# 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

Some network apps

- social networking
- Web
- text messaging
- e-mail
- multi-user network games
- streaming stored video (YouTube, Netflix)
- P2P file sharing
- voice over IP (Skype)
- real-time video conferencing (Zoom)
- Internet search
- remote login
- ...
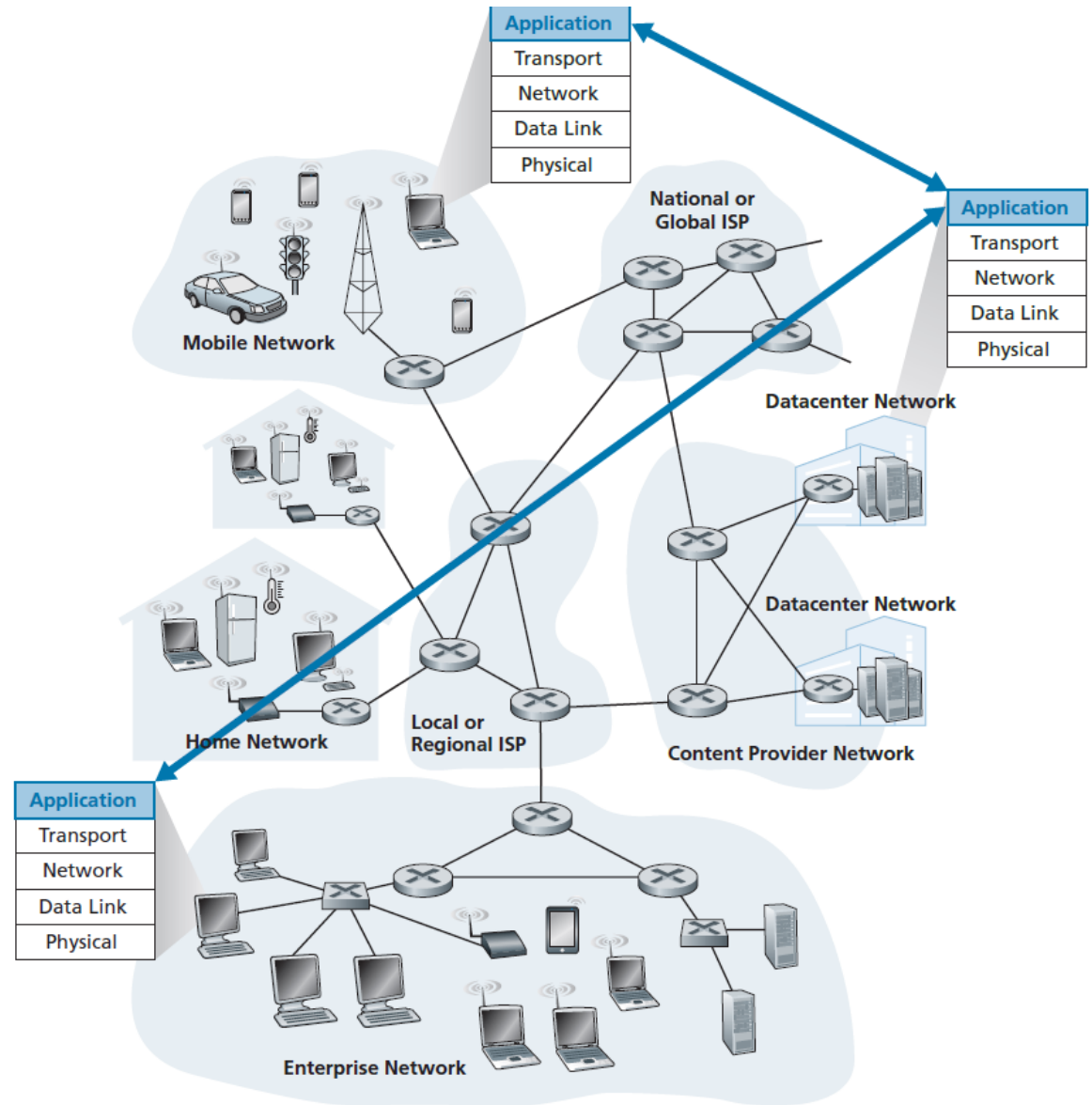
Creating a network app

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
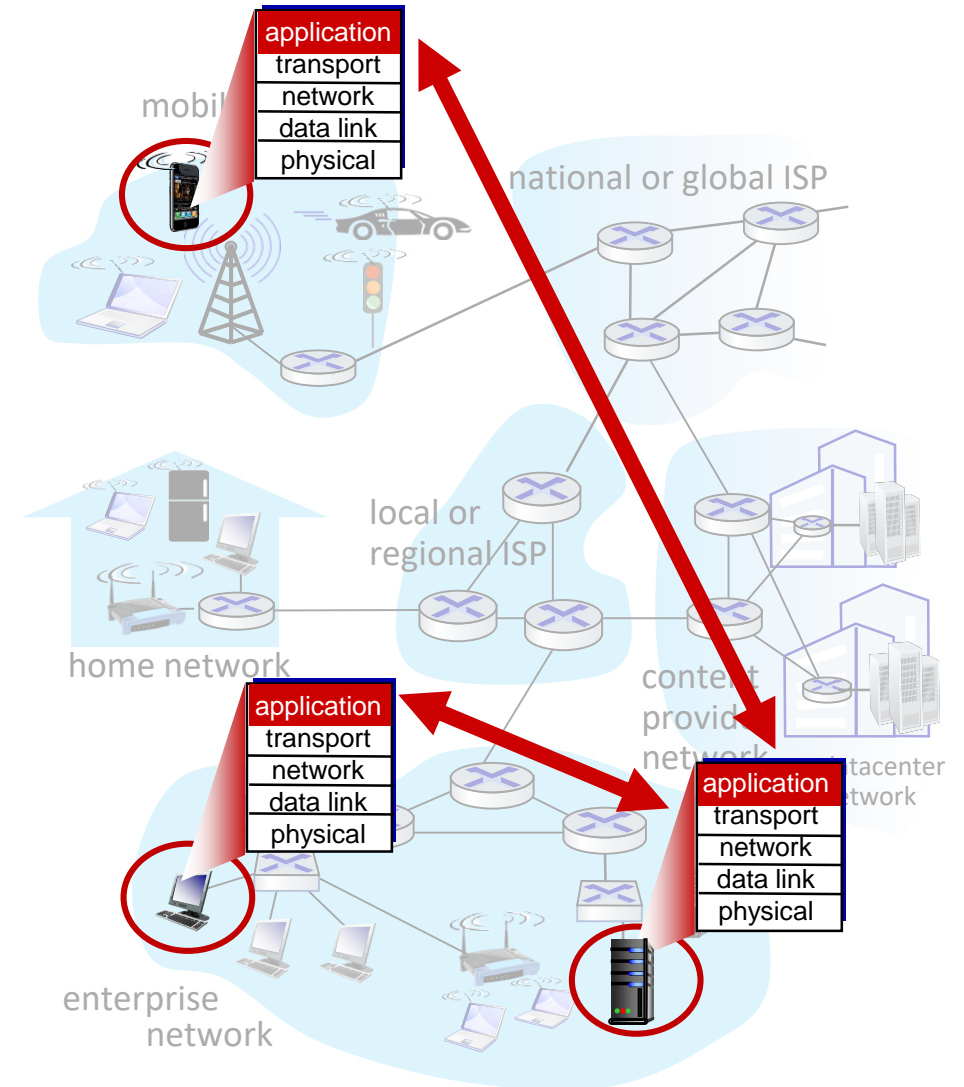
Client-server paradigm

server:
- always-on host
- permanent IP address
- often in data centers, for scaling

clients:
- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other
- examples: HTTP, IMAP, FTP

mobile network

national or global ISP

home network

local or regional ISP

content provider network

datacenter network

enterprise network
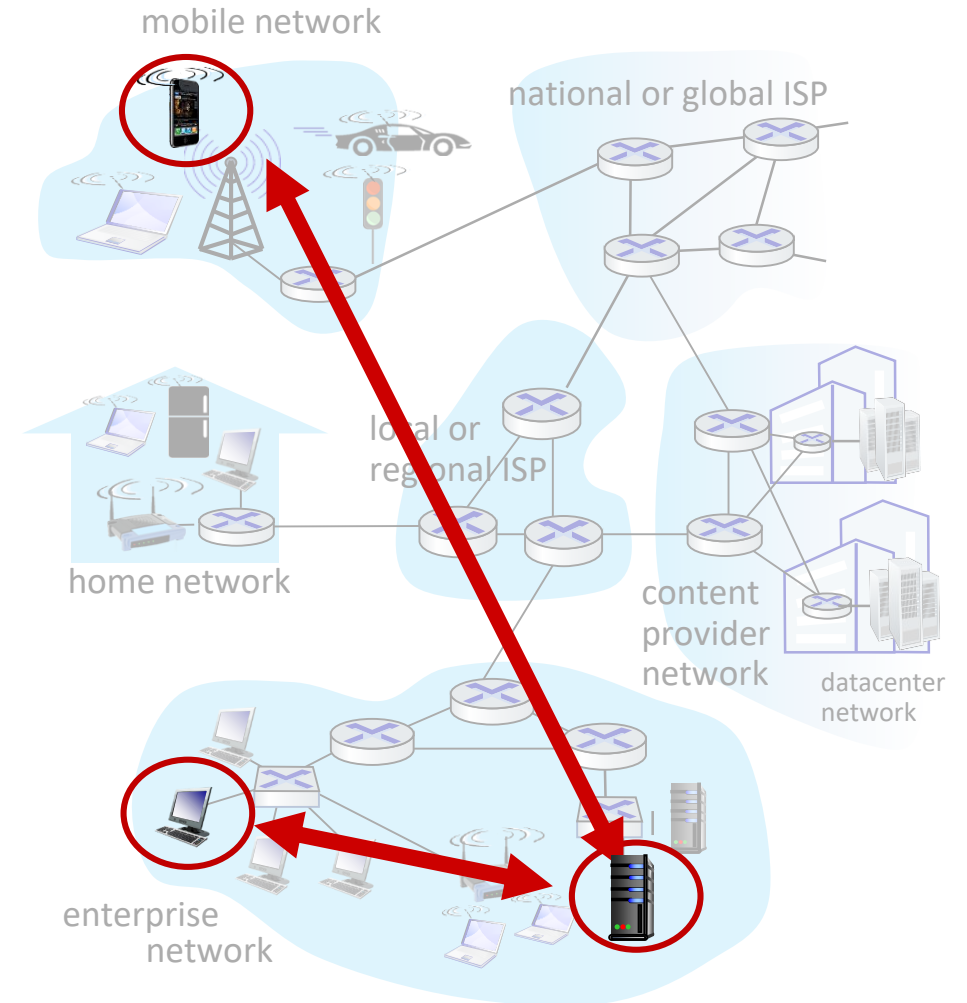
Peer-peer architecture

- *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
- example: P2P file sharing
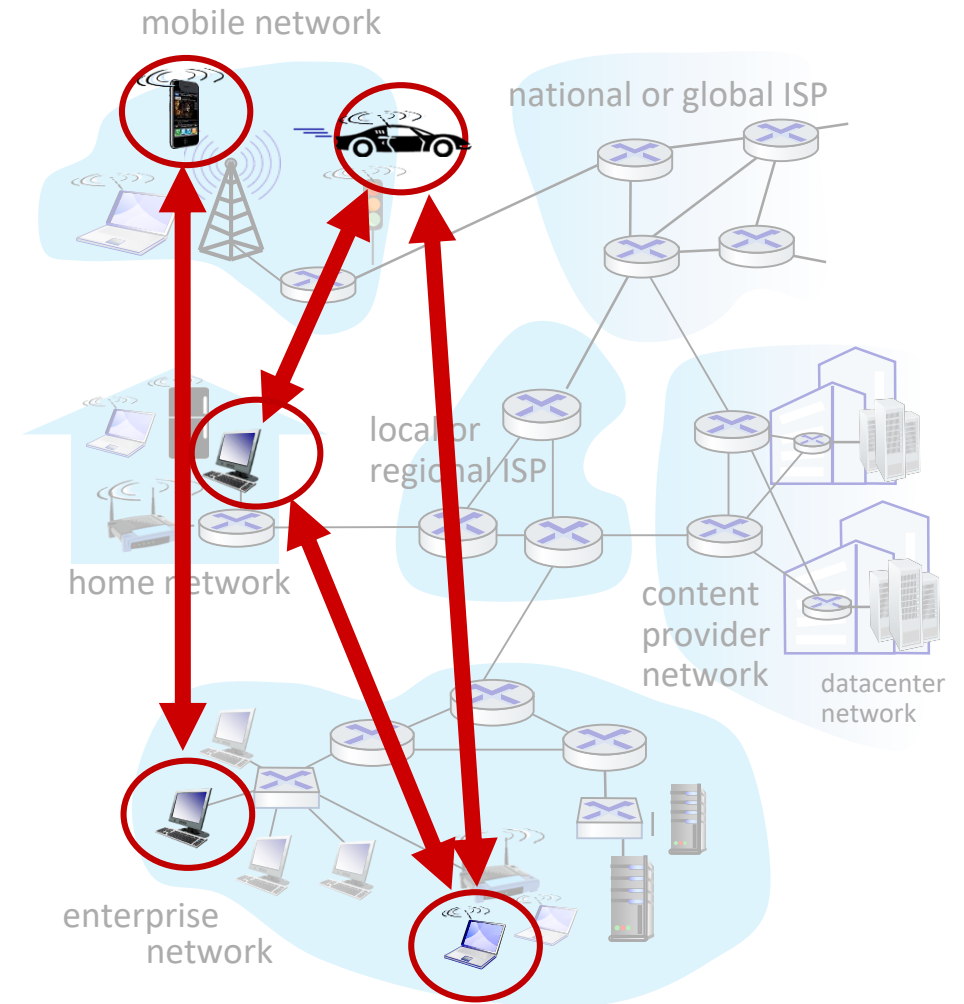
Processes communicating

