

Computer Networks

The Application Layer

Prof. Navrati Saxena
San Jose State University

Use Wireshark to look at captured HTTP request/response)

Maintaining user/server state: cookies

- Recall: HTTP GET/response interaction is *stateless*
- Often desired by website to identify users
 - Either to restrict or
 - Serve special content

Maintaining user/server state: cookies

Web sites and client browser use *cookies* to maintain some state between transactions

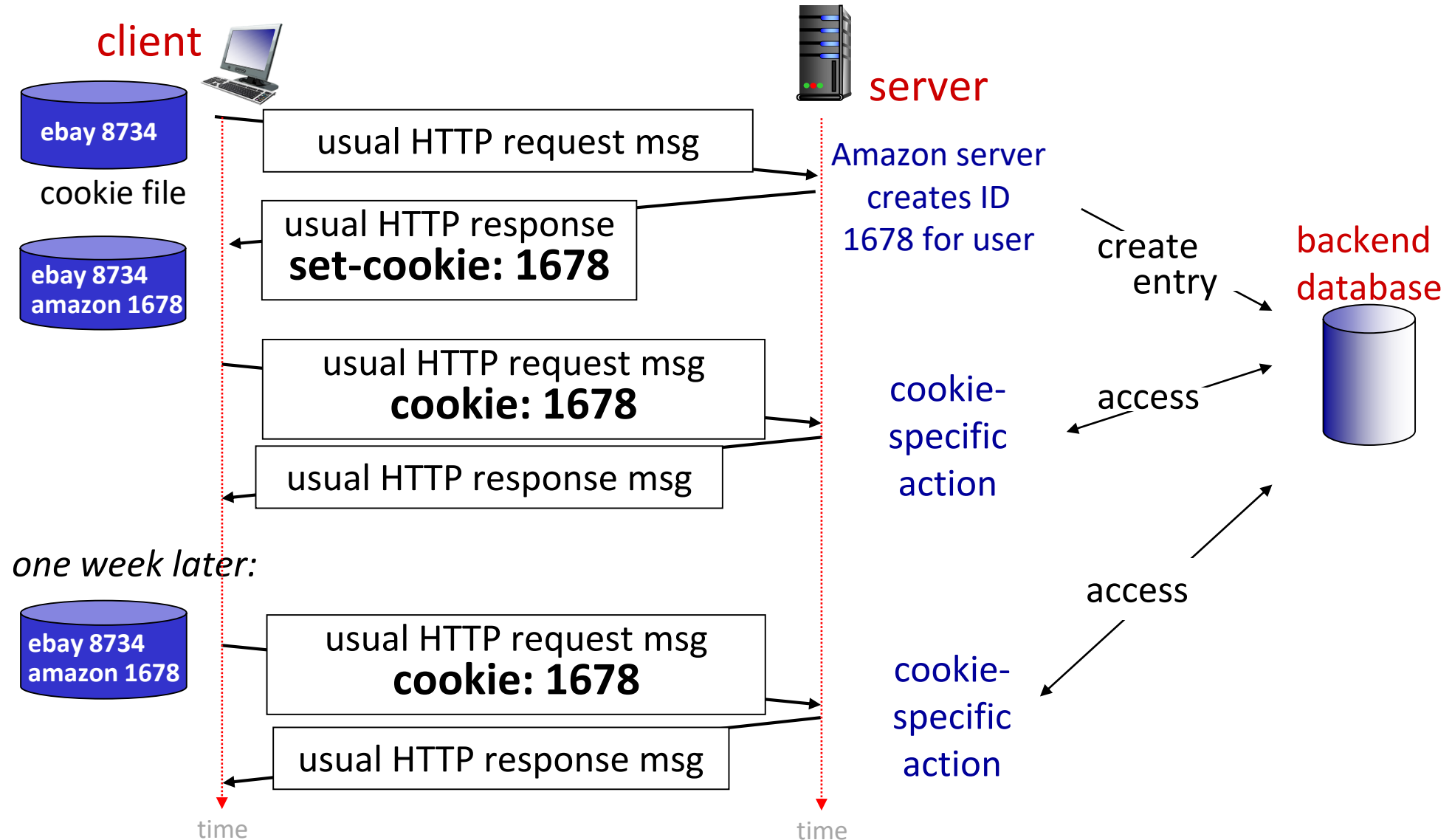
four components:

- 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 uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
 - unique ID (aka “cookie”)
 - entry in backend database for ID
- subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to “identify” Susan

Maintaining user/server state: cookies



HTTP cookies: comments

What cookies can be used for:

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

Challenge: How to keep state?

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

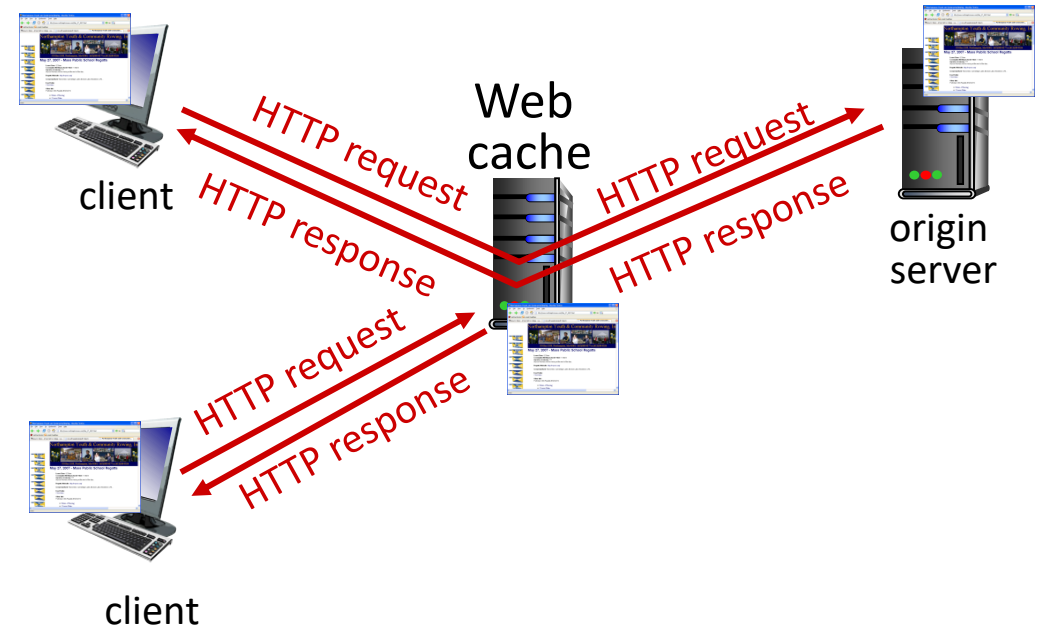
— aside —
cookies and privacy:

- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

Web caches

Goal: satisfy client requests without involving origin server
Purchased and installed by ISP

- user configures browser to point to a (local) *Web cache*
- browser sends all HTTP requests to cache
 - *if* object in cache: cache returns object to client
 - *else* cache requests object from origin server, caches received object, then returns object to client



Web caches (aka proxy servers)

- Web cache acts as both client and server
 - server for original requesting client
 - client to origin server

Why Web caching?

- reduce response time for client request
 - cache is closer to client
- reduce traffic on an institution's access link
- Internet is dense with caches
 - enables “poor” content providers to more effectively deliver content

Improves performance for all Apps!

Caching example

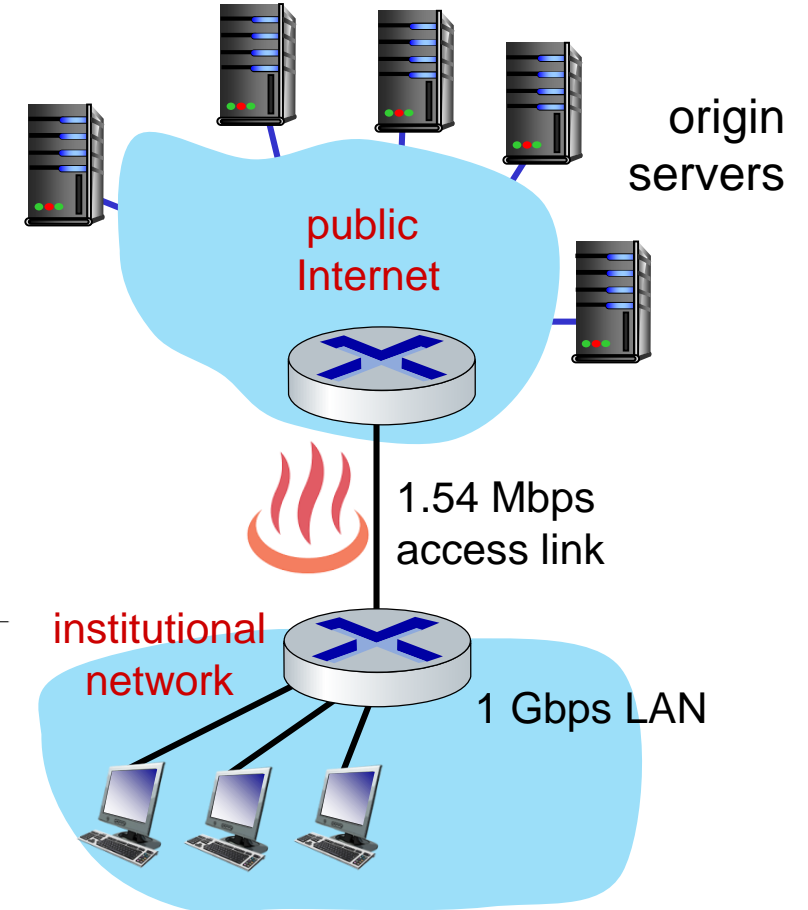
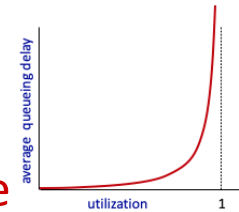
Scenario:

- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
 - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = **.97**
 - $[1.50/1.54 = 0.97]$
- LAN utilization: .0015
- end-end delay = Internet delay +
access link delay + LAN delay
= 2 sec + minutes + usecs

*problem: large
queueing delays
at high utilization!*



Option 1: buy a faster access link

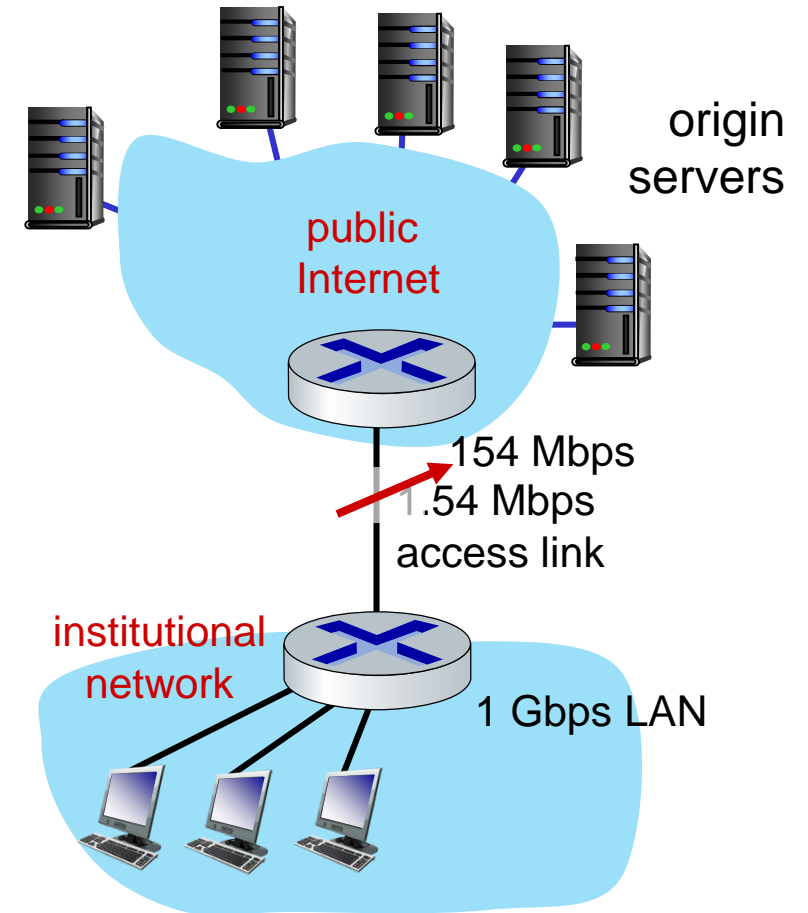
Scenario:

- access link rate: ~~1.54~~ 154 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
 - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = ~~.97~~ .0097
- LAN utilization: .0015
- end-end delay = Internet delay +
access link delay + LAN delay
= 2 sec + ~~minutes~~ + usecs

Cost: faster access link (expensive!) → msecs



Option 2: install a web cache

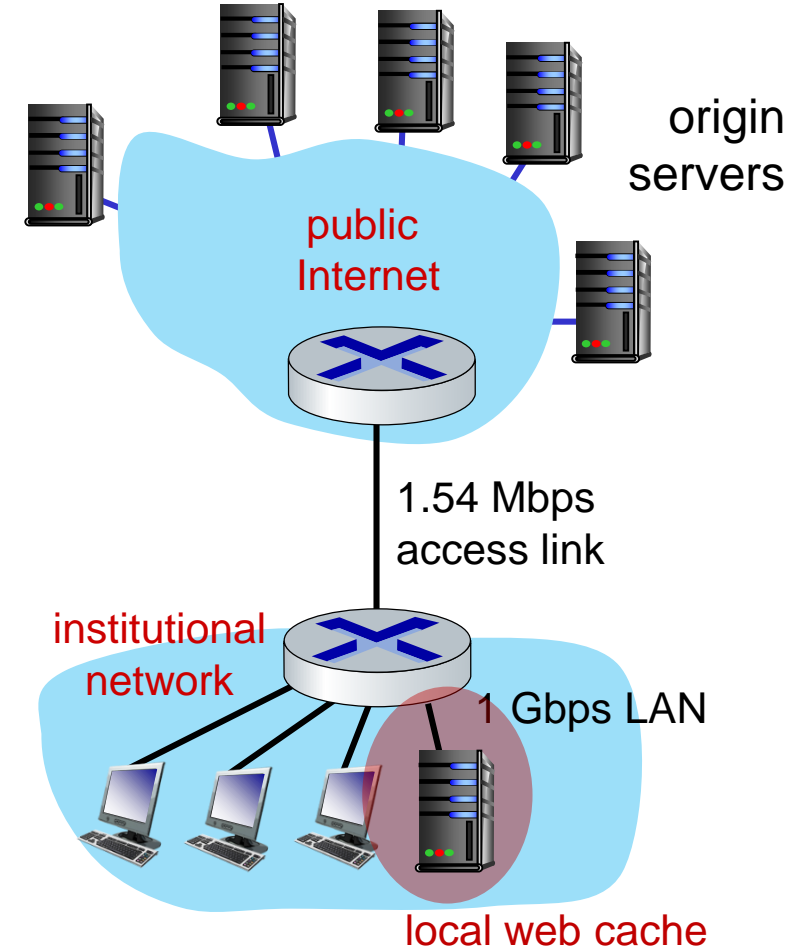
Scenario:

- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
 - avg data rate to browsers: 1.50 Mbps

Cost: web cache (cheap!)

Performance:

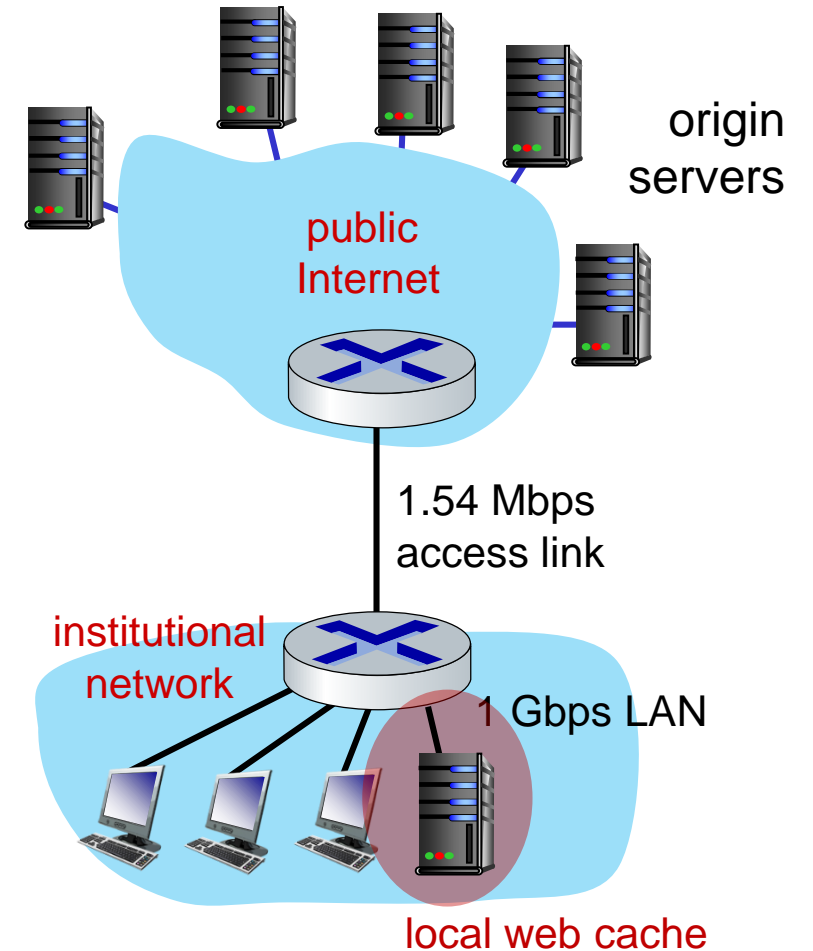
- LAN utilization: .?
 - access link utilization = ?
 - average end-end delay = ?
- How to compute link utilization, delay?*



Calculating access link utilization, end-end delay with cache:

suppose cache hit rate is 0.4:

- 40% requests served by cache, with low (msec) delay
- 60% requests satisfied at origin
 - rate to browsers over access link
 $= 0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$
 - access link utilization $= 0.9 / 1.54 = .58$ means low (msec) queueing delay at access link
- average end-end delay:
 $= 0.6 * (\text{delay from origin servers})$
 $+ 0.4 * (\text{delay when satisfied at cache})$
 $= 0.6 (2.01) + 0.4 (\sim \text{msecs}) = \sim 1.2 \text{ secs}$



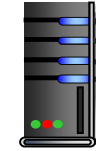
lower average end-end delay than with 154 Mbps link (and cheaper too!)

