

Computer and Communication Networks (2026)

Iordanis Koutsopoulos, Vasilios Siris, George Xylomenos
Professors, Department of Informatics, AUEB

Acknowledgment

Adapted from the slides accompanying the book «Computer Networking: A Top-Down Approach», 9th edition, by J.F Kurose and K.W. Ross, Pearson 2025.

Original slides copyright notice: 1996-2025, J.F Kurose and K.W. Ross, All Rights Reserved

Application layer: roadmap

- Principles of network applications
- Web and HTTP
- The Domain Name System DNS
- Video streaming and CDNs

Some network apps

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

Q: your favorites?

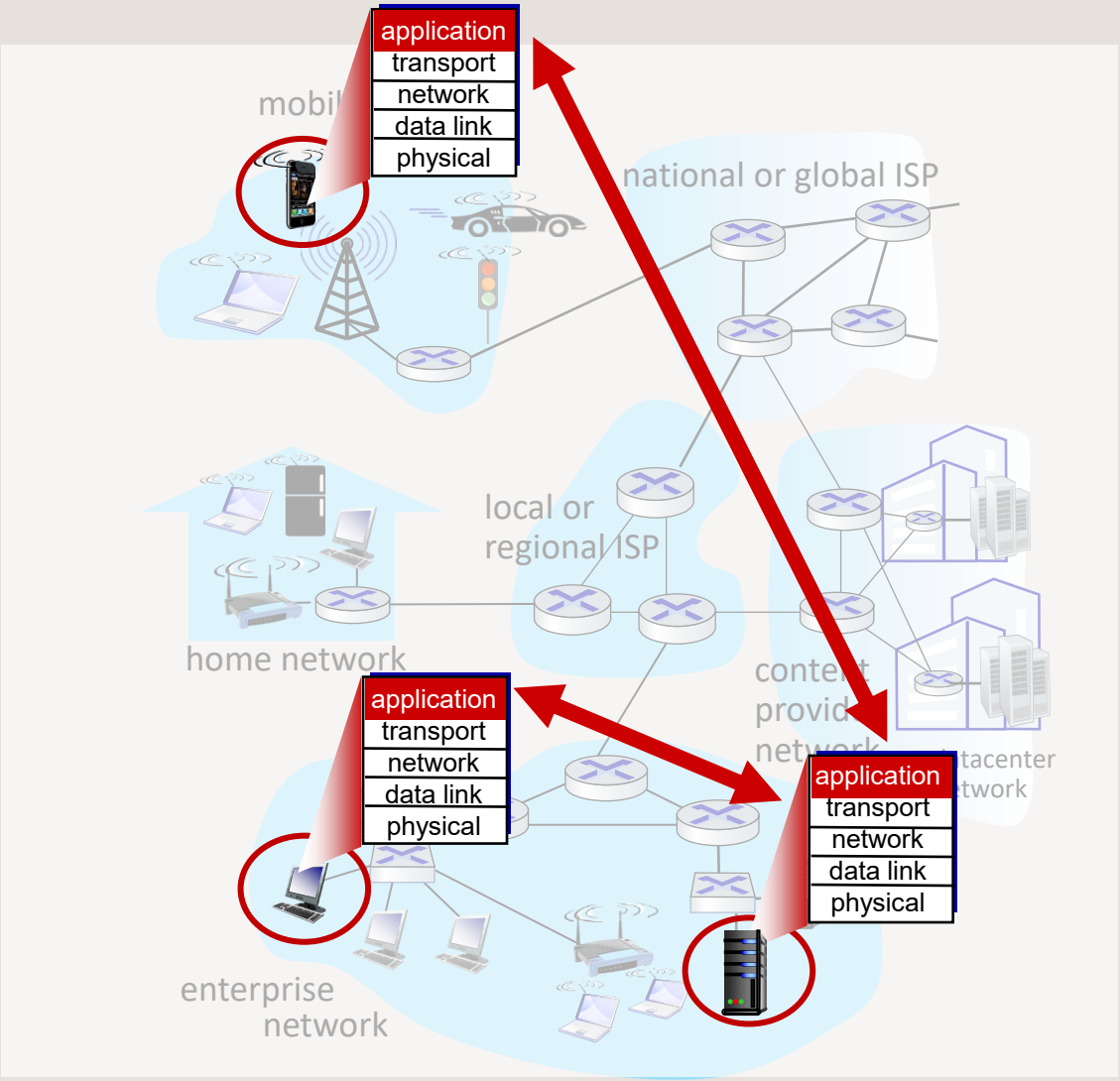
Creating a network app

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

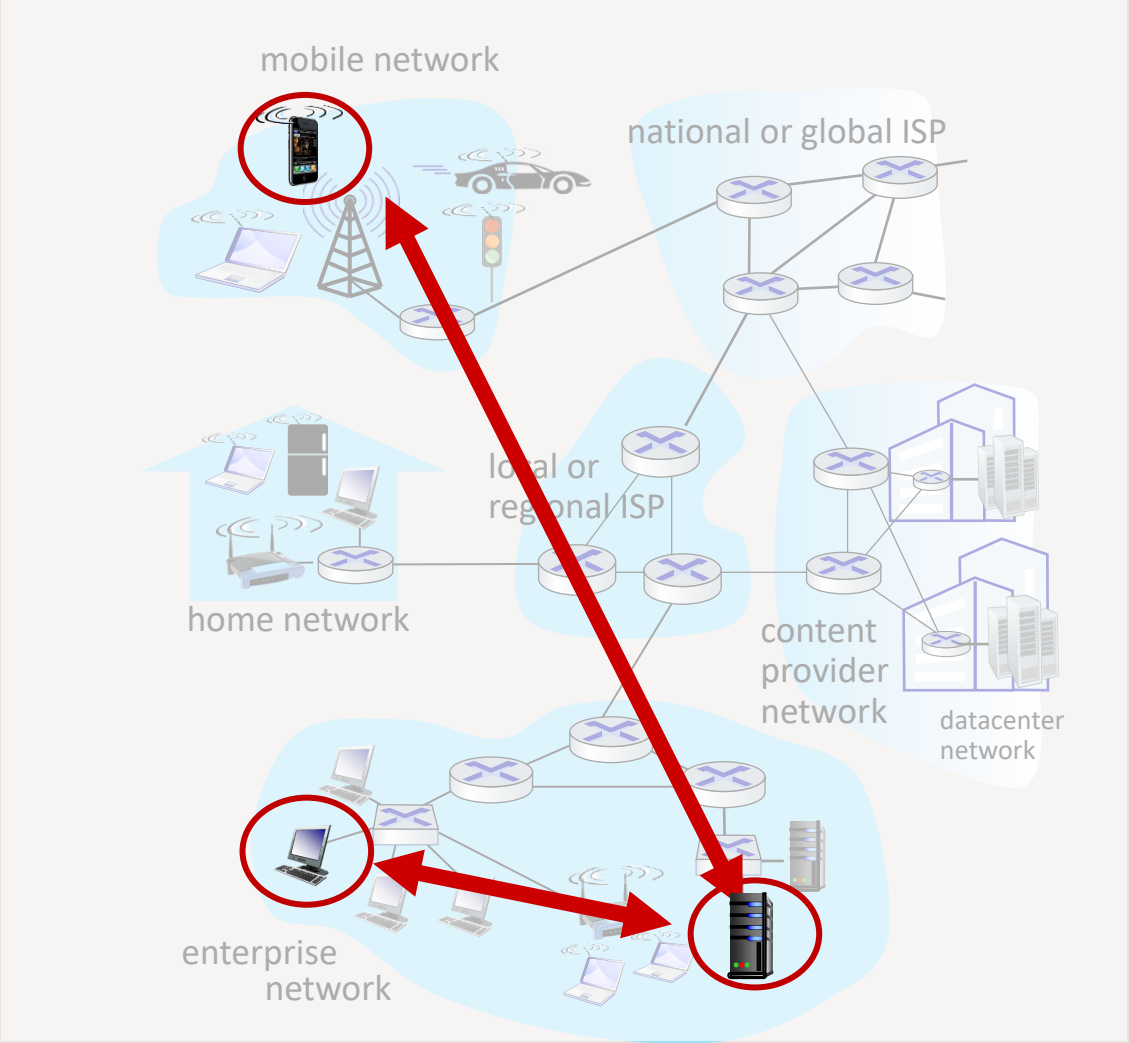
server:

- 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



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

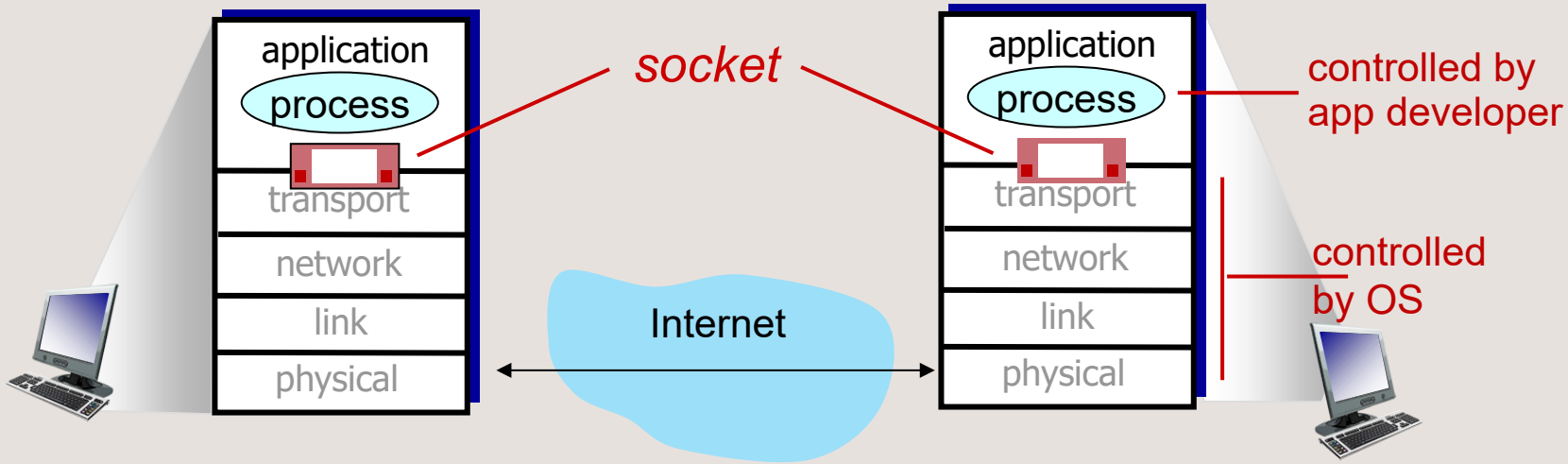
clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

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
- more shortly…

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., Zoom

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, …

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.

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

Application layer: roadmap

- Principles of network applications
- Web and HTTP
- The Domain Name System DNS
- Video streaming and CDNs

Web and HTTP

First, a quick review…

- 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

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

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

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

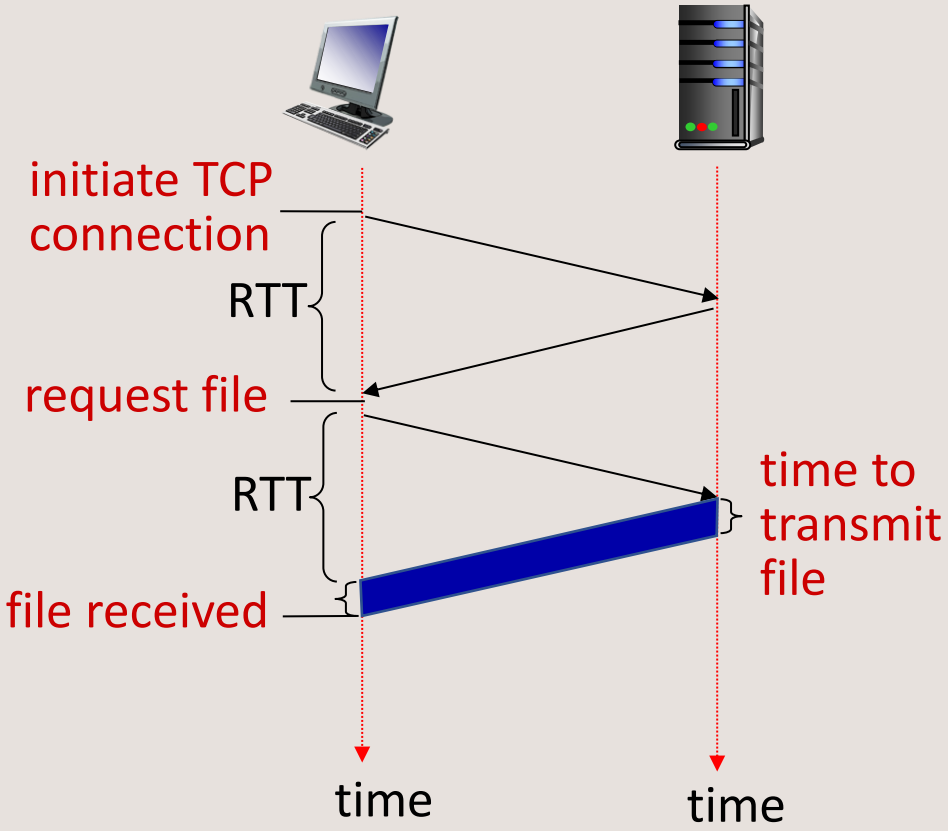
time

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)

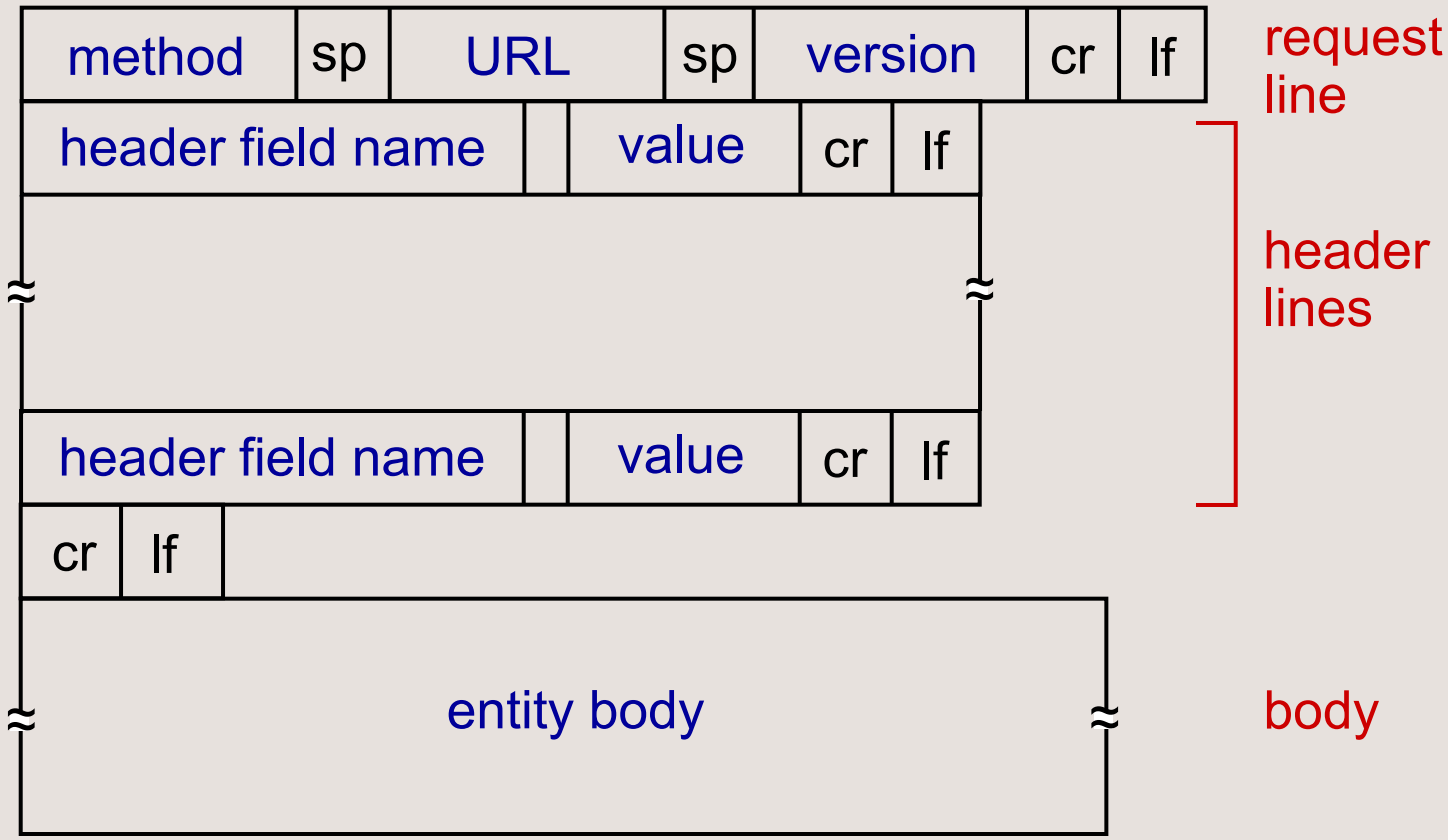
carriage return character
line-feed character

request line (GET, POST, HEAD commands)

