

Computer Networks and Applications

COMP 3331/COMP 9331

Week 2

Application Layer (Principles)

Chapter 2, Section 2.1

2. Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

2. Application layer

our goals:

- ❖ conceptual, implementation aspects of network application protocols
 - transport-layer service models
 - client-server paradigm
 - peer-to-peer paradigm
- ❖ learn about protocols by examining popular application-level protocols
 - HTTP
 - SMTP / POP3 / IMAP
 - DNS
- ❖ creating network applications
 - socket API

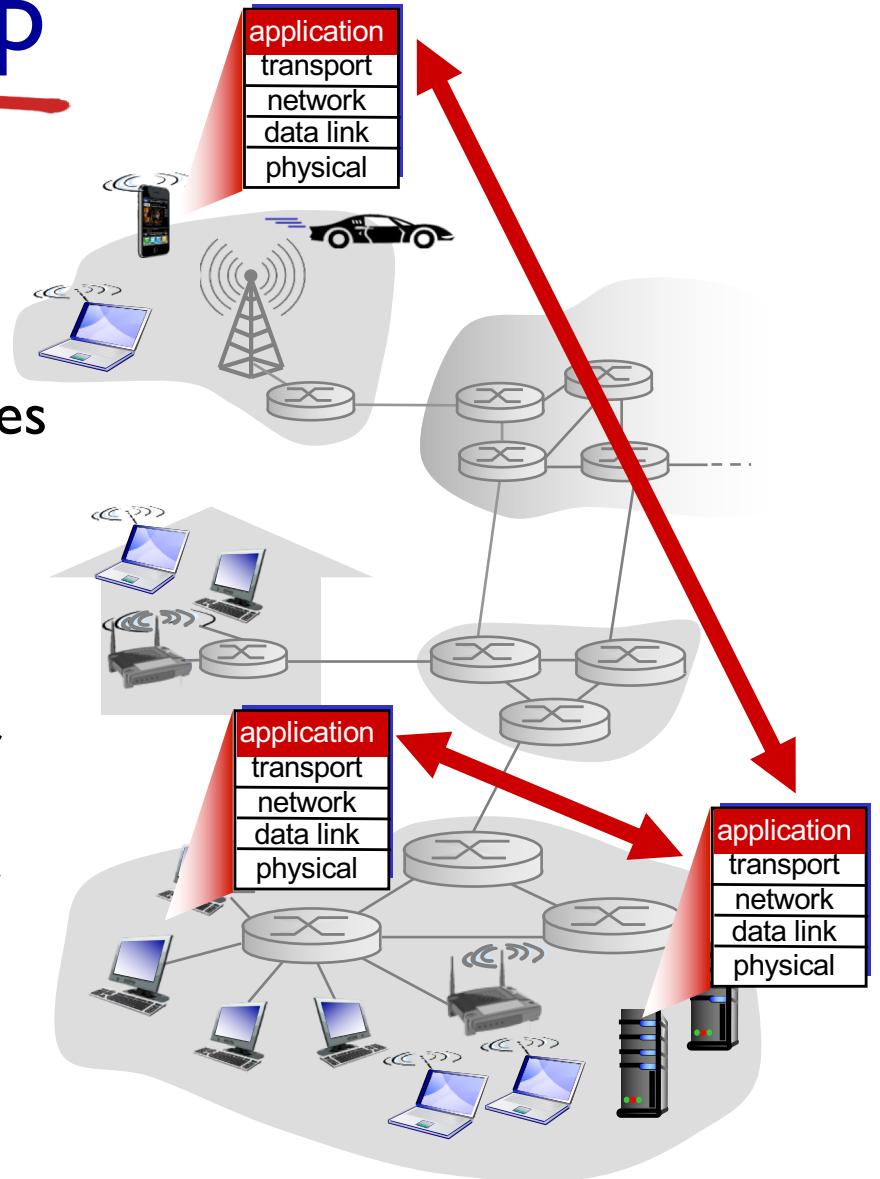
Creating a network app

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

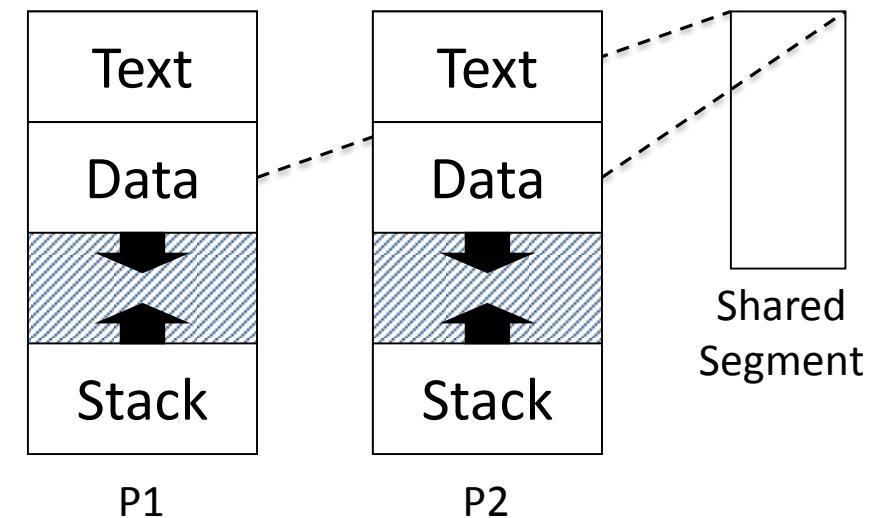
- ❖ network-core devices do not run user applications
- ❖ applications on end systems allows for rapid app development



Interprocess Communication (IPC)

- ❖ Processes talk to each other through Inter-process communication (IPC)

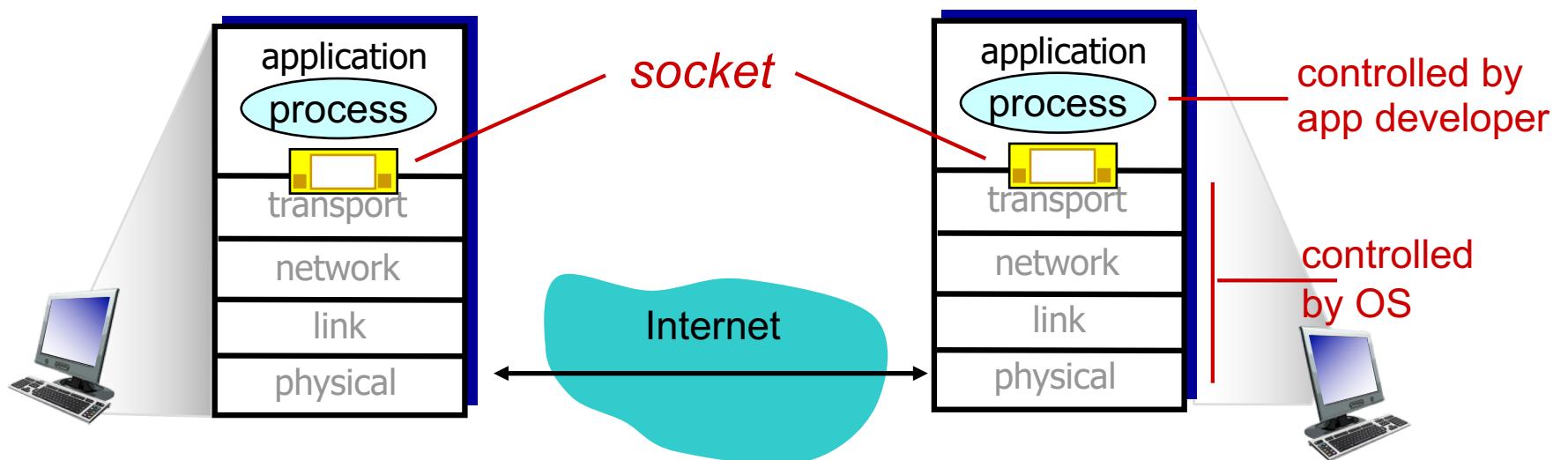
- ❖ On a single machine:
 - Shared memory



- ❖ Across machines:
 - We need other abstractions (message passing)

