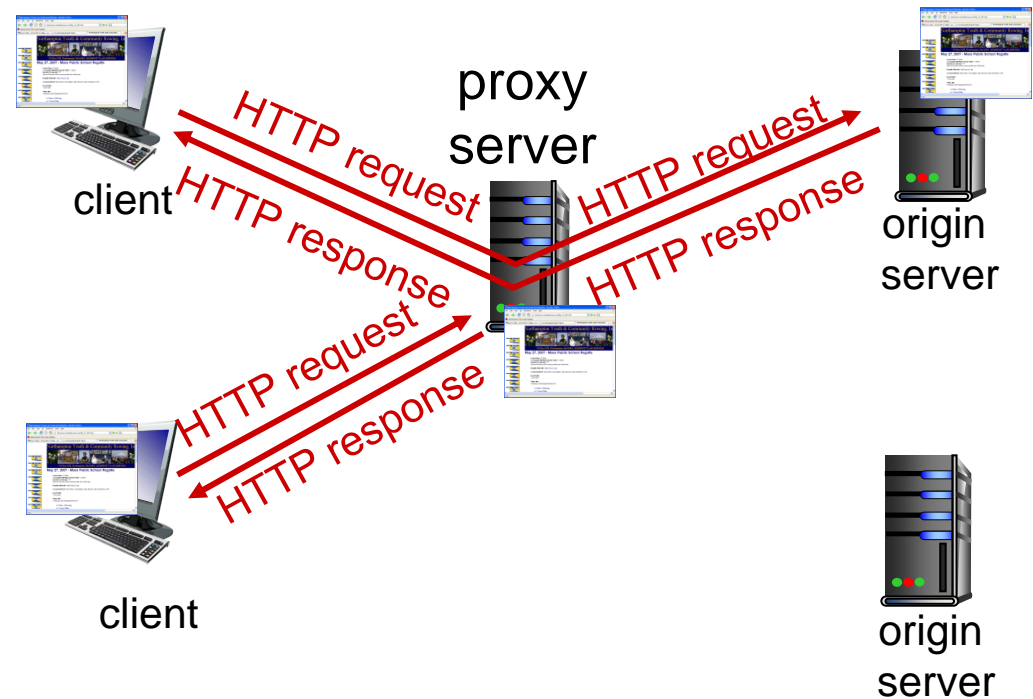
- **What does your digital footprint say about you?**



Web Caches (proxy server)

Goal: satisfy client request without involving origin server

- User sets browser: Web accesses via cache
- Browser sends all HTTP requests to cache
 - object in cache: cache returns object
 - else cache requests object from origin server, then returns object to client
 - What are the benefits of this?



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?

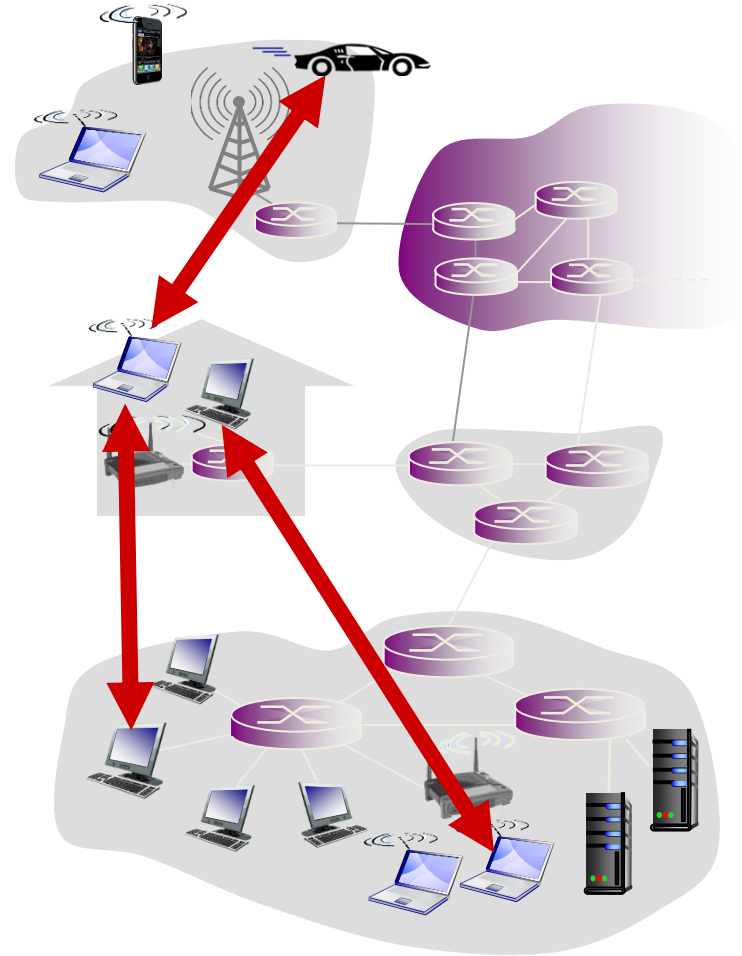
- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables “poor” content providers to effectively deliver content (so too does P2P file sharing)