Conditional GET

client

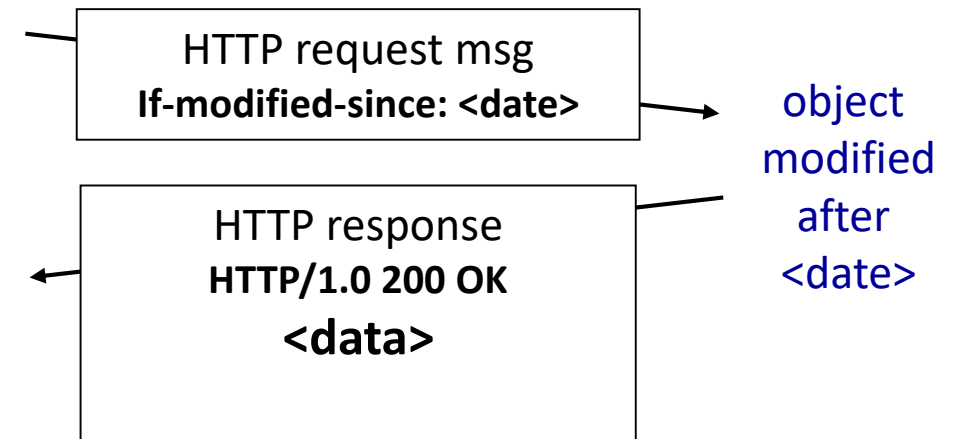
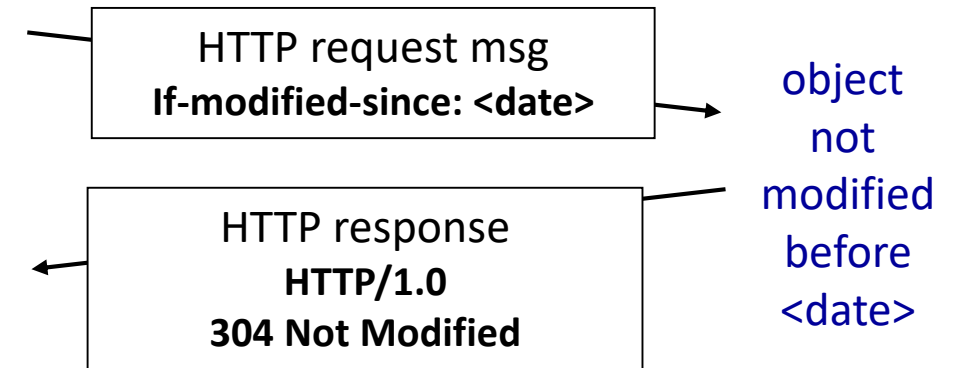


server



Goal: don't send object if cache has up-to-date cached version

- no object transmission delay (or use of network resources)
- **client:** specify date of cached copy in HTTP request
If-modified-since: <date>
- **server:** response contains no object if cached copy is up-to-date:
HTTP/1.0 304 Not Modified



HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP1.1:

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (**head-of-line (HOL) blocking**) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission

HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

- methods, status codes, most header fields unchanged from HTTP 1.1
- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)

HTTP/2 to HTTP/3

HTTP/2 over single TCP connection means:

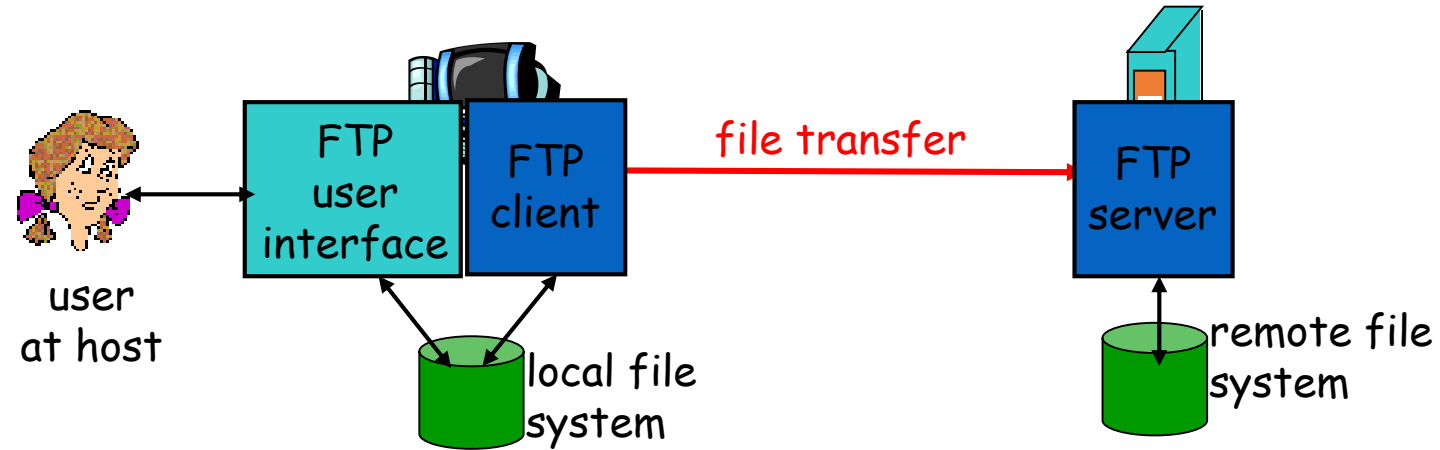
- recovery from packet loss still stalls all object transmissions
 - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
- no security over vanilla TCP connection
- **HTTP/3**: adds security, per object error- and congestion-control (more pipelining) over UDP

Application layer: overview

- Principles of network applications
- Web and HTTP
- FTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P Applications



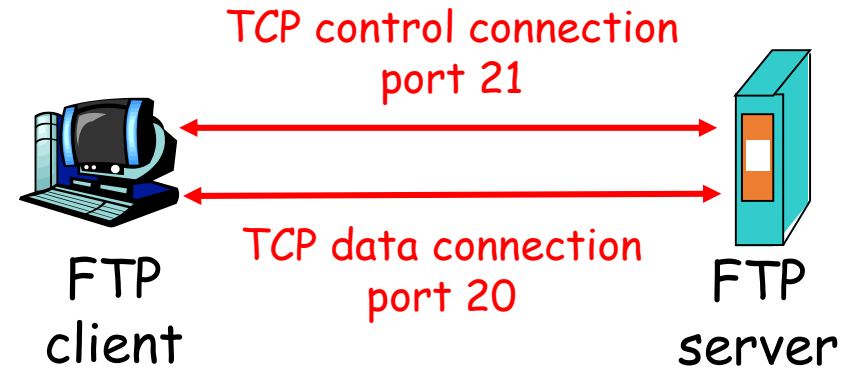
FTP: the file transfer protocol



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

FTP: separate control, data connections

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



- server opens another TCP data connection to transfer another file.
- control connection: **"out of band"**
- FTP server maintains "state": current directory, earlier authentication

HTTP vs FTP

- Both run on TCP
- FTP uses 2 parallel TCP connection to transfer files
 - Control connection
 - Sending control information between 2 hosts: identification, authorization, password, change remote directory, commands to “put” and “get” files
 - Data connection
 - Actually send the file
- FTP sends control information – *out-of-band*
- HTTP sends control information – *in-band*
- FTP maintains *state*
- HTTP is *stateless*

Application layer: overview

- Principles of network applications
- Web and HTTP
- FTP
- E-mail, SMTP
- The Domain Name System DNS
- P2P Applications