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

The slides are based on the slides by Computer Networking: A Top-Down Approach 8ᵗʰ edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-6

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
  - two sockets involved: one on each side
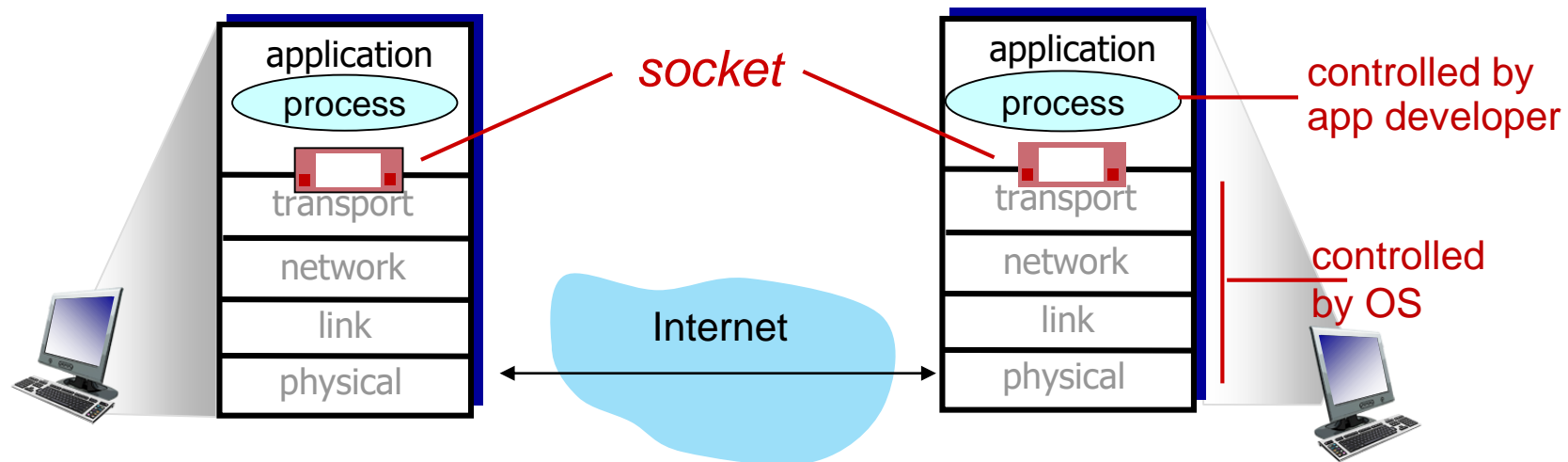
Addressing processes

- 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

An application-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, everyone has access to protocol definition
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:
- e.g., Skype, Zoom

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-9

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/download	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, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	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

- *connection-oriented:* setup required between client and server processes

- *does not provide:* timing, minimum throughput guarantee, security

UDP service:

- *unreliable data transfer* between sending and receiving process

- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Internet applications, and transport protocols

application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP 1.1 [RFC 7320]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7320], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

Securing TCP

Vanilla TCP & UDP sockets:

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

Transport Layer Security (TLS)

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

TLS implemented in application layer

- apps use TLS libraries, that use TCP in turn
- cleartext sent into "socket" traverse Internet *encrypted*

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

Web and HTTP

- web page consists of *objects,* each of which can be stored on different Web servers

- object can be HTML file, JPEG image, Java applet, audio file,...

- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,

`www.someschool.edu/someDept/pic.gif`

host name   path name

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-16

HTTP overview

HTTP: hypertext transfer protocol

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

PC running
Firefox browser

HTTP request

HTTP response

HTTP request

HTTP response

server running
Apache Web
server

iPhone running
Safari browser

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020
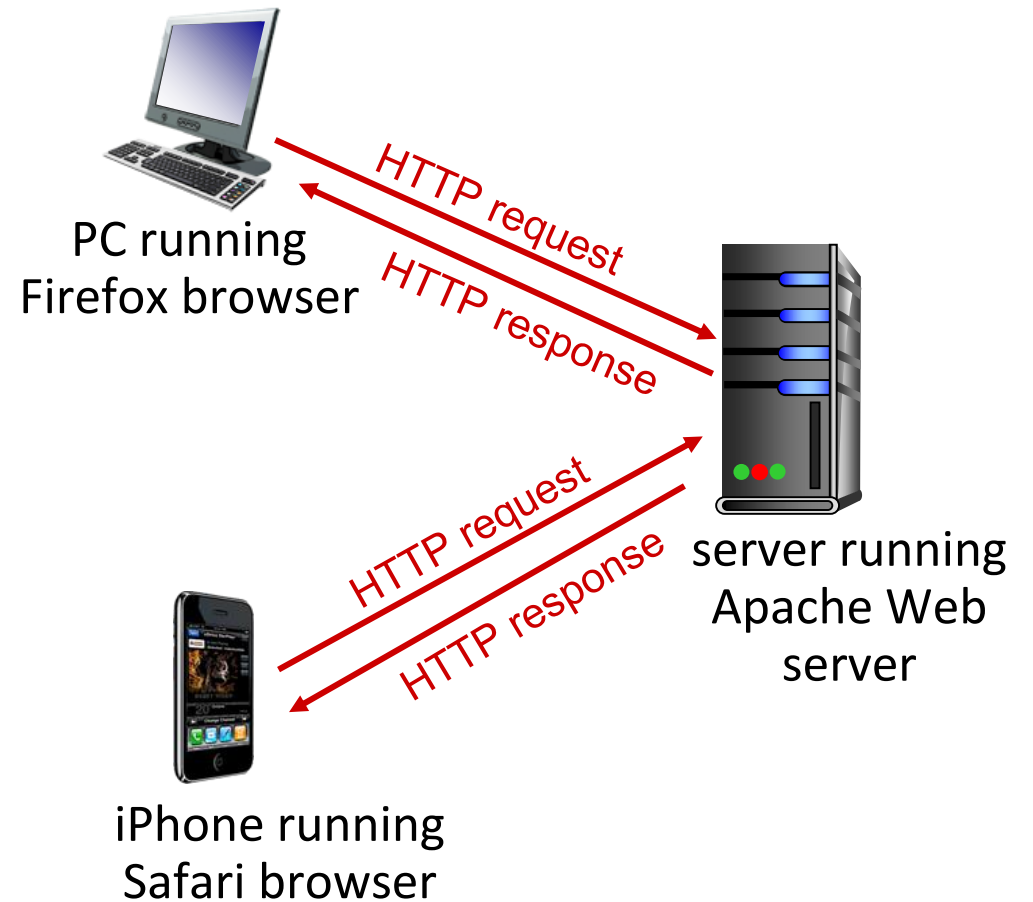
HTTP overview (continued)

HTTP uses TCP:

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

HTTP is "stateless"

- server maintains *no* information about past client requests

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

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-18

HTTP connections: two types

Non-persistent HTTP

1. TCP connection opened

2. at most one object sent over TCP connection

3. TCP connection closed

downloading multiple objects required multiple connections

Persistent HTTP

- TCP connection opened to a server

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

- TCP connection closed

Non-persistent HTTP: example

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

(containing text, references to 10 jpeg images)

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

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80 "accepts" connection, notifying client

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

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

time

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Non-persistent HTTP: example (cont.)

User enters URL: `www.someSchool.edu/someDepartment/home.index`
(containing text, references to 10 jpeg images)

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
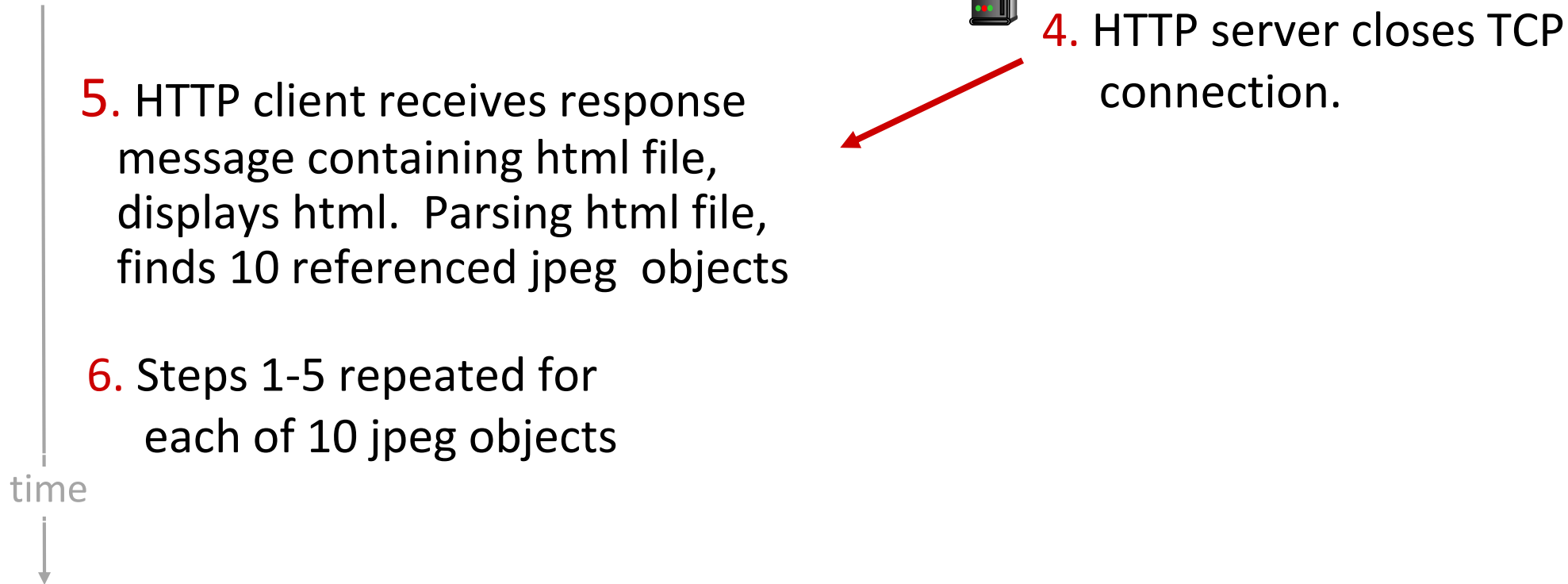
time

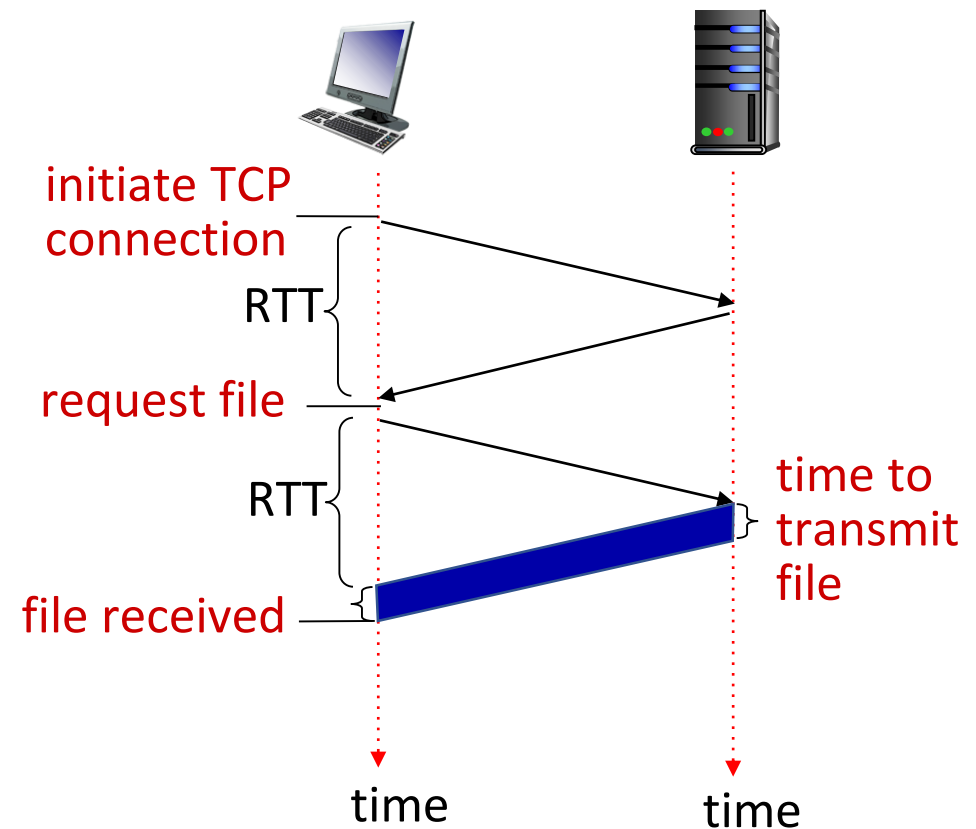# Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time (per object):
- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- object/file transmission time