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 (Muvi)
- VoIP (Skype)

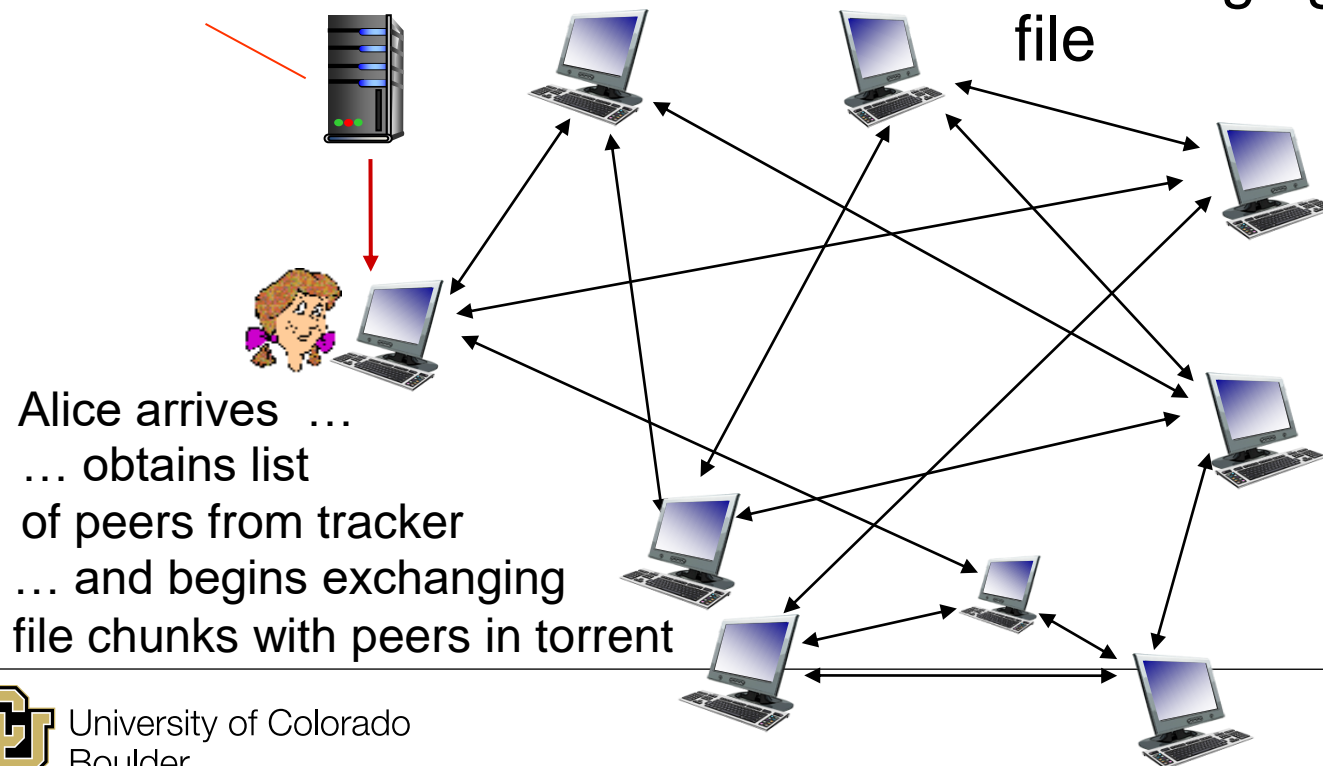


P2P File Distribution: BitTorrent

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

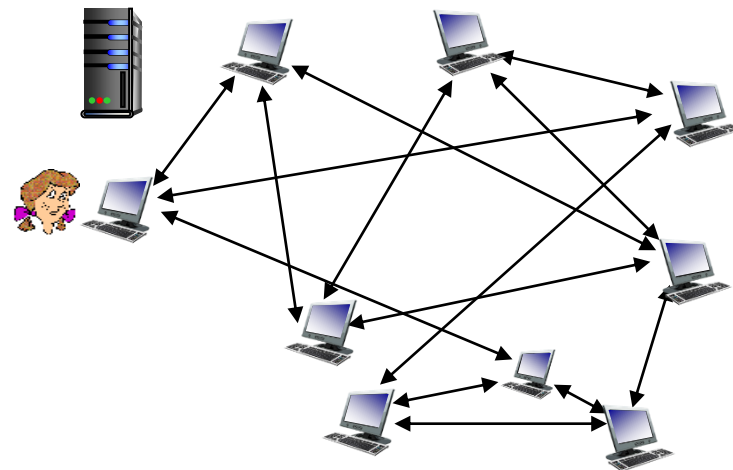
tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



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
- *What protocol does this sound like that we've already studied?*

- *Sending chunks:*

- 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 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
 - “optimistically unchoke” this peer
 - newly chosen peer may join top 4

Video Streaming and CDNs: context

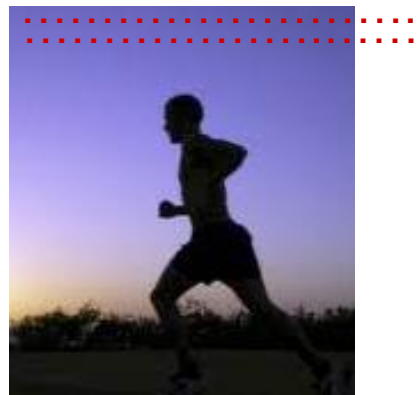
- **Video traffic: major consumer of Internet bandwidth**
 - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
 - ~2.25B YouTube users, ~75M Netflix users
- **Challenge: scale - how to reach ~2B users?**
 - single mega-video server won't work (why?)
- **Challenge: heterogeneity**
 - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor; location)
- ***Solution:* distributed, application-level infrastructure**



Multimedia: video

- Video: sequence of images displayed at constant rate
 - e.g., 24 images/sec
- Digital image: array of pixels
 - each pixel represented by bits
- Coding: use redundancy *within* and *between* images to decrease # bits used to encode image
 - spatial (within image)
 - temporal (from one image to next)

spatial coding example: instead of sending N values of same color (all purple), send only two values: color value (*purple*) and number of repeated values (N)



frame i

temporal coding example: instead of sending complete frame at $i+1$, send only differences from frame i



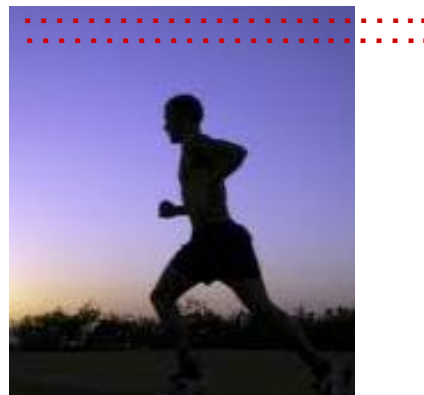
frame $i+1$



Multimedia: video

- **CBR: (constant bit rate):**
video encoding rate fixed
- **VBR: (variable bit rate):**
video encoding rate changes as amount of spatial, temporal coding changes
- **examples:**
 - MPEG 1 (CD-ROM)
1.5 Mbps
 - MPEG2 (DVD) 3-6 Mbps
 - MPEG4 (often used in Internet, < 1 Mbps)

spatial coding example: instead of sending N values of same color (all purple), send only two values: color value (*purple*) and number of repeated values (N)



frame i

temporal coding example: instead of sending complete frame at $i+1$, send only differences from frame i