Sockets

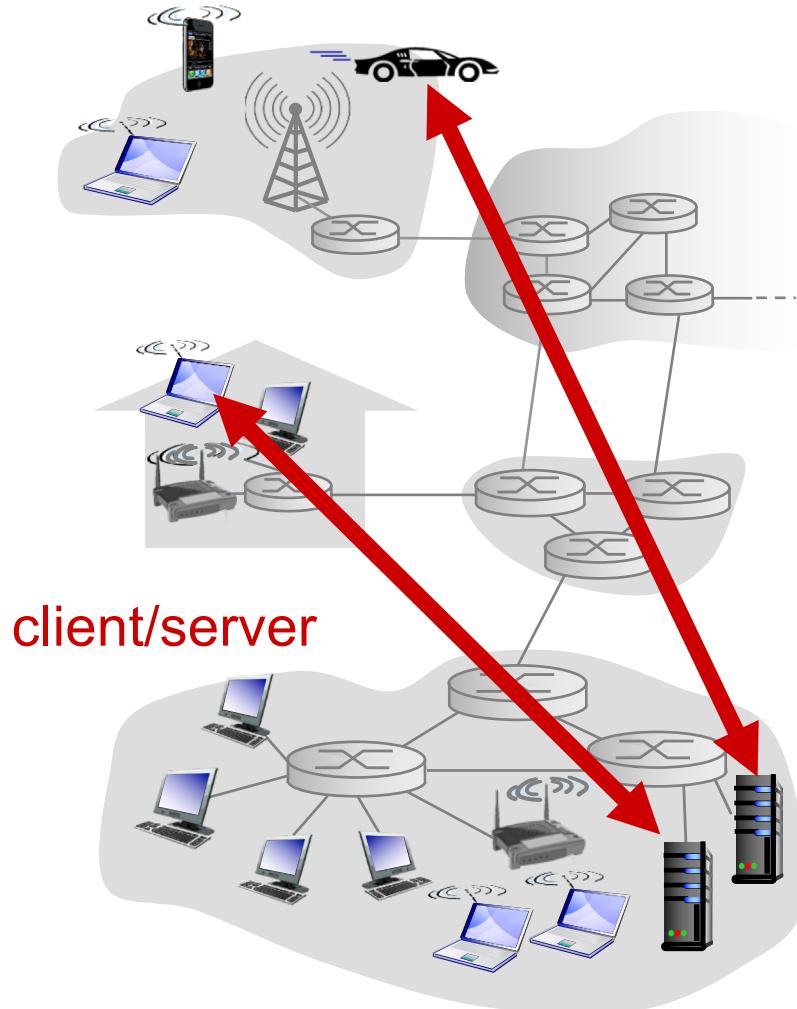
- ❖ process sends/receives messages to/from its **socket**
- ❖ socket analogous to door
 - sending process shoves message out door
 - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
- ❖ Application has a few options, OS handles the details



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 cse.unsw.edu.au web server:
 - IP address: 129.94.242.51
 - port number: 80

Client-server architecture



server:

- ❖ Exports well-defined request/response interface
- ❖ long-lived process that waits for requests
- ❖ Upon receiving request, carries it out

clients:

- ❖ Short-lived process that makes requests
- ❖ “User-side” of application
- ❖ Initiates the communication

Client versus Server

❖ Server

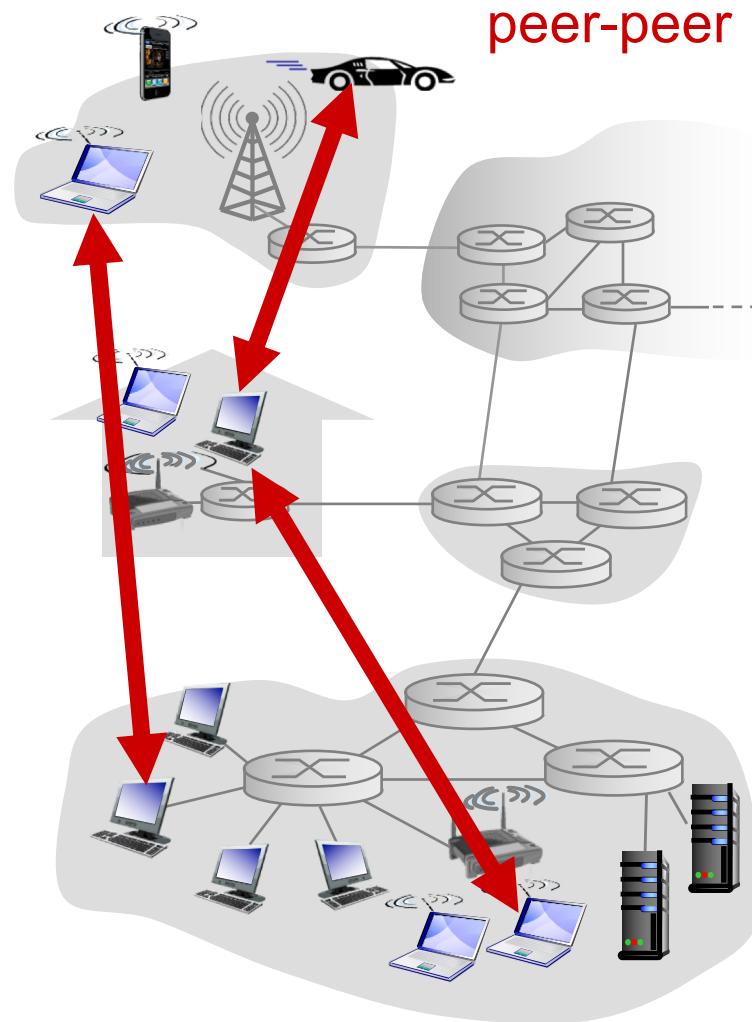
- Always-on host
- Permanent IP address (rendezvous location)
- Static port conventions (http: 80, email: 25, ssh:22)
- Data centres for scaling
- May communicate with other servers to respond

❖ Client

- May be intermittently connected
- May have dynamic IP addresses
- Do not communicate directly with each other

P2P architecture

- ❖ no always-on server
 - No permanent rendezvous involved
- ❖ arbitrary end systems (peers) directly communicate
- ❖ Symmetric responsibility (unlike client/server)
- ❖ Often used for:
 - File sharing (BitTorrent)
 - Games
 - Video distribution, video chat
 - In general: “distributed systems”



P2P architecture: Pros and Cons

+ 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

+ Speed: parallelism, less contention

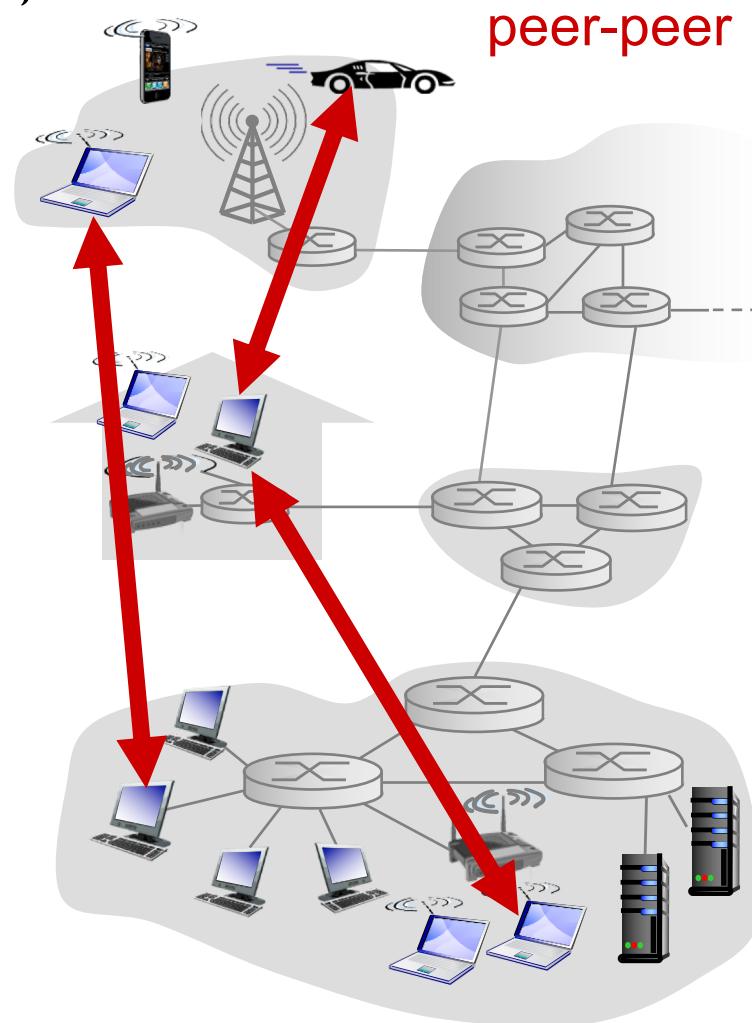
+ Reliability: redundancy, fault tolerance

+ Geographic distribution

-Fundamental problems of decentralized control

- State uncertainty: no shared memory or clock
- Action uncertainty: mutually conflicting decisions

-Distributed algorithms are complex



App-layer protocol defines

- ❖ types of messages exchanged,
 - e.g., request, response
- ❖ message syntax:
 - what fields in messages & how fields are delineated
- ❖ message semantics
 - meaning of information in fields
- ❖ rules for when and how processes send & respond to messages

open protocols:

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

proprietary protocols:

- ❖ e.g., Skype

What transport service does an app need?

data integrity

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

timing

- ❖ some apps (e.g., 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	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 50kbps-1Mbps video:100kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
Chat/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
- ❖ *does not provide*: timing, minimum throughput guarantee, security
- ❖ *connection-oriented*: setup required between client and server processes

UDP service:

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

Q: why bother? Why is there a UDP?