Non-persistent HTTP response time = 2RTT+ file transmission time

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open multiple parallel TCP connections to fetch referenced objects in parallel

Persistent HTTP (HTTP1.1):

- 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 (cutting response time in half)

HTTP request message

- two types of HTTP messages: *request, response*

- HTTP request message:
  - ASCII (human-readable format)

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

HTTP request message: general format

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-25
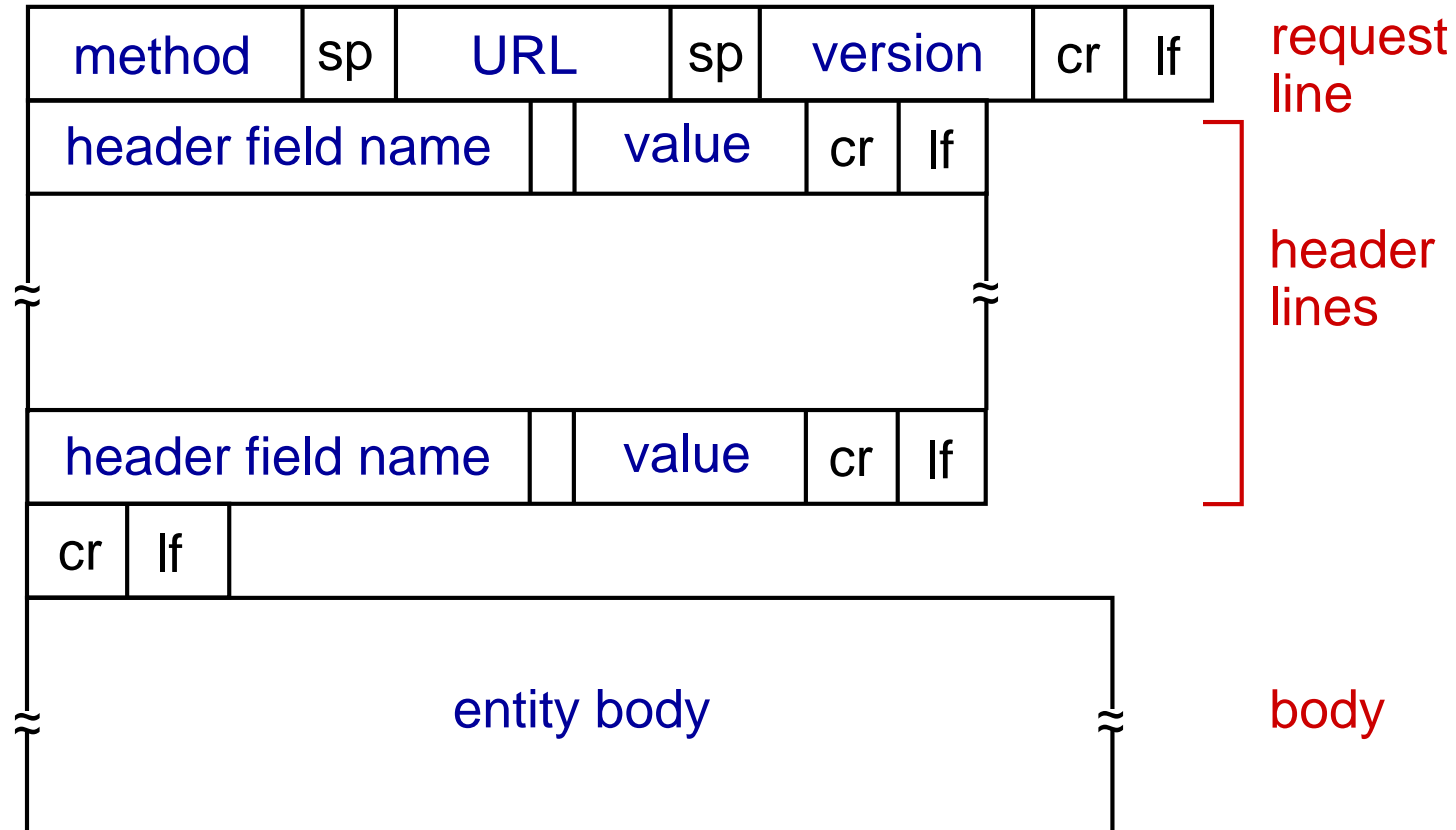
Other HTTP request messages

POST method:

- web page often includes form input

- user input sent from client to server in entity body of HTTP POST request message

GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a '?'):

www.somesite.com/search?name&telephoneNumber

HEAD method:

- for debugging

PUT method:

- uploads new file (object) to server

HTTP response message

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

HTTP response status codes

▪ status code appears in 1st line in server-to-client response message.
▪ some sample codes:

200 OK
  • request succeeded, requested object later in this message

301 Moved Permanently
  • requested object moved, new location specified later in this message (in Location: field)

400 Bad Request
  • request msg not understood by server

404 Not Found
  • requested document not found on this server

505 HTTP Version Not Supported

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

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-30
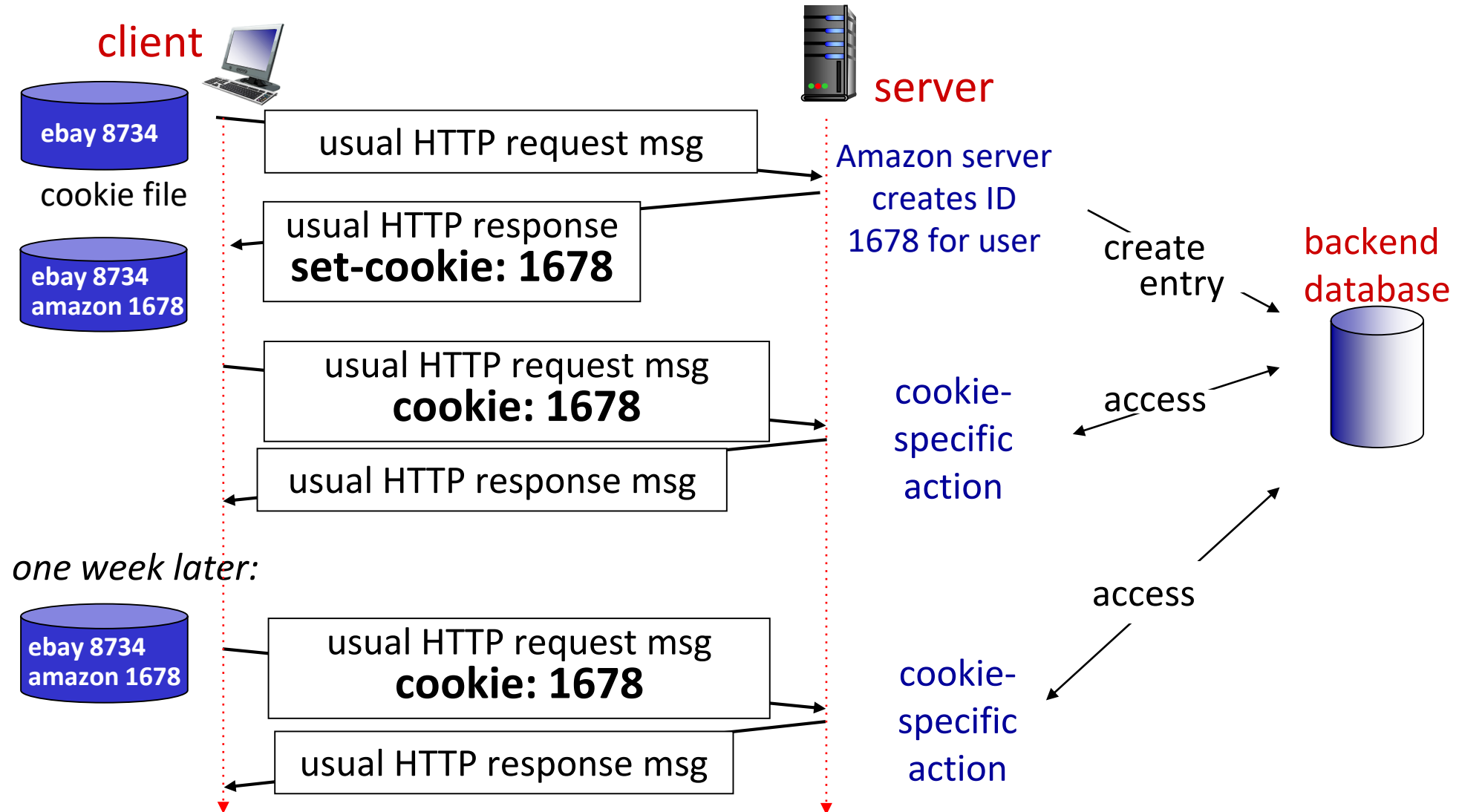
Web caches

Goal: satisfy client requests without involving origin server

- 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

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-31
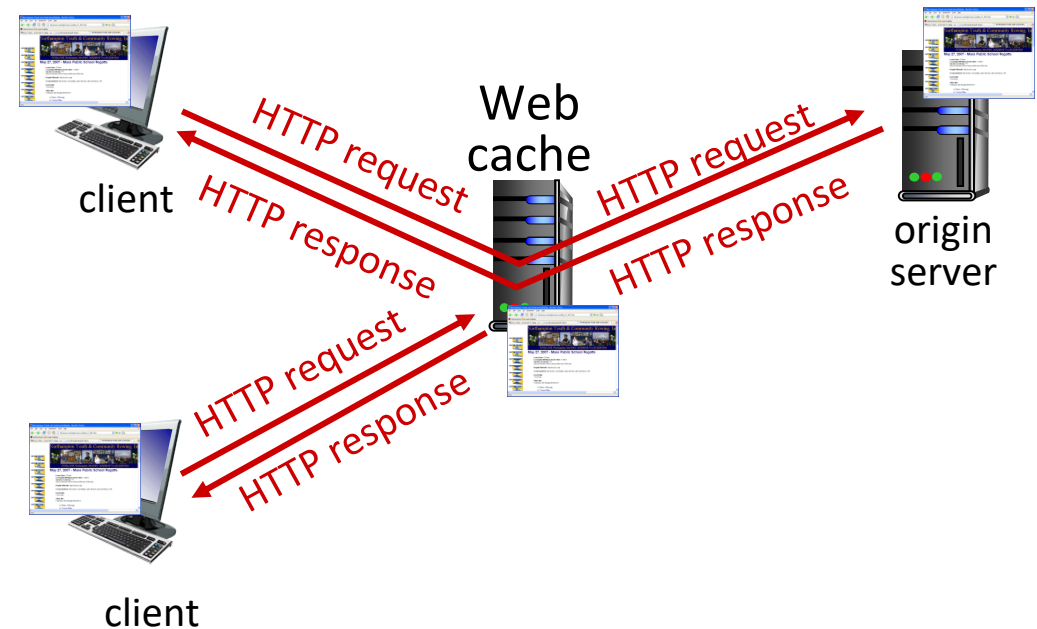
HTTP/2

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

HTTP/2: mitigating HOL blocking

HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects

objects delivered in order requested: O_2, O_3, O_4 wait behind O_1
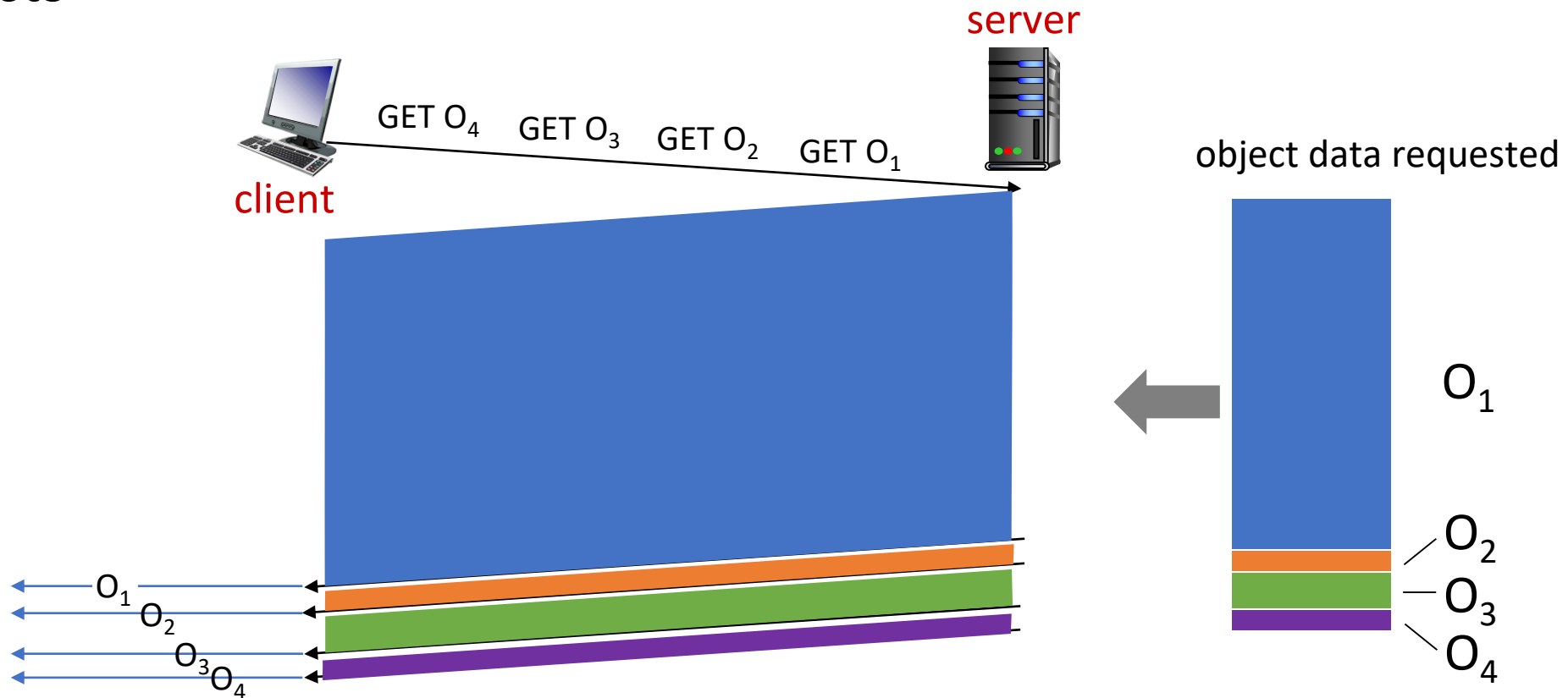
HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved