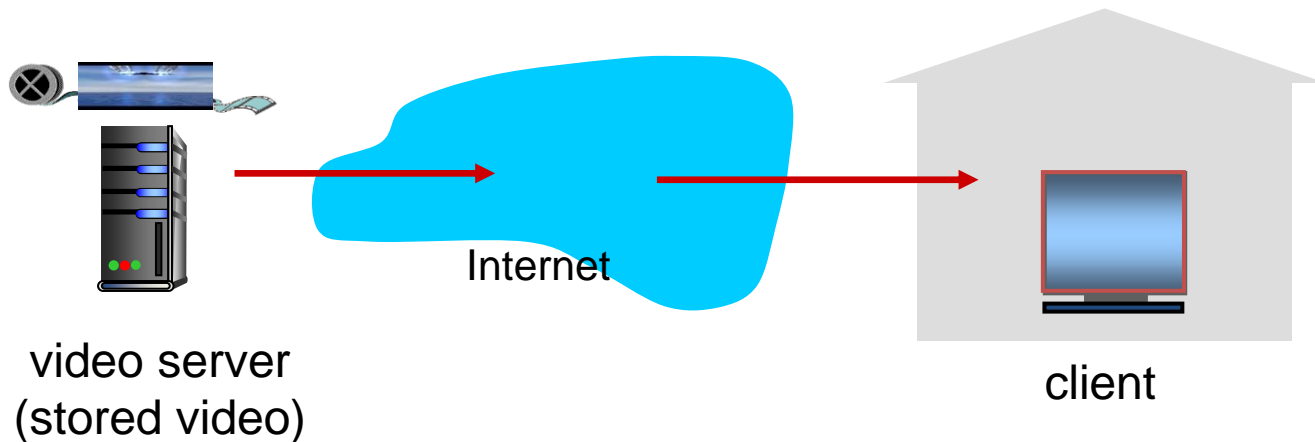


frame $i+1$



Streaming stored video:

simple scenario:



Streaming Multimedia: DASH

- **DASH:** *D*ynamic, *A*daptive *S*treaming over *H*TTP
- *Server:*
 - divides video file into multiple chunks
 - each chunk stored, encoded at different rates
 - *manifest file:* provides URLs for different chunks
- *Client:*
 - periodically measures server-to-client bandwidth
 - consulting manifest, requests one chunk at a time
 - ***chooses maximum coding rate sustainable given current bandwidth***
 - ***can choose different coding rates at different points in time (depending on available bandwidth at time)***

Streaming Multimedia: DASH

- *DASH: Dynamic, Adaptive Streaming over HTTP*
- “*intelligence*” at client: client determines
 - *when* to request chunk (so that buffer starvation, or overflow does not occur)
 - *what encoding rate* to request (higher quality when more bandwidth available)
 - *where* to request chunk (can request from URL server that is “close” to client or has high available bandwidth)

Content Distribution Networks (CDN)

- *Challenge*: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?
- *Option 1*: single, large “mega-server”
 - single point of failure
 - point of network congestion
 - long path to distant clients
 - multiple copies of video sent over outgoing link
 - *quite simply: this solution doesn't scale*