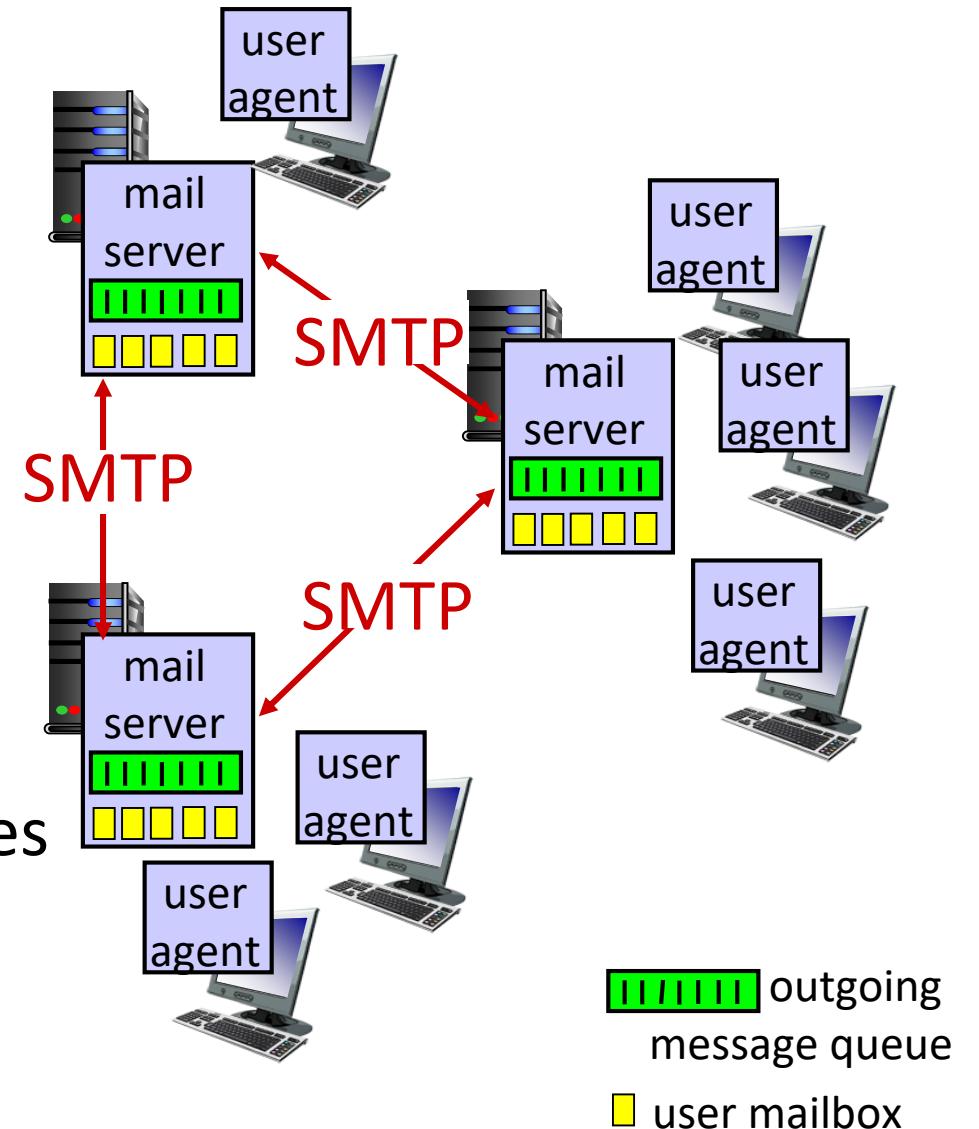
E-mail

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, iPhone mail client
- outgoing, incoming messages stored on server



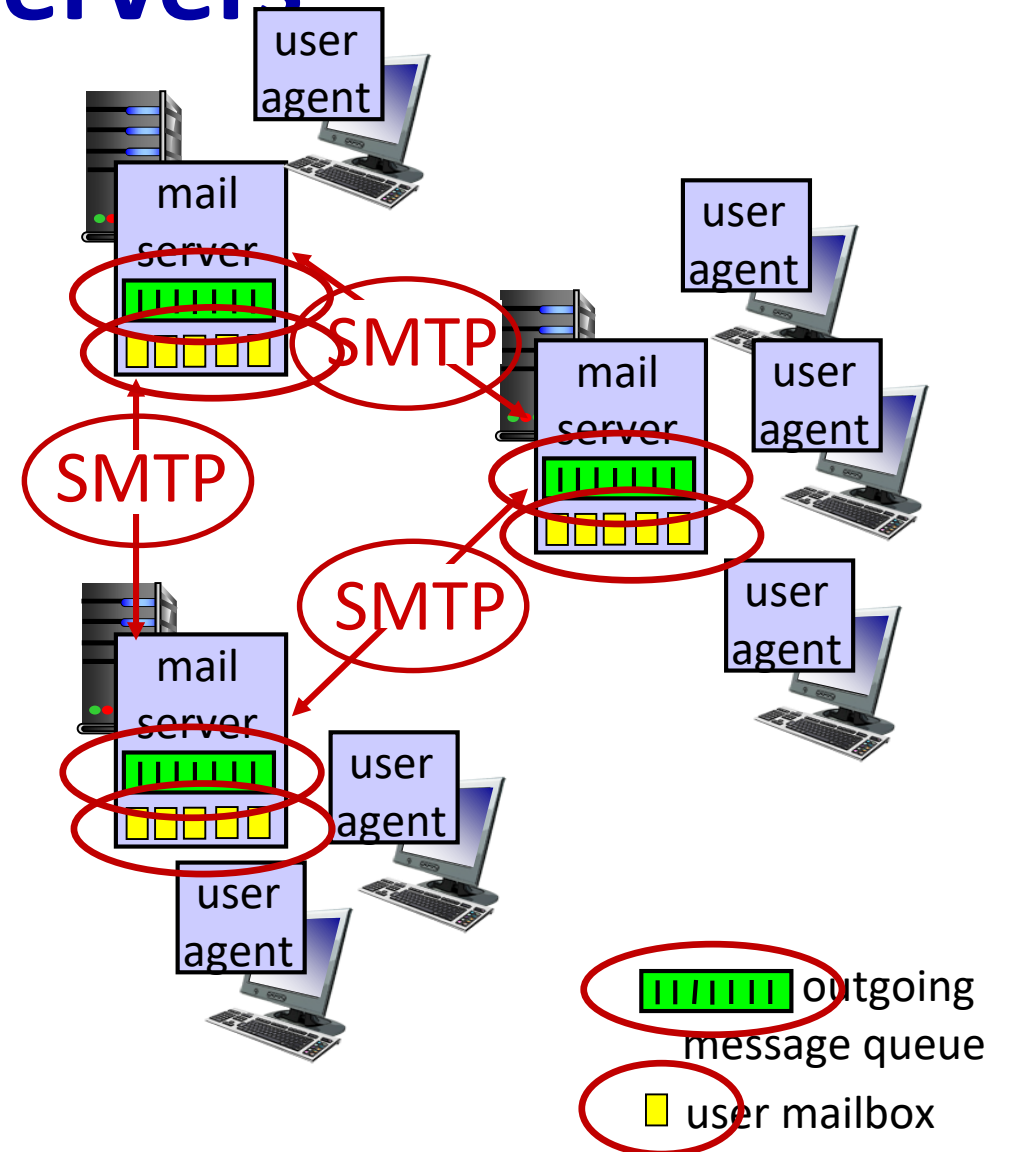
E-mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages

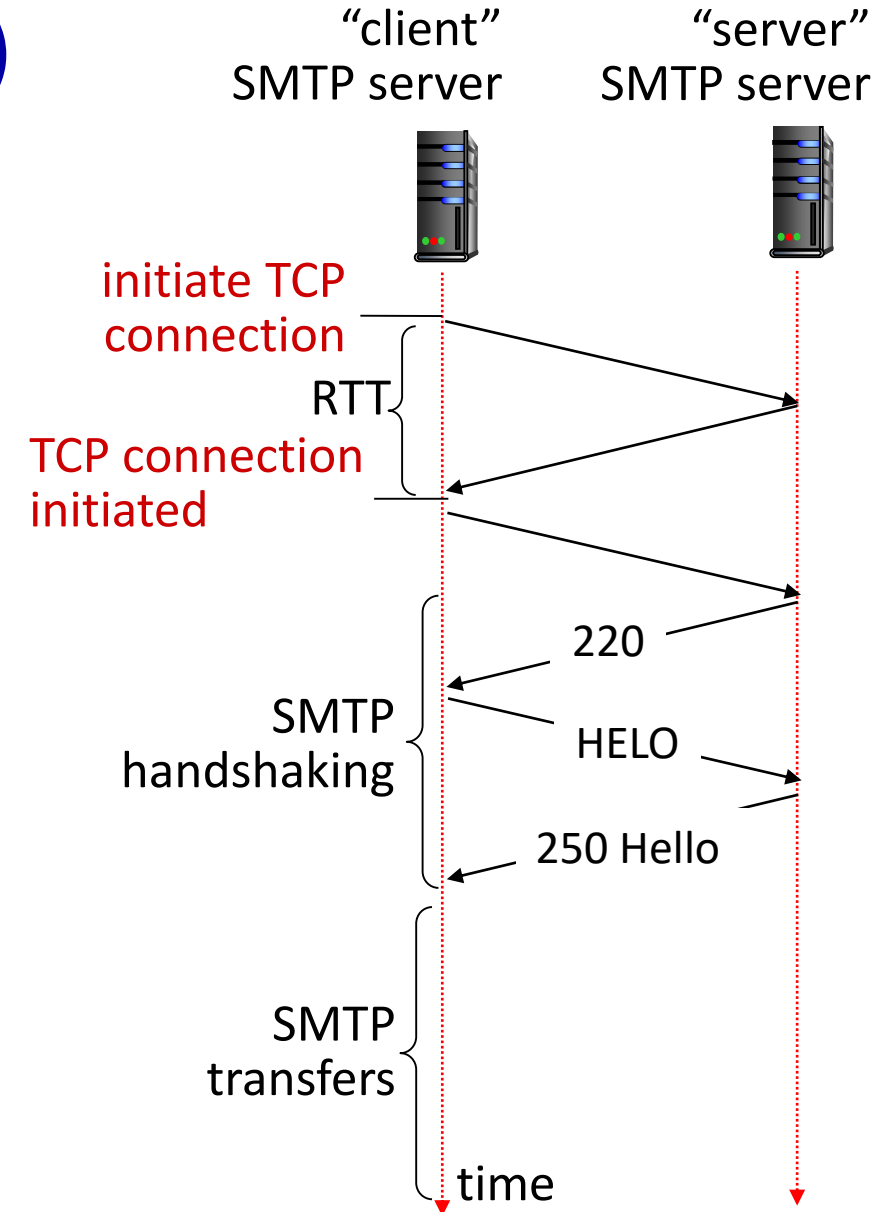
SMTP protocol between mail servers to send email messages