server

GET O_4 GET O_3 GET O_2 GET O_1

client

object data requested

O_2

O_4

O_3

O_1

O_1

O_2

O_3

O_4
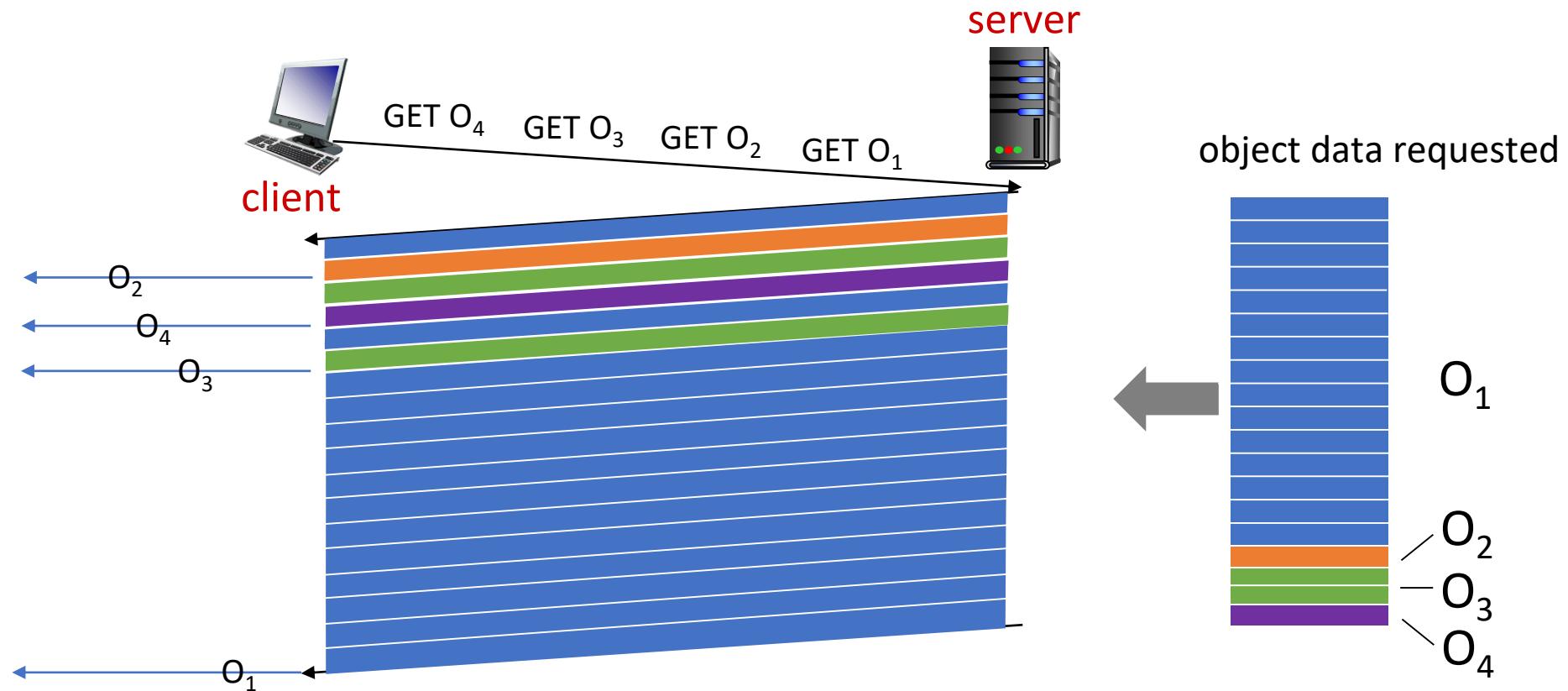
O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

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

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

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

user agent

mail server

SMTP

SMTP

user agent

mail server

user agent

SMTP

mail server

user agent

user agent

user agent

outgoing message queue
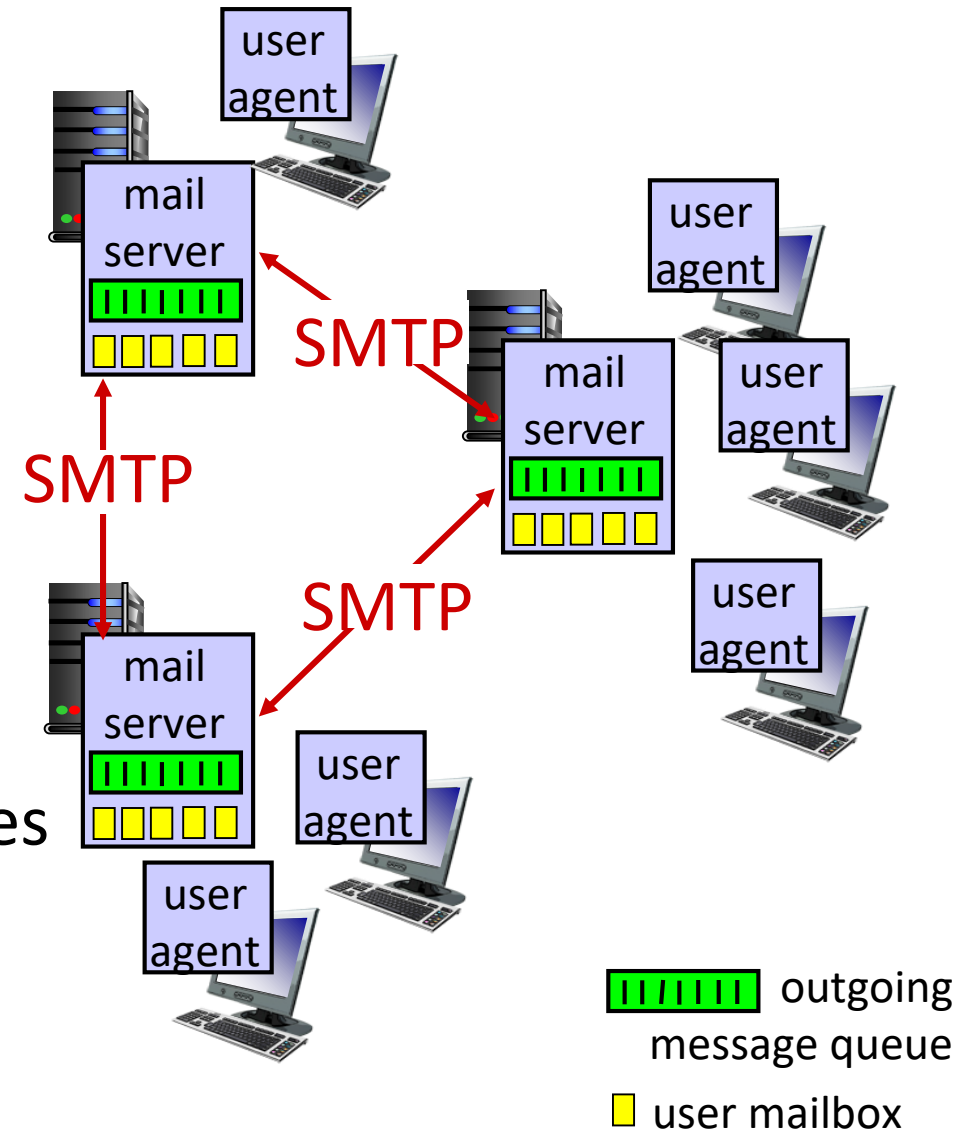
user mailbox

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

outgoing message queue

user mailbox

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-37
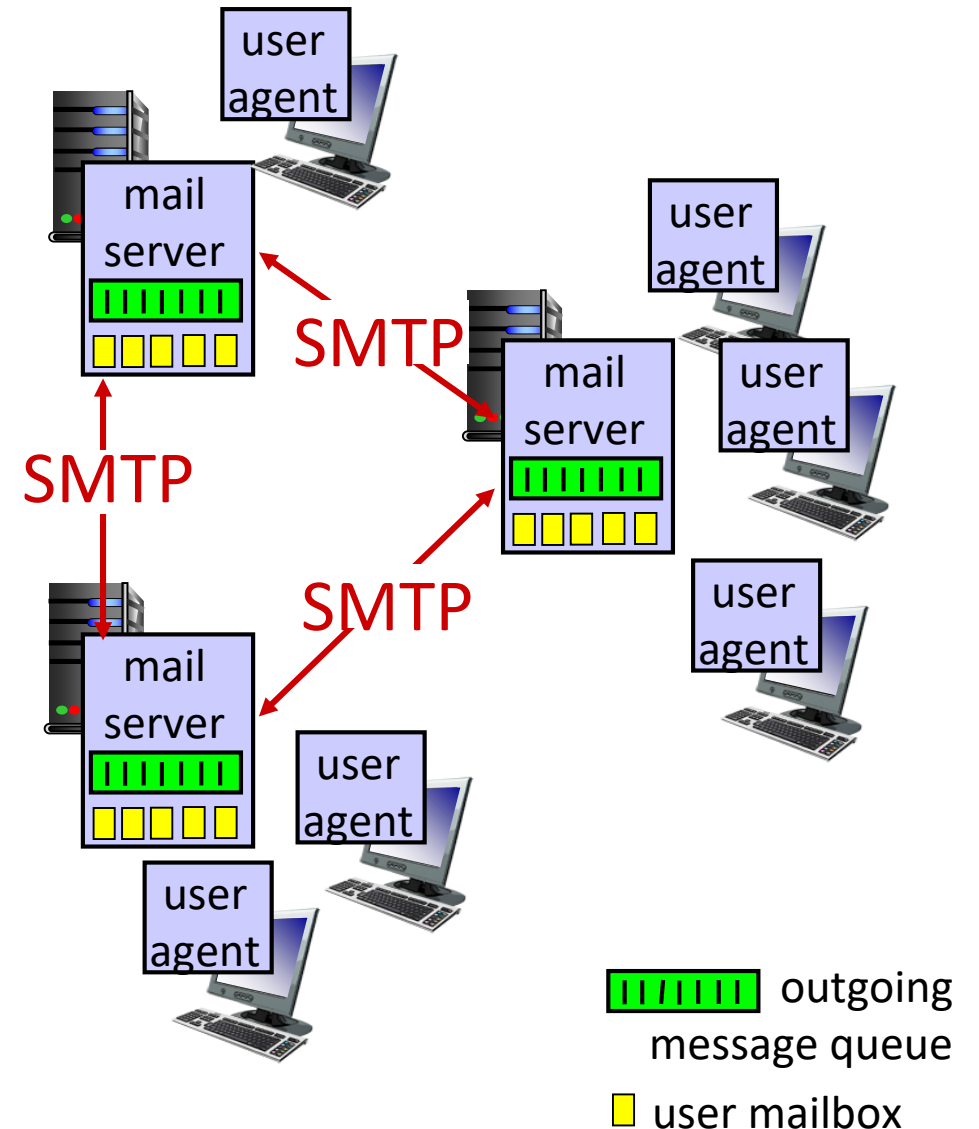
SMTP RFC (5321)