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

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

HTTP request message: general format



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/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 POST HTTP request message

HTTP response message

status line (protocol status code status phrase) → `HTTP/1.1 200 OK`

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

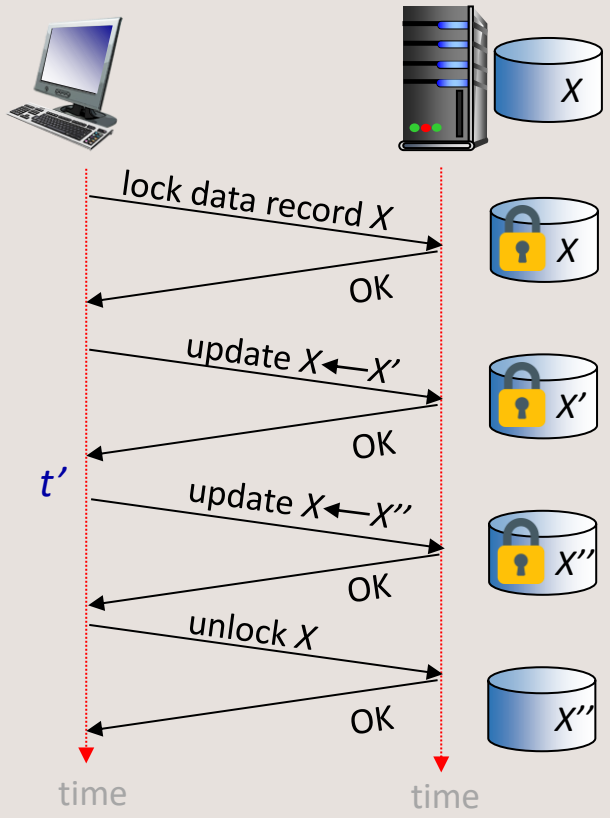
505 HTTP Version Not Supported

Maintaining user/server state: cookies

Recall: HTTP GET/response interaction is *stateless*

- no notion of multi-step exchanges of HTTP messages to complete a Web "transaction"
  - no need for client/server to track "state" of multi-step exchange
  - all HTTP requests are independent of each other
  - no need for client/server to "recover" from a partially-completed-but-never-completely-completed transaction

a stateful protocol: client makes two changes to X, or none at all



Q: what happens if network connection or client crashes at *t'* ?

Maintaining user/server state: cookies

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

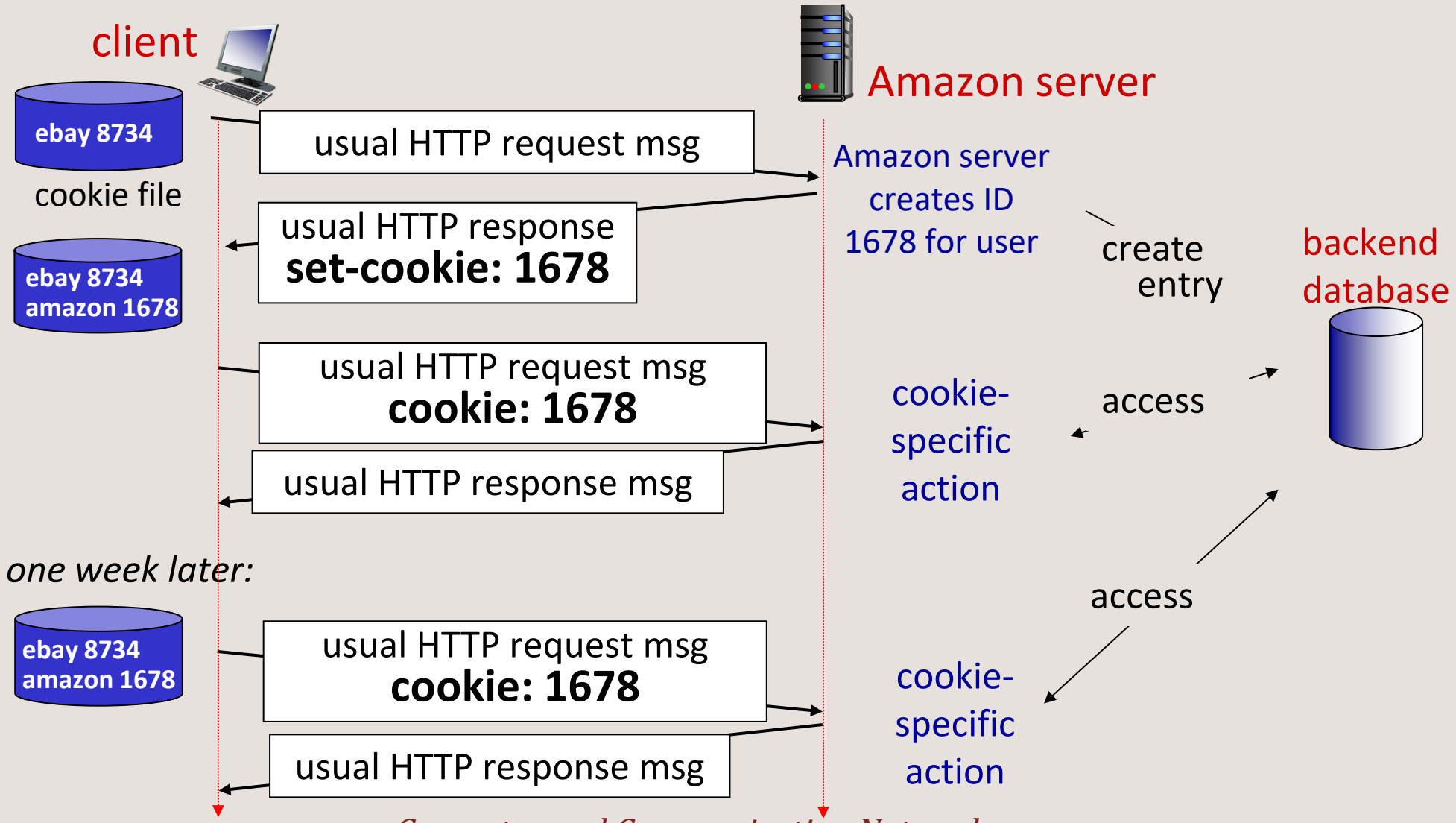
four components:

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:

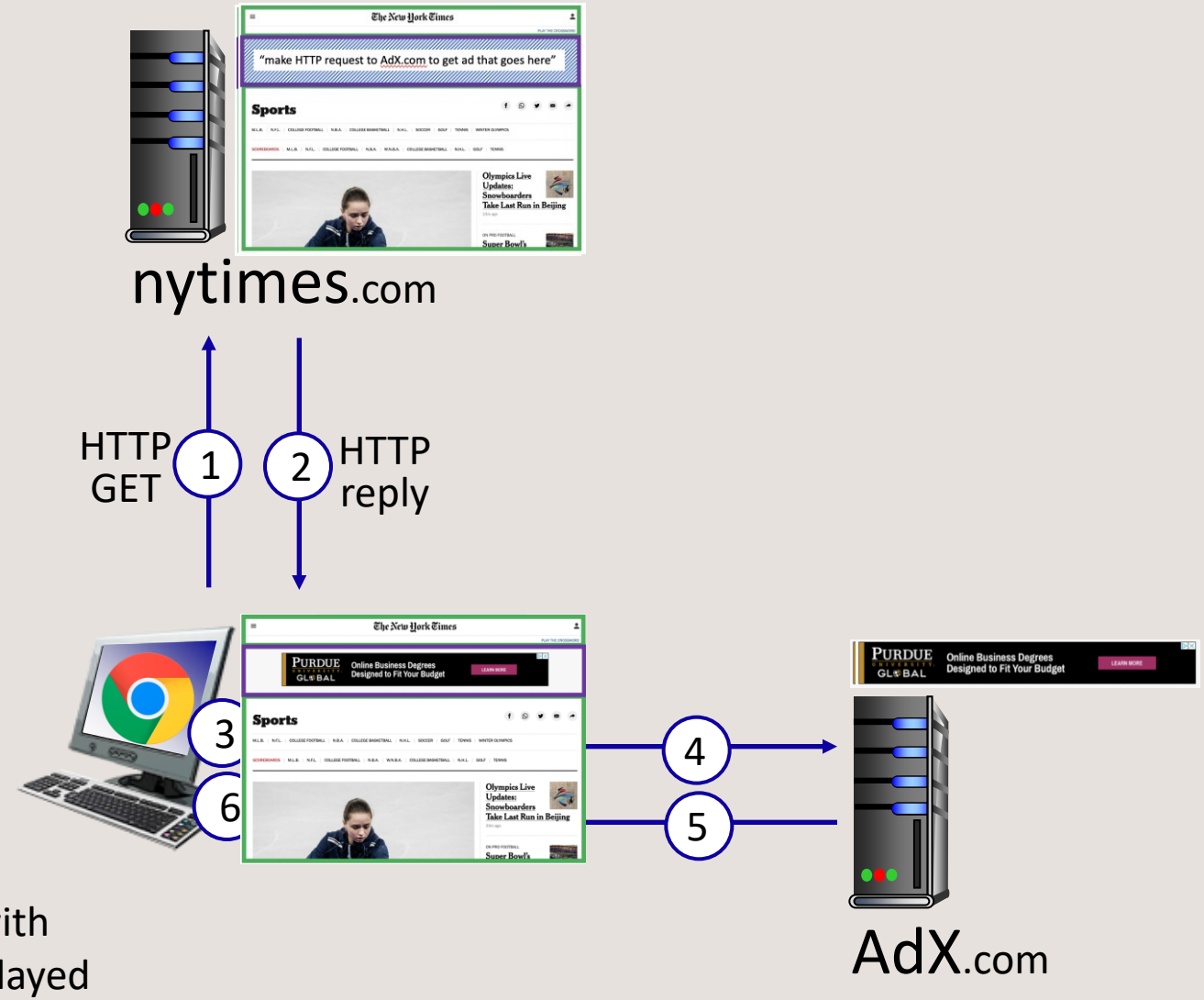
- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

Example: displaying a NY Times web page

1 2 GET base html file from nytimes.com

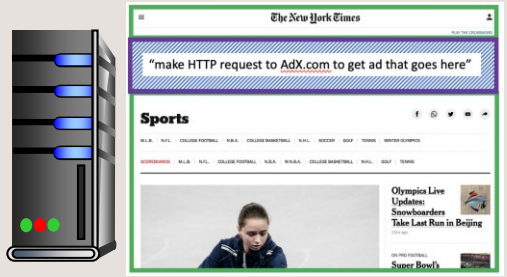
4 5 fetch ad from AdX.com

7 display composed page



NY times page with embedded ad displayed

Cookies: tracking a user's browsing behavior



nytimes.com (sports)



1634: sports, 2/15/22

"first party" cookie – from website you chose to visit (provides base html file)

HTTP GET

HTTP reply
Set cookie: 1634

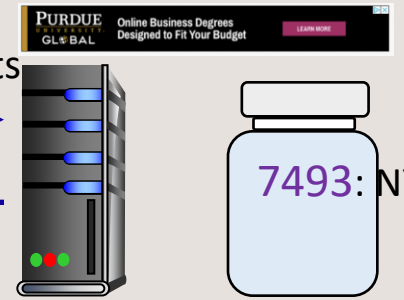


HTTP GET
Referrer: NY Times Sports

4

5

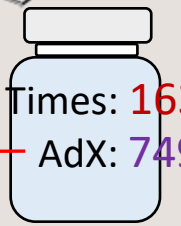
HTTP reply
Set cookie: 7493



AdX.com

7493: NY Times sports, 2/15/22

"third party" cookie – from website you did not choose to visit

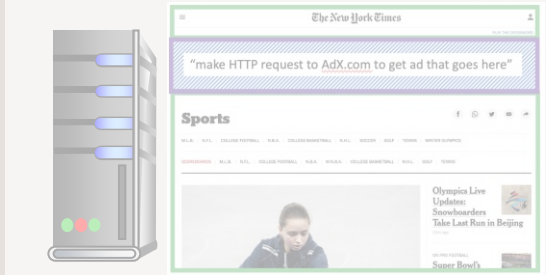


NY Times: 1634
AdX: 7493

Cookies: tracking a user's browsing behavior



socks.com

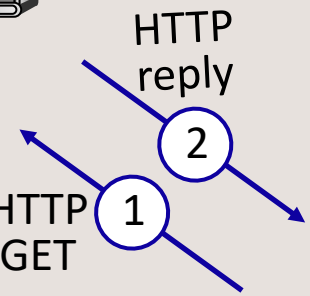


nytimes.com



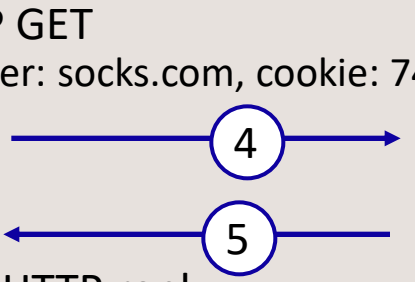
AdX:

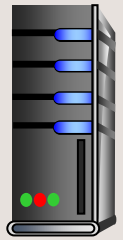
- *tracks my web browsing* over sites with AdX ads
- can return targeted ads based on browsing history



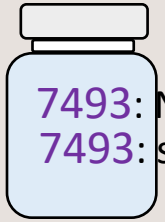






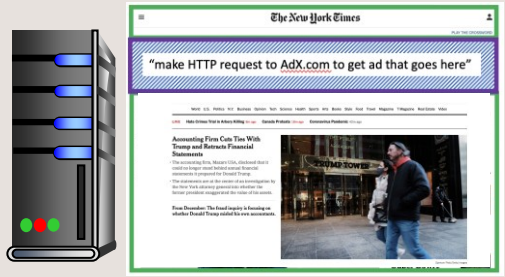


AdX.com



Cookies: tracking a user's browsing behavior (one day later)





nytimes.com (arts)

1634: sports, 2/15/22
1634: arts, 2/17/22

HTTP GET
cookie: 1634

HTTP reply
Set cookie: 1634



HTTP GET
Referrer: nytimes.com, cookie: 7493

4

5

HTTP reply
Set cookie: 7493
Returned ad for socks!

NY Times: 1634
AdX: 7493

AdX.com

7493: NY Times sports, 2/15/22
7493: socks.com, 2/16/22
7493: NY Times arts, 2/15/22

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

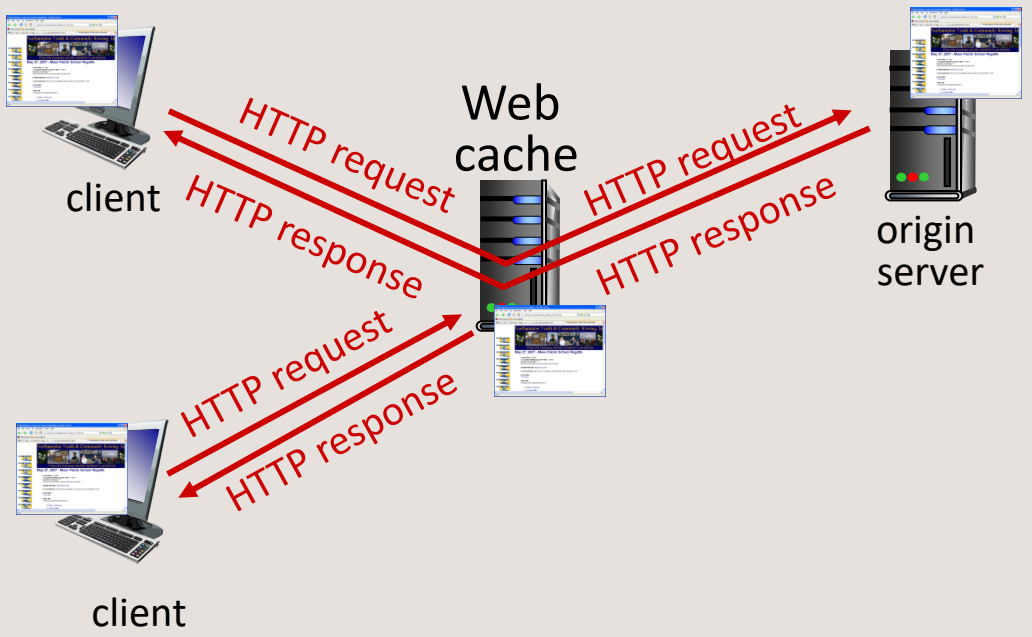
third party tracking via cookies:

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

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



Web caches (aka proxy servers)

- Web cache acts as both client and server
  - server for original requesting client
  - client to origin server
- server tells cache about object's allowable caching in response header:

```
Cache-Control: max-age=<seconds>
```

```
Cache-Control: no-cache
```

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

Browser caching: Conditional GET

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

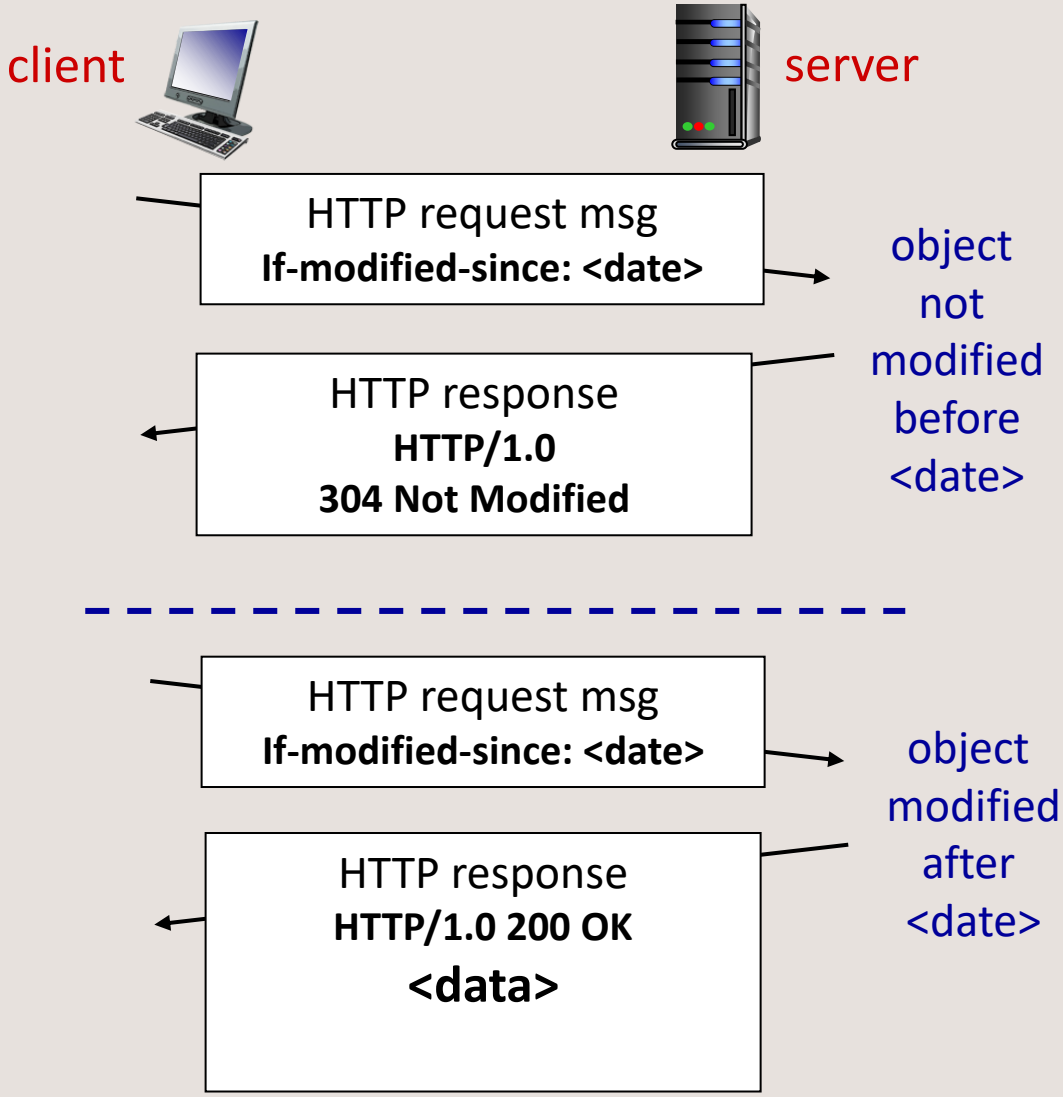
- no object transmission delay (or use of network resources)

- *client:* specify date of browser-cached copy in HTTP request

  If-modified-since: <date>

- *server:* response contains no object if browser-cached copy is up-to-date:

  HTTP/1.0 304 Not Modified



Application layer: roadmap

- Principles of network applications
- Web and HTTP
- The Domain Name System DNS
- Video streaming and CDNs

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)
  - *note:* core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

DNS: services, structure

DNS services:

- hostname-to-IP-address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

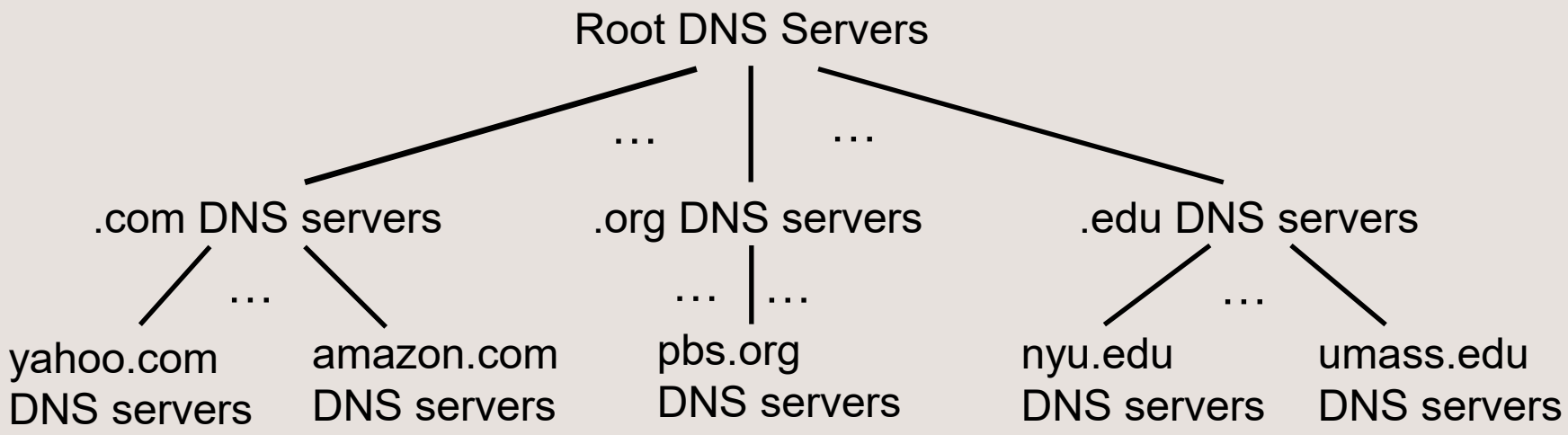
Q: Why not centralize DNS?