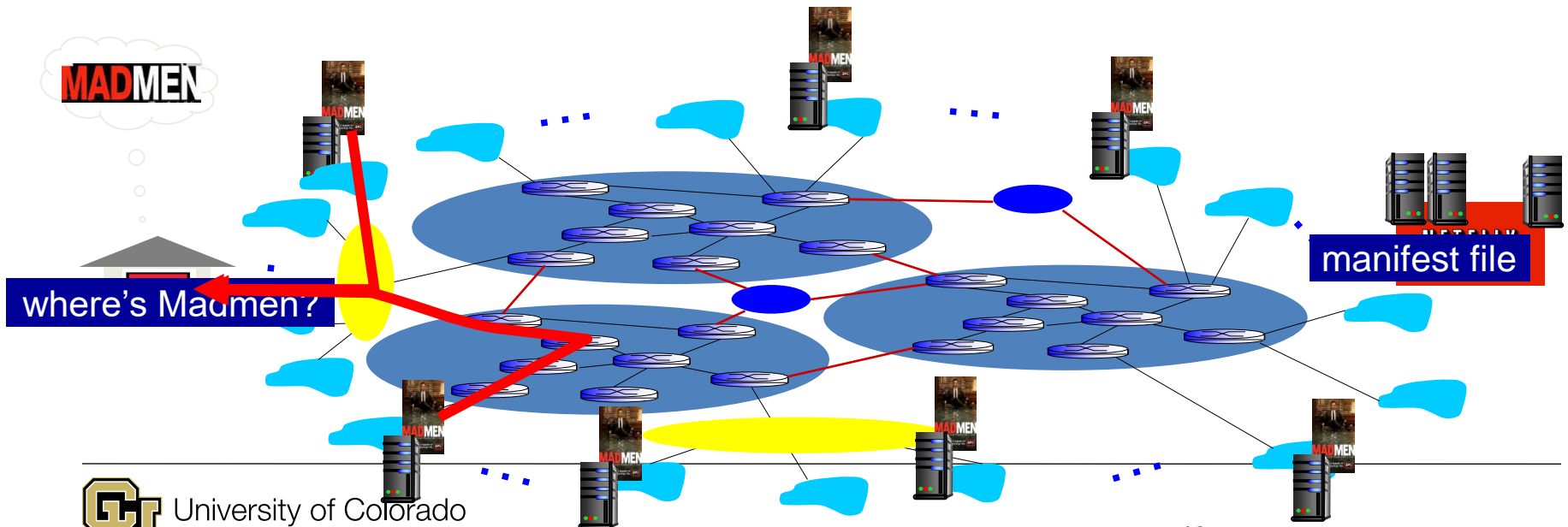
CDN

- *Challenge:* how to stream content (selected from millions of videos) to millions of simultaneous users?
- *Option 2:* store/serve multiple copies of videos at multiple geographically distributed sites (*CDN*)
 - *enter deep:* push CDN servers deep into many access networks
 - *close to users*
 - *used by Akamai, 1700 locations*
 - *bring home:* smaller number (10's) of larger clusters in POPs near (but not within) access networks
 - *used by Limelight*

