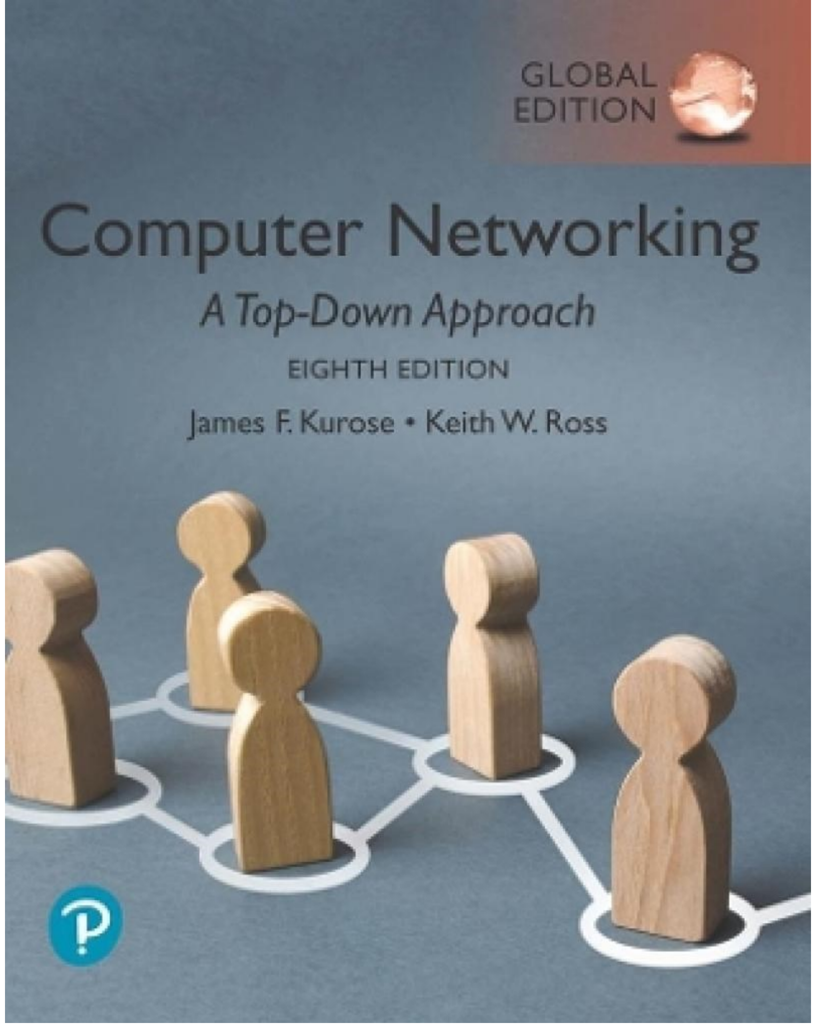
Chapter 2
Application Layer

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



Computer Networking: A Top-Down Approach
8^{th} edition n
Jim Kurose, Keith Ross
Pearson, 2020

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- 2.5 Peer-to-peer file distribution
- 2.6 Video streaming and content distribution networks

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- 2.5 Peer-to-peer file distribution
- 2.6 Video streaming and content distribution networks

Application layer: overview

Our goals:

- conceptual *and* implementation aspects of application-layer protocols
  - client-server model
  - peer-to-peer model
  - transport-layer service models
- learn about protocols by examining popular application-layer protocols
  - HTTP
  - SMTP
  - DNS
  - (video streaming systems, CDNs)
- programming network applications in C
  - socket API

Some network apps

- social networking
- Web
- text messaging
- e-mail
- streaming stored video (YouTube, Hulu, Netflix)
- P2P file sharing (BitTorrent)
- voice over IP (e.g., Skype)
- real-time video conferencing (e.g., Zoom)
- Internet search
- remote login
- …

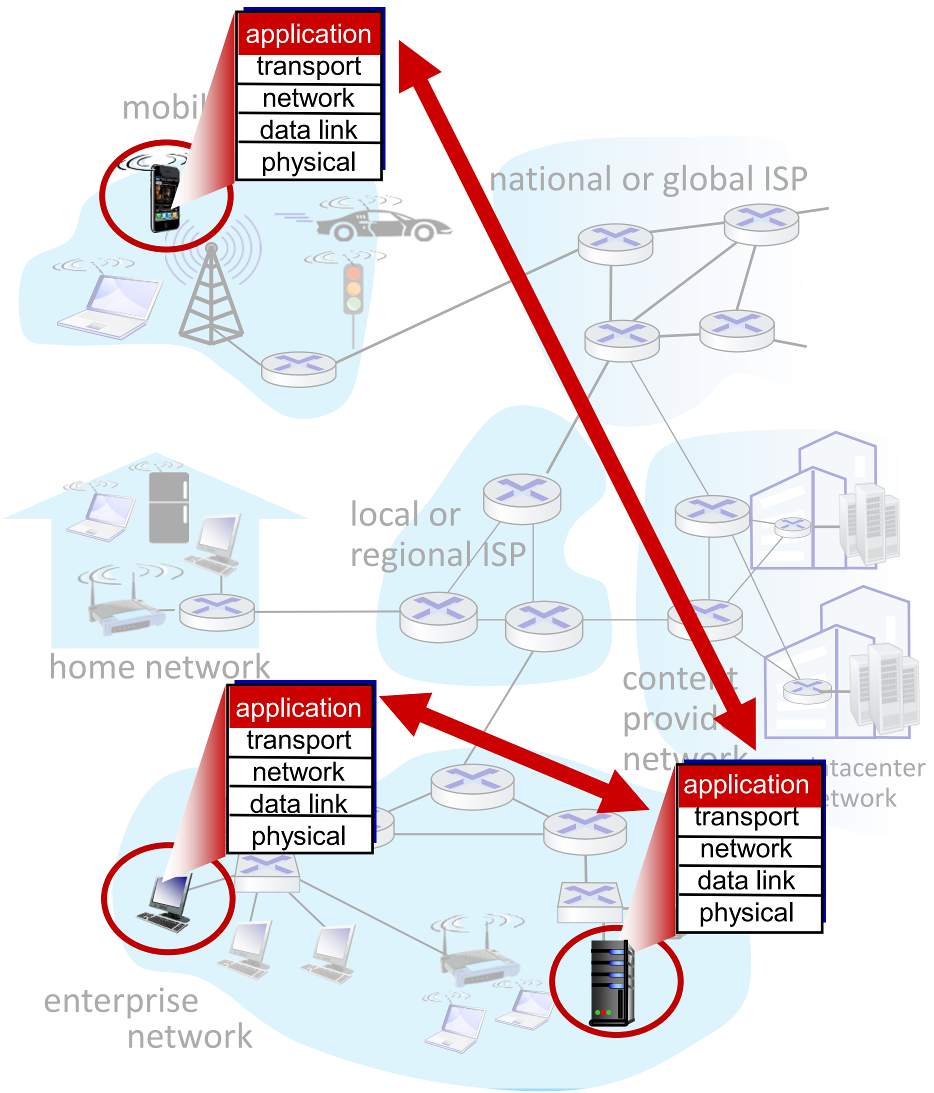
Creating a network app

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

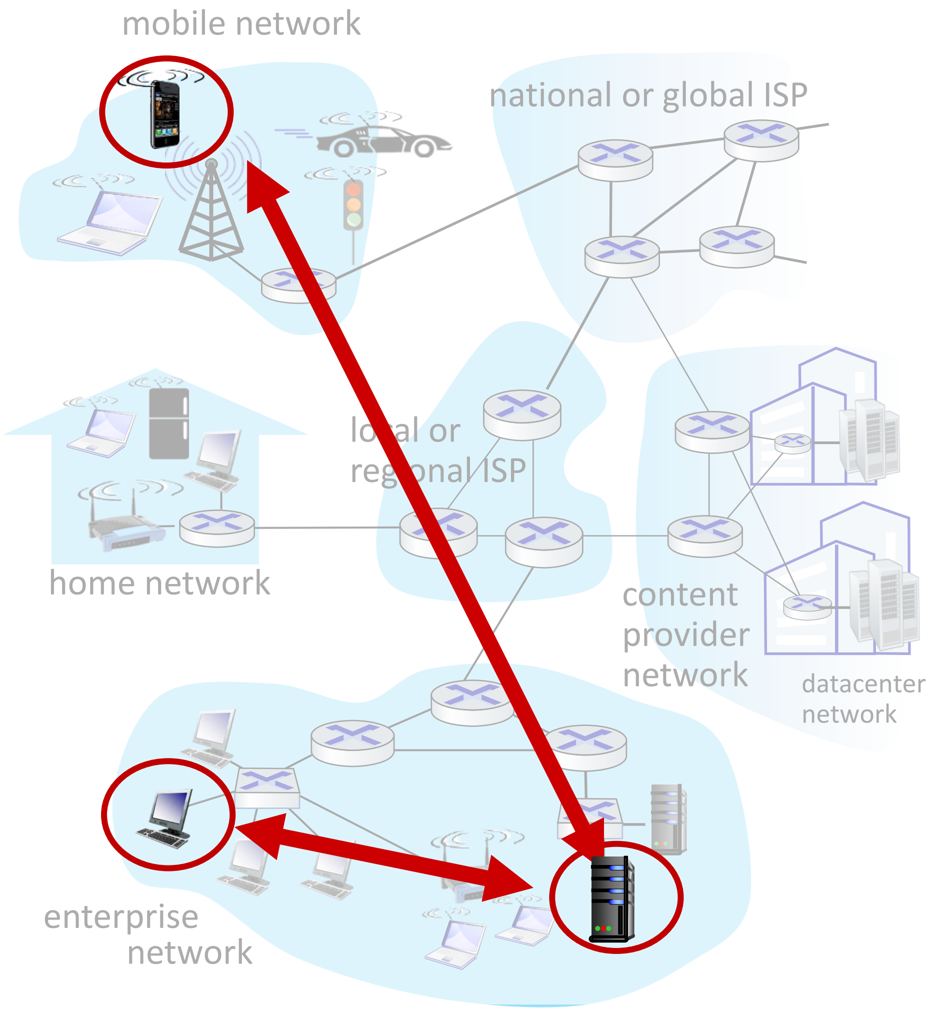
- network-core devices do not run user applications



Client-server model

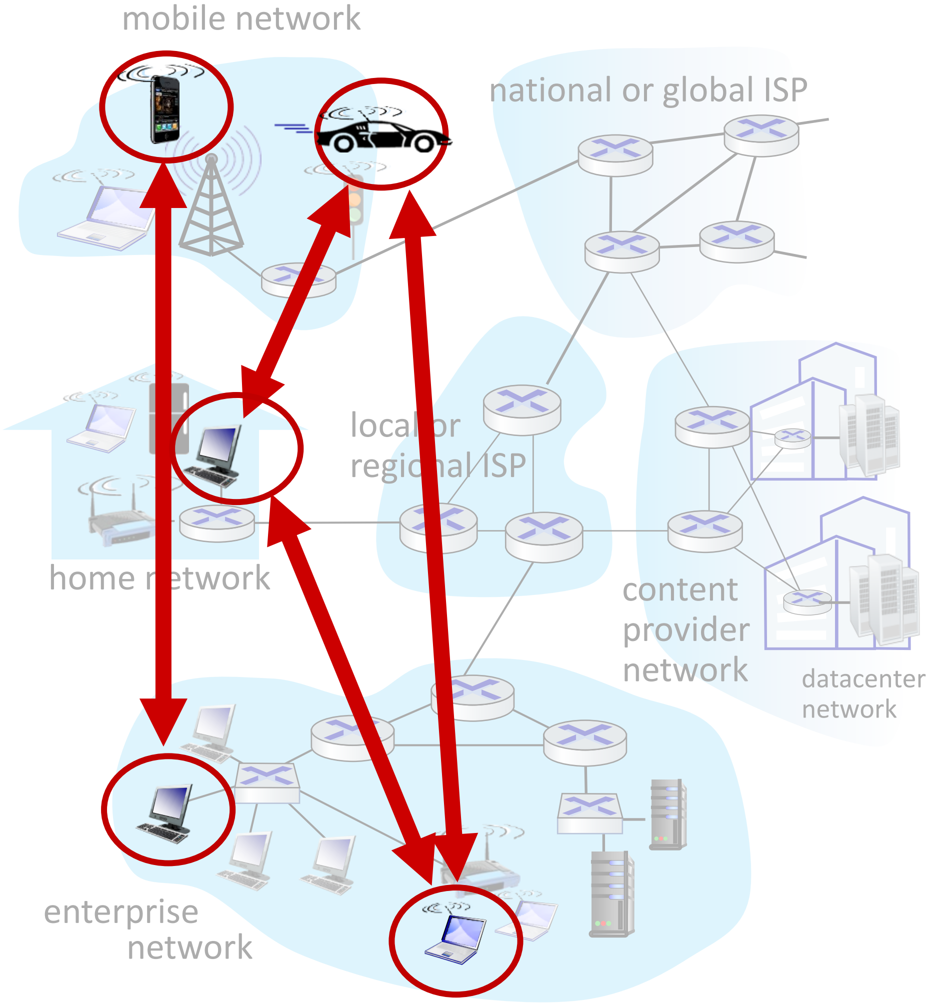
server:
- always-on host
- permanent IP address
- centralized
- often in data centers, for scaling