NOTE: More on transport later on

Internet apps: application, transport protocols

	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Computer Networks and Applications

COMP 3331/COMP 9331

Week 3

Application Layer (Web & HTTP)

Chapter 2, Section 2.2

2. Application Layer: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP



Self study

Self study

The Web – Precursor



Ted Nelson

- ❖ **1967, Ted Nelson, Xanadu:**
 - A world-wide publishing network that would allow information to be stored not as separate files but as connected literature
 - Owners of documents would be automatically paid via electronic means for the virtual copying of their documents
- ❖ **Coined the term “Hypertext”**

Self study

The Web – History



Tim Berners-Lee

- ❖ World Wide Web (WWW): a distributed database of “pages” linked through **Hypertext Transport Protocol (HTTP)**
 - First HTTP implementation - 1990
 - Tim Berners-Lee at CERN
 - HTTP/0.9 – 1991
 - Simple GET command for the Web
 - HTTP/1.0 – 1992
 - Client/Server information, simple caching
 - HTTP/1.1 – 1996
 - HTTP2.0 - 2015

<http://info.cern.ch/hypertext/WWW/TheProject.html>

Web and HTTP

First, a review...

- ❖ *web page* consists of *objects*
- ❖ object can be HTML file, JPEG image, Java applet, audio file,...
- ❖ web page consists of *base HTML-file* which includes *several referenced objects*
- ❖ each object is addressable by a *URL*, e.g.,