- **client**: sending mail server
- **“server”**: receiving mail server
- SMTP is older than HTTP
- Transfers messages from sender's mail server to recipient's



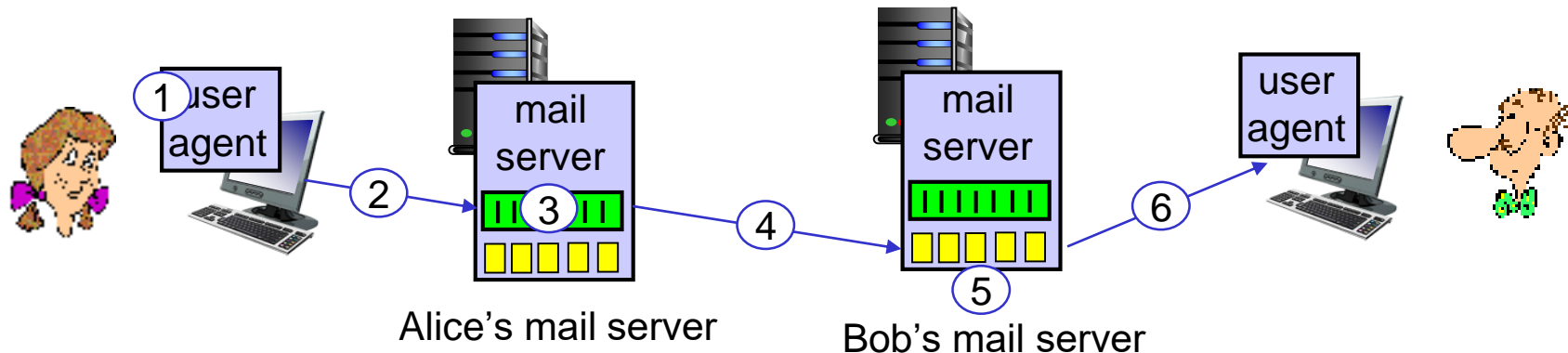
SMTP RFC (5321)

- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25
 - direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
 - SMTP handshaking (greeting)
 - SMTP transfer of messages
 - SMTP closure
- command/response interaction (like HTTP)
 - **commands**: ASCII text
 - **response**: status code and phrase



Scenario: Alice sends e-mail to Bob

- 1) Alice uses UA to compose e-mail message "to" bob@some school.edu
- 2) Alice's UA sends message to her mail server using SMTP; message placed in message queue
- 3) client side of SMTP at mail server 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 comparison with HTTP

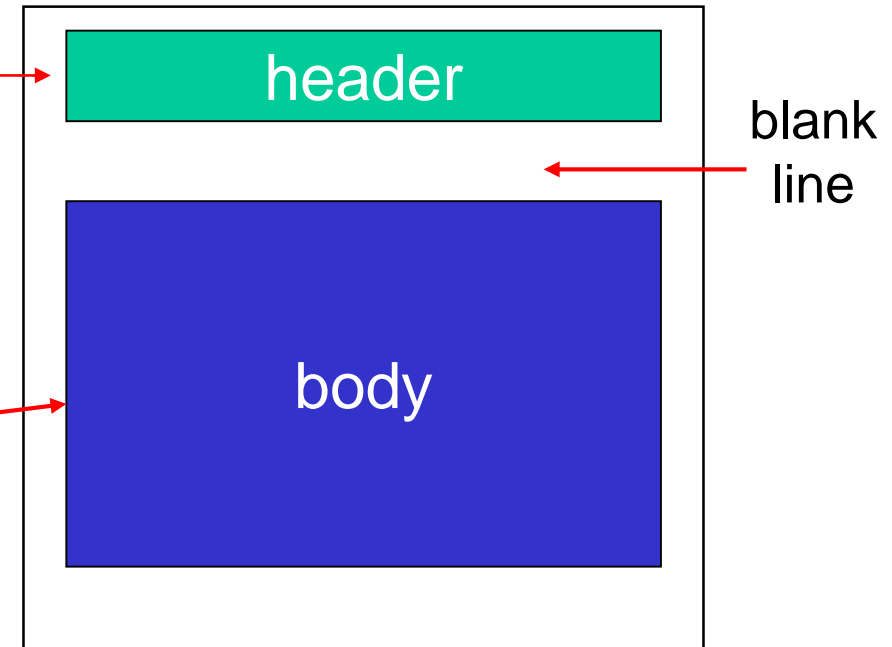
- Both transfer files
- Both uses persistent connections
- HTTP: client pull
- SMTP: client push
- SMTP requires message (header & body) to be in 7-bit ASCII
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in multipart message
- HTTP uses *in-band control*
- FTP uses *out-of-band control* – parallel TCP connections

Mail message format

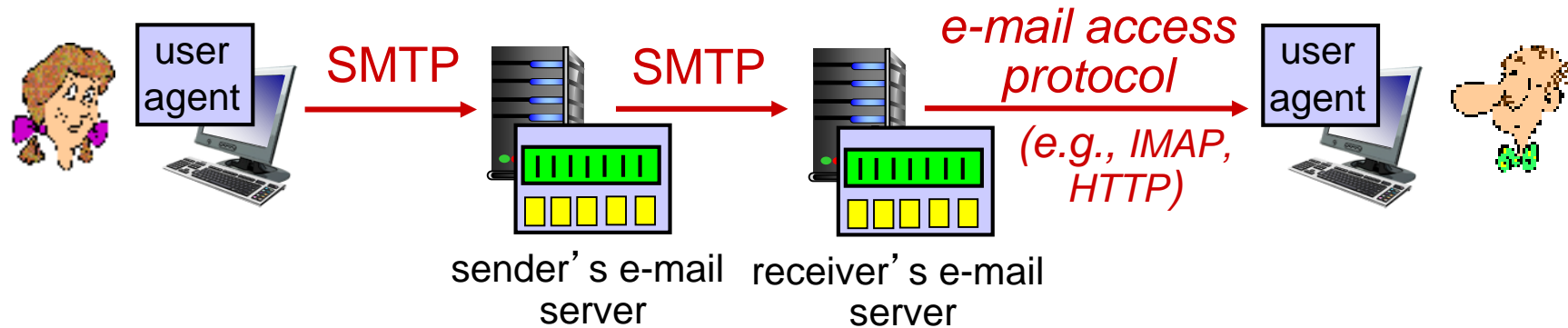
SMTP: protocol for exchanging e-mail messages, defined in RFC 5321 (like RFC 7231 defines HTTP)

RFC 2822 defines *syntax* for e-mail message itself (like HTML defines syntax for web documents)

- header lines, e.g.,
 - To:
 - From:
 - Subject:these lines, within the body of the email message area different from SMTP MAIL FROM:, RCPT TO: commands!
- Body: the “message” , ASCII characters only



Retrieving email: mail access protocols



- **SMTP:** delivery/storage of e-mail messages to receiver's server
- mail access protocol: retrieval from server
 - **IMAP:** Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
- **HTTP:** gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of SMTP (to send), IMAP (or POP) to retrieve e-mail messages

Application layer: overview

- Principles of network applications
- Web and HTTP
- FTP
- E-mail, SMTP
- The Domain Name System DNS
- P2P Applications



DNS: Domain Name System

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- Hostname - e.g., sjsu.edu - used by humans
- IP address (32 bit) - used for addressing datagrams

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, DNS servers communicate to *resolve* names (address/name translation)
 - *note*: core Internet function, implemented as application-layer protocol
 - complexity at network's "edge"

DNS: services, structure

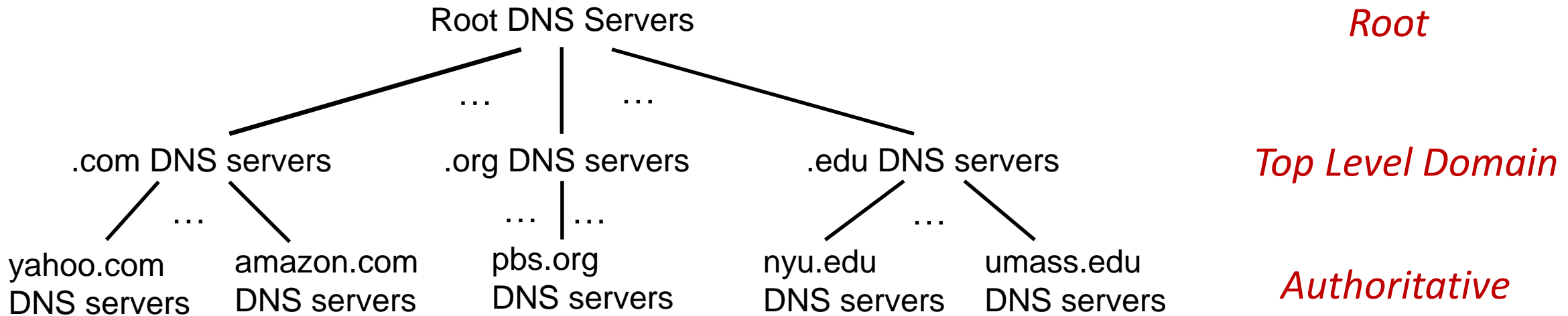
DNS services:

- hostname-to-IP-address translation
- load distribution
 - replicated Web servers: many IP addresses correspond to one name
- Runs over UDP
- Uses port 53

Q: Why not centralize DNS?