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



Peer-to-peer (P2P) model

- *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

Request-response

client process: process that initiates communication using a request

server process:

- process that waits to be contacted
- sends a response to the client

- note: applications with P2P model have client processes & server processes

Addressing processes

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

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

What transport service does an app need?

reliability

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

timing, delay

- 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, …

Internet transport protocols services

TCP (Transmission Control Protocol):

- *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/delay, minimum throughput guarantee

UDP (User Datagram Protocol):

- *fast packet transmission*
- *unreliable data transfer* between sending and receiving process
- *does not provide* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Q: why bother? *Why* is there a UDP?

Internet 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
	HTTP/3 [RFC 9114]	UDP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7320], DASH	UDP or TCP

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- 2.5 Peer-to-peer file distribution
- 2.6 Video streaming and content distribution networks

2.2 **Web and HTTP**

- 2.2.1 Overview of HTTP
- 2.2.2 Non-persistent and persistent connections
- 2.2.3 HTTP message format
- 2.2.4 Cookies
- ~~2.2.5 Web caching~~
- 2.2.6 HTTP/2

2.2 Web and HTTP

HTTP: transfer protocol
HTML: markup language

A quick HTML review…

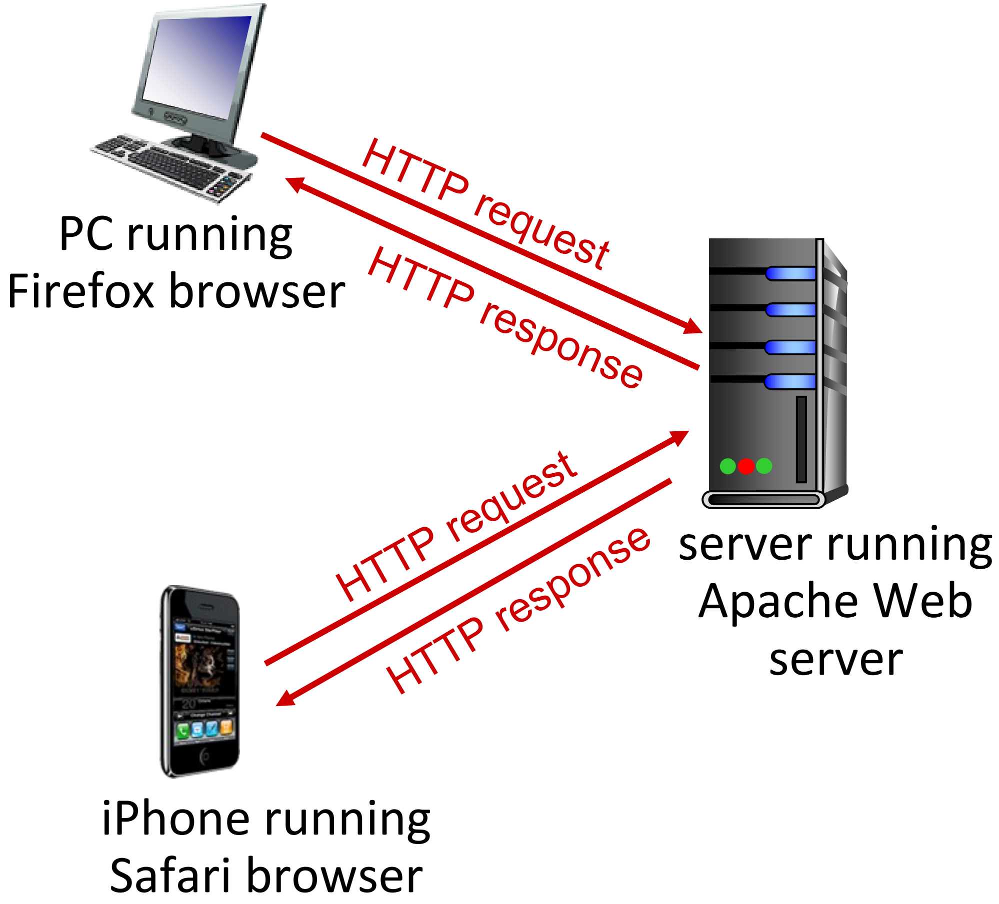
- web page consists of *base HTML-file* which *references several objects, each* addressable by a *URL*
  - objects can be HTML files, JPEG images, Java applets, audio files, etc.
  - Each object has an address and can stored on different Web servers

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

2.2.1 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)

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

2.2.2 HTTP connections: two types

Non-persistent HTTP/1.0 [1996]

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/1.1 [1997]

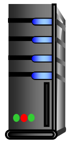
- TCP connection opened to a server
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed

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

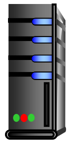
2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

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

time

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

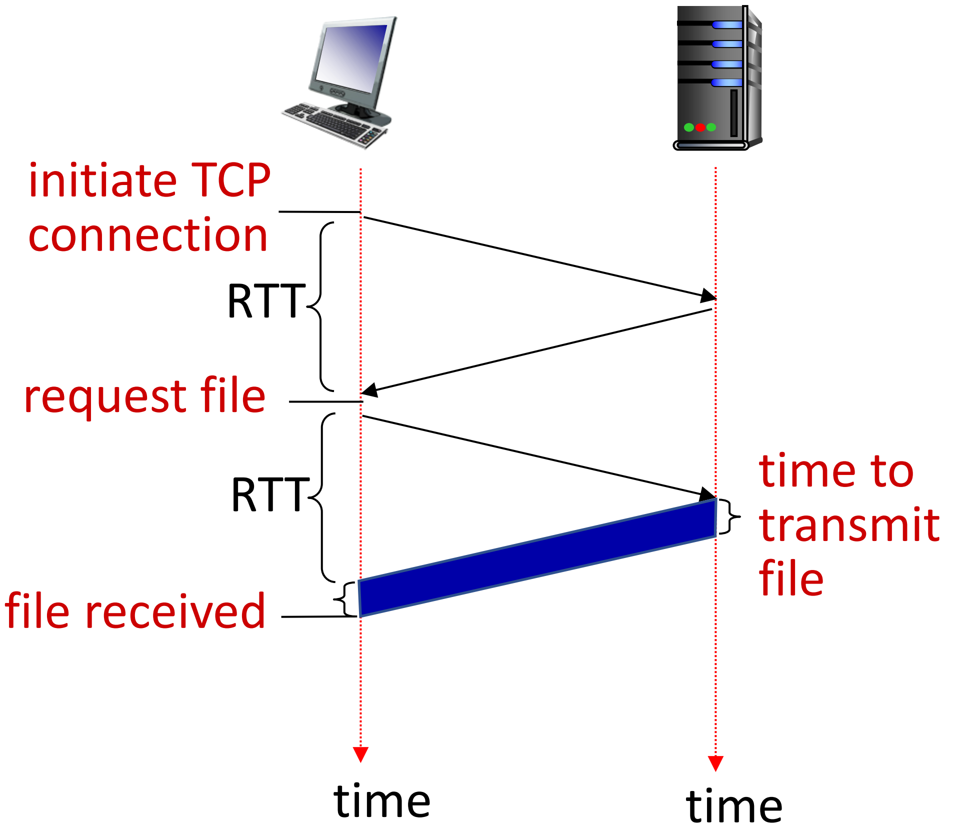
time

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 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 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 messages

- Two types of HTTP messages:
  - requests sent by the client to trigger an action on the server
  - responses are the answer returned the server in response to a request
- HTTP messages are normally machine generated
- HTTP messages are text-based (ASCII), and easy to read and understand

2.2.3 HTTP/1.1 request message example

Start line

Header lines (optional)

Empty line

```
GET /index.html HTTP/1.1
Host: www-net.cs.umass.edu
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
   10.15; rv:80.0) Gecko/20100101 Firefox/80.0
Accept: text/html,application/xhtml+xml
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Connection: keep-alive
\r\n
```

carriage return character (0x0D)

newline character (0x0A)

HTTP request methods

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/animalsearch?monkeys&banana`

HEAD method:

- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

PUT method:

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of PUT HTTP request message

HTTP/1.1 response message example

Start line

```
HTTP/1.1 200 OK
```

Header lines (optional)

```
Date: Tue, 08 Sep 2020 00:53:20 GMT
Server: Apache/2.4.6 (CentOS)
   OpenSSL/1.0.2k-fips PHP/7.4.9
   mod_perl/2.0.11 Perl/v5.16.3
Last-Modified: Tue, 01 Mar 2016 18:57:50 GMT
ETag: "a5b-52d015789ee9e"
Accept-Ranges: bytes
Content-Length: 2651
Content-Type: text/html; charset=UTF-8
```

Empty line

```
\r\n
```

Body (requested object)

```
data data data data data ...
```

Comparison of HTTP request and response

Request

```
POST / HTTP/1.1
Host: developer.mozilla.org
User-Agent: curl/8.6.0
Accept: */*
Content-Type: application/json
Content-Length: 345

{
"data": "ABC123"
}
```

Start line

Headers

Empty line

Body

Response

```
HTTP/1.1 403 Forbidden
Server: Apache
Date: Fri, 21 Jun 2024 12:52:39 GMT
Content-Length: 678
Content-Type: text/html
Cache-Control: no-store

<!DOCTYPE html>
<html lang="en">
(more data…)
```

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

Web-pages and HTML

- A very simple webpage that shows how its HTML code and the layout corresponds

URL: $\boxed{\texttt{http://}}\underbrace{\texttt{pracnet.net}}_{\text{host name}}\underbrace{\texttt{/simple.html}}_{\text{path name}}$

Default HTTP server port 80. No encryption

Trying out HTTP using netcat

1. start command prompt, and type

```
C:\temp>ncat pracnet.net 80
```

- opens TCP connection to pracnet.net at port 80 (default HTTP server port)
- anything typed in will be sent to port 80

2. type in a GET HTTP request:

```
GET /simple.html HTTP/1.1
Host: pracnet.net
```