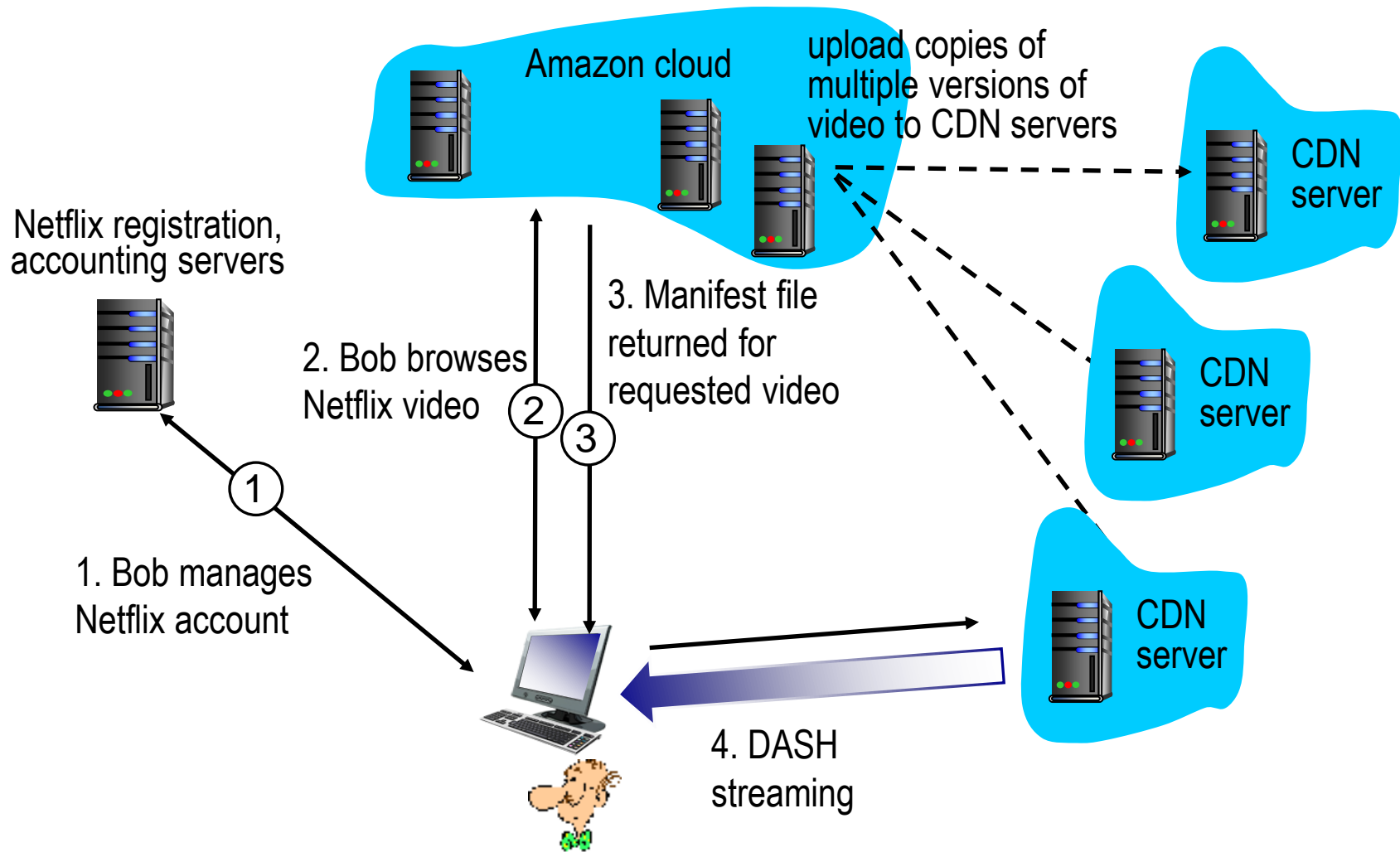


Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
 - e.g. Netflix stores copies of Mad Men
- Subscriber requests content from CDN
 - directed to nearby copy, retrieves content