- 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

"client" SMTP server   "server" SMTP server

initiate TCP connection

RTT

TCP connection initiated

SMTP handshaking
- 220
- HELO
- 250 Hello

SMTP transfers

time

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-38
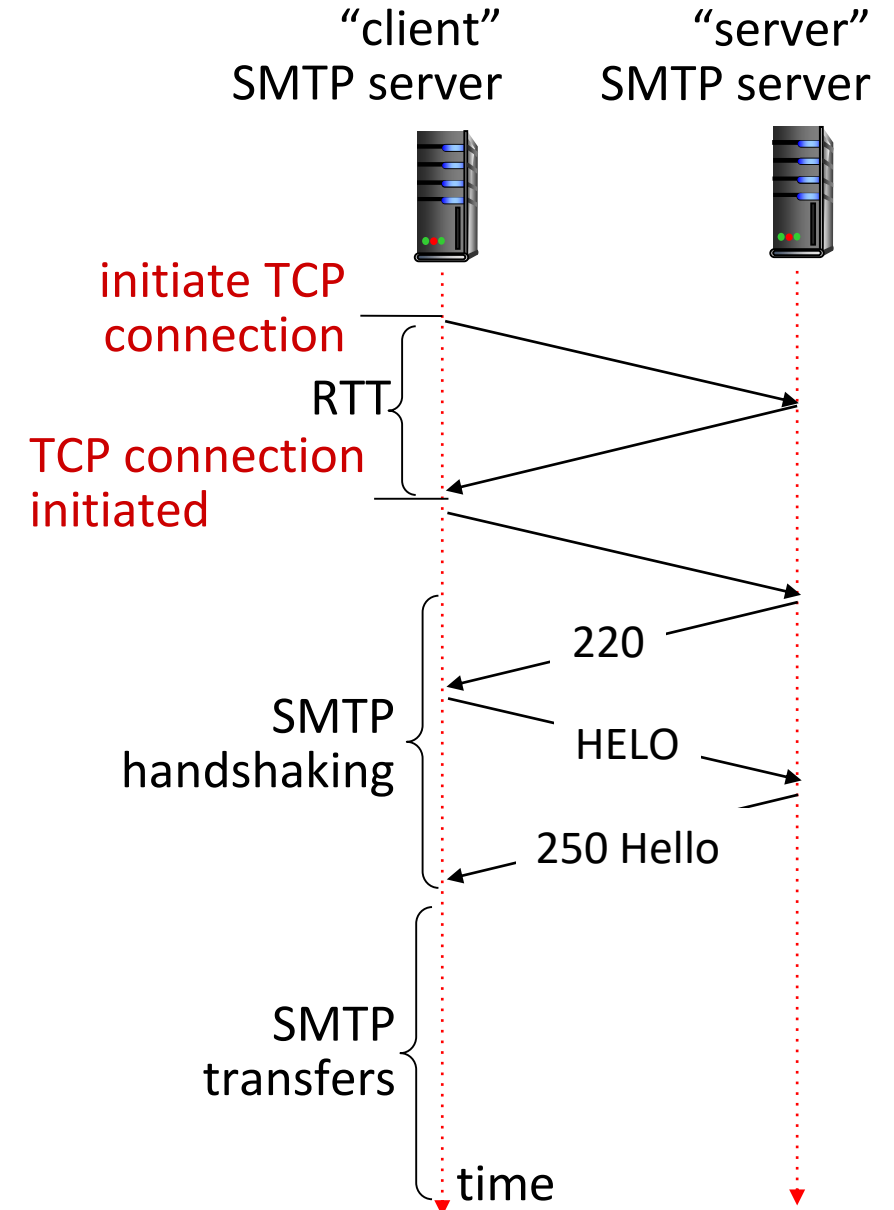
Scenario: Alice sends e-mail to Bob

1) Alice uses UA to compose e-mail message "to" bob@someschool.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