www.someschool.edu/someDept/pic.gif

host name

path name

Web and HTTP

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hyperlink Example</title>
    </head>
    <body>
        <p>Click the following link</p>
        <a href = "http://www.cnn.com" target ="_self">CNN</a>
    </body>
</html>
```

Uniform Resource Locator (URL)

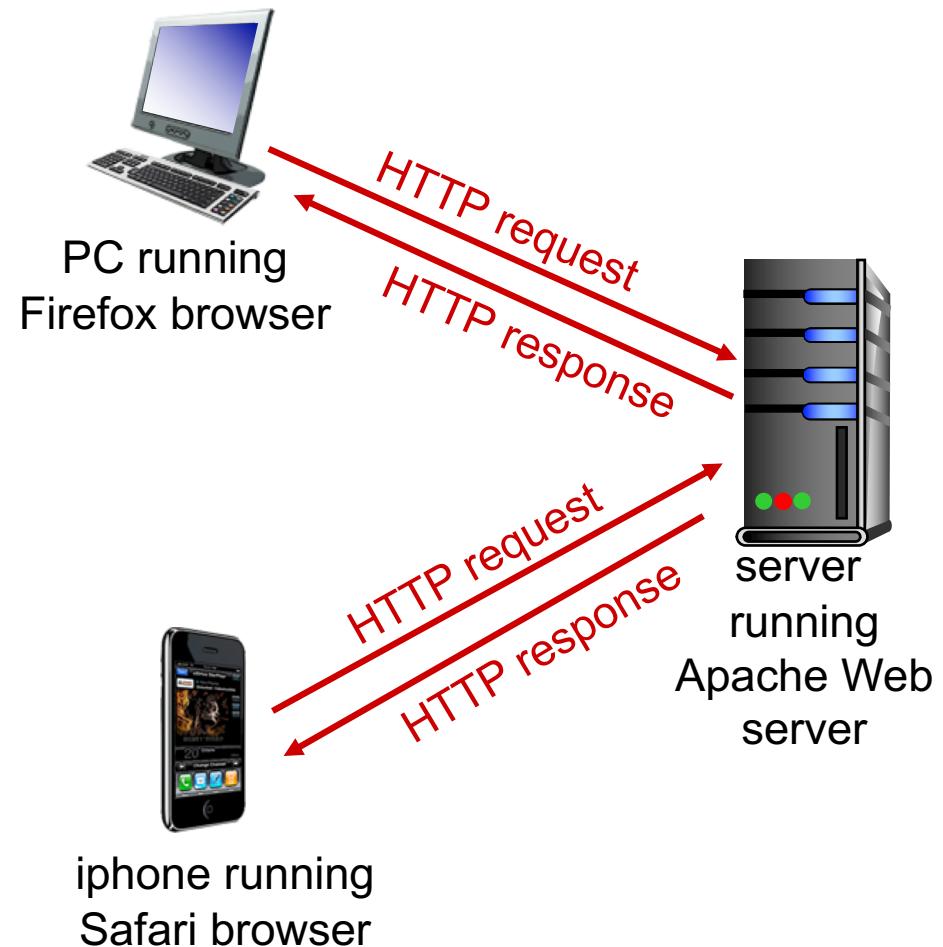
protocol://host-name[:port]/directory-path/resource

- ❖ *protocol*: http, ftp, https, smtp etc.
- ❖ *hostname*: DNS name, IP address
- ❖ *port*: defaults to protocol's standard port; e.g. http: 80 https: 443
- ❖ *directory path*: hierarchical, reflecting file system
- ❖ *resource*: Identifies the desired resource

HTTP overview

HTTP: hypertext transfer protocol

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



HTTP overview (continued)

uses TCP:

- ❖ client initiates TCP connection (creates socket) to server, 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 request message

- ❖ two types of HTTP messages: *request, response*
- ❖ **HTTP request message:**
 - ASCII (human-readable format)

request line
(GET, POST,
HEAD commands)

header
lines

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

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

carriage return character

line-feed character

HTTP response message

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\nDate: Sun, 26 Sep 2010 20:09:20 GMT\r\nServer: Apache/2.0.52 (CentOS) \r\nLast-Modified: Tue, 30 Oct 2007 17:00:02  
GMT\r\nETag: "17dc6-a5c-bf716880"\r\nAccept-Ranges: bytes\r\nContent-Length: 2652\r\nKeep-Alive: timeout=10, max=100\r\nConnection: Keep-Alive\r\nContent-Type: text/html; charset=ISO-8859-  
1\r\n\r\ndata data data data 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 msg

301 Moved Permanently

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

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

451 Unavailable for Legal Reasons

429 Too Many Requests

418 I'm a Teapot

HTTP is all text

- ❖ Makes the protocol simple
 - Easy to delineate messages (\r\n)
 - (relatively) human-readable
 - No issues about encoding or formatting data
 - Variable length data
- ❖ Not the most efficient
 - Many protocols use binary fields
 - Sending "12345678" as a string is 8 bytes
 - As an integer, 12345678 needs only 4 bytes
 - Headers may come in any order
 - Requires string parsing/processing

Request Method types (“verbs”)

HTTP/1.0:

- ❖ GET
 - Request page
- ❖ POST
 - Uploads user response to a form
- ❖ HEAD
 - asks server to leave requested object out of response

HTTP/1.1:

- ❖ GET, POST, HEAD
- ❖ PUT
 - uploads file in entity body to path specified in URL field
- ❖ DELETE
 - deletes file specified in the URL field
- ❖ TRACE, OPTIONS, CONNECT, PATCH
 - For persistent connections

Uploading form input

POST method:

- ❖ web page often includes form input
- ❖ input is uploaded to server in entity body

Get (in-URL) method:

- ❖ uses GET method
- ❖ input is uploaded in URL field of request line:

`www.somesite.com/animalsearch?monkeys&banana`

User-server state: cookies

many Web sites use cookies

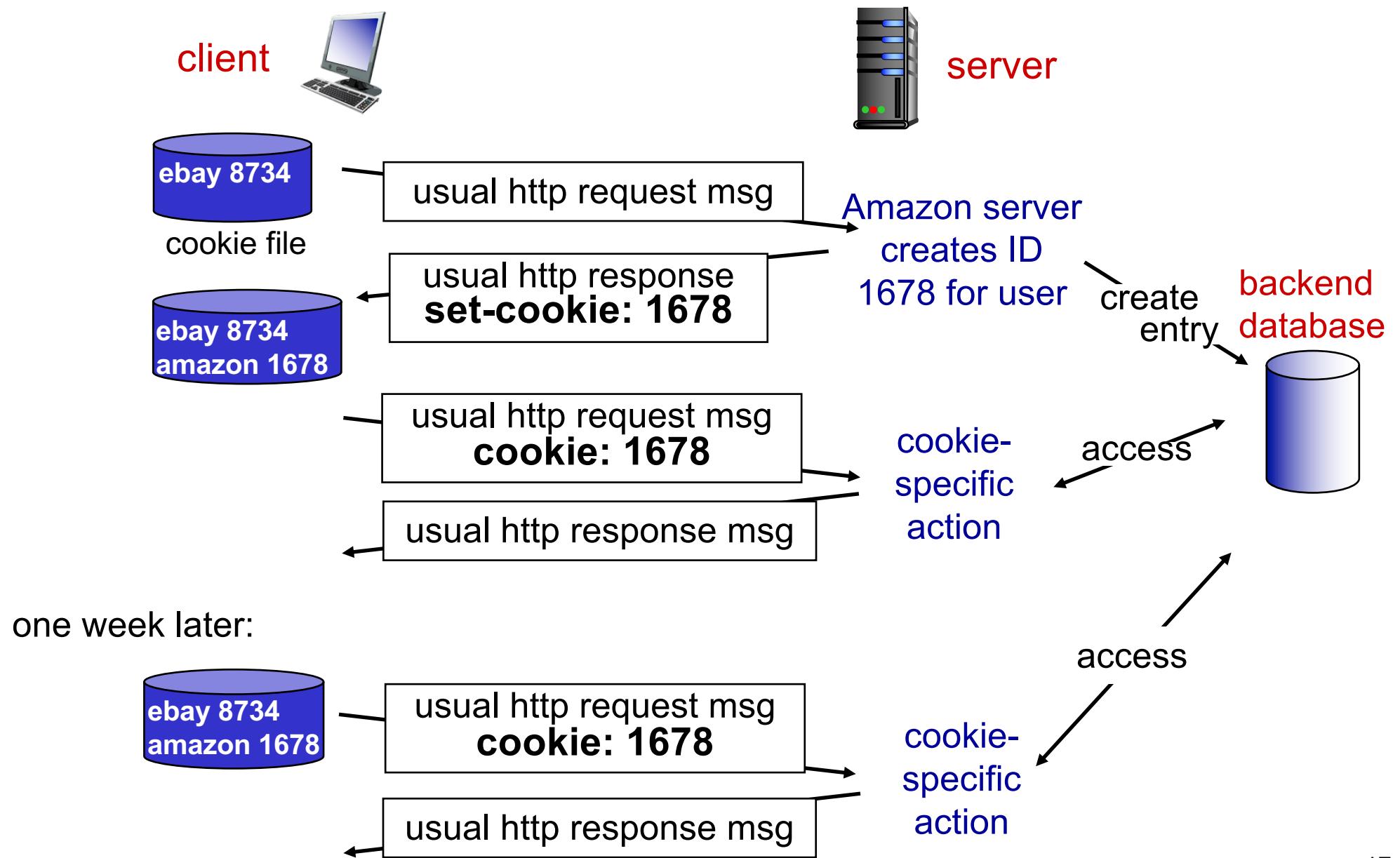
four components:

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

example:

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

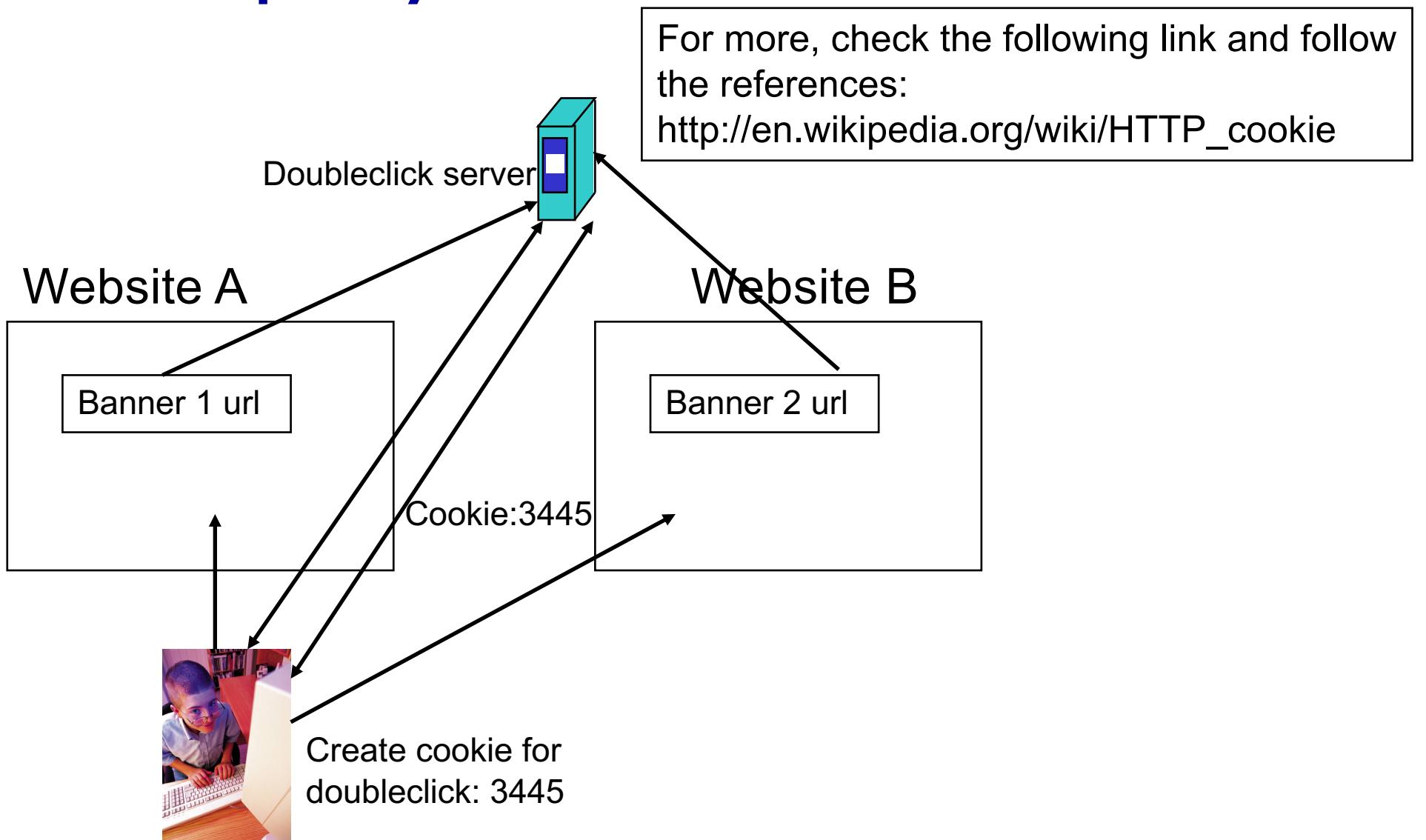
Cookies: keeping “state” (cont.)



The Dark Side of Cookies

- ❖ Cookies permit sites to learn a lot about you
- ❖ You may supply name and e-mail to sites (and more)
- ❖ 3rd party cookies (from ad networks, etc.) can follow you across multiple sites
 - Ever visit a website, and the next day ALL your ads are from them ?
 - Check your browser's cookie file (cookies.txt, cookies.plist)
 - Do you see a website that you have never visited
- ❖ You COULD turn them off
 - But good luck doing anything on the Internet !!

Third party cookies



Performance of HTTP

- Page Load Time (PLT) as the metric
 - From click until user sees page
 - Key measure of web performance
- Depends on many factors such as
 - page content/structure,
 - protocols involved and
 - Network bandwidth and RTT

Performance Goals

- ❖ User
 - fast downloads
 - high availability
- ❖ Content provider
 - happy users (hence, above)
 - cost-effective infrastructure
- ❖ Network (secondary)
 - avoid overload

Solutions?

- ❖ User
 - fast downloads
 - high availability
- ❖ Content provider
 - happy users (hence, above)
 - cost-effective infrastructure
- ❖ Network (secondary)
 - avoid overload

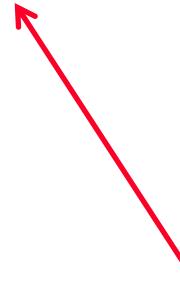
Improve HTTP to
achieve faster
downloads



Solutions?

- ❖ User
 - fast downloads
 - high availability
 - ❖ Content provider
 - happy users (hence, above)
 - cost-effective delivery infrastructure
 - ❖ Network (secondary)
 - avoid overload
-
- Improve HTTP to achieve faster downloads
- Caching and Replication
- Caching and Replication

Solutions?

- ❖ User
 - fast downloads
 - high availability
 - ❖ Content provider
 - happy users (hence, above)
 - cost-effective delivery infrastructure
 - ❖ Network (secondary)
 - avoid overload
- Improve HTTP to
achieve faster
downloads
- Caching and Replication
- Exploit economies of scale
(Webhosting, CDNs, datacenters)
- 

How to improve PLT

- Reduce content size for transfer
 - Smaller images, compression
- Change HTTP to make better use of available bandwidth
 - Persistent connections and pipelining
- Change HTTP to avoid repeated transfers of the same content
 - Caching and web-proxies
- Move content closer to the client
 - CDNs

HTTP Performance

- ❖ Most Web pages have multiple objects
 - e.g., HTML file and a bunch of embedded images
- ❖ How do you retrieve those objects (naively)?
 - *One item at a time*
- ❖ New TCP connection per (small) object!

non-persistent HTTP

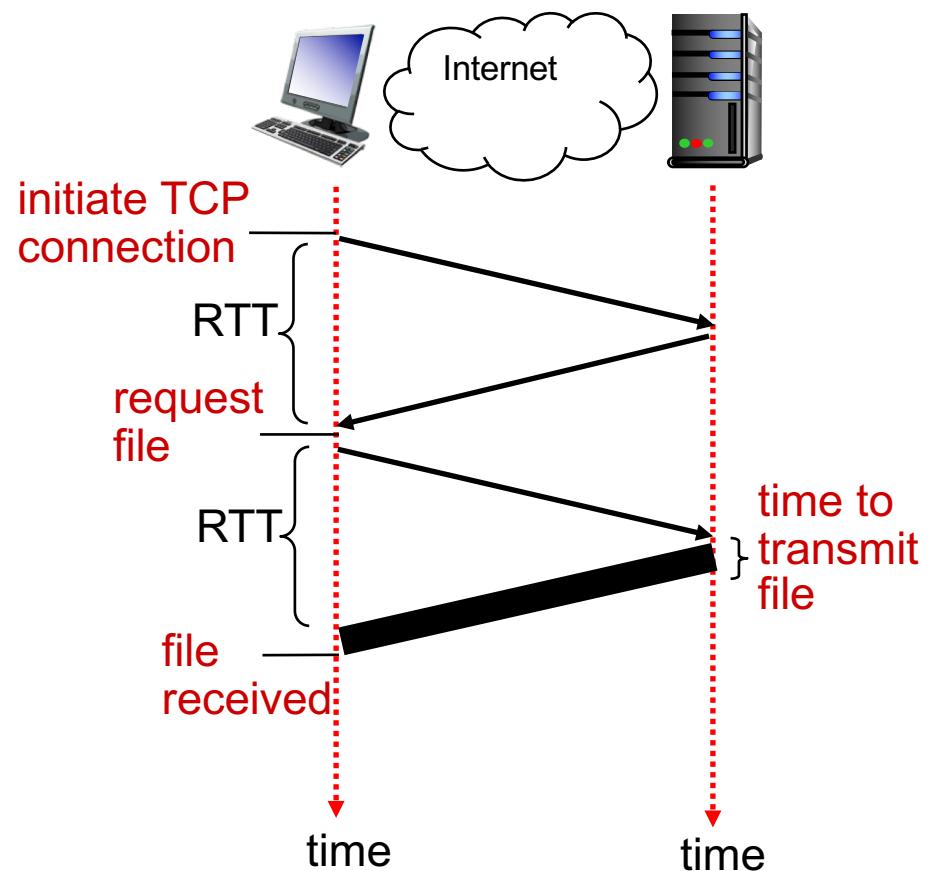
- ❖ at most one object sent over TCP connection
 - connection then closed
- ❖ downloading multiple objects required multiple connections

Non-persistent HTTP: response time

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