 - may choose different copy if network path congested (how could it know this?)



Case Study: Netflix



Questions?



Client can't connect to web server

Three Locations w/ single web server

Appendix

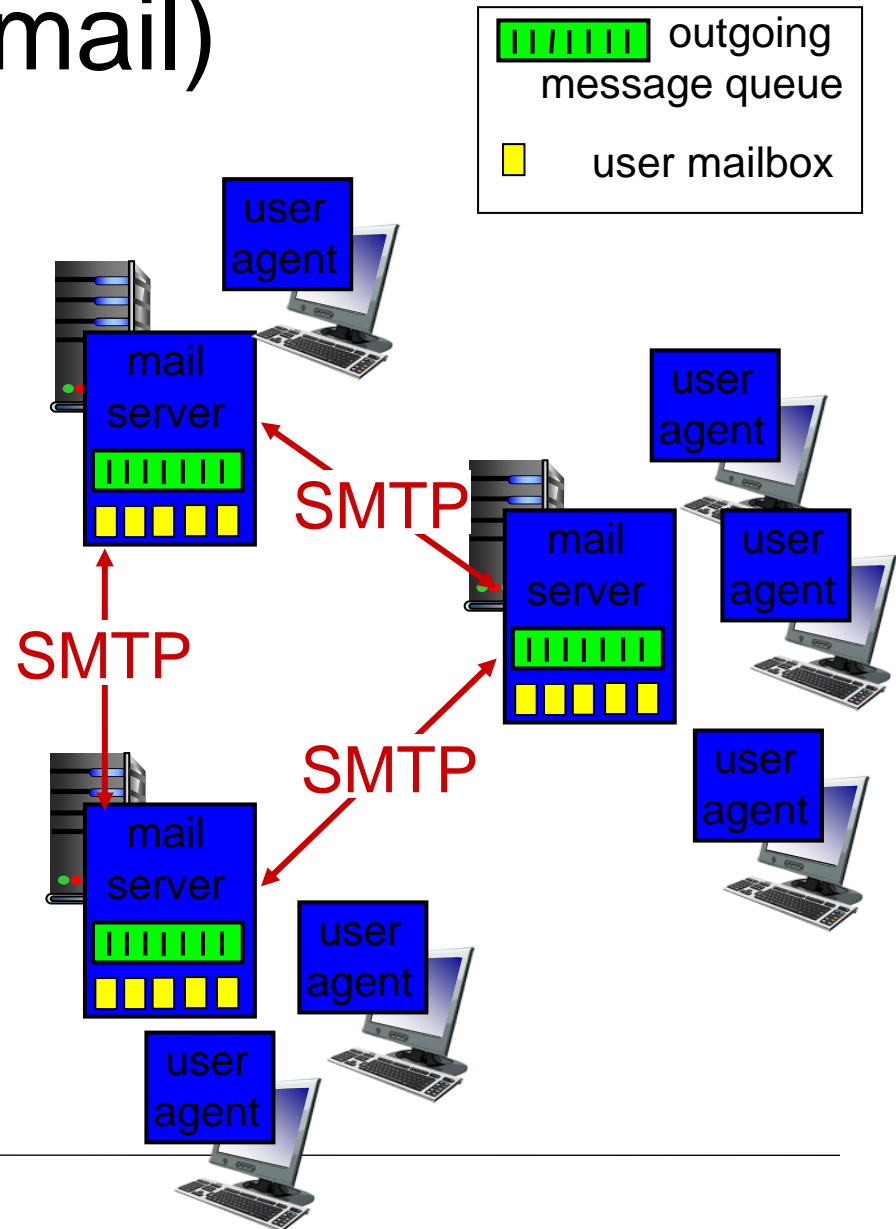
Electronic mail (e-mail)