- hit return twice

3. look at response message sent by HTTP server!

```
C:\temp>ncat pracnet.net 80
GET /simple.html HTTP/1.1
Host: pracnet.net

HTTP/1.1 200 OK
Connection: Keep-Alive
Keep-Alive: timeout=5, max=100
content-type: text/html
last-modified: Tue, 12 Sep 2023 18:12:09 GMT
accept-ranges: bytes
content-length: 408
date: Fri, 17 Jan 2025 23:47:48 GMT
server: LiteSpeed

<HTML>

<HEAD>
  <TITLE>This is a Simple HTML Page</TITLE>
  <link rel="icon" href="favicon.png" />
</HEAD>

<BODY>

  <H1>This is a Title</H1>
  <hr>

  <H2>This is a Subtitle</H2>
  <p>This is some text.</p>
```

```html
<HTML>

<HEAD>
  <TITLE>This is a Simple HTML Page</TITLE>
  <link rel="icon" href="favicon.png" />
</HEAD>

<BODY>

  <H1>This is a Title</H1>
  <hr>

  <H2>This is a Subtitle</H2>
  <p>This is some text.</p>
  <p>This is some <strong>bold</strong> text.</p>
  <p>This is some <em>italic</em> text.</p>
  <p>This is an image: <BR />
    <img src="//pracnet.net/favicon.ico">
  </p>
  <hr>

</BODY>

</HTML>
```

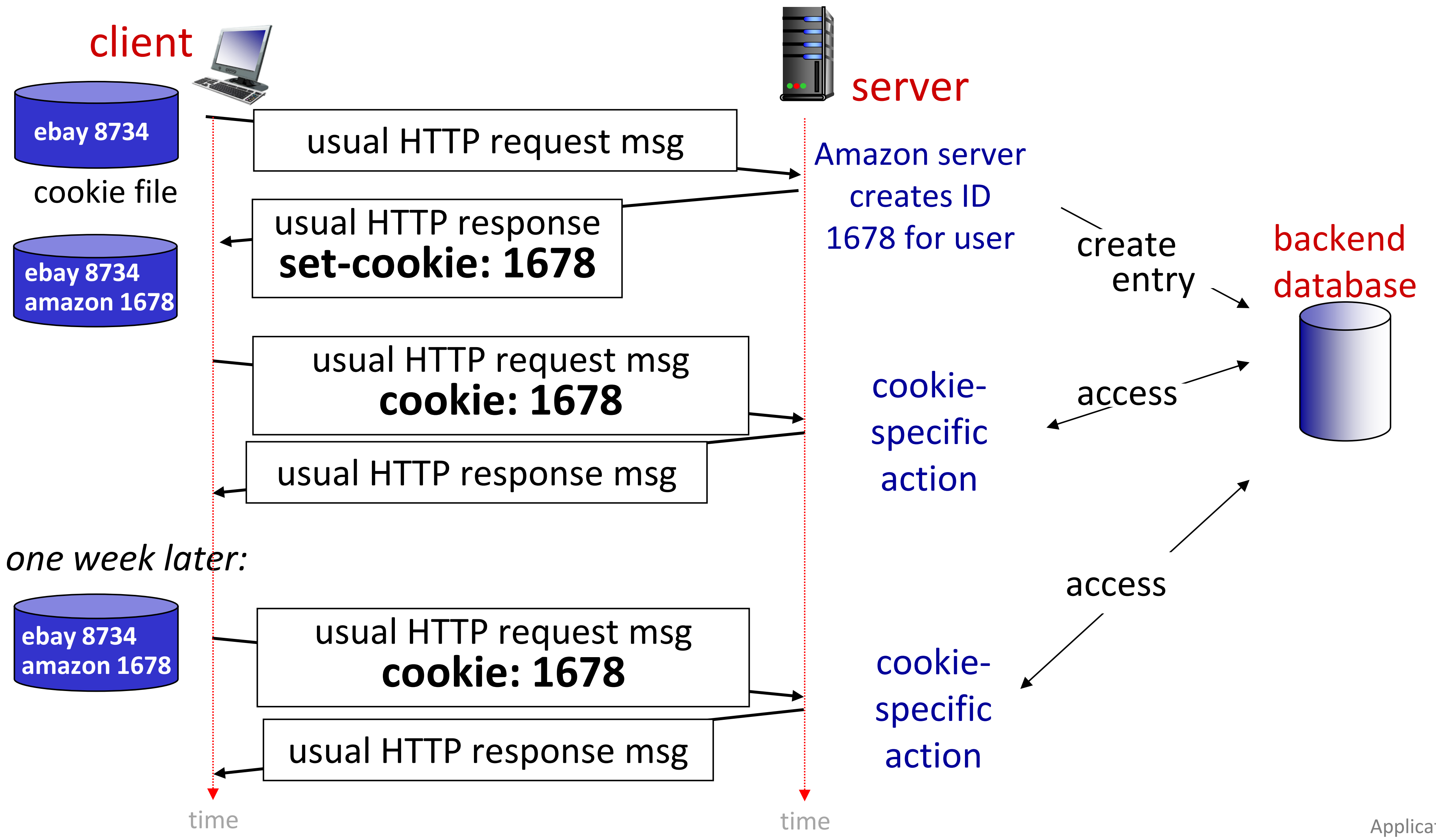
2.2.4 Maintaining user/server state: cookies

Web sites and browsers use *cookies* to maintain state

- Cookies are unique long-term identifiers
- allows to identify web browsers
- keep track of user activities

1. A cookie is selected randomly first time a user visits a webpage
   - The server includes the cookie in the response
2. 3) The cookie is stored locally by the browser and in the Web site's database
3. subsequent HTTP requests from the same browser to this site will contain cookie ID value, allowing site to "identify" the browser

Maintaining user/server state: cookies



Cookies: tracking a user's browsing behavior

Cookies can be used to:

- track user behavior on a given website (first party cookies)
- track user behavior across multiple websites (third party cookies) without user ever choosing to visit tracker site (!)
- tracking may be *invisible* to user:
  - rather than displayed ad triggering HTTP GET to tracker, could be an invisible link

third party tracking via cookies:

- disabled by default in Firefox, Safari browsers
- to be disabled in Chrome browser in 2023

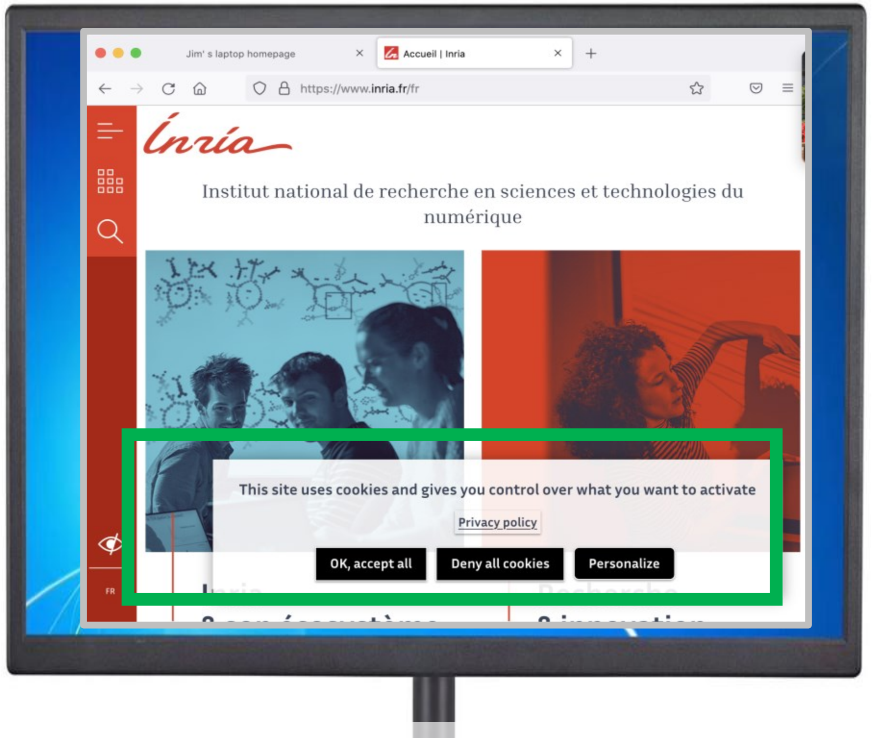
GDPR (EU General Data Protection Regulation) and cookies

"Natural persons may be associated with online identifiers […] such as internet protocol addresses, cookie identifiers or other identifiers […].

This may leave traces which, in particular when combined with unique identifiers and other information received by the servers, may be used to create profiles of the natural persons and identify them."

GDPR, recital 30 (May 2018)

when cookies can identify an individual, cookies are considered personal data, subject to GDPR personal data regulations



User has explicit control over whether or not cookies are allowed

Browser caching: Conditional GET

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

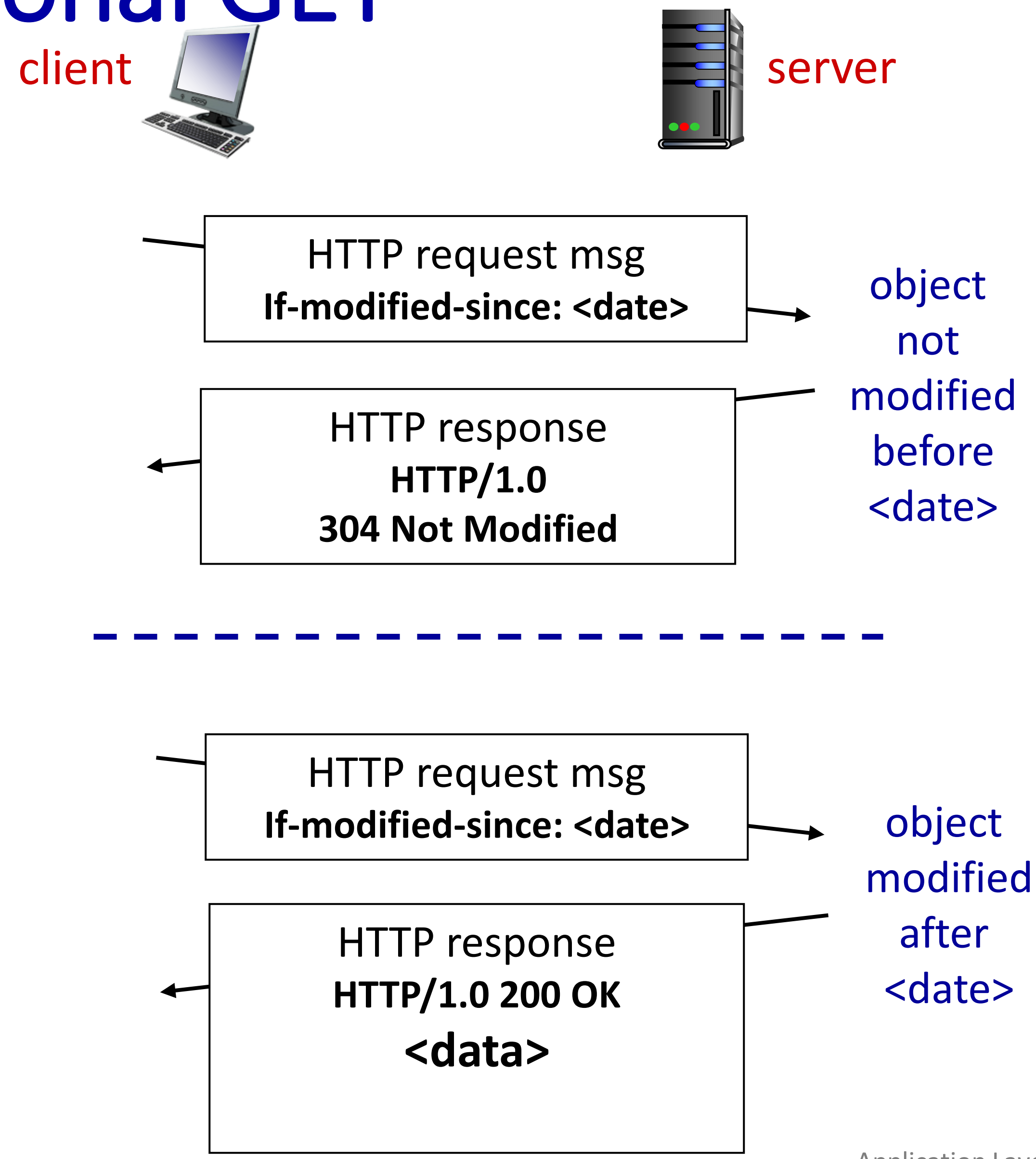
- no object transmission delay (or use of network resources)