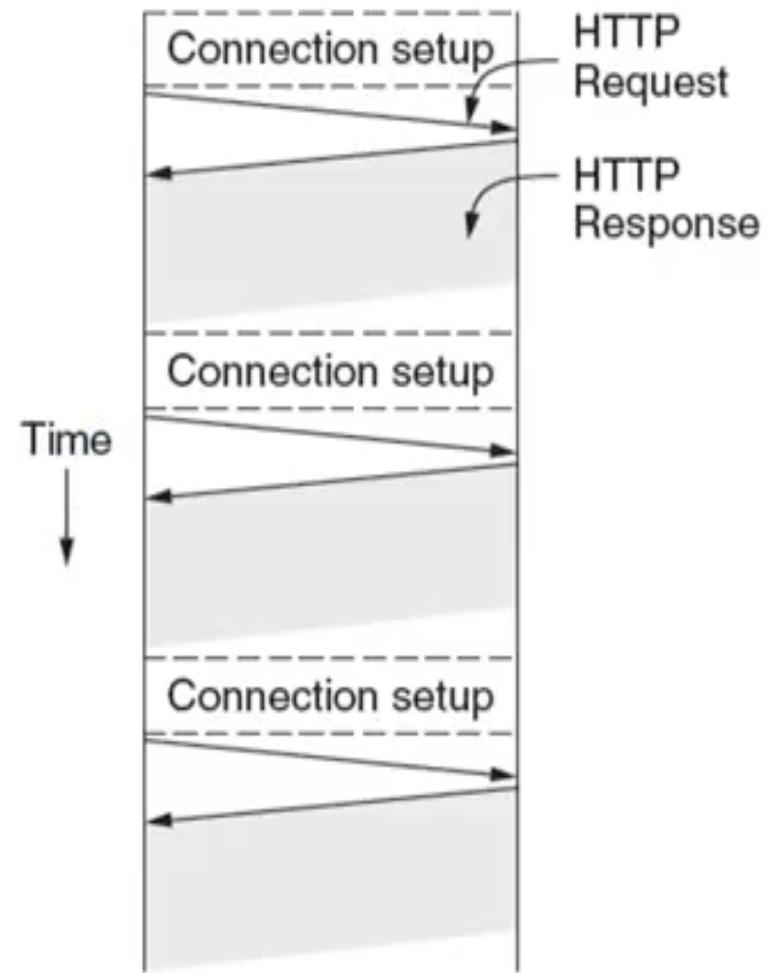
HTTP response time:

- ❖ one RTT to initiate TCP connection
- ❖ one RTT for HTTP request and first few bytes of HTTP response to return
- ❖ file transmission time
- ❖ non-persistent HTTP response time = $2\text{RTT} + \text{file transmission time}$



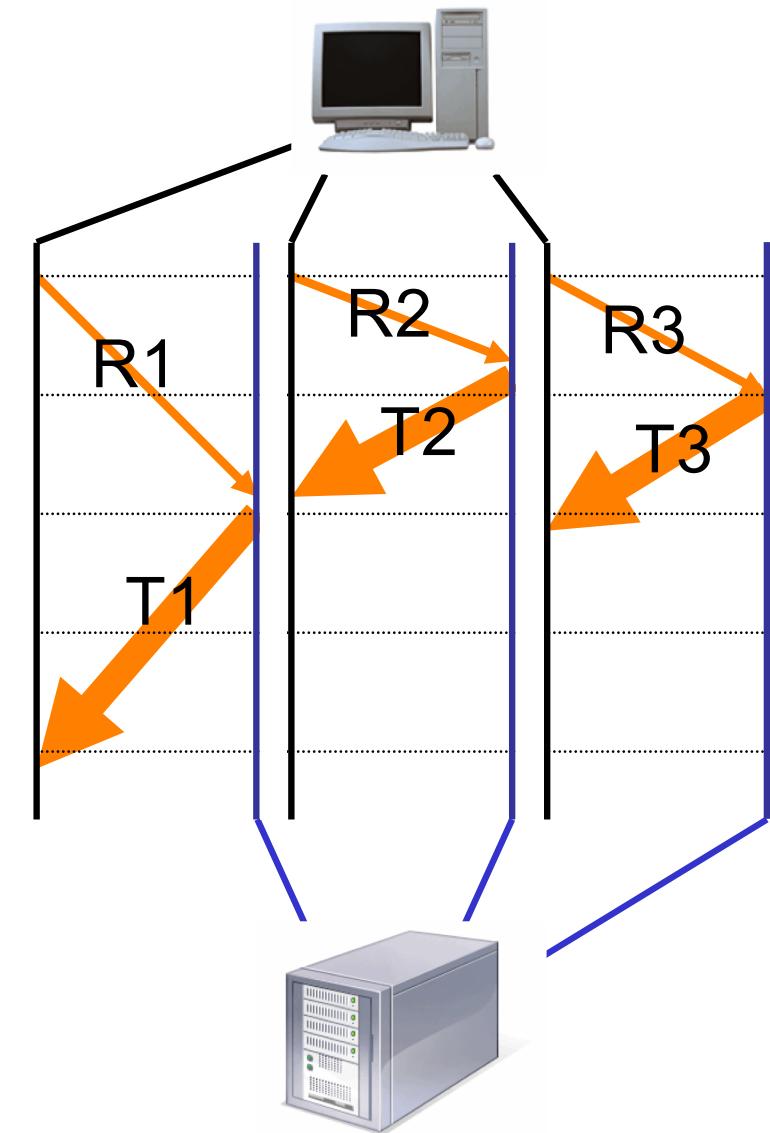
HTTP/1.0

- Non-Persistent: One TCP connection to fetch one web resource
- Fairly poor PLT
- 2 Scenarios
 - Multiple TCP connections setups to the **same server**
 - Sequential request/responses even when resources are located on **different servers**
- Multiple TCP slow-start phases (more in lecture on TCP)



Improving HTTP Performance: Concurrent Requests & Responses

- ❖ Use multiple connections *in parallel*
- ❖ Does not necessarily maintain order of responses



Quiz: Parallel HTTP Connections



- ❖ What are potential downsides of parallel HTTP connections, i.e. can opening too many parallel connections be harmful and if so in what way?

Persistent HTTP

Persistent HTTP

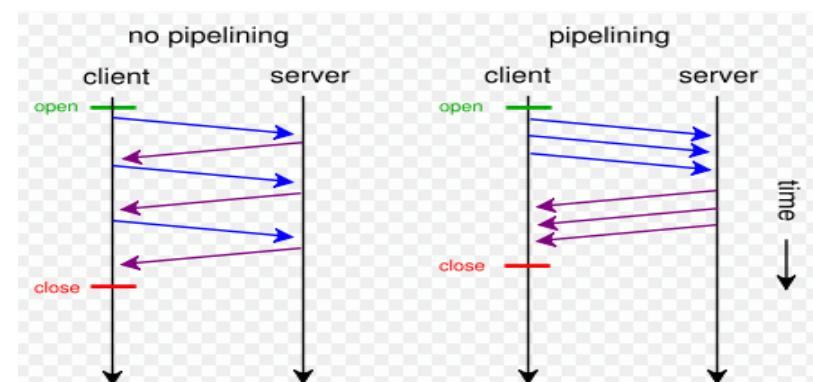
- ❖ server leaves TCP connection open after sending response
- ❖ subsequent HTTP messages between same client/server are sent over the same TCP connection
- ❖ Allow TCP to learn more accurate RTT estimate (APPARENT LATER IN THE COURSE)
- ❖ Allow TCP congestion window to increase (APPARENT LATER)
- ❖ i.e., leverage previously discovered bandwidth (APPARENT LATER)

Persistent without pipelining:

- ❖ client issues new request only when previous response has been received
- ❖ one RTT for each referenced object

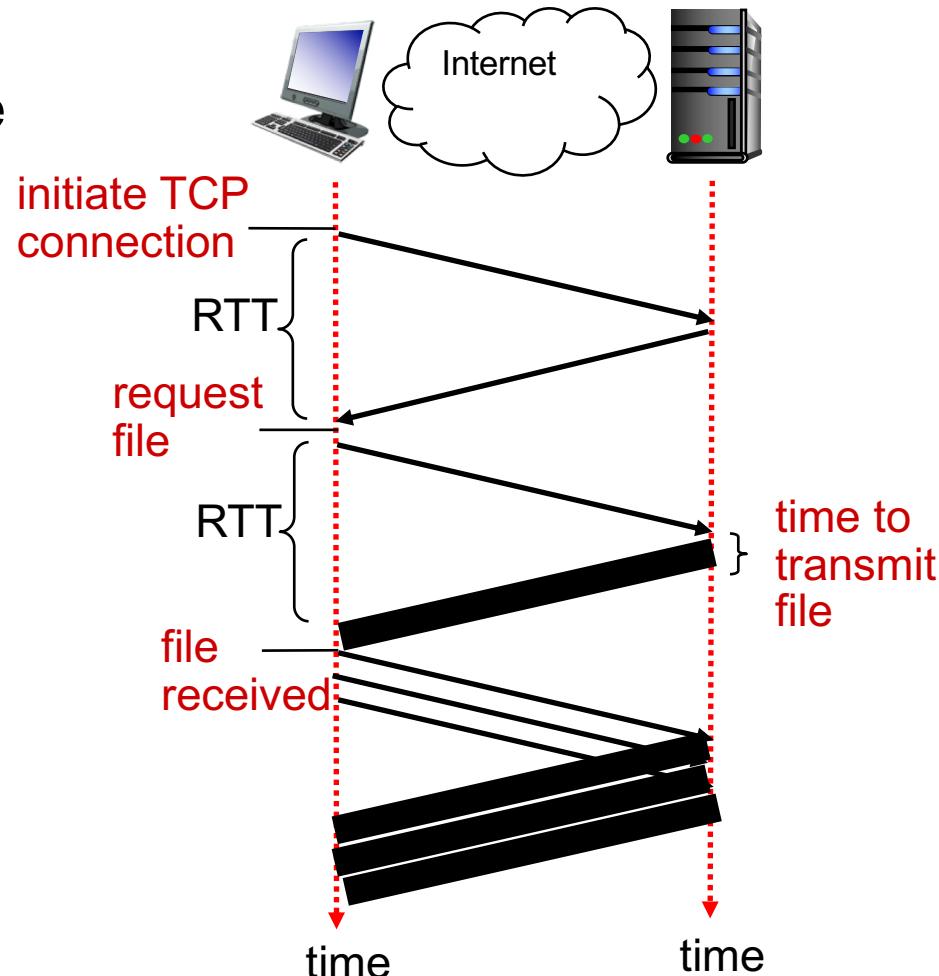
Persistent with pipelining:

- ❖ introduced in HTTP/1.1
- ❖ client sends requests as soon as it encounters a referenced object
- ❖ as little as one RTT for all the referenced objects



HTTP 1.1: response time with pipelining

Website with one index page and three embedded objects



How to improve PLT

- Reduce content size for transfer
 - Smaller images, compression
- Change HTTP to make better use of available bandwidth
 - Persistent connections and pipelining
- Change HTTP to avoid repeated transfers of the same content
 - Caching and web-proxies
- Move content closer to the client
 - CDNs

Improving HTTP Performance: Caching

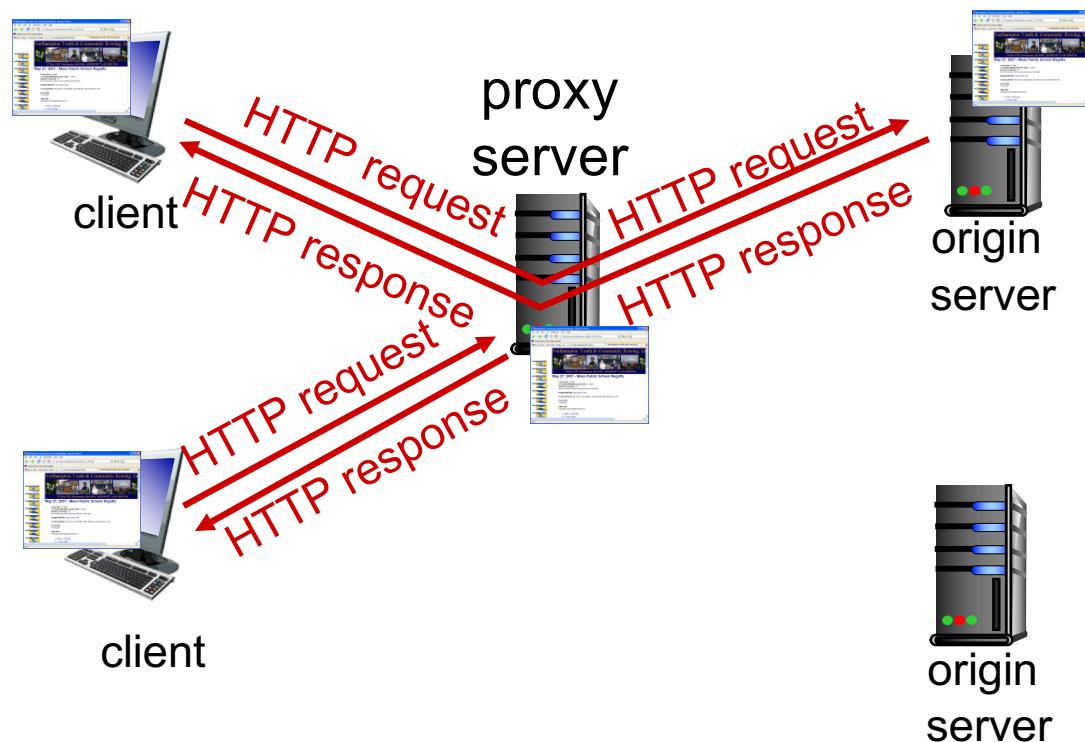
- ❖ Why does caching work?
 - Exploits *locality of reference*

- ❖ How well does caching work?
 - Very well, up to a limit
 - Large overlap in content
 - But many unique requests

Web caches (proxy server)

goal: satisfy client request without involving origin server

- ❖ user sets browser: Web accesses via cache
- ❖ browser sends all HTTP requests to cache
 - object in cache: cache returns object
 - else cache requests object from origin server, then returns object to client



More about Web caching

- ❖ cache acts as both client and server
 - server for original requesting client
 - client to origin server
- ❖ typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