Three major components:

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

User Agent

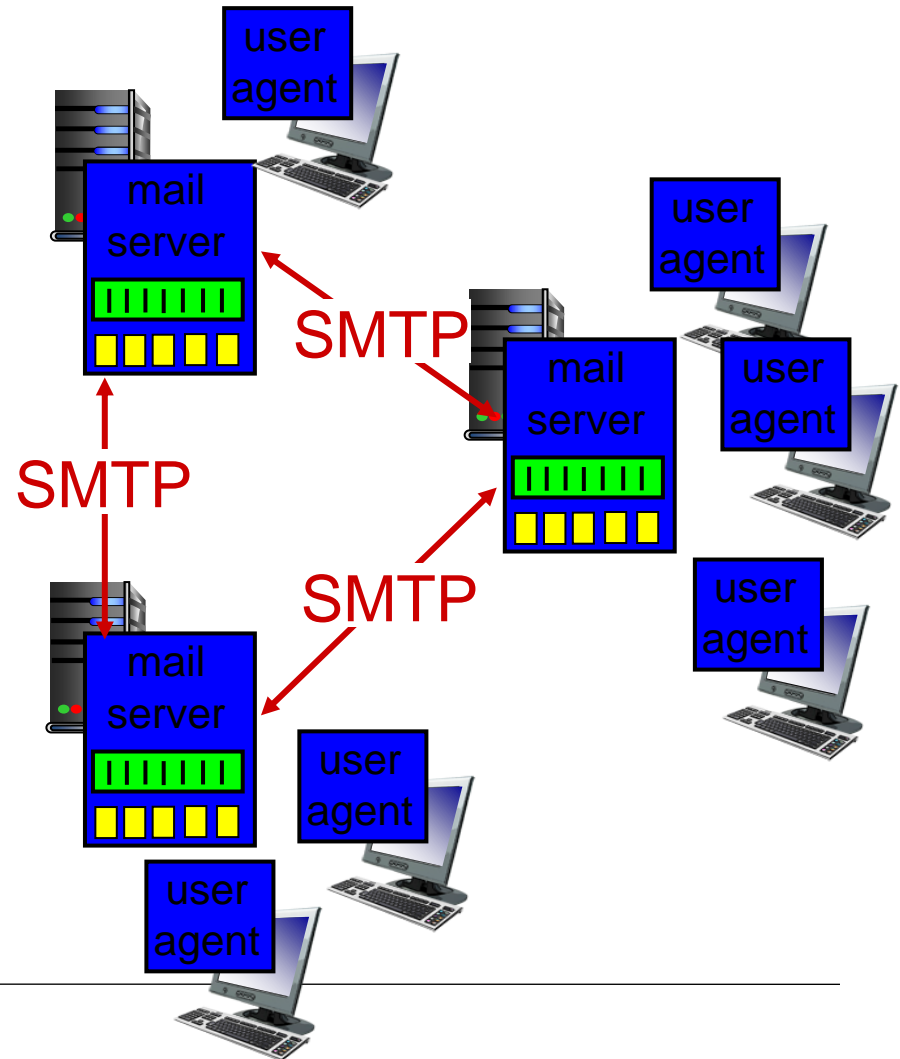
- **a.k.a. “mail reader”**
- **composing, editing, reading mail messages**
- **e.g., Outlook, Thunderbird, iPhone mail client**
- **outgoing, incoming messages stored on server**