- *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



2.2.6 HTTP/2

Key goal: reduce delay in multi-object HTTP requests, avoid parallel TCP connections

HTTP1.1: introduced multiple, pipelined GETs over single TCP connection

- 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)

HTTP/2

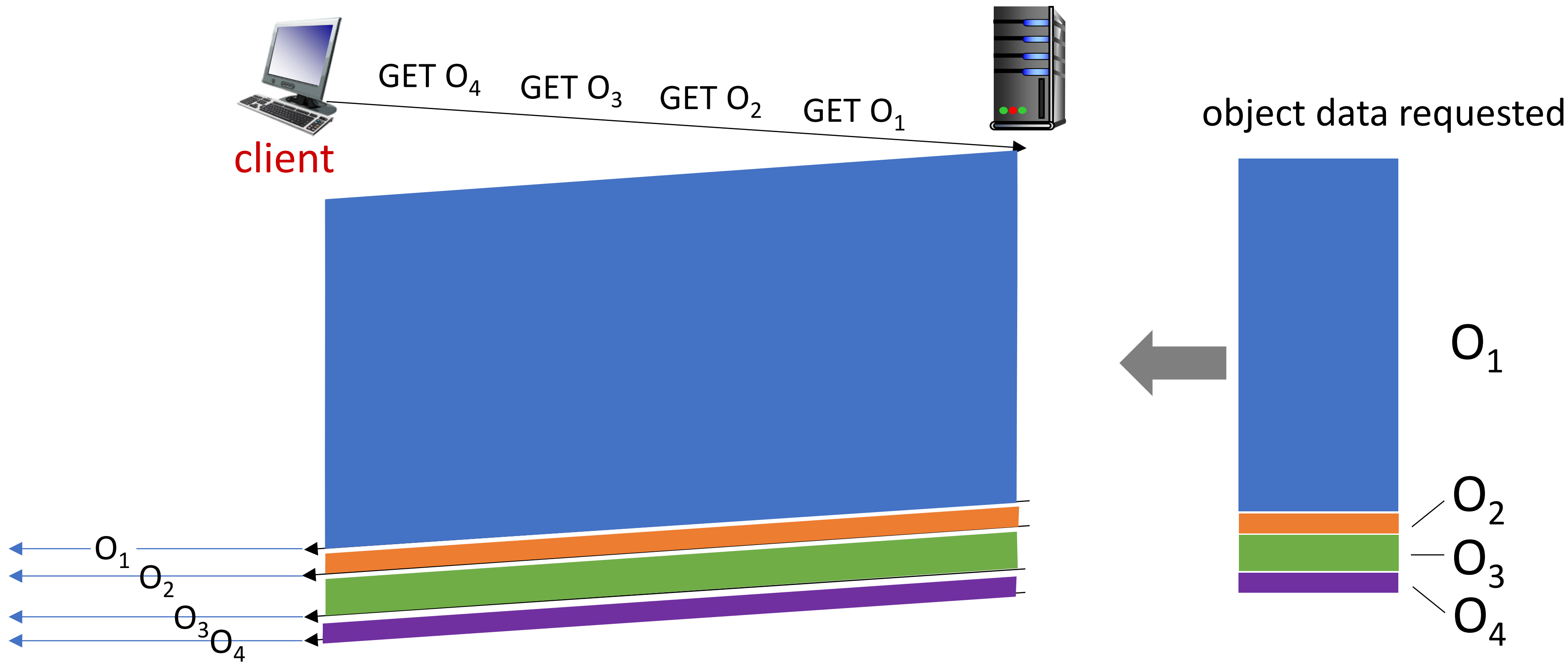
Key goal: reduce delay in multi-object HTTP requests, avoid parallel TCP connections

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, but represented in compressed binary frames
- transmission order of requested objects based on client-specified object priority
- push unrequested objects to client
- mitigate HOL blocking by dividing objects into frames, schedule frames using interleaving

HTTP/2: mitigating HOL blocking

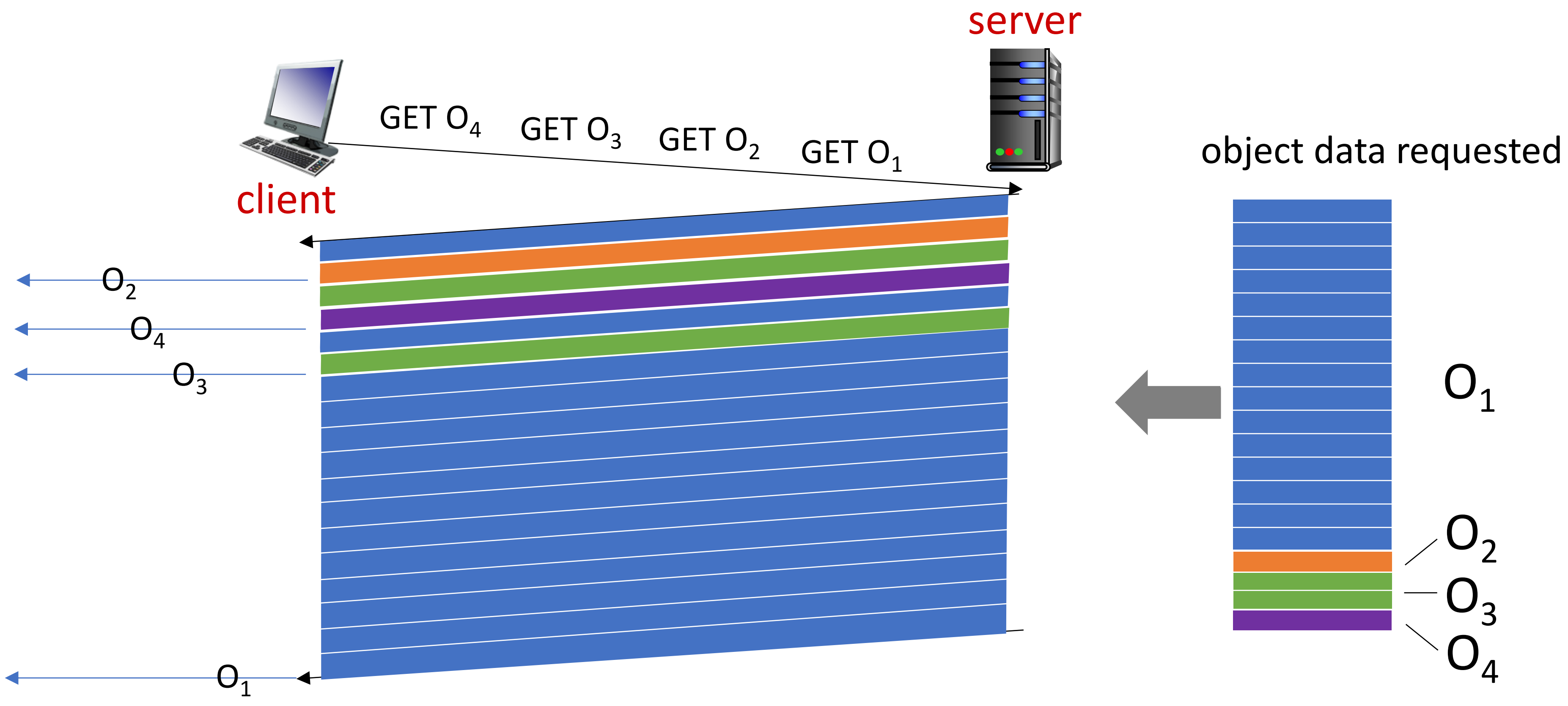
HTTP 1.1: client requests 1 large object (e.g., a big chunk of a video file) and 3 smaller objects



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



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- 2.5 Peer-to-peer file distribution
- 2.6 video streaming and content distribution networks

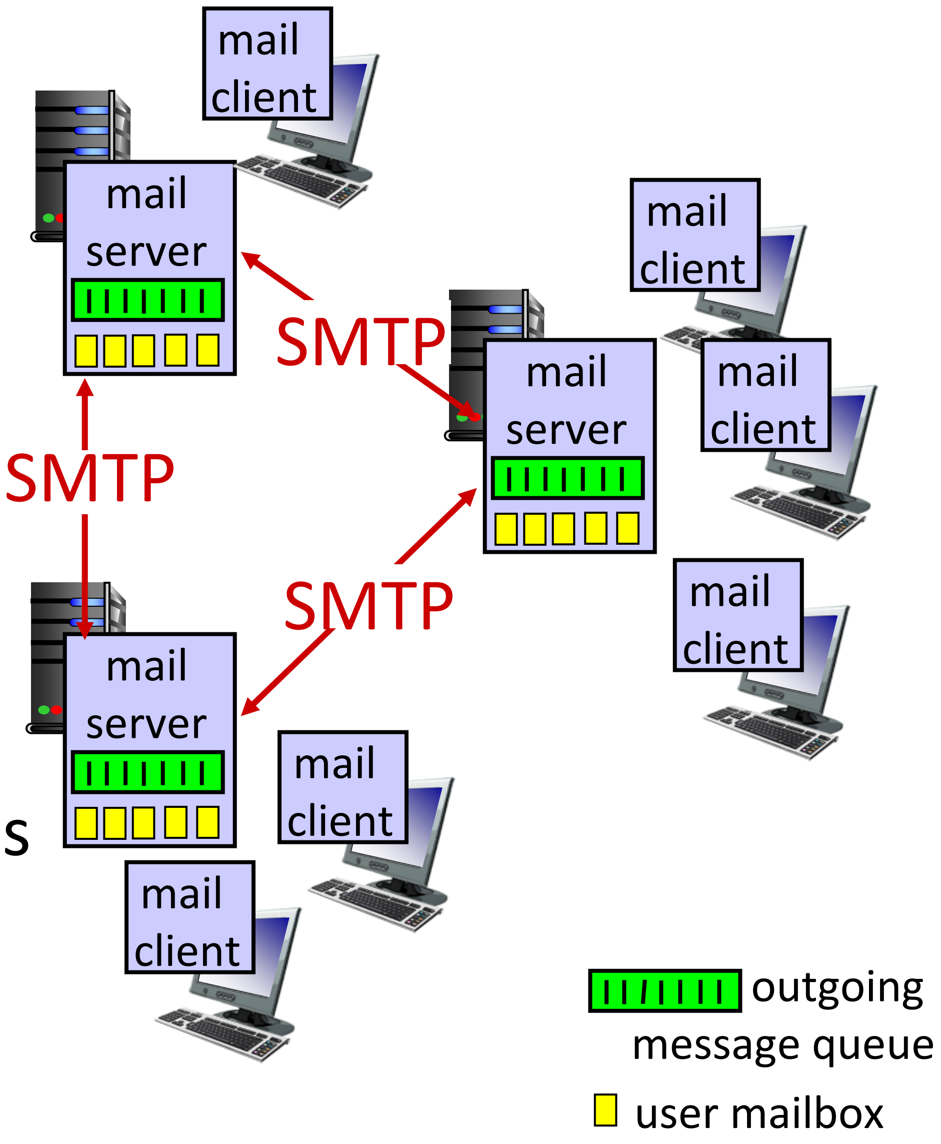
2.3 E-mail, SMTP, IMAP

- 2.3.1 SMTP
- 2.3.2 Mail message formats
- 2.3.3 Mail access protocols

2.3 E-mail

Three major components:

1. mail client
2. mail servers
3. simple mail transfer protocol: SMTP

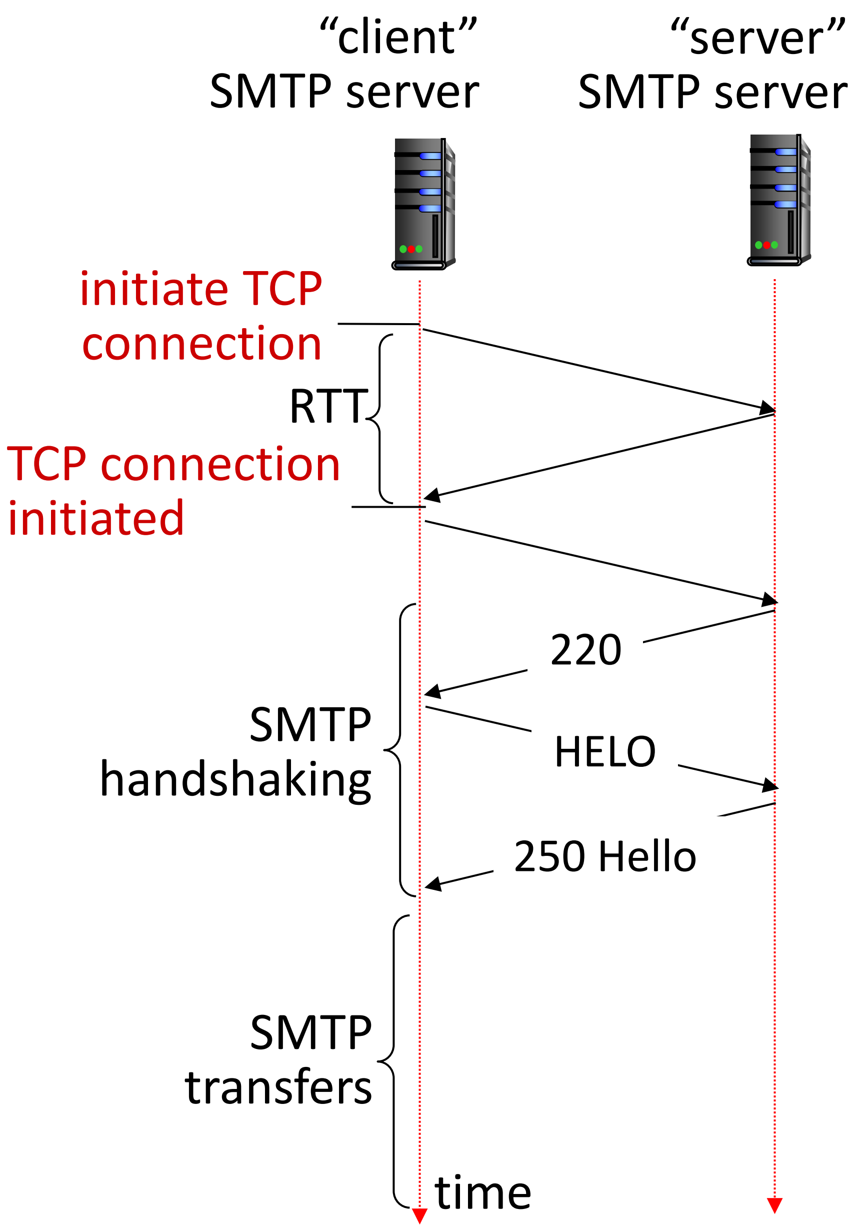


Mail client

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

2.3.1 SMTP RFC (~~821,~~ ~~2811,~~ 5321)

- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25 (or TLS port 587)
  - 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 MC to compose e-mail message “to” bob@someschool.edu

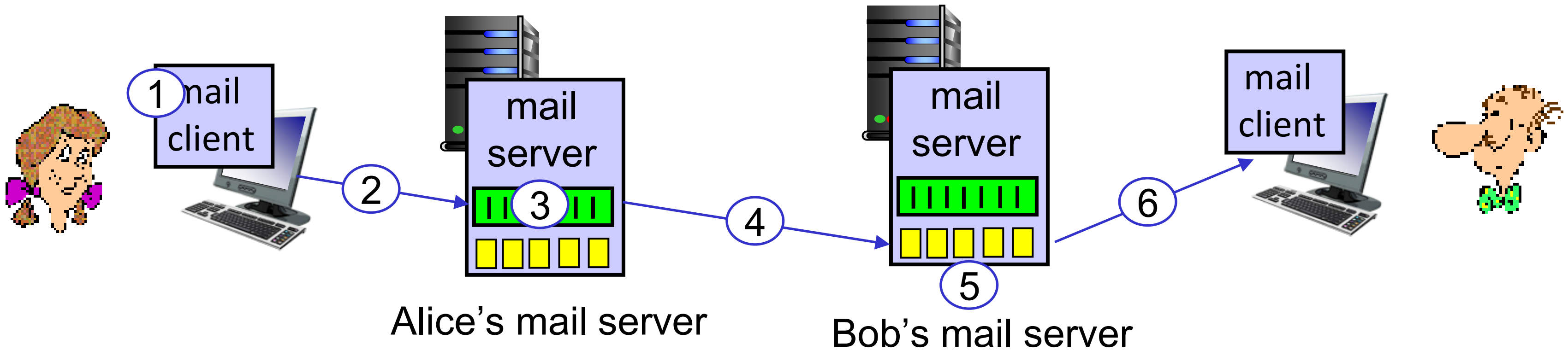
2) Alice’s MC sends message to her mail server using SMTP

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

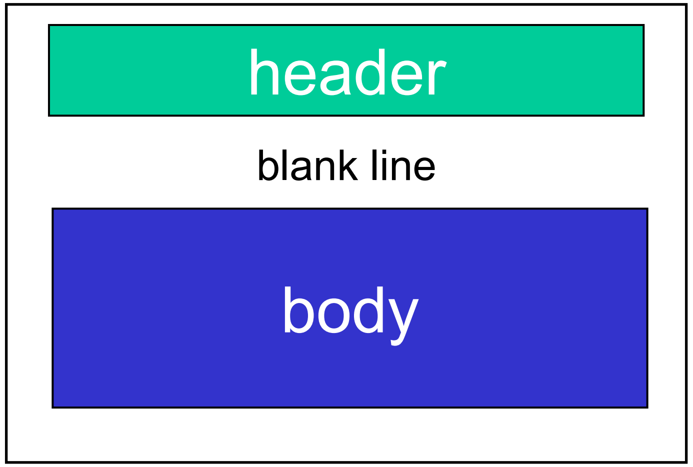
5) Bob’s mail server places the message in Bob’s mailbox

6) Bob invokes his mail client to read message



Sample SMTP interaction

S: 220 smtp.example.com
C: HELO relay.example.org
S: 250 Hello relay.example.org
C: MAIL FROM:<alice@example.org>
S: 250 Ok
C: RCPT TO:<bob@example.com>
S: 250 Ok
C: DATA
S: 354 Ready to receive
C: From: “Alice" <alice@example.org>
C: To: “Bob“<bob@example.com>
C: Date: Tue, 15 Jan 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice.
C: This is a test message
C: Your friend,
C: Bob
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye



https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol

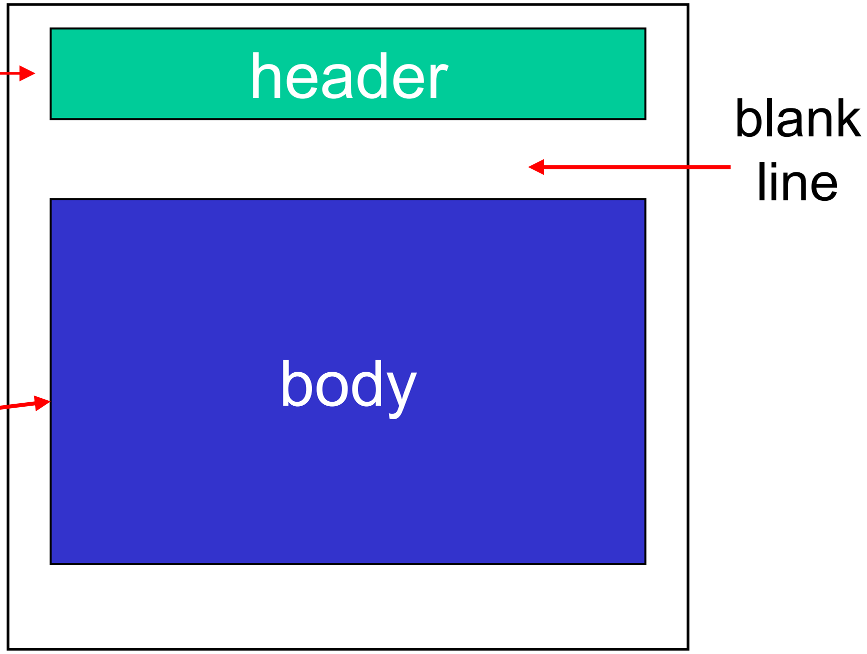
2.3.2 Mail message format

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

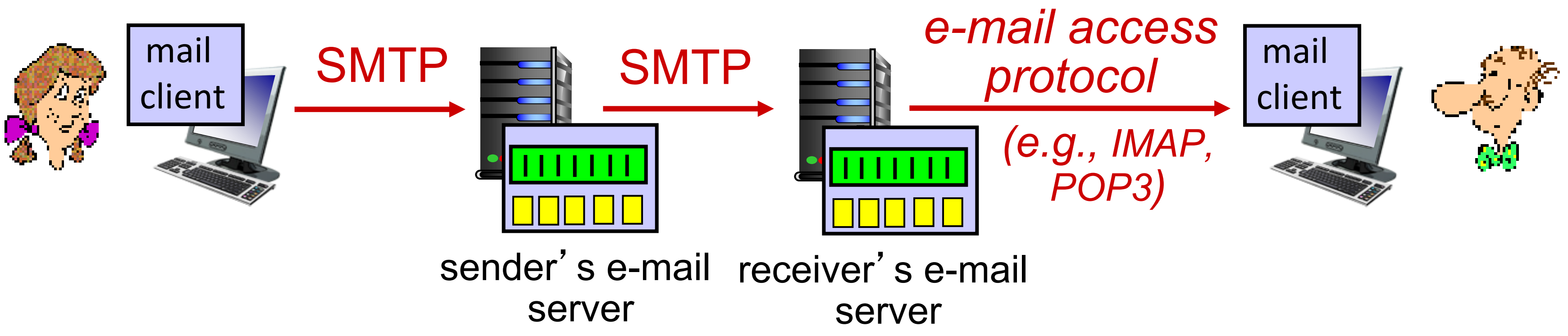
RFC 5322 defines *syntax* for e-mail message itself (like HTML defines syntax for web documents)

- 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



2.3.3 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]: “views” messages stored on server, IMAP provides retrieval, deletion
  - POP3: Post Office Protocol version 3. The email client downloads emails to client
- HTTP: gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of SMTP (to send), IMAP (or POP3) to retrieve e-mail messages

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- 2.5 Peer-to-peer file distribution
- 2.6 video streaming and content distribution networks

DNS: Domain Name System

Domain names:

- uia.no, youtube.com

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., vg.no - used by humans

- Hostname-to-IP-address translation
  - Internet “phone book”
  - Domains vs. hosts

Domain Name System (DNS):

1. *distributed database* implemented in hierarchy of many *name servers*
2. *DNS protocol*
   - *application-layer protocol:* hosts, DNS servers communicate to *resolve* names (address/name translation)
   - Uses the UDP transport layer protocol using port 53
- complexity at network’s “edge”

uia.no → 158.37.242.21, 158.37.218.21, 129.240.118.130, 158.37.242.20, 158.37.218.20

DNS: services, structure

DNS services:

- hostname-to-IP-address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
  - e.g.: uia.no → mx.uhpost.no
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

Q: Why not centralize DNS?

- single point of failure
- no redundancy
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!