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

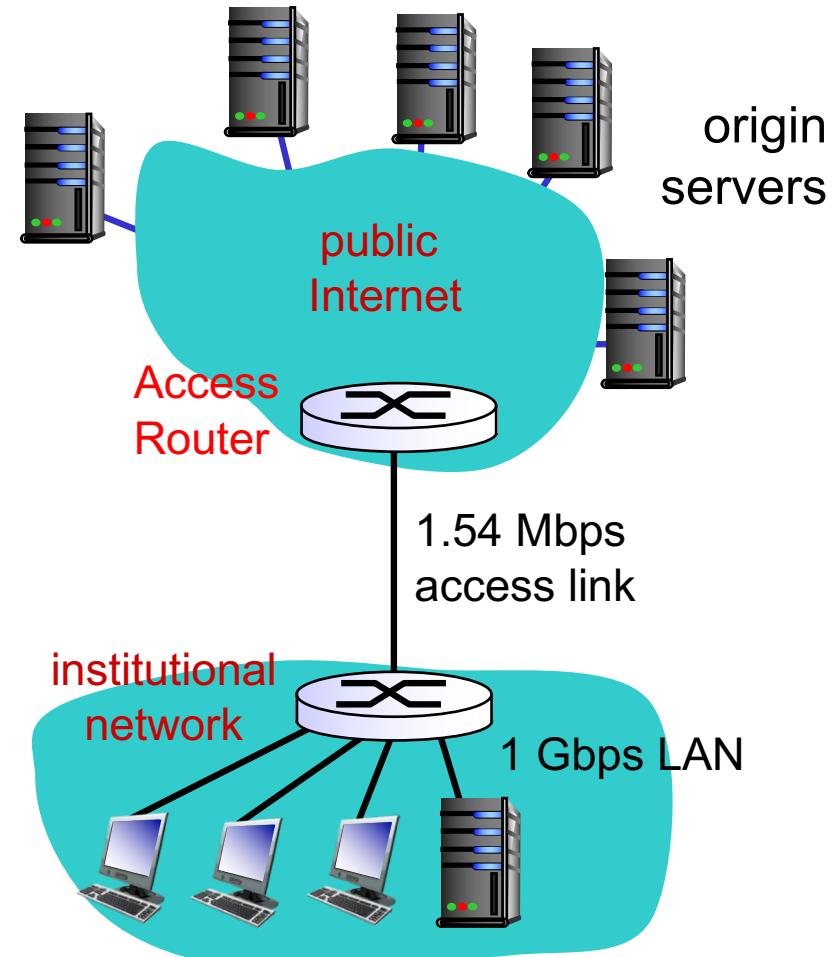
Caching example:

assumptions:

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

consequences:

- ❖ LAN utilization: 0.15%
- ❖ access link utilization = **99%**
- ❖ total delay = Internet delay +
access delay + LAN delay
= 2 sec + minutes + usecs



problem!

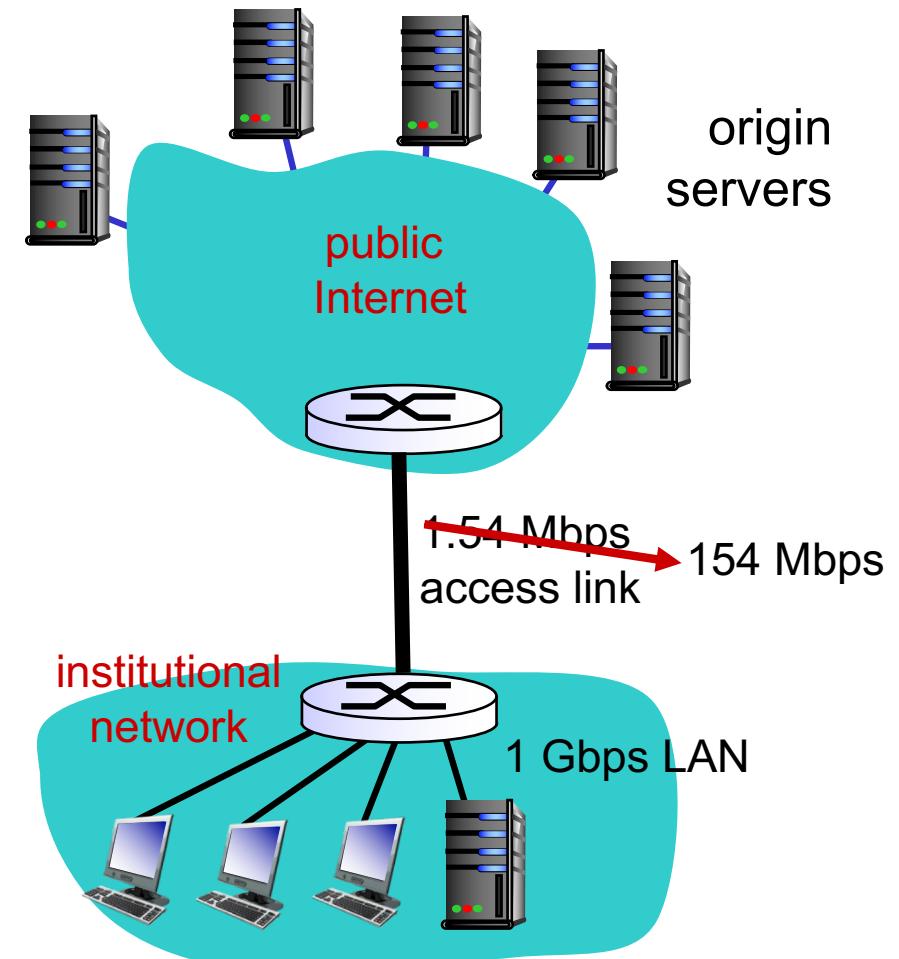
Caching example: fatter access link

assumptions:

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

consequences:

- ❖ LAN utilization: 0.15%
- ❖ access link utilization = ~~99%~~ 0.99%
- ❖ total delay = Internet delay + access delay + LAN delay
= 2 sec + minutes + usecs
msecs



Cost: increased access link speed (not cheap!)

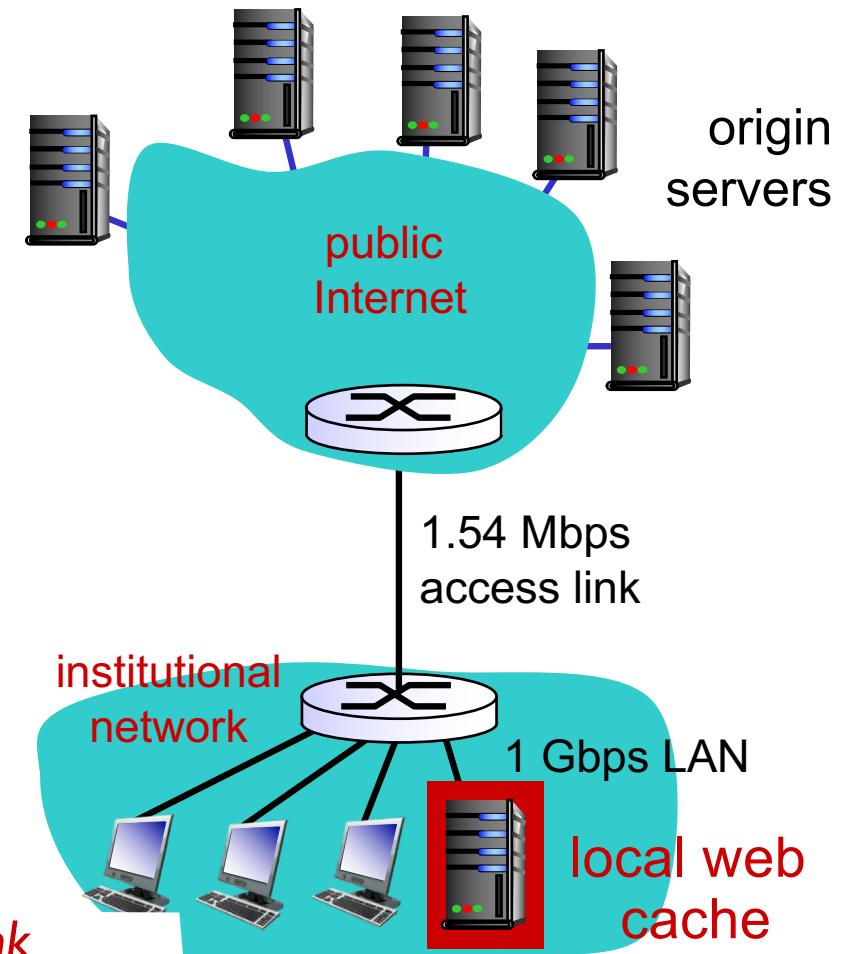
Caching example: install local cache

assumptions:

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

consequences:

- ❖ LAN utilization: ?
- ❖ access link utilization = ?
- ❖ total delay = ? *How to compute link utilization, delay?*

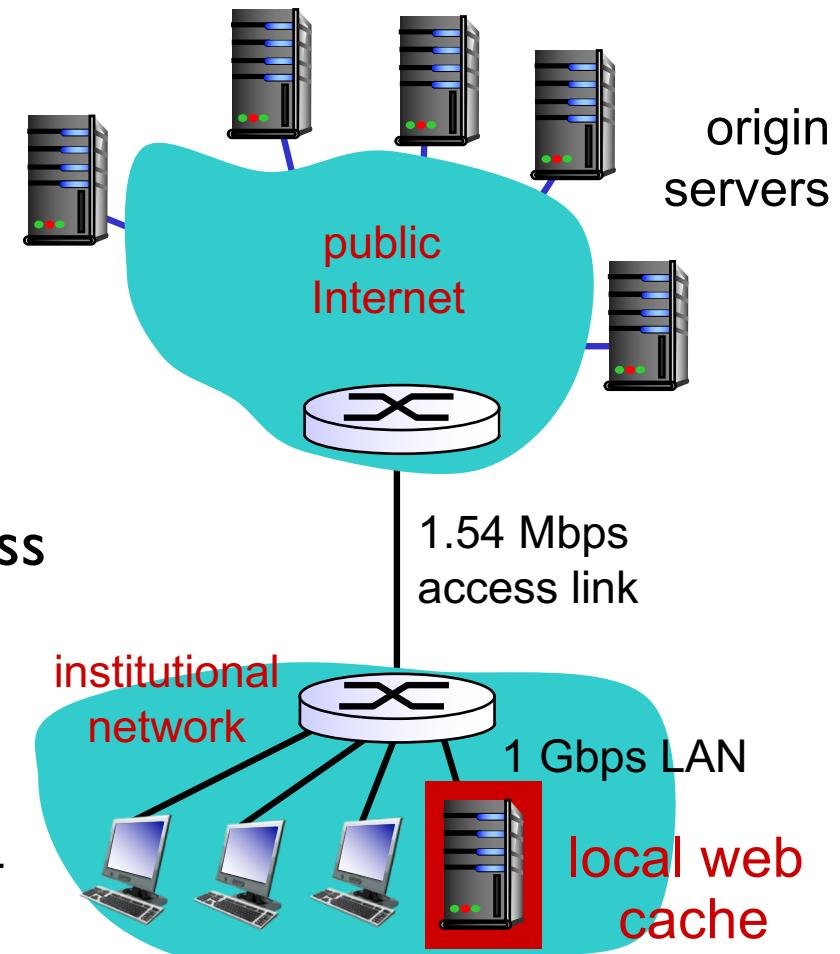


Cost: web cache (cheap!)

Caching example: install local cache

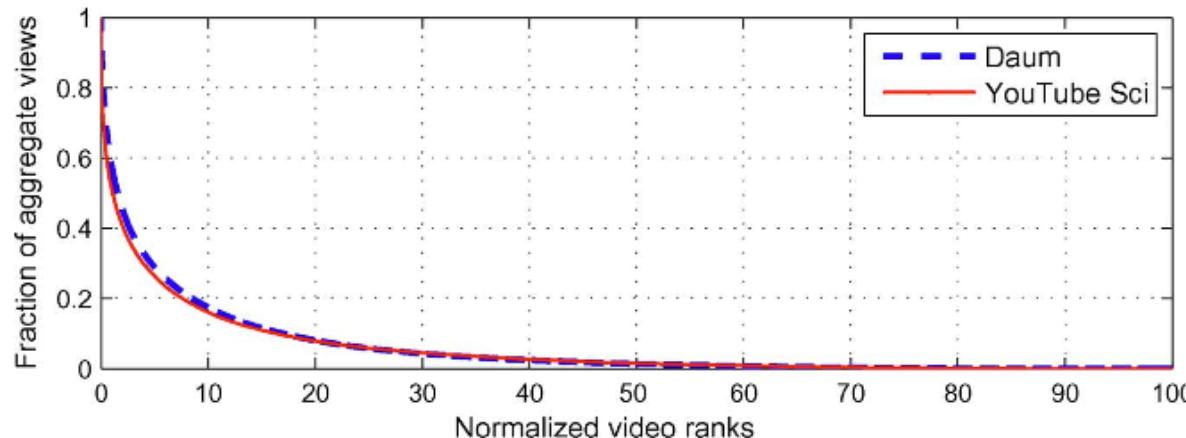
Calculating access link utilization, delay with cache:

- ❖ suppose cache hit rate is 0.4
 - 40% requests satisfied at cache,
60% requests satisfied at origin
- ❖ access link utilization:
 - 60% of requests use access link
- ❖ data rate to browsers over access link = $0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$
 - utilization = $0.9 / 1.54 = .58$
- ❖ total delay
 - $= 0.6 * (\text{delay from origin servers}) + 0.4 * (\text{delay when satisfied at cache})$
 - $= 0.6 (2.01) + 0.4 (\sim \text{msecs})$
 - $= \sim 1.2 \text{ secs}$
 - less than with 154 Mbps link (and cheaper too!)



But what is the likelihood of cache hits?

- ❖ Distribution of web object requests generally follows a Zipf-like distribution
- ❖ *The probability that a document will be referenced k requests after it was last referenced is roughly proportional to 1/k*. That is, web traces exhibit excellent **temporal locality**.



Video content exhibits similar properties: 10% of the top popular videos account for nearly 80% of views, while the remaining 90% of videos account for total 20% of requests.

Paper – <http://yongyeol.com/papers/cha-video-2009.pdf>

Paper – “Web Caching and Zipf-like Distributions: Evidence and Implications”
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.8742&rep=rep1&type=pdf>

Conditional GET

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

- no object transmission delay
- lower link utilization

- ❖ **cache:** 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

client



server



HTTP request msg
If-modified-since: <date>

HTTP response
HTTP/1.0
304 Not Modified

object
not
modified
before
<date>

HTTP request msg
If-modified-since: <date>

HTTP response
HTTP/1.0 200 OK
<data>

object
modified
after
<date>

Example Cache Check Request

GET / HTTP/1.1

Accept: */*

Accept-Language: en-us

Accept-Encoding: gzip, deflate