- single point of failure
- 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

DNS: a distributed, hierarchical database



Root

Top Level Domain

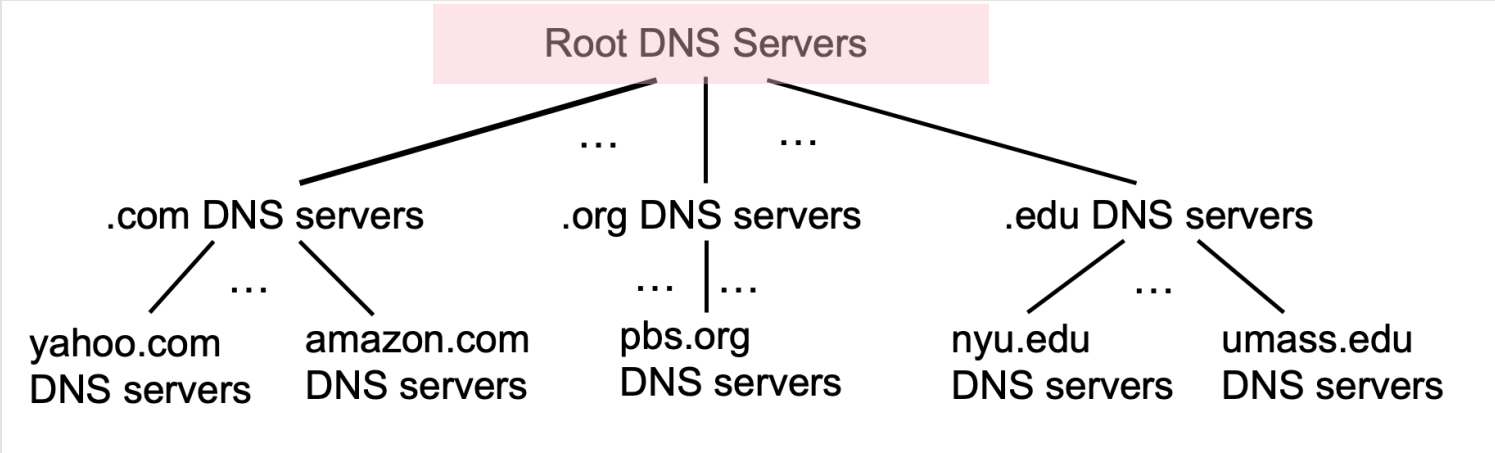
Authoritative

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

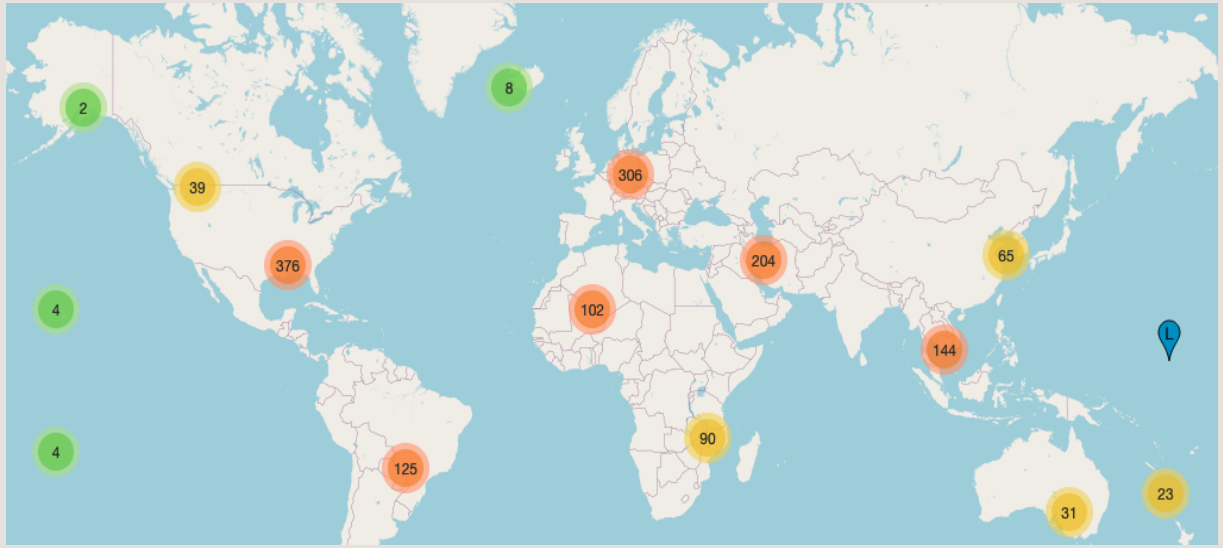
DNS: root name servers

- 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)



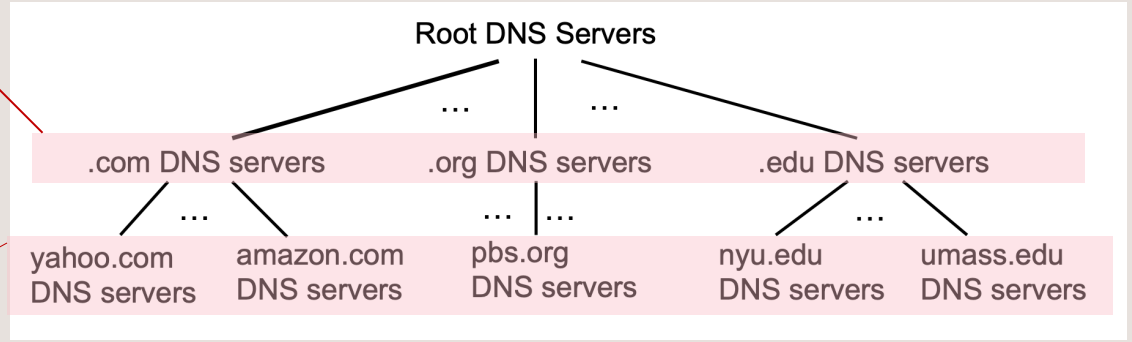
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain

13 logical root name "servers" worldwide; each "server," replicated many times

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

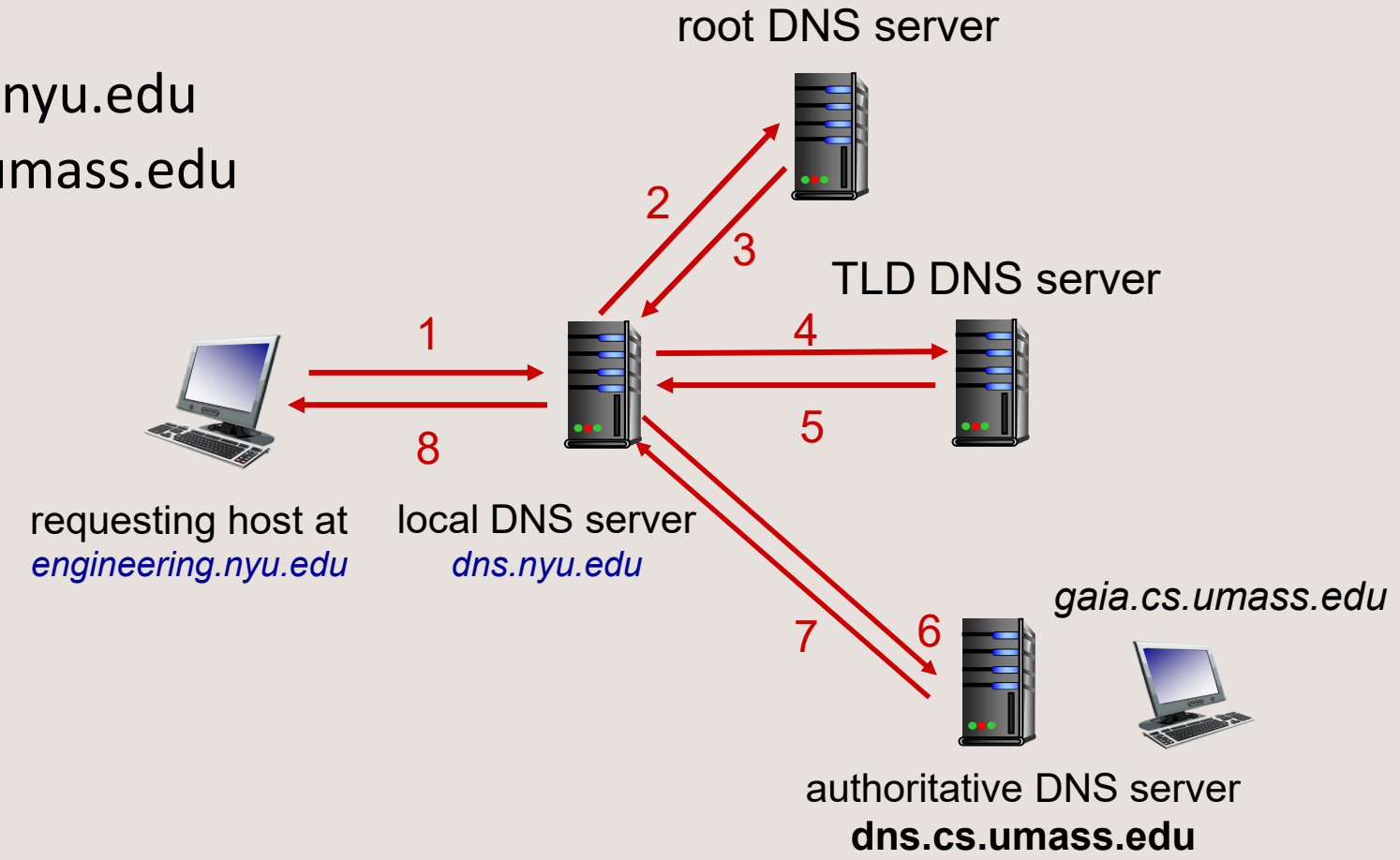
- when host makes DNS query, it is sent to its *local* DNS server
  - Local DNS server returns reply, answering:
    - from its local cache of recent name-to-address translation pairs (possibly out of date!)
    - forwarding request into DNS hierarchy for resolution
  - each ISP has local DNS name server;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