- Comcast DNS servers alone: 600B DNS queries/day
- Akamai DNS servers alone: 2.2T DNS queries/day

Thinking about the DNS

Very large distributed database:

- ~ billion records, each simple

handles many *trillions* of queries/day:

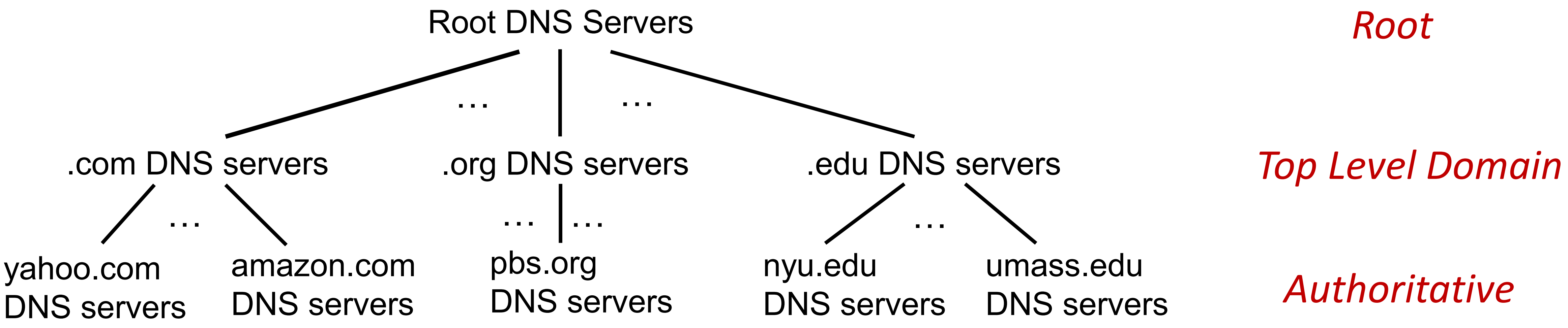
- *performance matters:* almost every Internet transaction interacts with DNS - msecs count!

organizationally, physically decentralized:

- millions of different organizations responsible for their records

"bulletproof": reliability, security

DNS: a distributed, hierarchical database



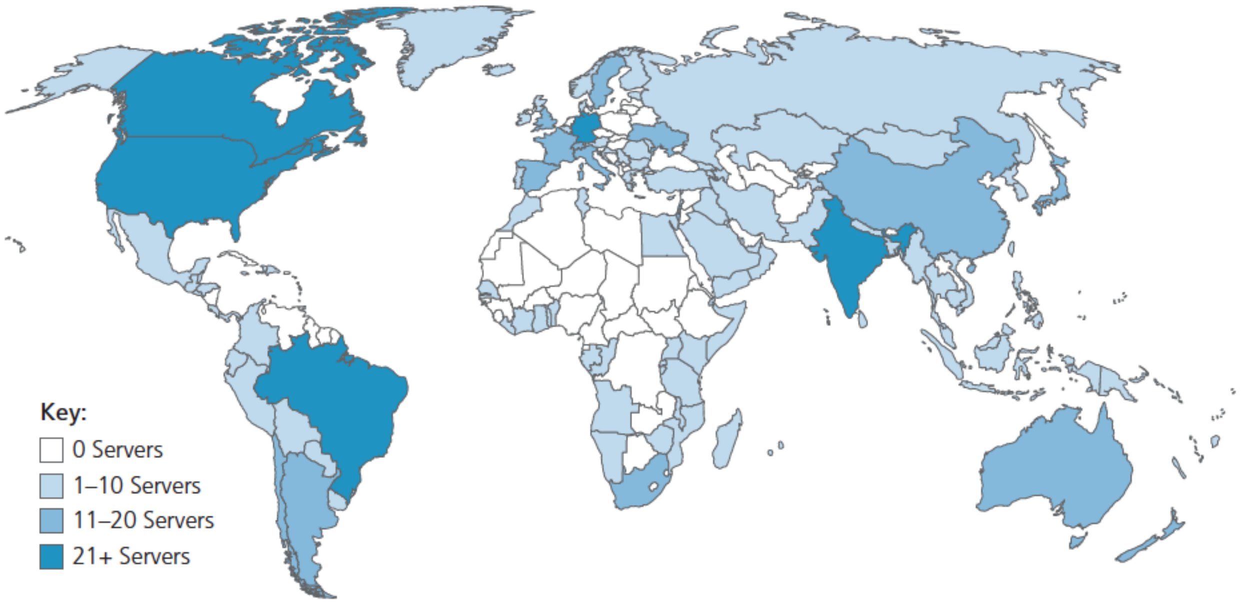
Client wants IP address for www.amazon.com; 1st approximation:

- 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

- 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)
e.g., A.ROOT-SERVERS.NET fixed IP 198.41.0.4



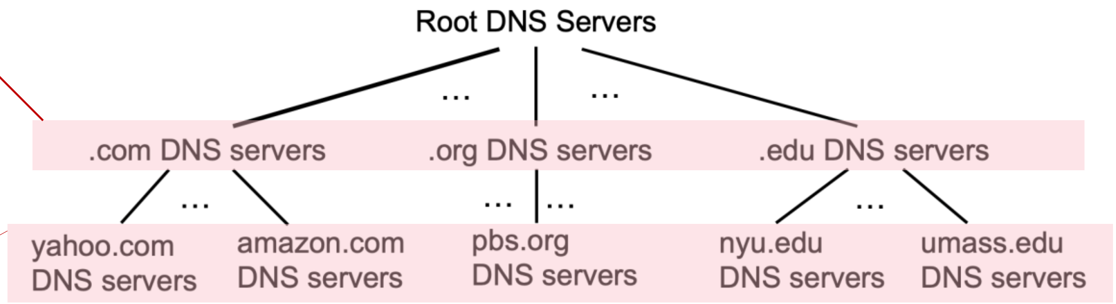
Top-Level Domain, and authoritative servers

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
- Answers requests by returning a list of the authoritative name servers

1058 TLDs :

- 730 generic TLD
- 301 country code 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

- Local DNS servers don't strictly belong to hierarchy
- When a host makes DNS query, it is sent to its *local* DNS server:
  - checks its local cache of recent name-to-address translation pairs (possibly out of date), or
  - forwarding request into DNS hierarchy for resolution

- Each ISP has a local DNS name server; to find yours:

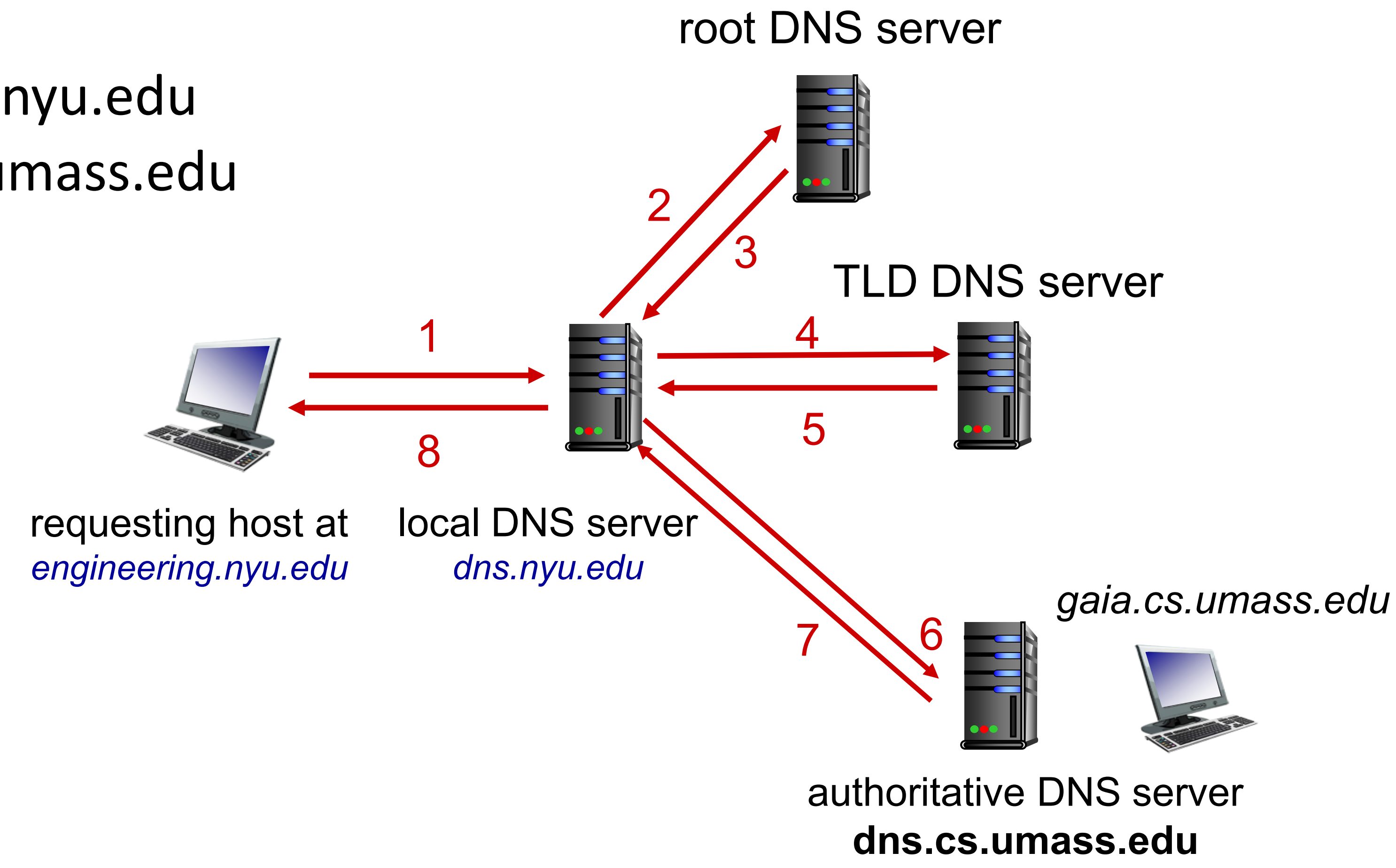
  - C:\> **`ipconfig /all`**

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"



Looking up a host’s IP address

```
C:\Users\sigurde>nslookup uia.no
Server:  nv2.ti.telenor.net
Address:  2001:4600:4:1fff::253

Non-authoritative answer:
Name:    uia.no
Addresses:  2001:700:100:118::130
          129.240.118.130
```

```
C:\Users\sigurde>nslookup uia.no
Server:  adgrm01.uia.no
Address:  158.37.218.20

Name:    uia.no
Addresses:  2001:700:100:118::130
          2001:700:1500:7250::242:20
          2001:700:1500:7250::242:21
          2001:700:1501:7250::218:20
          2001:700:1501:7250::218:21
          158.37.218.21
          129.240.118.130
          158.37.242.20
          158.37.218.20
          158.37.242.21
```

Looking up IP address of an authorative DNS server

```
C:\Users\sigurde>nslookup -query=soa uia.no
Server:  nv2.ti.telenor.net
Address:  2001:4600:4:1fff::253

Non-authoritative answer:
uia.no
        primary name server = ns1.uia.no
        responsible mail addr = hostmaster.uia.no
        serial  = 2025012300
        refresh = 28800 (8 hours)
        retry   = 3600 (1 hour)
        expire  = 86400 (1 day)
        default TTL = 3600 (1 hour)
```

```
C:\Users\sigurde>nslookup uia.no ns1.uia.no
Server:  ns1.uia.no
Address:  2001:700:1500:d270::245:200

Name:    uia.no
Addresses:  2001:700:100:118::130
          129.240.118.130
```

DNS records

DNS: distributed database storing resource records

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

type=A

- `name` is hostname
- `value` is IPv4 address

www.demosite.com. 3600 A 207.124.120.25

type=CNAME

- `name` refers to a “canonical” name specified in `value`

shop.example.com. 3600 CNAME shops.myshopify.com.

type=NS

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

demosite.com. 3600 NS ns1.demosite.net.

type=MX

- `name` is domain (e.g., uia.no)
- `value` is name of SMTP mail server for this domain

demosite.com. 3600 MX 10 mail1.demosite.com.