Electronic mail: mail servers

mail servers:

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



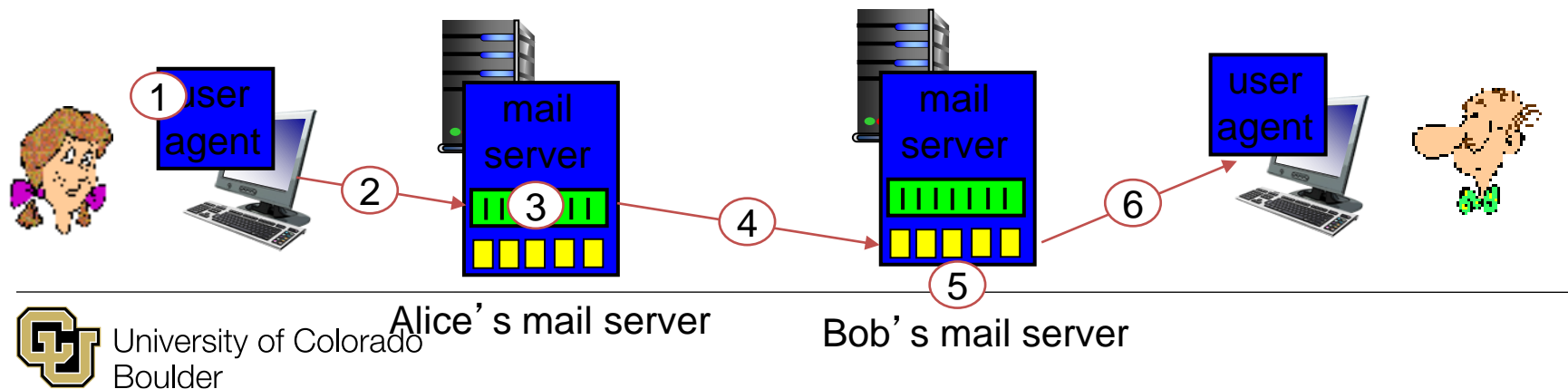
Electronic Mail: SMTP [RFC 2821]

- **uses TCP to reliably transfer email message from client to server, port 25**
- **direct transfer: sending server to receiving server**
- **three phases of transfer**
 - handshaking (greeting)
 - transfer of messages
 - closure
- **command/response interaction (like HTTP)**
 - **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



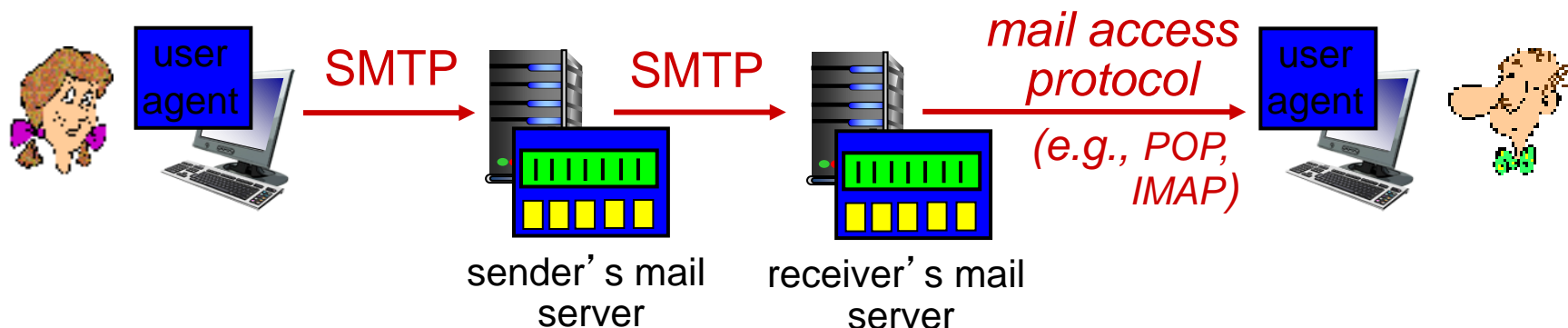
SMTP: Summary

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

comparison with HTTP:

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

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 messages 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

- **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