Alice's mail server

Bob's mail server

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-39
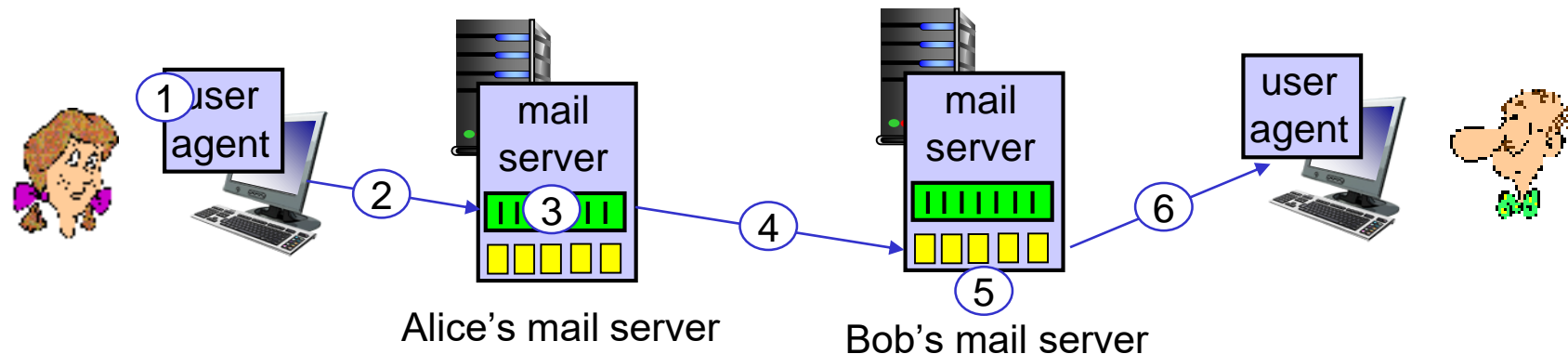
Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
```

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-40

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

header

blank line

body
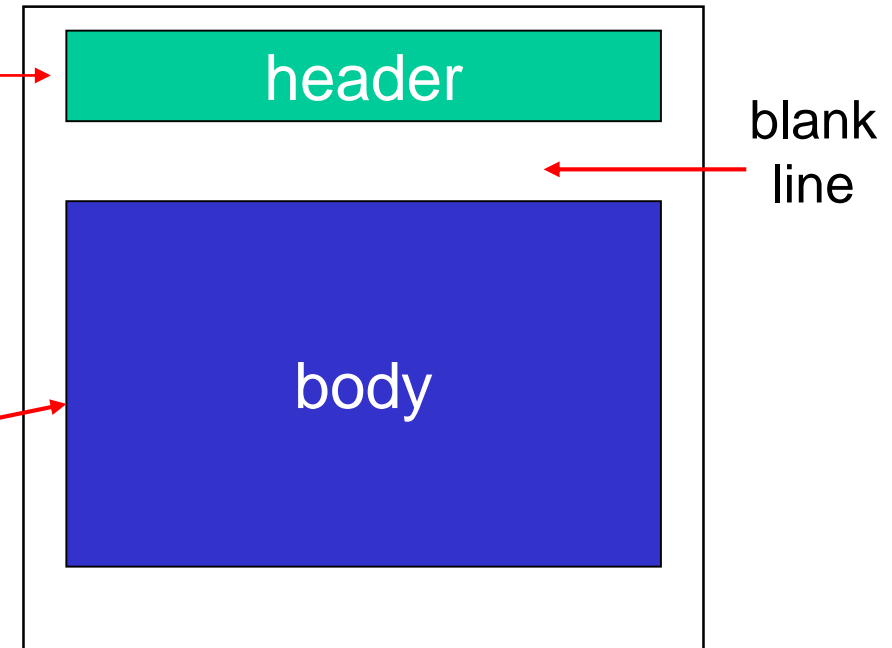
Retrieving email: mail access protocols
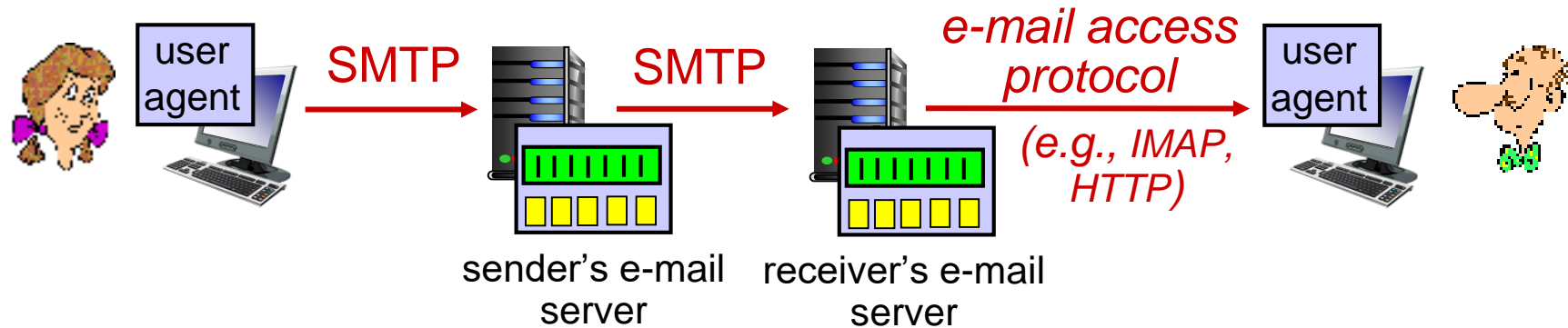
- SMTP: delivery/storage of e-mail messages to receiver's server

- 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 STMP (to send), IMAP (or POP) to retrieve e-mail messages

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-42

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

DNS: Domain Name System

people: many identifiers:
- SSN, name, passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., cs.umass.edu - 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, DNS servers communicate to *resolve* names (address/name translation)

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

DNS: services

- hostname-to-IP-address translation

- host aliasing
  - canonical, alias names

    relay1.west-coast.enterprise.com

    enterprise.com, www.enterprise.com

- mail server aliasing

    relay1.west-coast.yahoo.com

    yahoo.com

- load distribution
  - replicated Web servers: many IP addresses correspond to one name

Thinking about the DNS

humongous distributed database:
- ~ billion records, each simple

handles many *trillions* of queries/day:
- *many* more reads than writes
- *performance matters:* almost every Internet transaction interacts with DNS - msecs count!

organizationally, physically decentralized:
- millions of different organizations responsible for their records

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-46

DNS: a distributed, hierarchical database

Root DNS Servers

Root

.com DNS servers   .org DNS servers   .edu DNS servers

Top Level Domain

yahoo.com
DNS servers

amazon.com
DNS servers

pbs.org
DNS servers

nyu.edu
DNS servers

umass.edu
DNS servers

Authoritative
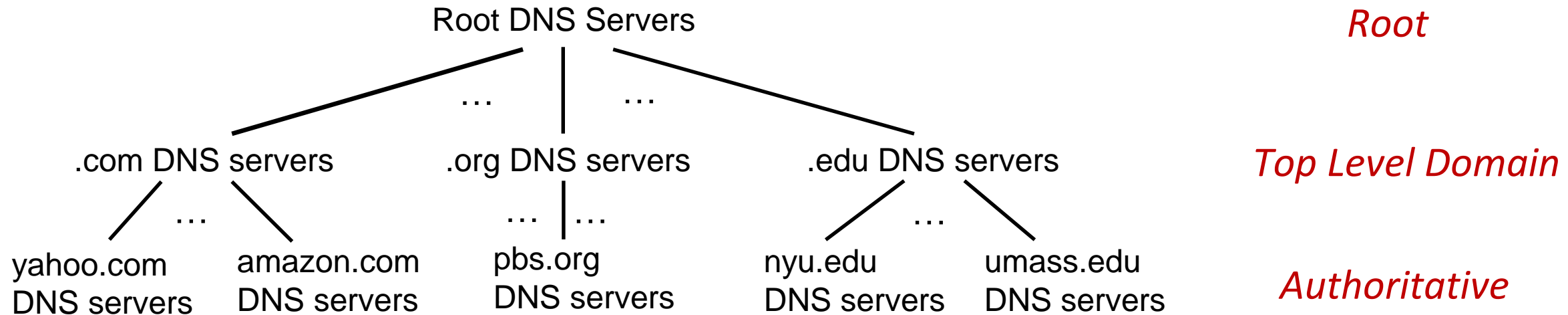
Client wants IP address for www.amazon.com

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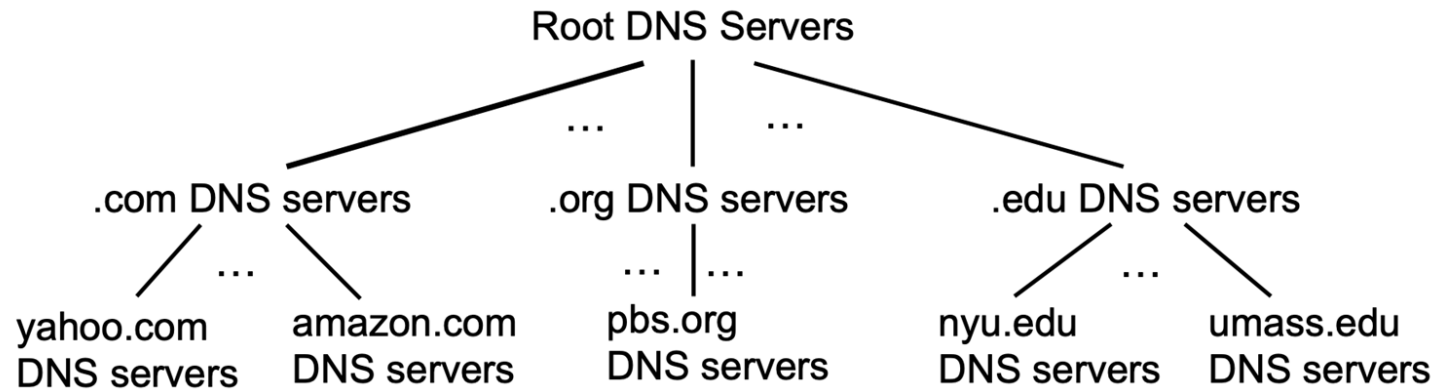
13 logical root name "servers" worldwide each "server" replicated many times (~200 servers in US)

Key:
- 0 Servers
- 1–10 Servers
- 11–20 Servers
- 21+ Servers

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
- Verisign: authoritative registry for .com
- Educause: .edu TLD

Root DNS Servers

... ...

.com DNS servers   .org DNS servers   .edu DNS servers

...   ...|...   ...

yahoo.com
DNS servers   amazon.com
DNS servers   pbs.org
DNS servers   nyu.edu
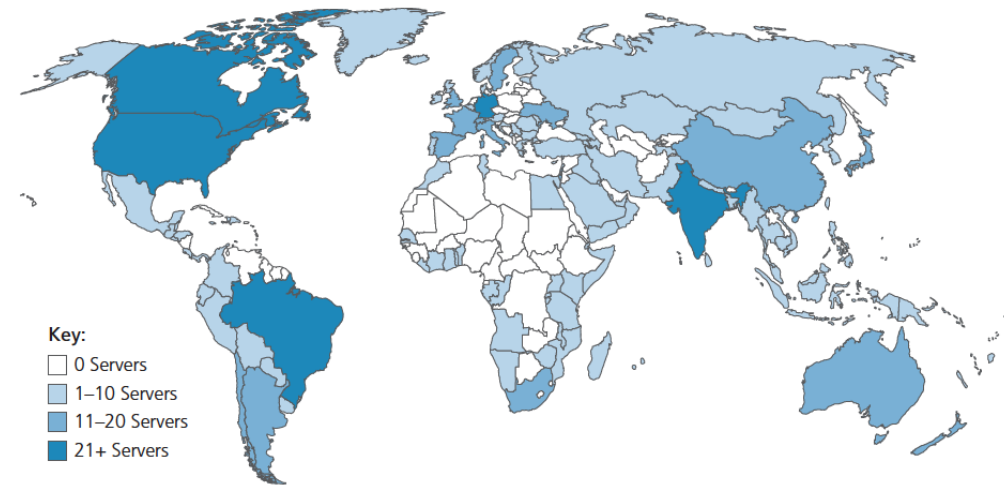DNS servers   umass.edu
DNS servers
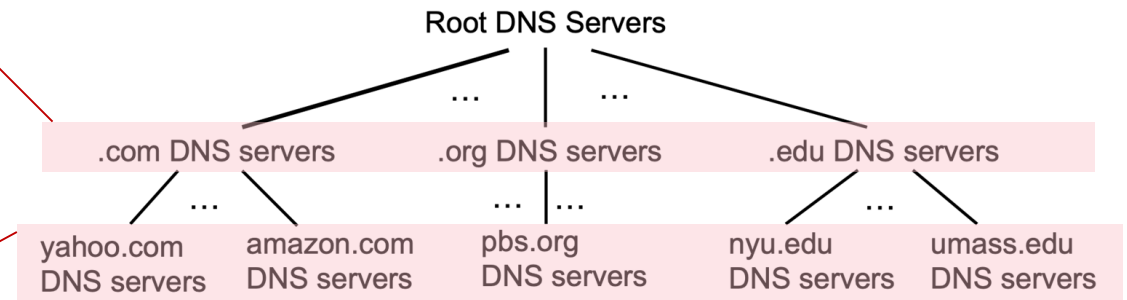
authoritative DNS servers:

- 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

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-49

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`

- local DNS server doesn't strictly belong to hierarchy

DNS name resolution: iterated query

Example: host at engineering.nyu.edu
wants IP address for gaia.cs.umass.edu

Iterated query:
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"

root DNS server

TLD DNS server

requesting host at
engineering.nyu.edu

local DNS server
dns.nyu.edu

gaia.cs.umass.edu

authoritative DNS server
dns.cs.umass.edu

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-51
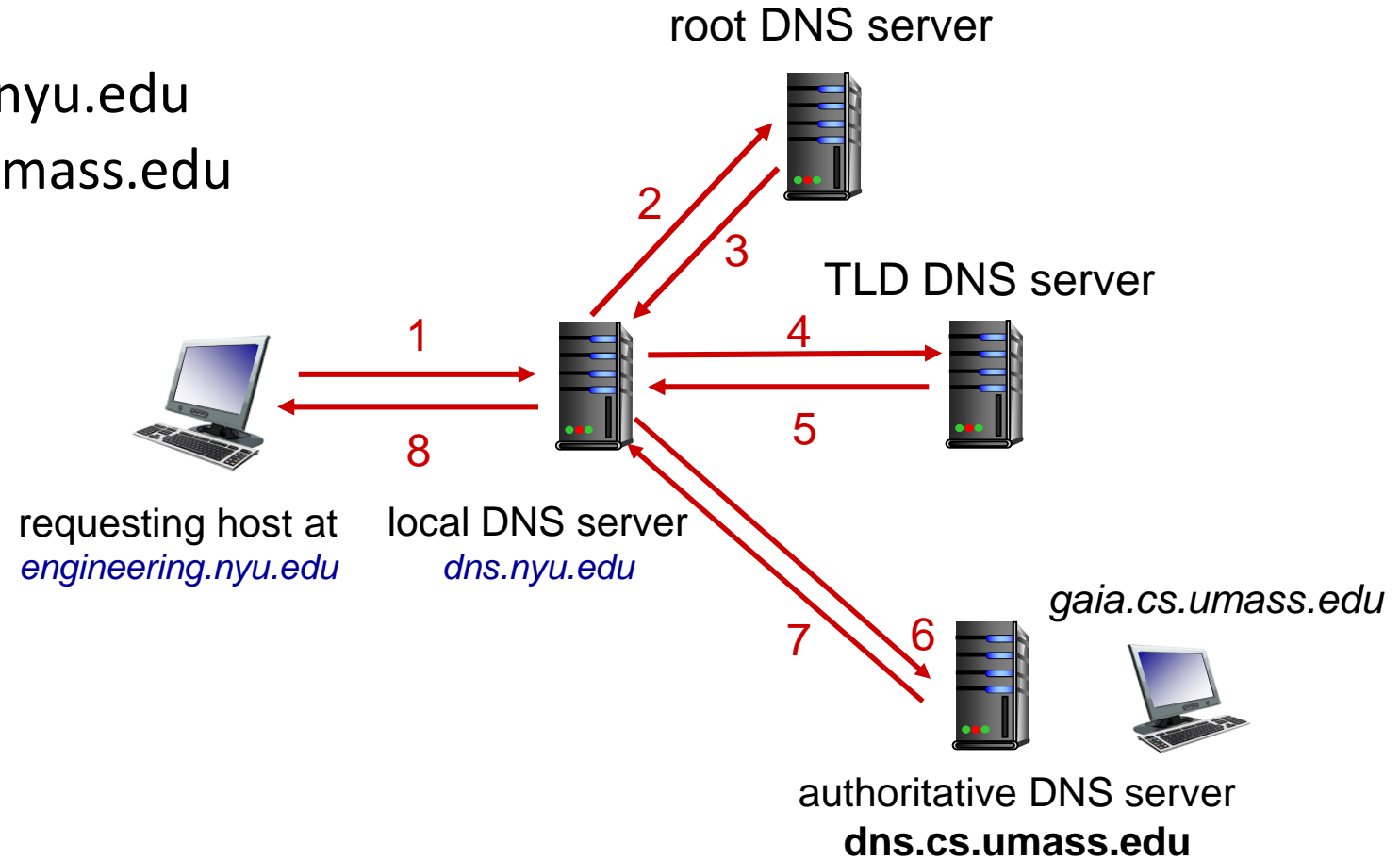
DNS name resolution: recursive query

Example: host at engineering.nyu.edu
wants IP address for gaia.cs.umass.edu

Recursive query:
- puts burden of name resolution on contacted name server

root DNS server

TLD DNS server

requesting host at
engineering.nyu.edu

local DNS server
dns.nyu.edu

gaia.cs.umass.edu

authoritative DNS server
dns.cs.umass.edu
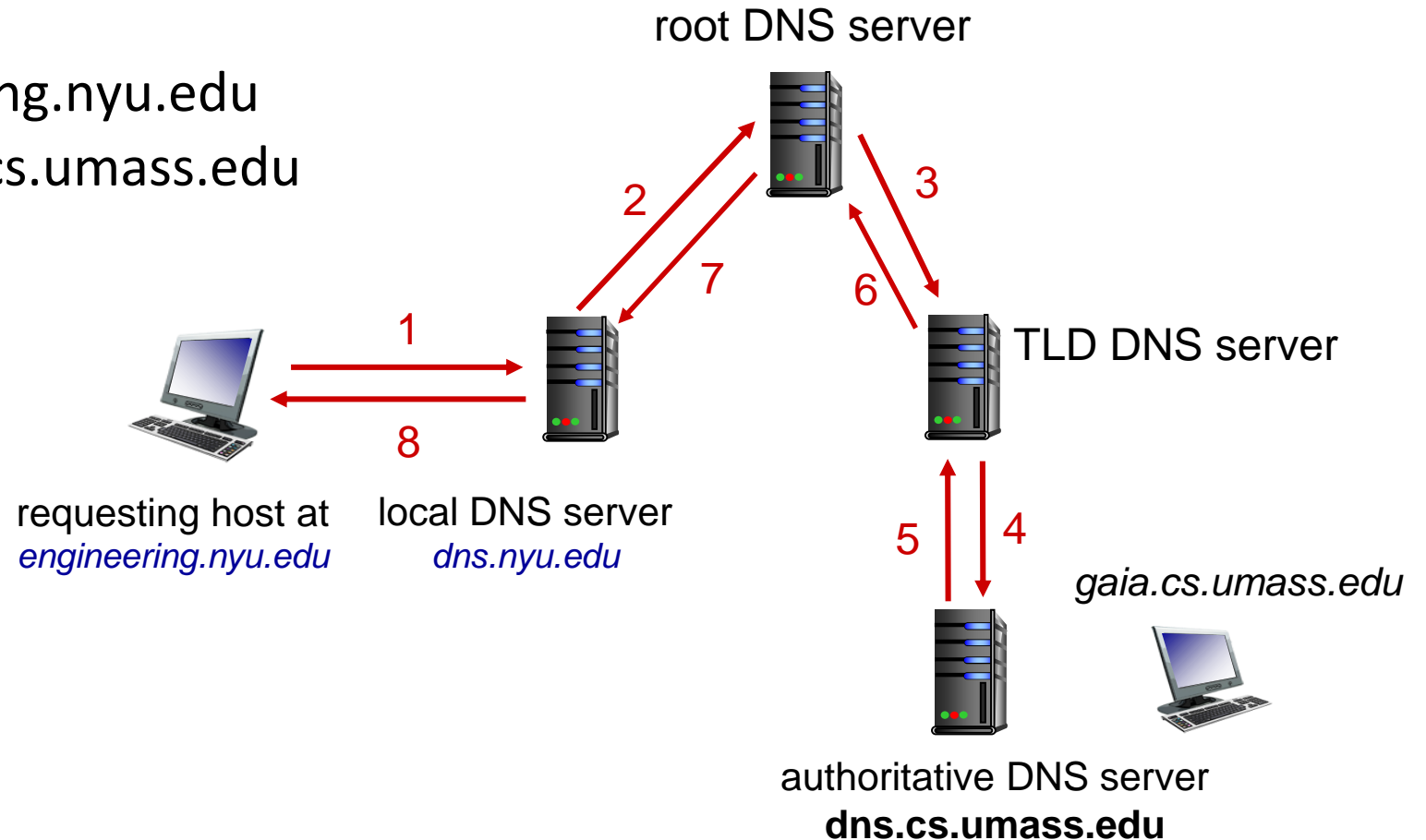
Caching DNS Information

- once (any) name server learns mapping, it *caches* mapping, and i*mmediately* returns a cached mapping in response to a query
  - caching improves response time
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers

- cached entries may be *out-of-date*
  - if named host changes IP address, may not be known Internet-wide until all TTLs expire!
  - *best-effort name-to-address translation!*

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 (e.g. 145.37.93.126)

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

type=CNAME
- `name` is alias name for some "canonical" (the real) name
- www.ibm.com is really servereast.backup2.ibm.com
- `value` is canonical name

type=MX
- `value` is name of SMTP mail server associated with `name`

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-54

DNS protocol messages

DNS *query* and *reply* messages, both have same *format:*

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

Identification	Flags	
Number of questions	Number of answer RRs	12 bytes
Number of authority RRs	Number of additional RRs	
Questions (variable number of questions)		Name, type fields for a query
Answers (variable number of resource records)		RRs in response to query
Authority (variable number of resource records)		Records for authoritative servers
Additional information (variable number of resource records)		Additional "helpful" info that may be used

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-55

DNS protocol messages