DNS records

DNS: distributed database storing resource records

type=SOA

- A start of authority (SOA) is a DNS record with information about authoritative name-server for a domain name

```
C:\Users\sigurde>nslookup -query=a uia.no
Server:  adgrm01.uia.no
Address:  158.37.218.20

Name:    uia.no
Addresses:  158.37.218.21
          129.240.118.130
          158.37.242.20
          158.37.218.20
          158.37.242.21
```

```
Address:  2001:4600:4:1fff::253

Non-authoritative answer:
uia.no  nameserver = ns2.uia.no
uia.no  nameserver = nn.uninett.no
uia.no  nameserver = ns1.uia.no

nn.uninett.no   AAAA IPv6 address = 2001:700:0:503::aa:5302
ns1.uia.no      AAAA IPv6 address = 2001:700:1500:d270::245:200
ns2.uia.no      AAAA IPv6 address = 2001:700:1501:d270::221:200
nn.uninett.no   internet address = 158.38.0.181
ns1.uia.no      internet address = 158.37.245.200
ns2.uia.no      internet address = 158.37.221.200
```

```
Non-authoritative answer:
uia.no  MX preference = 10, mail exchanger = mx.uhpost.no
```

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!*

Locally cached DNS information

```
C:\Users\sigurde>ipconfig /displaydns

Windows IP Configuration

    finn.no
    ----------------------------------------
    Record Name . . . . . : finn.no
    Record Type . . . . . : 1
    Time To Live  . . . . : 297
    Data Length . . . . . : 4
    Section . . . . . . . : Answer
    A (Host) Record . . . : 35.228.105.46


    finn.no
    ----------------------------------------
    No records of type AAAA
```

Registering domain info into the DNS

example: new startup "Network Utopia"

- register name networkuptopia.com at *DNS registrar* (I Norge: www.uniweb.no, www.norid.no)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts NS and A resource records into .com TLD server:

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

    `(dns1.networkutopia.com, 212.212.212.1, A)`
  - type A record mapping www.networkuptopia.com to an IP address
  - type MX record for networkutopia.com

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

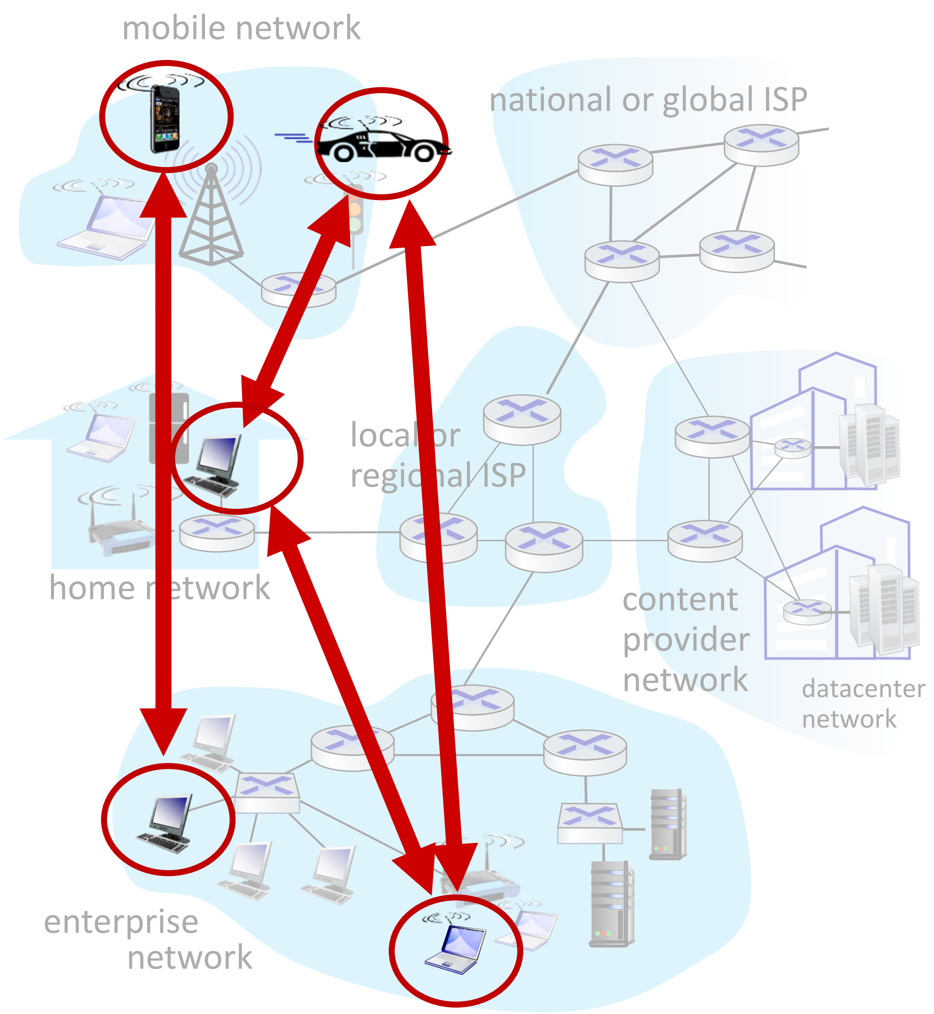
- intercept DNS queries, returning bogus replies
  - DNS cache poisoning
  - RFC 4033: DNSSEC authentication services

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- 2.5 Peer-to-peer file distribution
- 2.6 video streaming and content distribution networks

Peer-to-peer (P2P) model

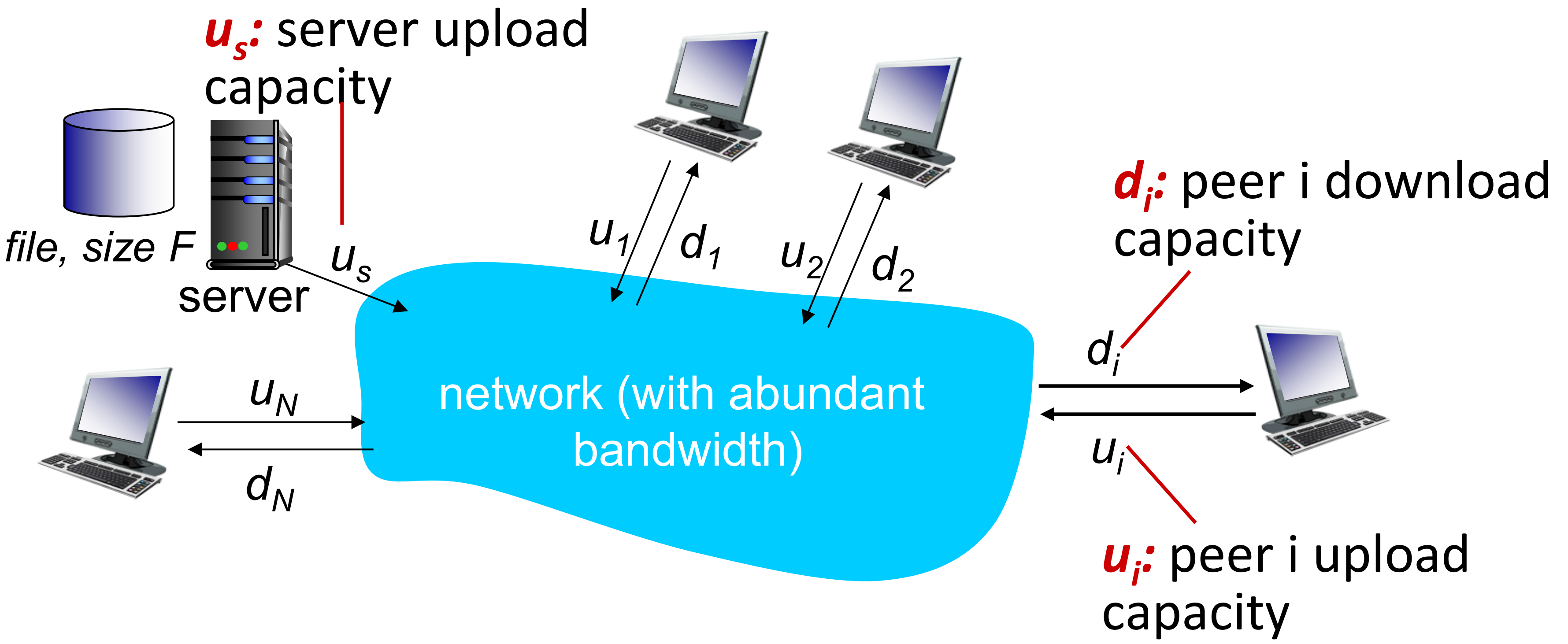
- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, and new service demands
- peers are intermittently connected and change IP addresses
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



File distribution: client-server vs P2P

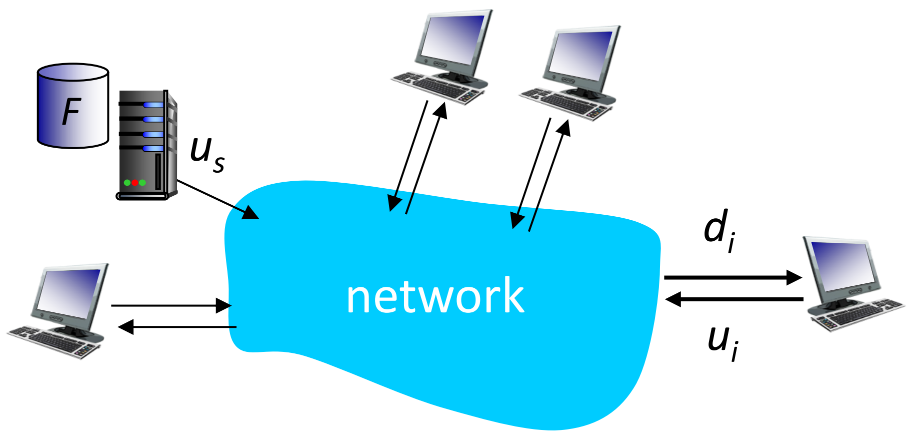
Q: how much time to distribute file (size *F*) from one server to *N peers*?