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

DNS: a distributed, hierarchical database

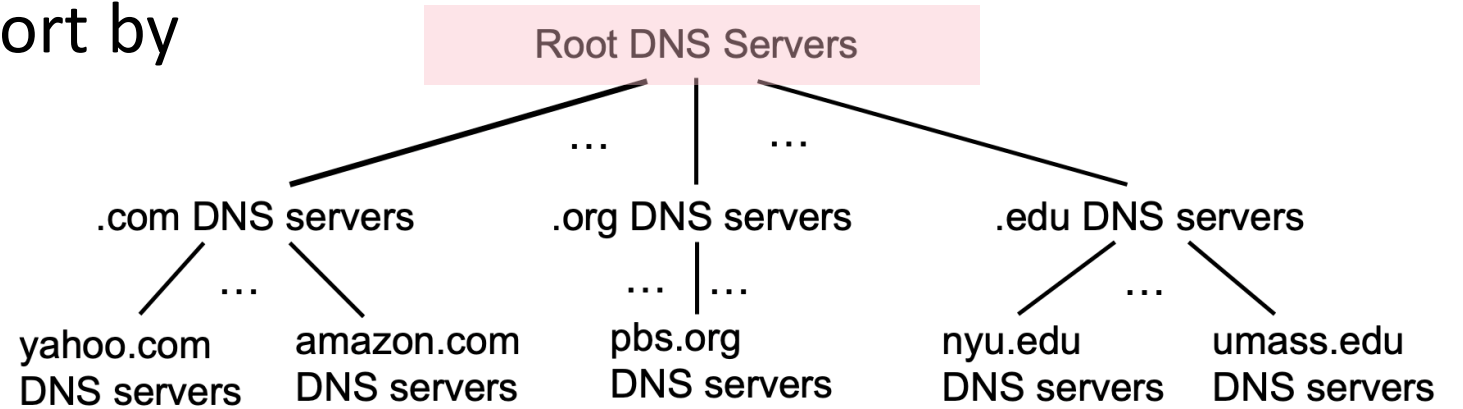


Client wants IP address for www.amazon.com; 1st approximation:

- 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

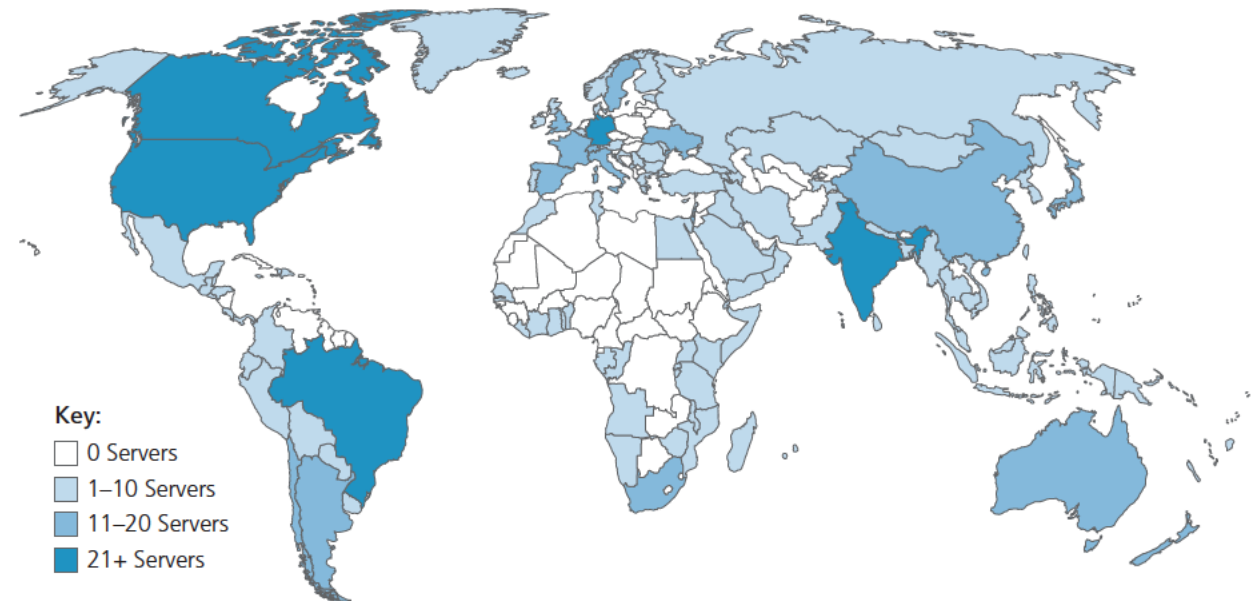
- official, contact-of-last-resort by name servers that can not resolve name



DNS: root name servers

- official, contact-of-last-resort by name servers that can not resolve name
- *incredibly important* Internet function
 - Internet couldn't function without it!
 - DNSSEC – provides security (authentication, message integrity)
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain

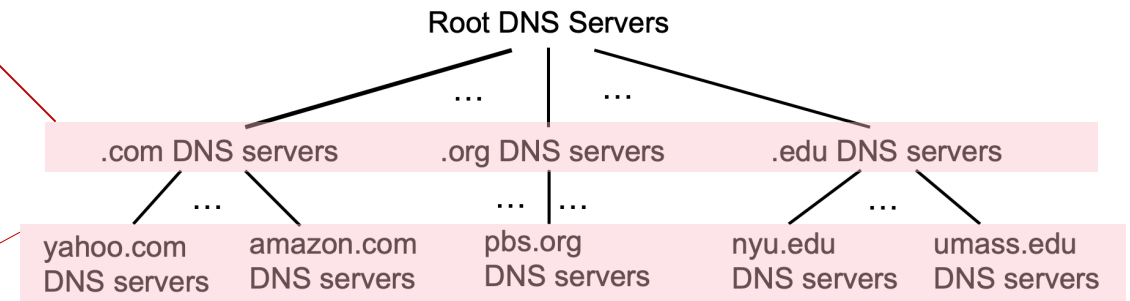
13 logical root name “servers”
worldwide each “server” replicated
many times (~200 servers in US)



Top-Level Domain, and authoritative servers

Top-Level Domain (TLD) servers:

- responsible for .com, .org, .net, .edu, .aero, .jobs, .museums, and all top-level country domains, e.g.: .cn, .uk, .fr, .ca, .jp
- Network Solutions: authoritative registry for .com, .net TLD
- Educause: .edu 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 servers

- when host makes DNS query, it is sent to its *local* DNS server
 - Local DNS server returns reply, answering:
 - from its local cache of recent name-to-address translation pairs (possibly out of date!)
 - forwarding request into DNS hierarchy for resolution
 - each ISP has local DNS name server; to find yours:
 - MacOS: `% scutil --dns`
 - Windows: `>ipconfig /all`

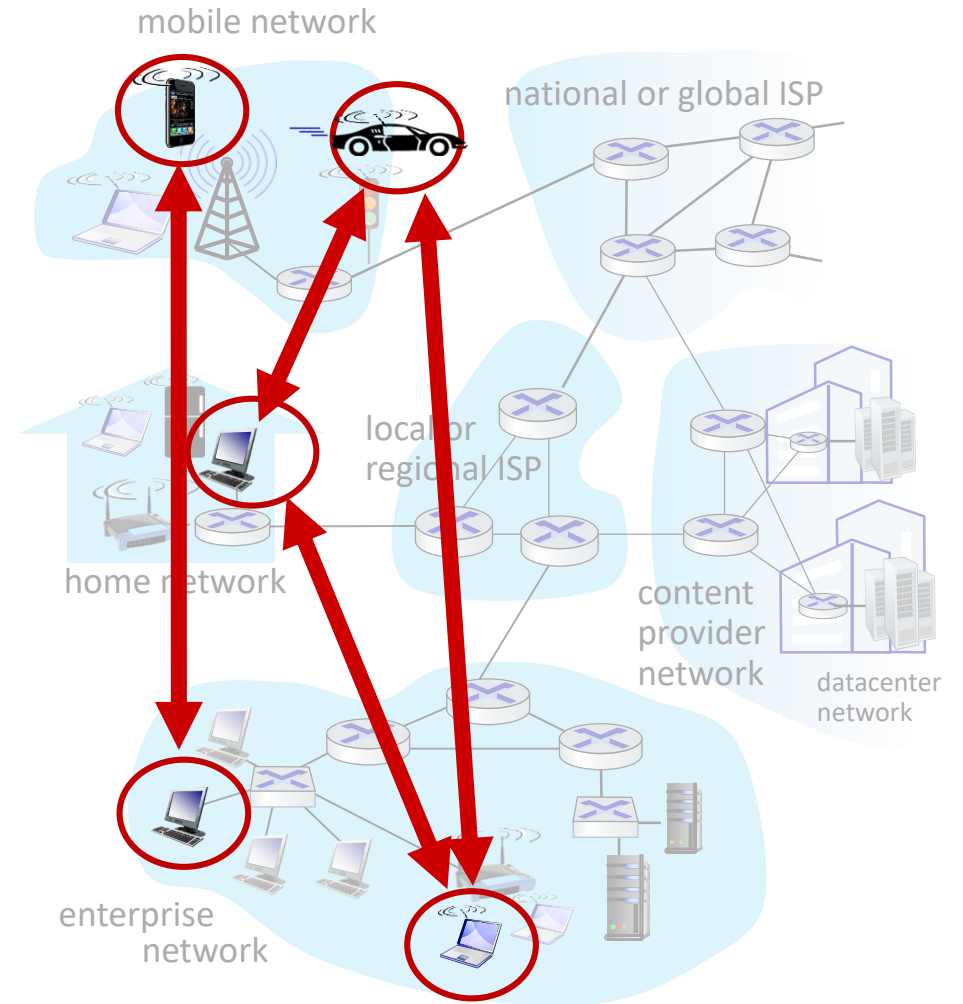
Application layer: overview

- Principles of network applications
- Web and HTTP
- FTP
- E-mail, SMTP
- The Domain Name System DNS
- P2P Applications