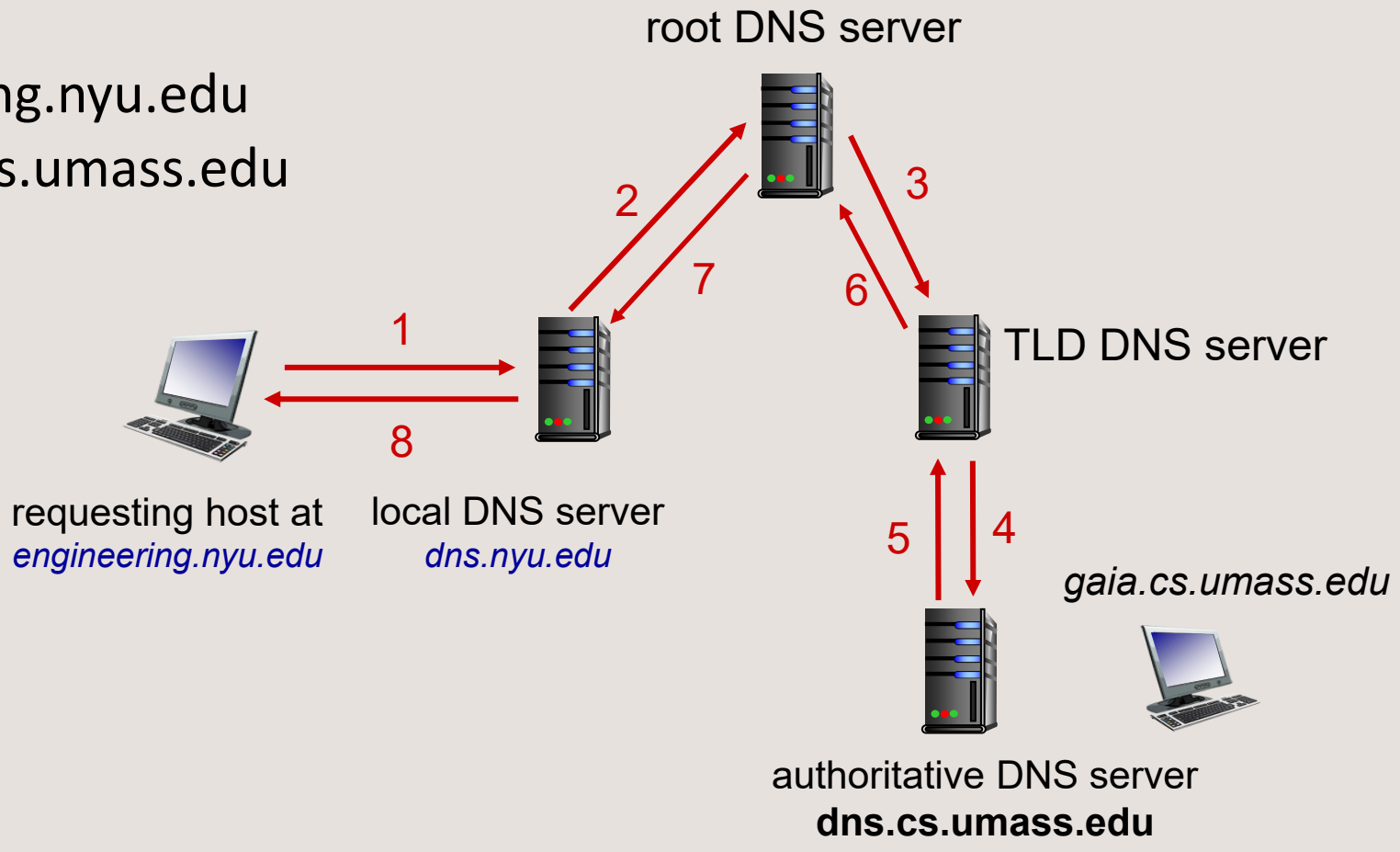


DNS name resolution: recursive query

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
- heavy load at upper levels of hierarchy?



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
- `value` is IP address

type=NS

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

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`

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

Application layer: roadmap

- Principles of network applications
- Web and HTTP
- The Domain Name System DNS
- Video streaming and CDNs