- peer upload/download capacity is limited resource



File distribution time: client-server

- *server transmission:* must sequentially send (upload) N file copies:
  - time to upload one copy: F/u_s
  - time to upload N copies: NF/u_s



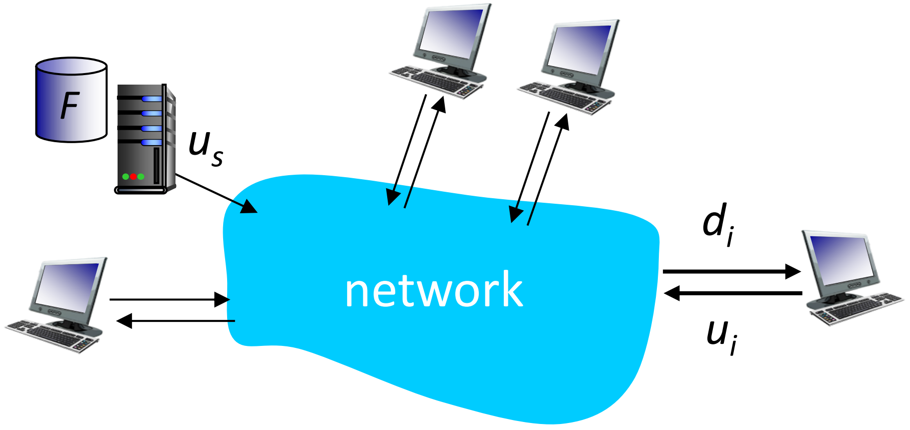
- *client:* each client must download file copy
  - max client download time: F/d_{min}
  - where d_{min} = lowest client download rate

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

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

increases linearly in N

File distribution time: P2P



- *server transmission:* must upload at least one copy:
  - time to upload one copy: $\frac{F}{u_s}$
- *client:* each client must download file copy
  - max client download time: $\frac{F}{d_{min}}$
- *clients:* upload *NF* bits
  - total upload time: $\frac{N \cdot F}{u_s + \sum u_i}$

minimum time to distribute F to N clients using P2P approach

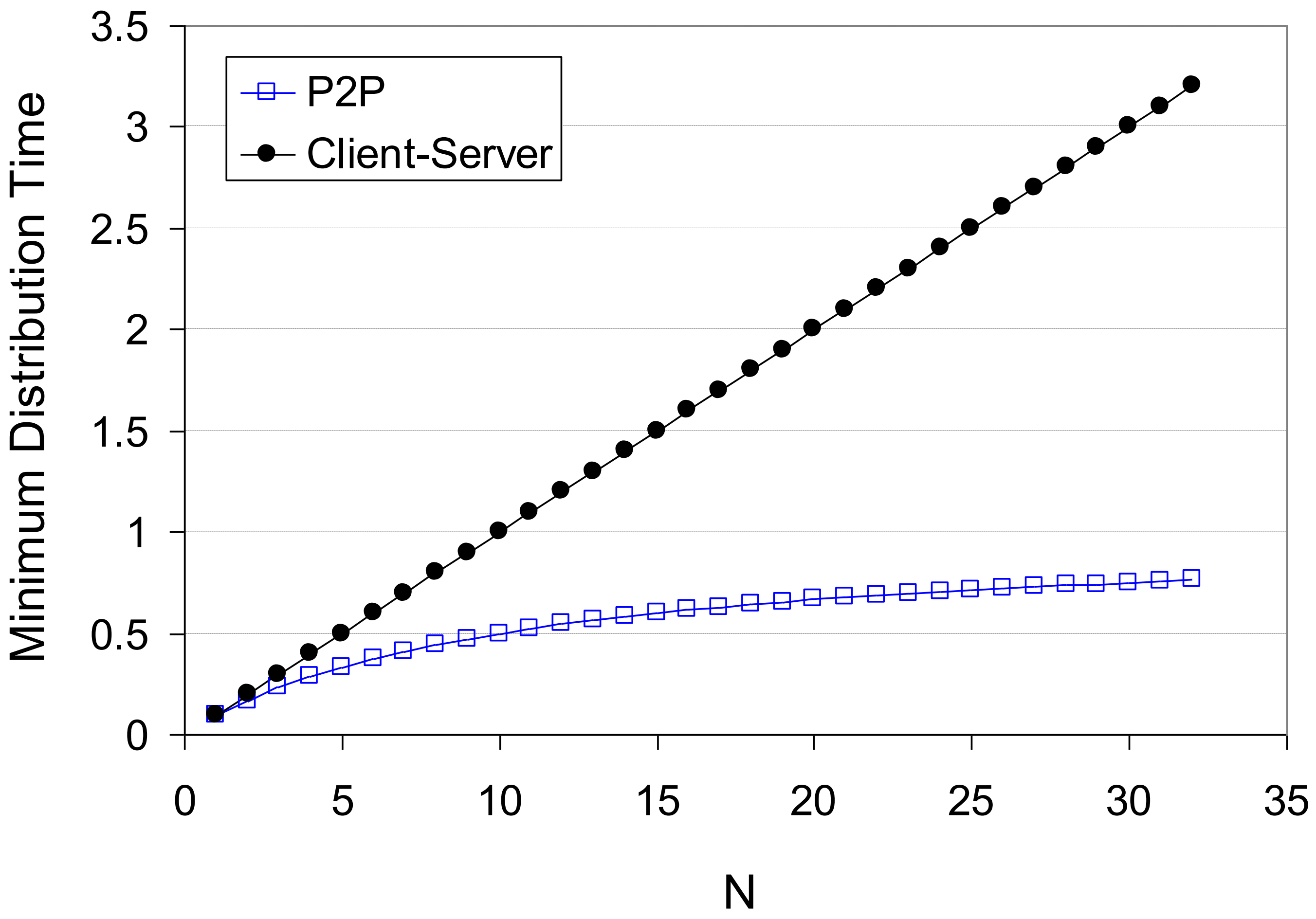
$$D_{P2P} > max\{ \frac{F}{u_s}, \frac{F}{d_{min}}, \frac{N \cdot F}{u_s + \sum u_i} \}$$

increases linearly in *N* …

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

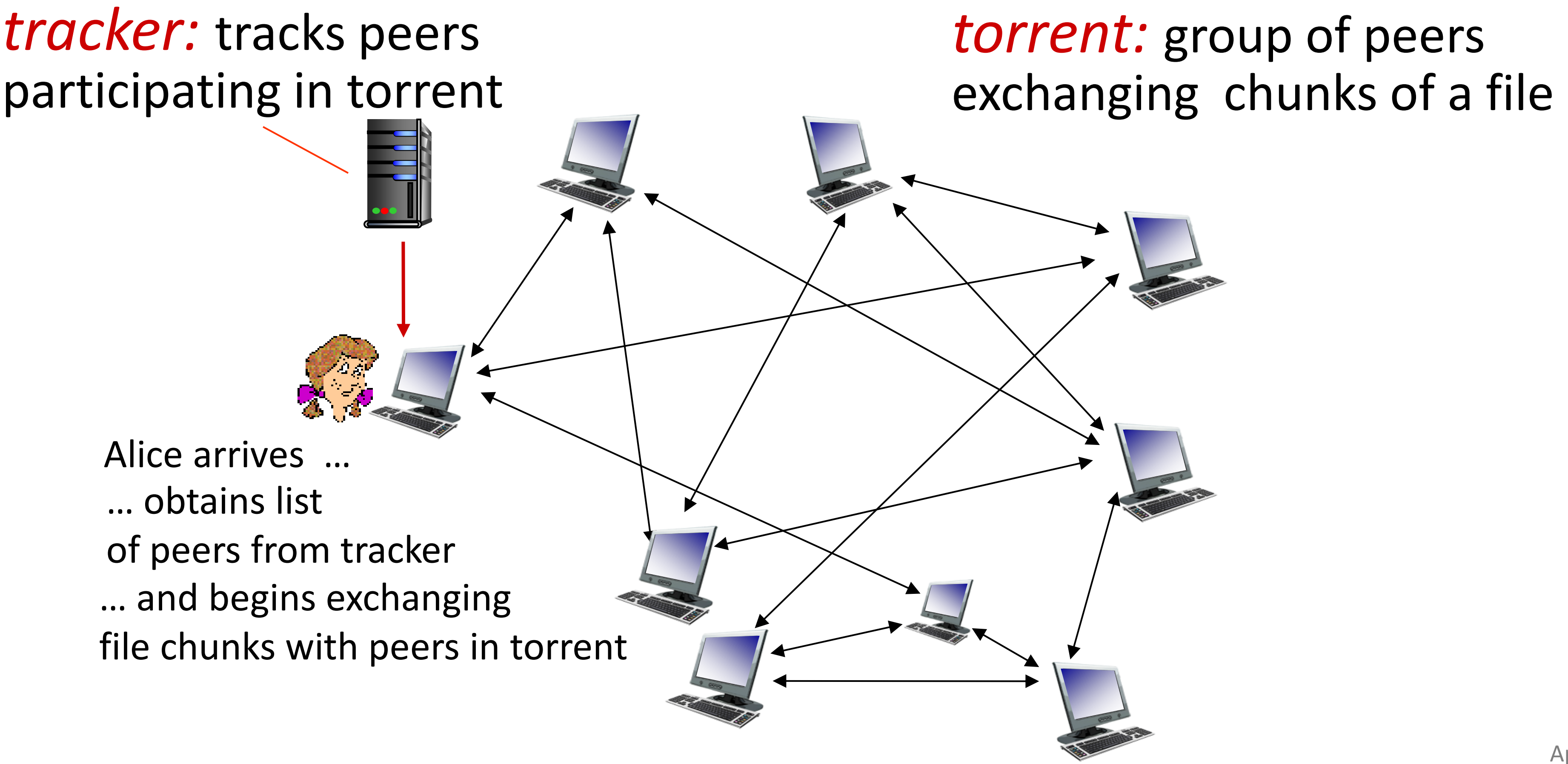
Client-server vs. P2P: example

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



P2P file distribution: BitTorrent

- file divided into 512KB (or less) chunks called pieces
- peers in torrent send/receive file piece by piece



Application layer: overview

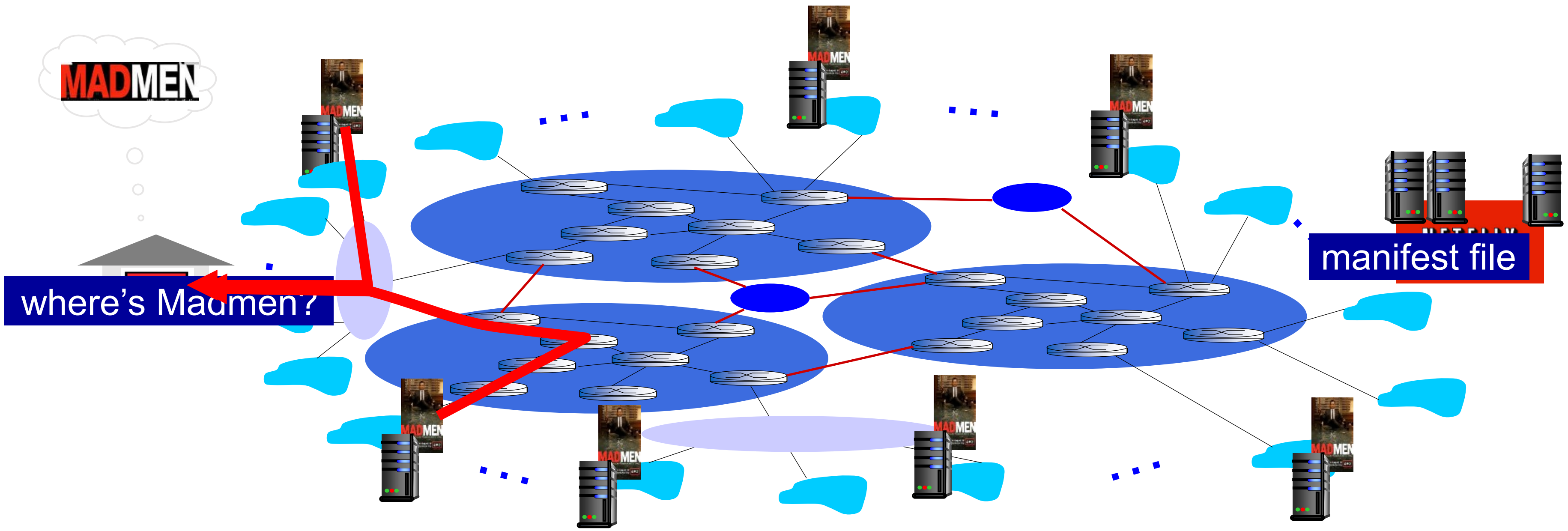
- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- 2.5 Peer-to-peer file distribution
- 2.6 Video streaming and content distribution networks

2.6 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 ~1 billion users?
- *challenge:* heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure

Content distribution networks (CDNs)

- CDN: stores copies of video content at CDN nodes
- subscriber requests content, service provider returns manifest
  - using manifest, client retrieves content at highest supportable rate
  - may choose different rate or copy if network path congested



Chapter 2: Summary

our study of network application layer is now complete!

- application models
  - 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, CDNs
- UDP og TCP-socket programmering i neste kapittel

Chapter 2: Summary

Most importantly: learned about *protocols*!

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - *headers*: fields giving info about data
  - *data:* info (payload) being communicated

important themes:

- centralized vs. decentralized
- stateless vs. stateful
- scalability
- reliable vs. unreliable message transfer
- “complexity at network edge”