Peer-to-peer (P2P) architecture

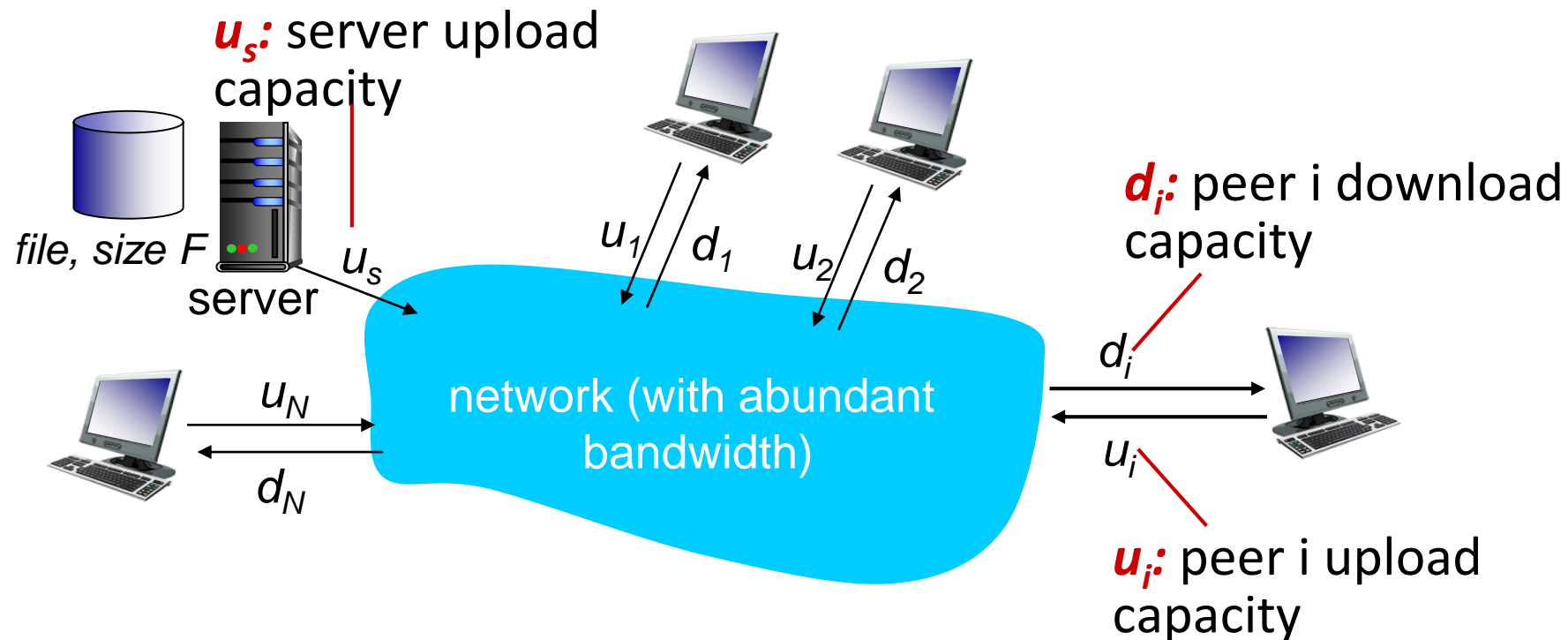
- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
 - *self scalability* – new peers bring new service capacity, and new service demands
- peers are intermittently connected and change IP addresses
 - complex management
- examples: P2P file sharing (BitTorrent), streaming (KanKan), VoIP (Skype)



File distribution: client-server vs P2P

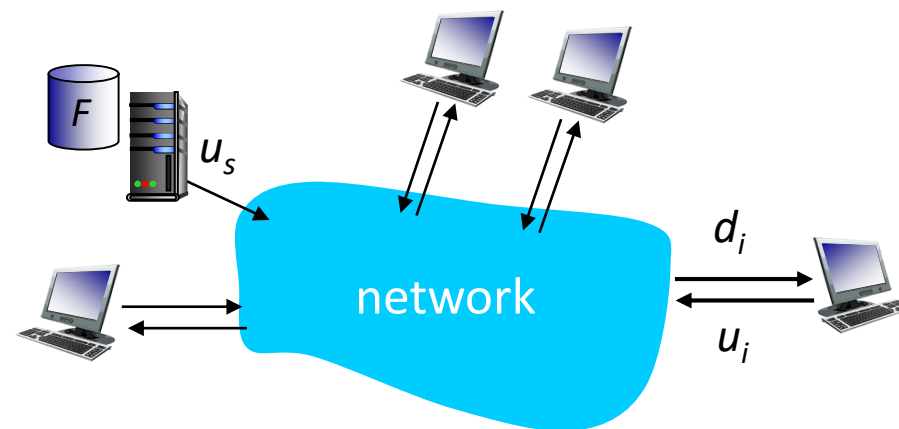
Q: how much time to distribute file (size F) from one server to N peers?

- 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*

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

increases linearly in N

File distribution time: P2P

- *server transmission*: must upload at least one copy:

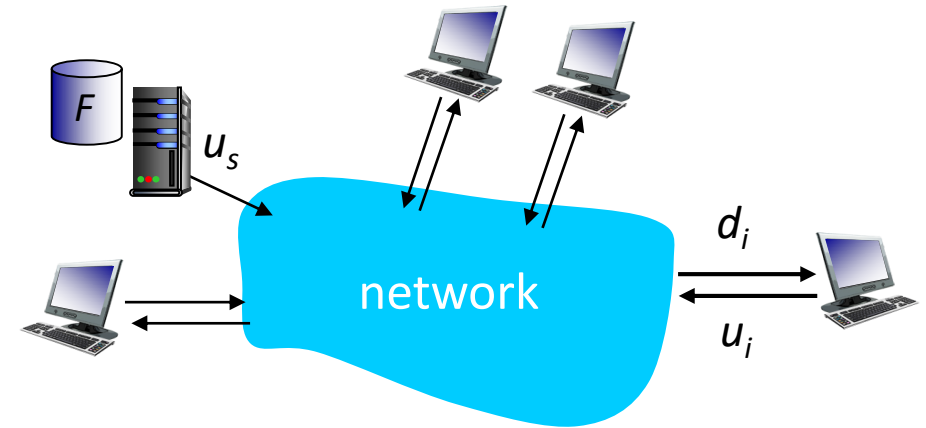
- time to send one copy: F/u_s

- *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 $u_s + \sum u_i$



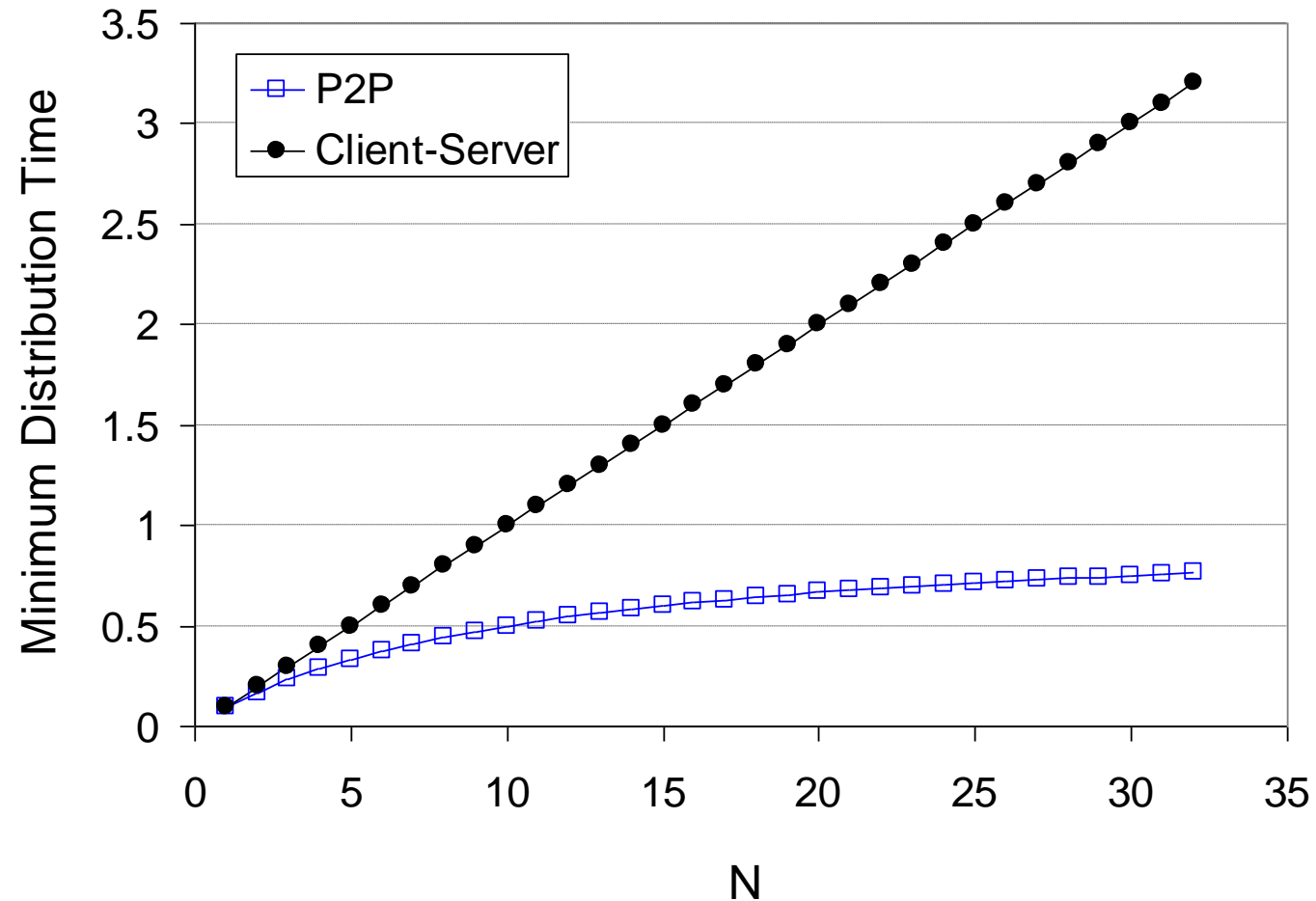
*time to distribute F
to N clients using
P2P approach*