DNS *query* and *reply* messages, both have same *format:*

← 2 bytes →	← 2 bytes →
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

name, type fields for a query —— questions (variable # of questions)

RRs in response to query —— answers (variable # of RRs)

records for authoritative servers —— authority (variable # of RRs)

additional " helpful" info that may be used —— additional info (variable # of RRs)

Getting your info into the DNS

example: new startup "Network Utopia"

- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)

  - provide names, IP addresses of authoritative name server (primary and secondary)

  - registrar inserts NS, A RRs into .com TLD server:

    ```
    (networkutopia.com, dns1.networkutopia.com, NS)
    (dns1.networkutopia.com, 212.212.212.1, A)
    ```

- create authoritative server locally with IP address `212.212.212.1`

  - type A record for www.networkuptopia.com

  - type MX record for networkutopia.com

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-57

DNS security

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
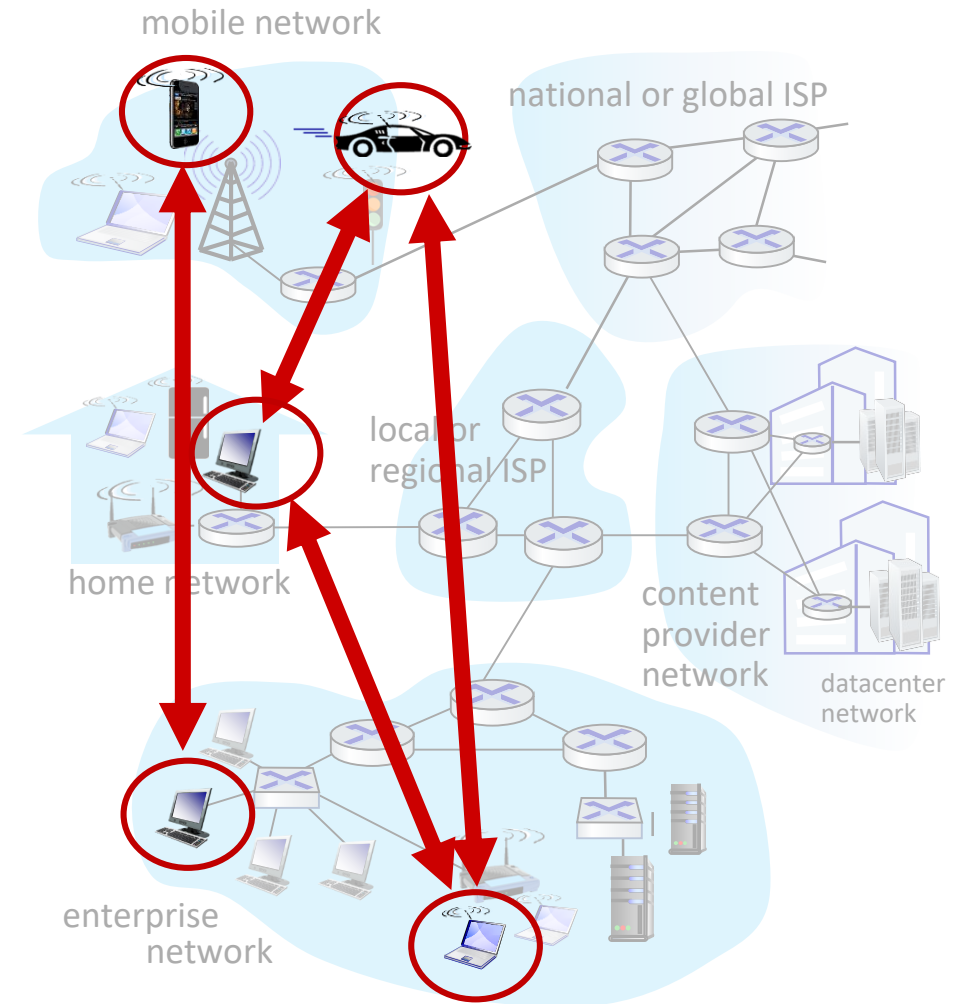 - potentially more dangerous

Spoofing attacks

- intercept DNS queries, returning bogus replies
  - DNS cache poisoning
  - RFC 4033: DNSSEC authentication services

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

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020
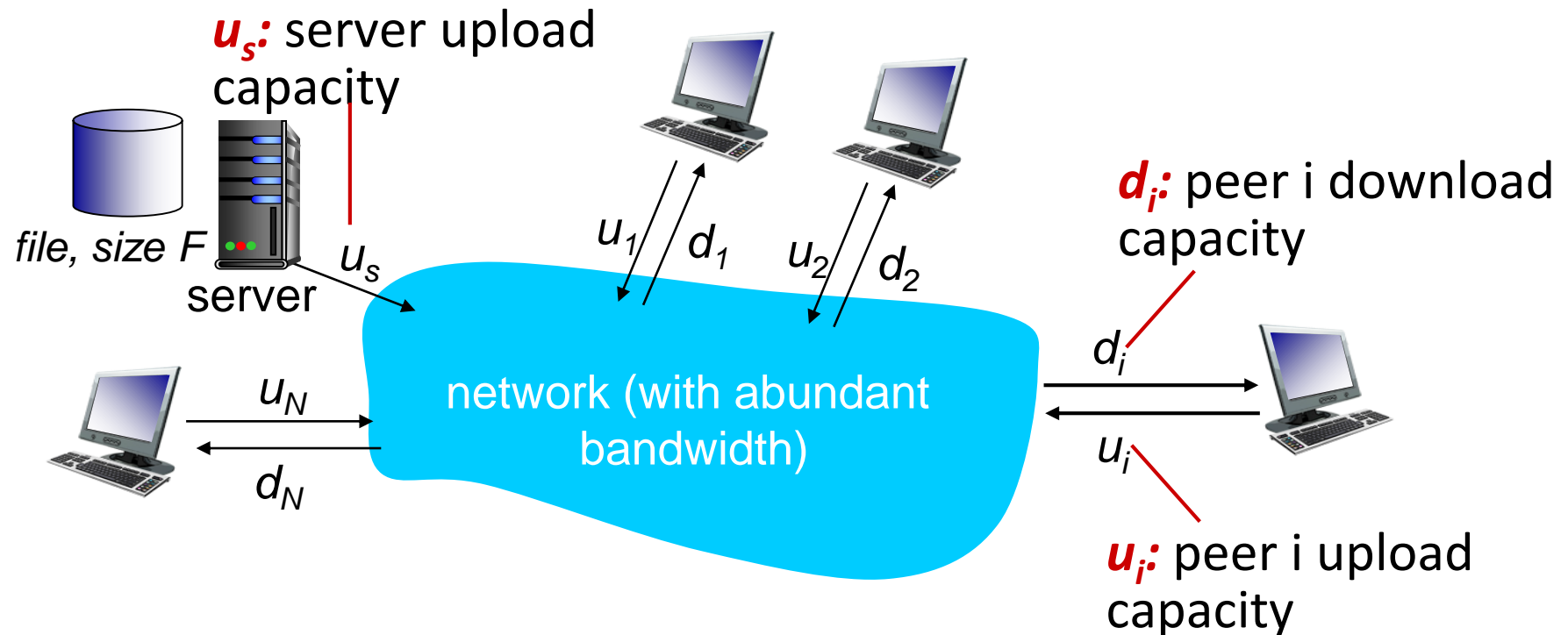
Peer-to-peer (P2P) architecture

- *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)

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-60

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

u_s: server upload capacity

d_i: peer i download capacity

u_i: peer i upload capacity

file, size F

server

u_s

u_1 d_1 u_2 d_2

u_N

d_N

network (with abundant bandwidth)

d_i

u_i

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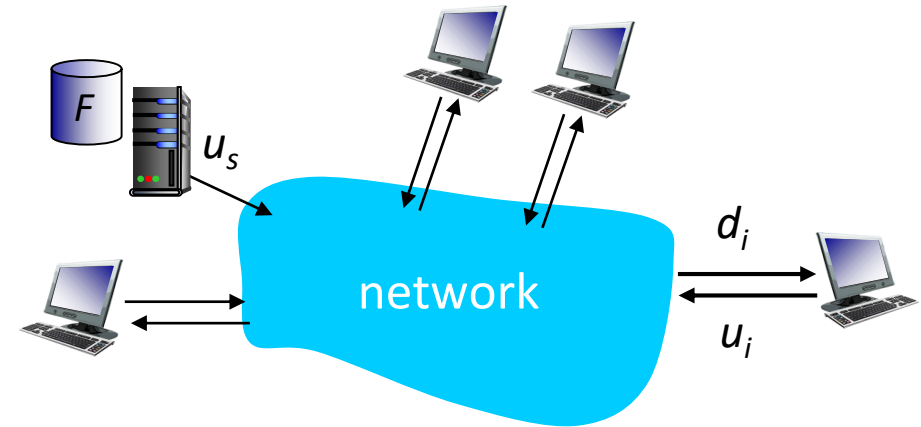
time to distribute *F* to *N* clients using client-server approach

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

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 $u_s + \Sigma u_i$
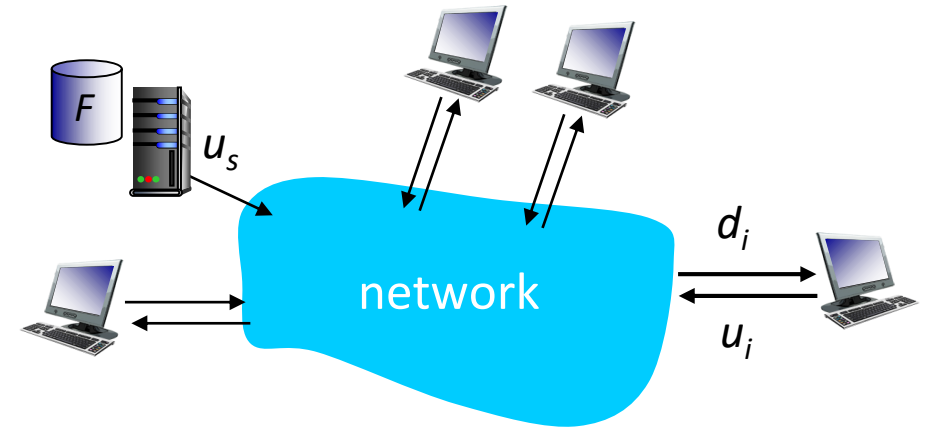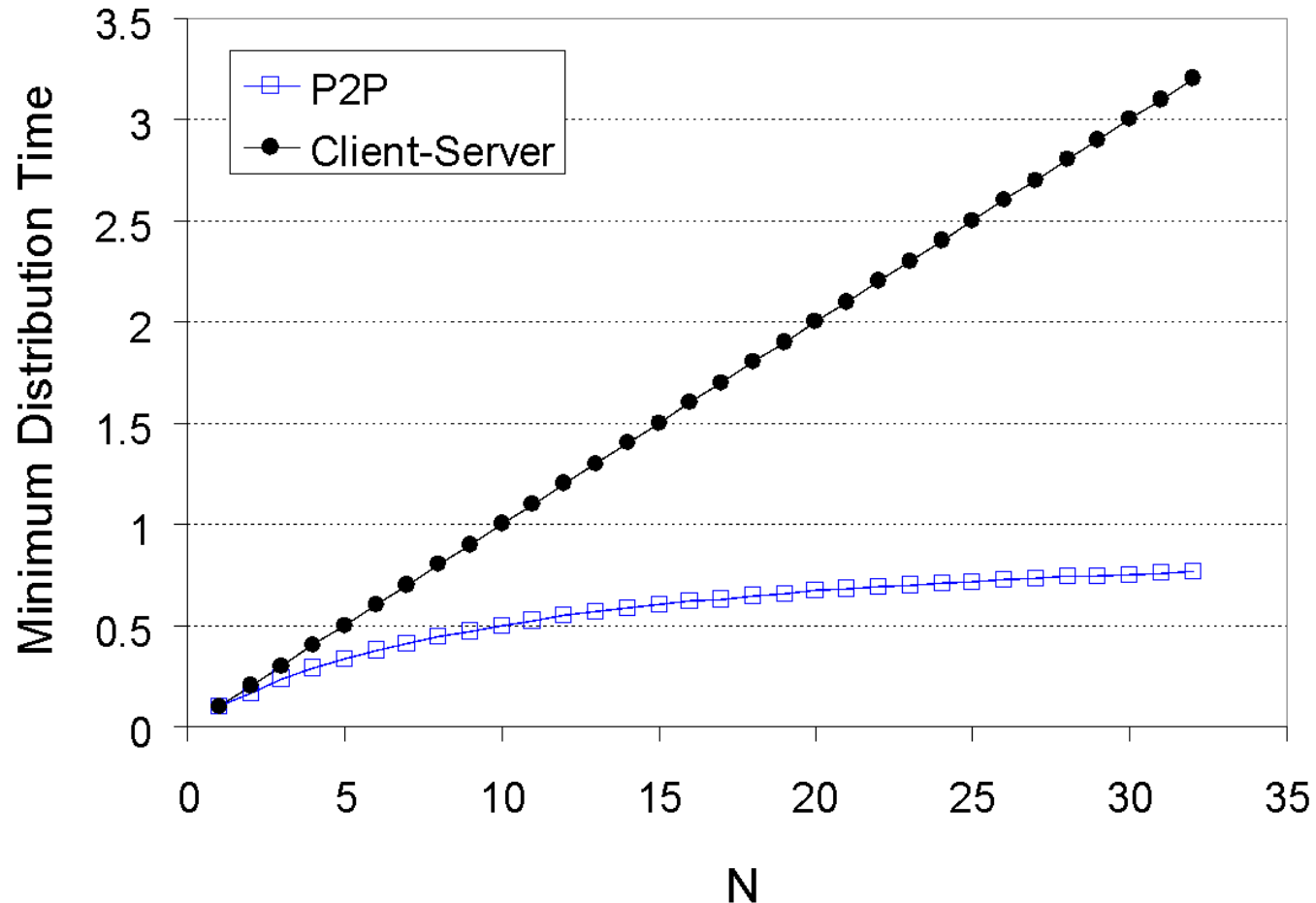

> *time to distribute F to N clients using P2P approach*
>
> $$D_{P2P} > max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

Client-server vs. P2P: example

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

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-64

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*)
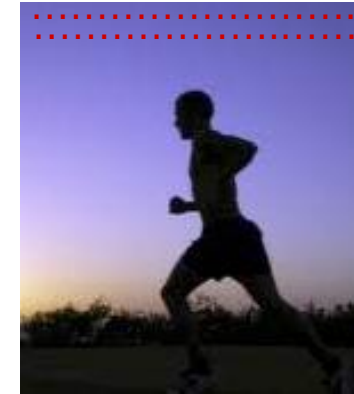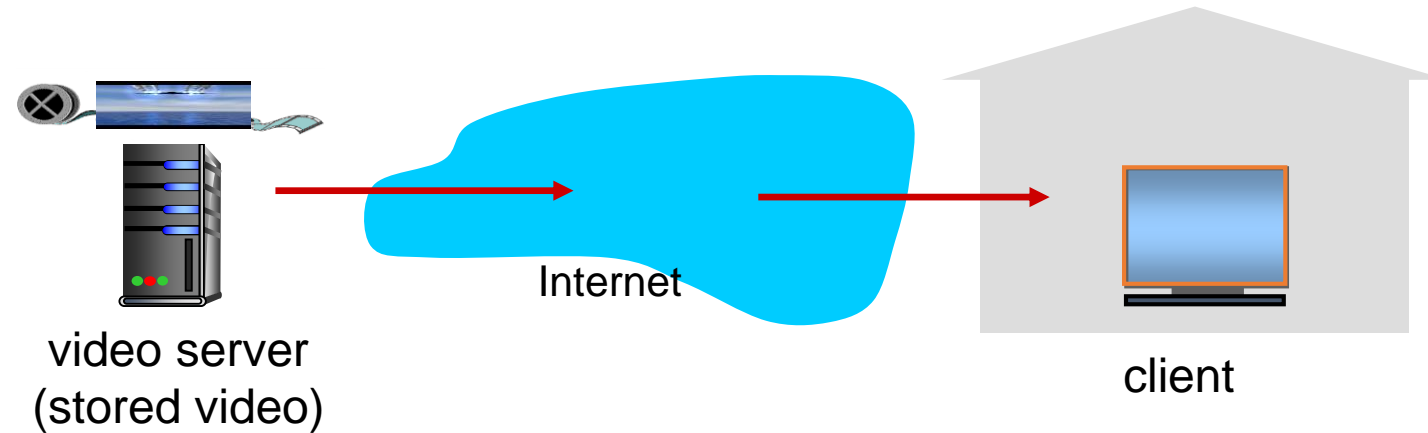
frame *i*

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:

video server
(stored video)

Internet

client

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-68
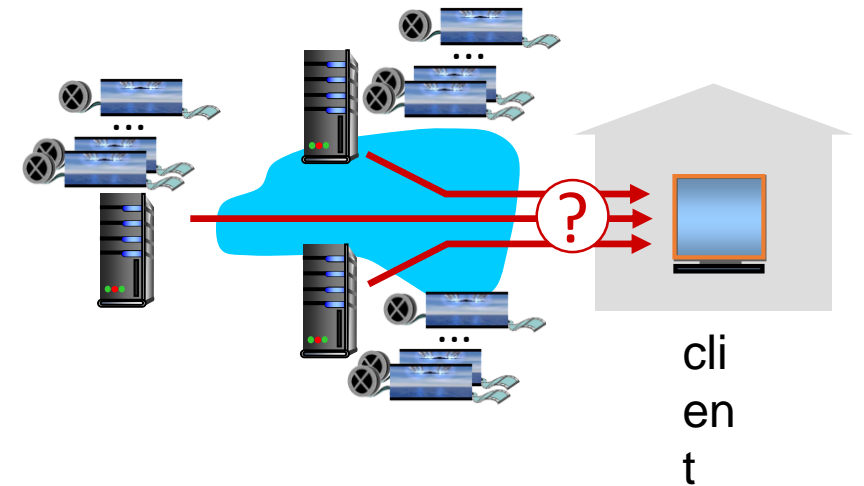
Streaming multimedia: DASH

*D*ynamic, *A*daptive *S*treaming over *H*TTP

server:

- divides video file into multiple chunks
- each chunk encoded at multiple different rates
- different rate encodings stored in different files
- files replicated in various CDN nodes
- *manifest file:* provides URLs for different chunks

cli en t

client:

- periodically estimates 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), and from different servers

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-69

Content distribution networks (CDNs)

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 (and possibly congested) path to distant clients

....quite simply: this solution *doesn't scale*

Content distribution networks (CDNs)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands 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
    - Akamai: 240,000 servers deployed in > 120 countries (2015)
  - *bring home:* smaller number (10's) of larger clusters in POPs near access nets
    - used by Limelight

The slides are based on the slides by Computer Networking: A Top-Down Approach 8ᵗʰ edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-71

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**

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-72

Socket programming

goal: learn how to build client/server applications that communicate using sockets

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

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-73
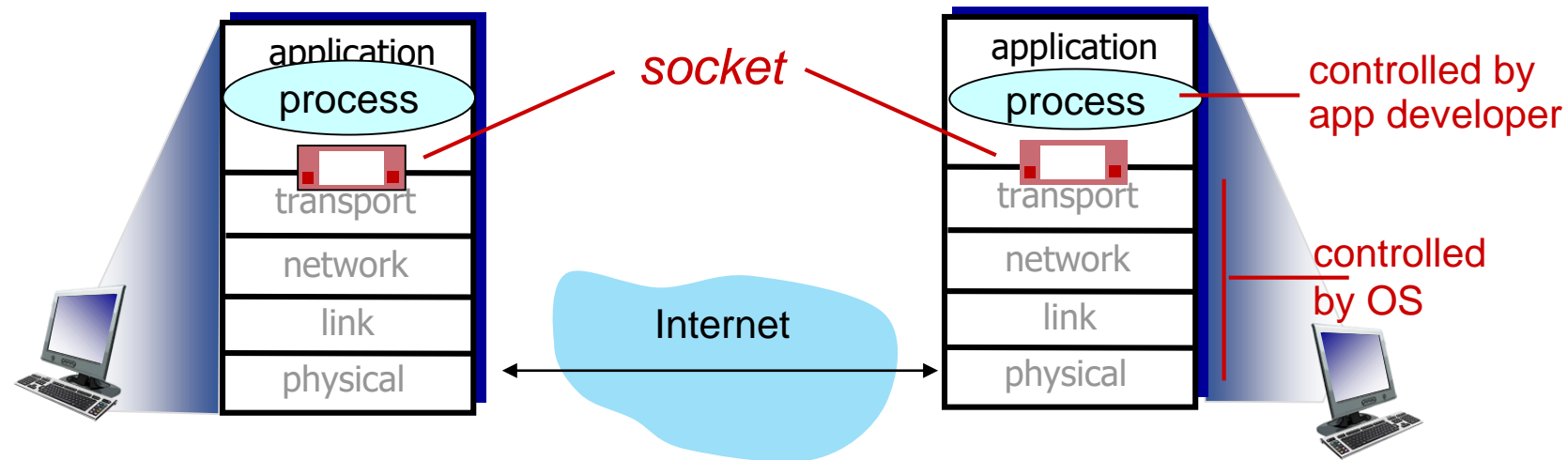
Socket programming

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

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-75

Client/server socket interaction: UDP

server (running on serverIP)

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

Create datagram with serverIP address
And port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Example app: UDP client

Python UDPClient

include Python's socket library ⟶ `from socket import *`

`serverName = 'hostname'`

`serverPort = 12000`

create UDP socket for server ⟶ `clientSocket = socket(AF_INET,`
`                                 SOCK_DGRAM)`

get user keyboard input ⟶ `message = raw_input('Input lowercase sentence:')`

attach server name, port to message; send into socket ⟶ `clientSocket.sendto(message.encode(),`
`                                 (serverName, serverPort))`

read reply characters from socket into string ⟶ `modifiedMessage, serverAddress =`
`                                 clientSocket.recvfrom(2048)`

print out received string and close socket ⟶ `print modifiedMessage.decode()`

`clientSocket.close()`

Example app: UDP server

Python UDPServer