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



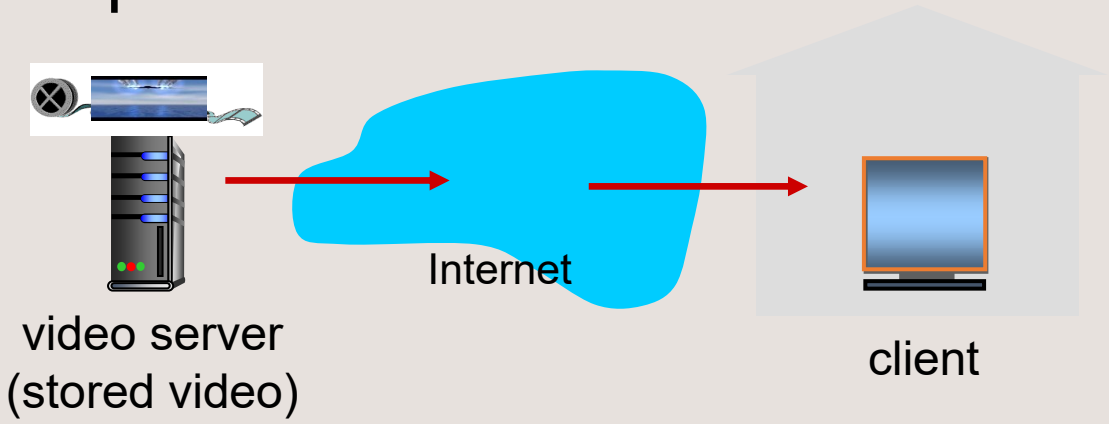






Streaming stored video

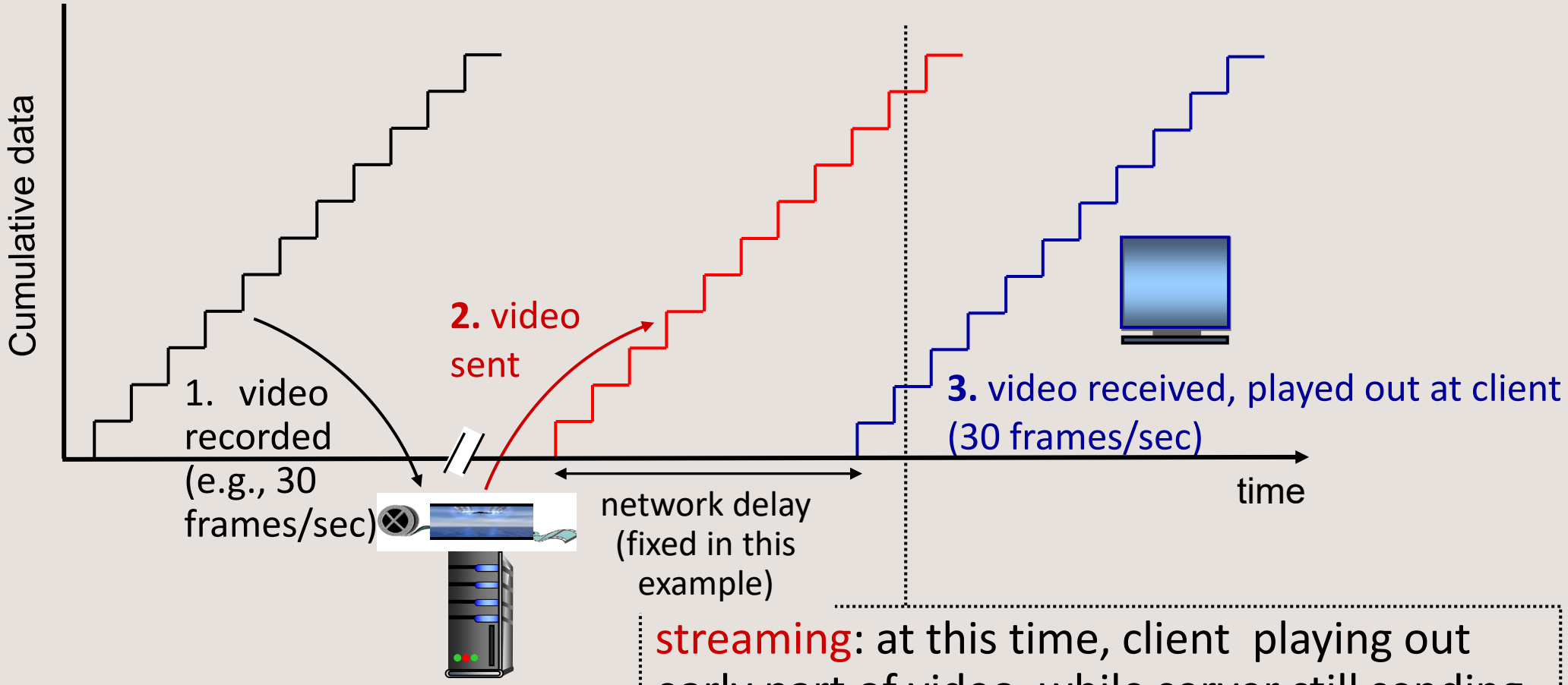
simple scenario:



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

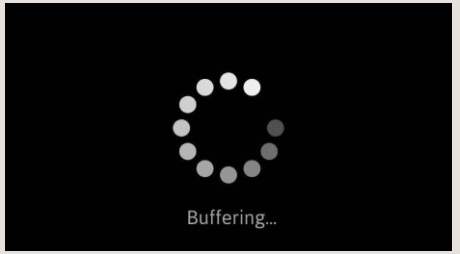
Streaming stored video



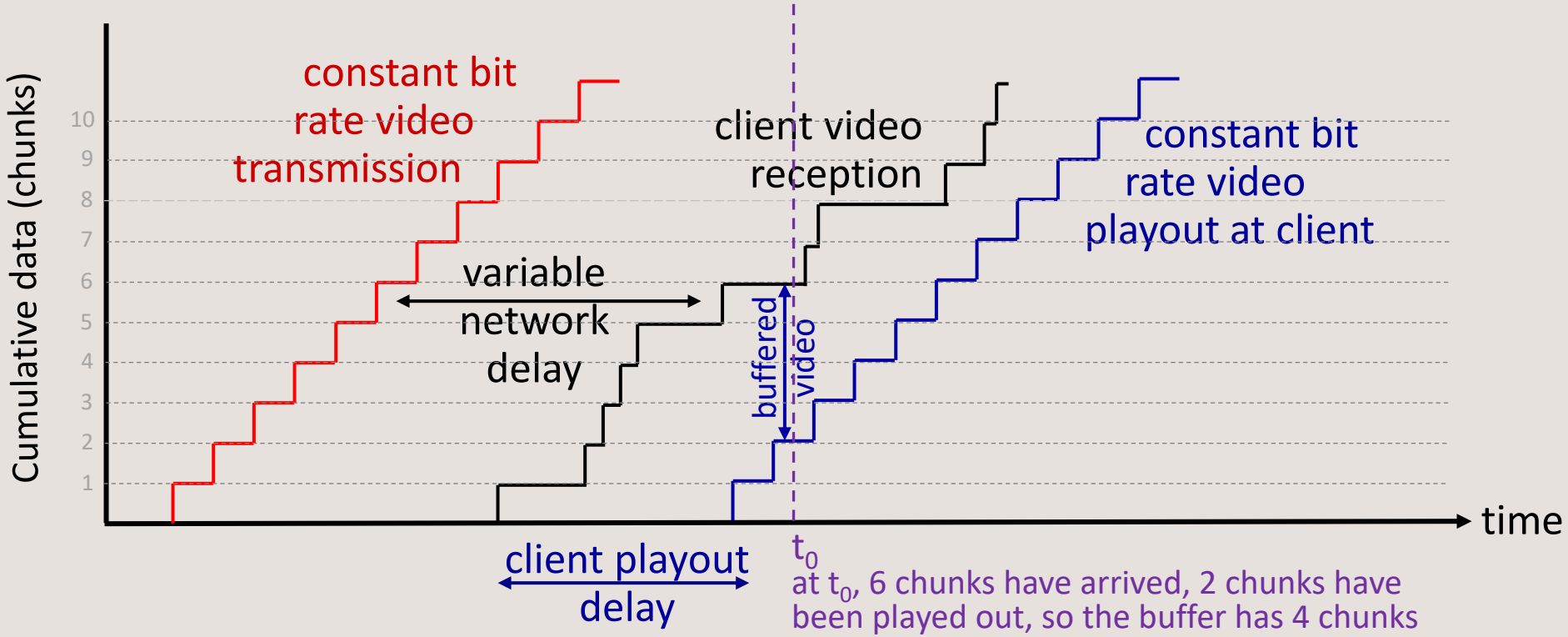
streaming: at this time, client playing out early part of video, while server still sending later part of video

Streaming stored video: challenges

- continuous playout constraint: during client video playout, playout timing must match original timing
  - … but network delays are variable (jitter), so will need client-side buffer to match continuous playout constraint
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through video
  - video packets may be lost, retransmitted



Streaming stored video: playout buffering



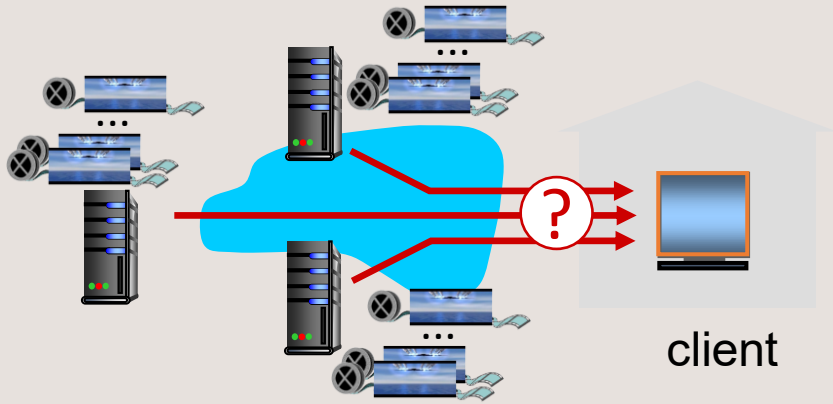
- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

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

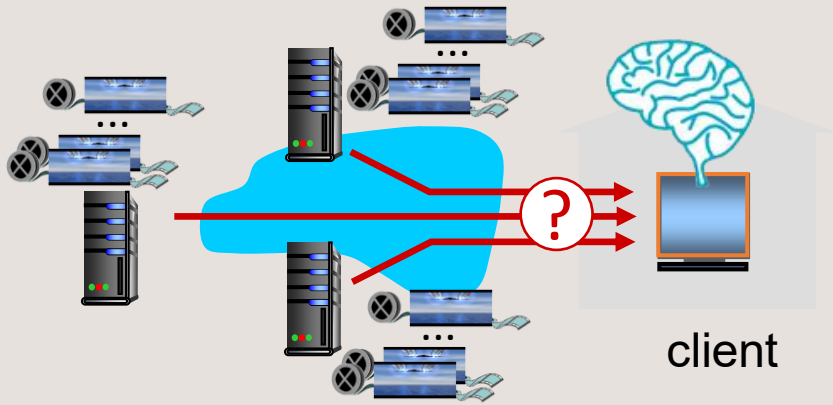


client:

- 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

Streaming multimedia: DASH

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



Streaming video = encoding + DASH + playout buffering

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





How does Netflix work?

- Netflix: stores copies of content (e.g., MADMEN) at its (worldwide) OpenConnect 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