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

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, $u_s = 10u$, $d_{min} \geq u_s$



Application layer: overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP



Video Streaming and CDNs: context

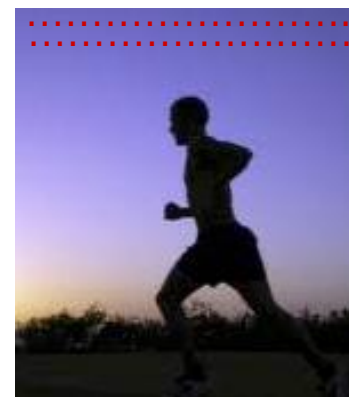
- stream video traffic: major consumer of Internet bandwidth
 - Netflix, YouTube, Amazon Prime: 80% of residential ISP traffic (2020)
- *challenge*: scale - how to reach ~1B users?
- *challenge*: heterogeneity
 - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *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

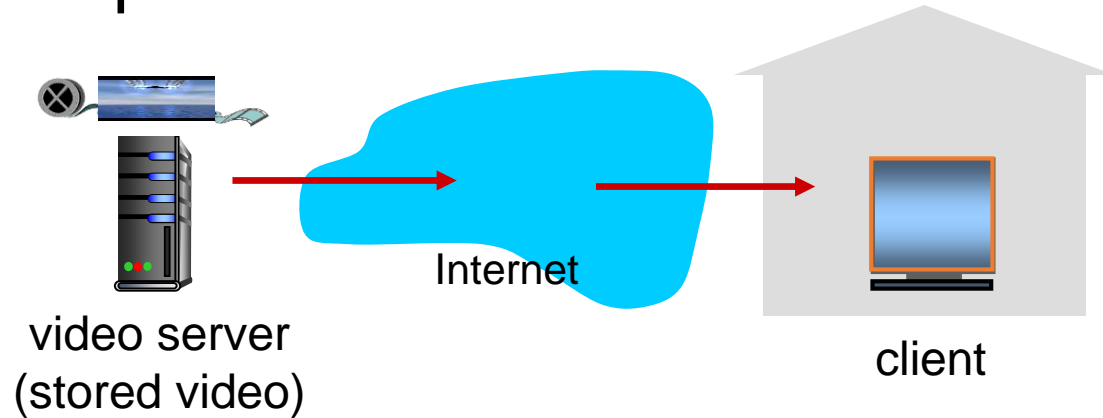
temporal coding example: instead of sending complete frame at $i+1$, send only differences from frame i



frame $i+1$

Streaming stored video

simple scenario:



Main challenges:

- server-to-client bandwidth will *vary* over time, with changing network congestion levels (in house, access network, network core, video server)
- packet loss, delay due to congestion will delay playout, or result in poor video quality

Application Layer: Overview

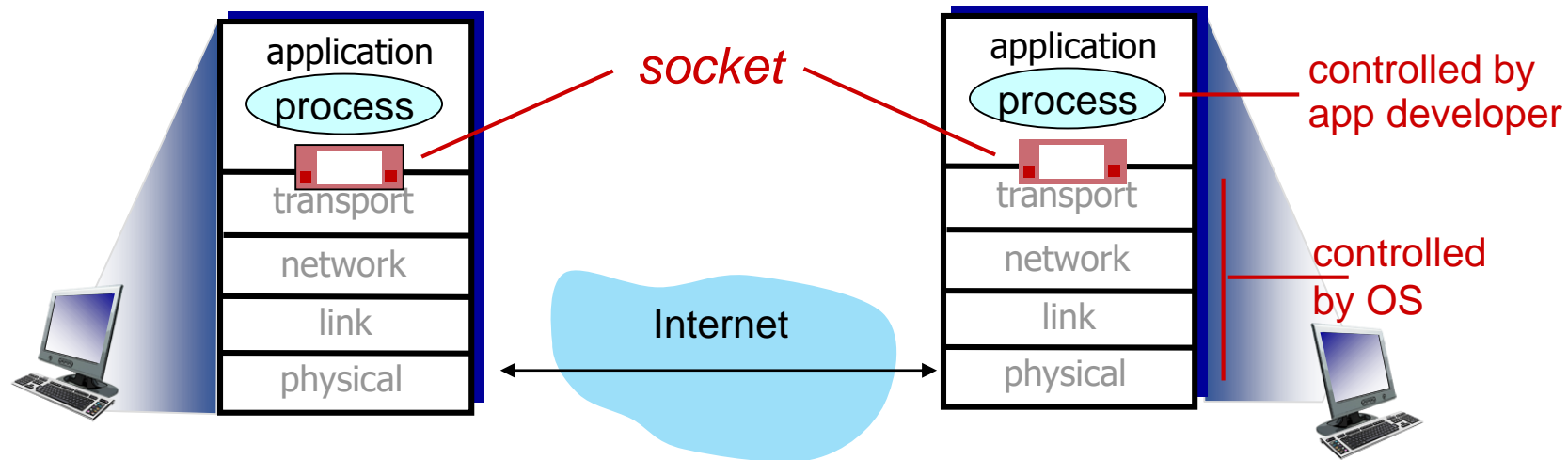
- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP



Socket programming

Learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



Socket programming

Two socket types for two transport services:

- *UDP*: unreliable datagram
- *TCP*: reliable, byte stream-oriented

Application Example:

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

Socket programming with UDP

UDP: no “connection” between client and server:

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes (“datagrams”) between client and server processes

Chapter 2: Summary

- 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
 - SMTP, IMAP
 - DNS
 - P2P: BitTorrent
- video streaming
- socket programming:
TCP, UDP sockets

Chapter 2

Application Layer

A note on the use of these PowerPoint slides:

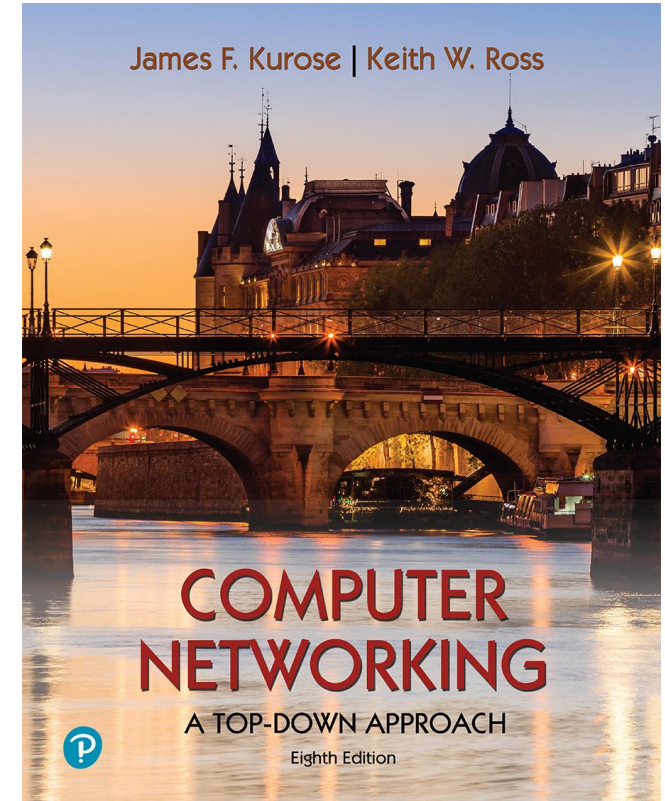
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2020
J.F Kurose and K.W. Ross, All Rights Reserved



Computer Networking: A Top-Down Approach

8th edition n
Jim Kurose, Keith Ross
Pearson, 2020

Thank you