```python
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```

create UDP socket ⟶

bind socket to local port number 12000 ⟶

loop forever ⟶

Read from UDP socket into message, getting client's address (client IP and port) ⟶

send upper case string back to this client ⟶

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-78

Socket programming **with TCP**

Client must contact server

- server process must first be running

- server must have created socket (door) that welcomes client's contact
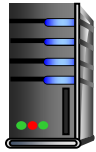
Client contacts server by:

- Creating TCP socket, specifying IP address, port number of server process

- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
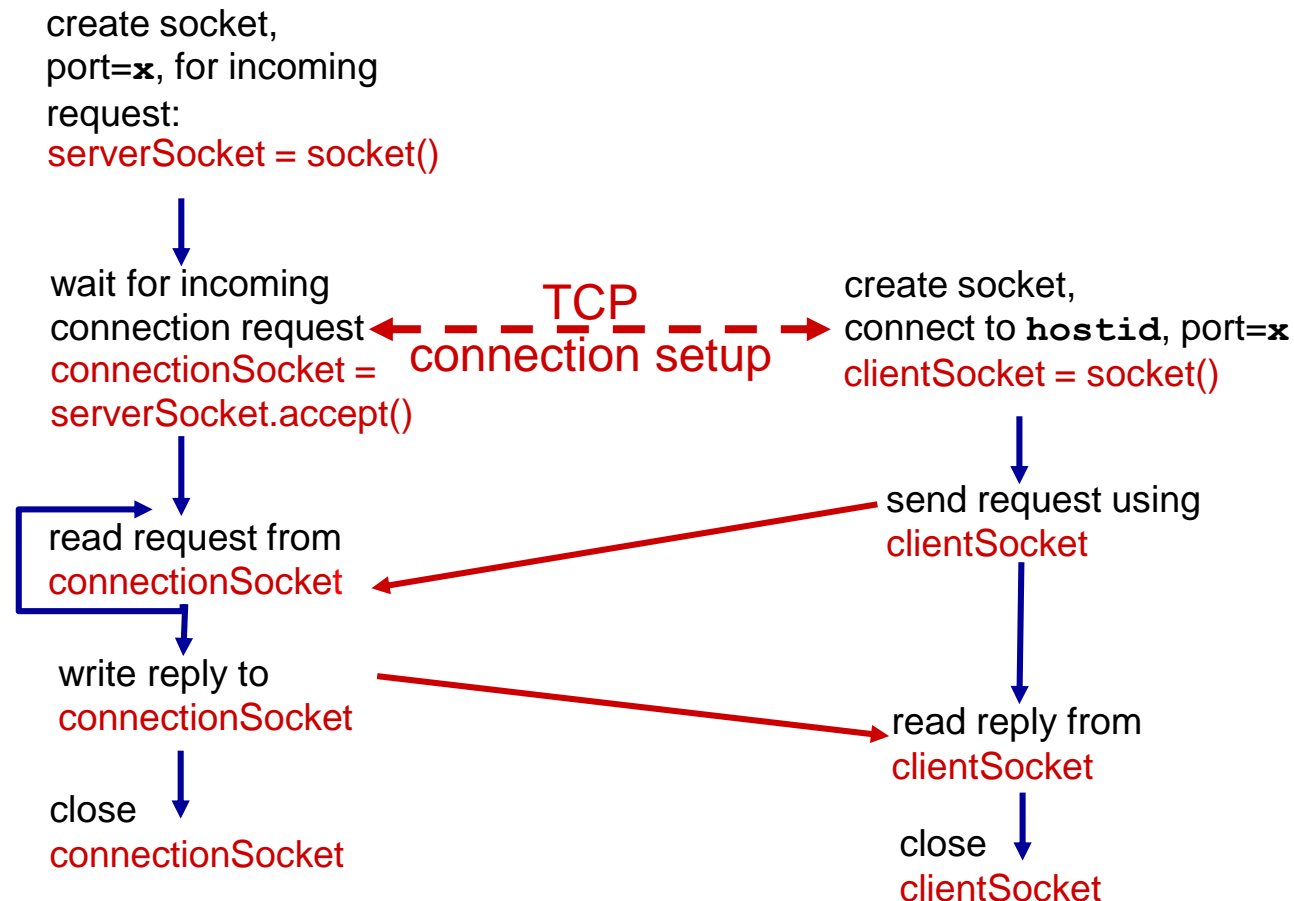 - *source* port numbers used to distinguish clients

Application viewpoint

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server processes

Client/server socket interaction: TCP

server (running on hostid)   **client**

create socket,
port=**x**, for incoming
request:
serverSocket = socket()

↓

wait for incoming
connection request ◄ - - - TCP - - - ► create socket,
connection request   connection setup   connect to **hostid**, port=**x**
connectionSocket =   clientSocket = socket()
serverSocket.accept()

read request from
connectionSocket ◄——————— send request using
                          clientSocket

write reply to
connectionSocket ———————► read reply from
                          clientSocket

close   close
connectionSocket   clientSocket

Example app: TCP client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(2048)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000 →

No need to attach server name, port →

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Example app: TCP server

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(2048).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                                        encode())
connectionSocket.close()
```

create TCP welcoming socket ⟶

server begins listening for incoming TCP requests ⟶

loop forever ⟶

server waits on accept() for incoming requests, new socket created on return ⟶

read bytes from socket (but not address as in UDP) ⟶

close connection to this client (but *not* welcoming socket) ⟶

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-82

General flow

The slides are based on the slides by Computer Networking: A Top-Down Approach 8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Application Layer: 2-83