If-Modified-Since: Mon, 29 Jan 2001 17:54:18 GMT

If-None-Match: "7a11f-10ed-3a75ae4a"

User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT
5.0)

Host: www.intel-iris.net

Connection: Keep-Alive

Example Cache Check Response

HTTP/1.1 304 Not Modified

Date: Tue, 27 Mar 2001 03:50:51 GMT

Server: Apache/1.3.14 (Unix) (Red-Hat/Linux) mod_ssl/2.7.1
OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod_perl/1.24

Connection: Keep-Alive

Keep-Alive: timeout=15, max=100

ETag: "7a11f-10ed-3a75ae4a"

Improving HTTP Performance: Replication

- ❖ Replicate popular Web site across many machines
 - Spreads load on servers
 - Places content closer to clients
 - Helps when content isn't cacheable
- ❖ Problem:
 - Want to direct client to particular replica
 - Balance load across server replicas
 - Pair clients with nearby servers
 - Expensive
- ❖ Common solution:
 - DNS returns different addresses based on client's geo location, server load, etc.

Improving HTTP Performance: CDN

- ❖ Caching and replication as a service
- ❖ Integrate forward and reverse caching functionality
- ❖ Large-scale distributed storage infrastructure (usually) administered by one entity
 - e.g., Akamai has servers in 20,000+ locations
- ❖ Combination of (pull) caching and (push) replication
 - **Pull:** Direct result of clients' requests
 - **Push:** Expectation of high access rate
- ❖ Also do some processing
 - Handle *dynamic* web pages
 - *Transcoding*
 - *Maybe do some security function – watermark IP*

What about HTTPS?

- ❖ HTTP is insecure
- ❖ HTTP basic authentication: password sent using base64 encoding (can be readily converted to plaintext)
- ❖ HTTPS: HTTP over a connection encrypted by Transport Layer Security (TLS)
- ❖ Provides:
 - Authentication
 - Bidirectional encryption
- ❖ Widely used in place of plain vanilla HTTP



What's on the horizon: HTTP/2

- ❖ Google SPDY (speedy) -> HTTP/2: (RFC 7540 May 2015)
- ❖ Better content structure
- ❖ Improvements
 - Servers can **push** content and thus reduce overhead of an additional request cycle
 - Fully multiplexed
 - Requests and responses are sliced in smaller chunks called frames, frames are tagged with an ID that connects data to the request/response
 - overcomes Head-of-line blocking in HTTP 1.1
 - Prioritisation of the order in which objects should be sent (e.g. CSS files may be given higher priority)
 - Data compression of HTTP headers
 - Some headers such as cookies can be very long
 - Repetitive information

More details: <https://http2.github.io/faq/>
Demo: <https://http2.akamai.com/demo>⁴⁸

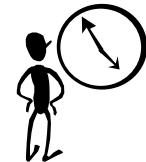
Summary

- ❖ Application Layer (Chapter 2)

- Principles of Network Applications
- HTTP

- ❖ Next:

- E-mail
- DNS



**Reading Exercise
Chapter 2: 2.3 – 2.4**

Computer Networks and Applications

COMP 3331/COMP 9331

Week 3

Application Layer (Email, DNS)

Reading Guide: Chapter 2, Sections 2.3, 2.4

Application Layer: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

Self study

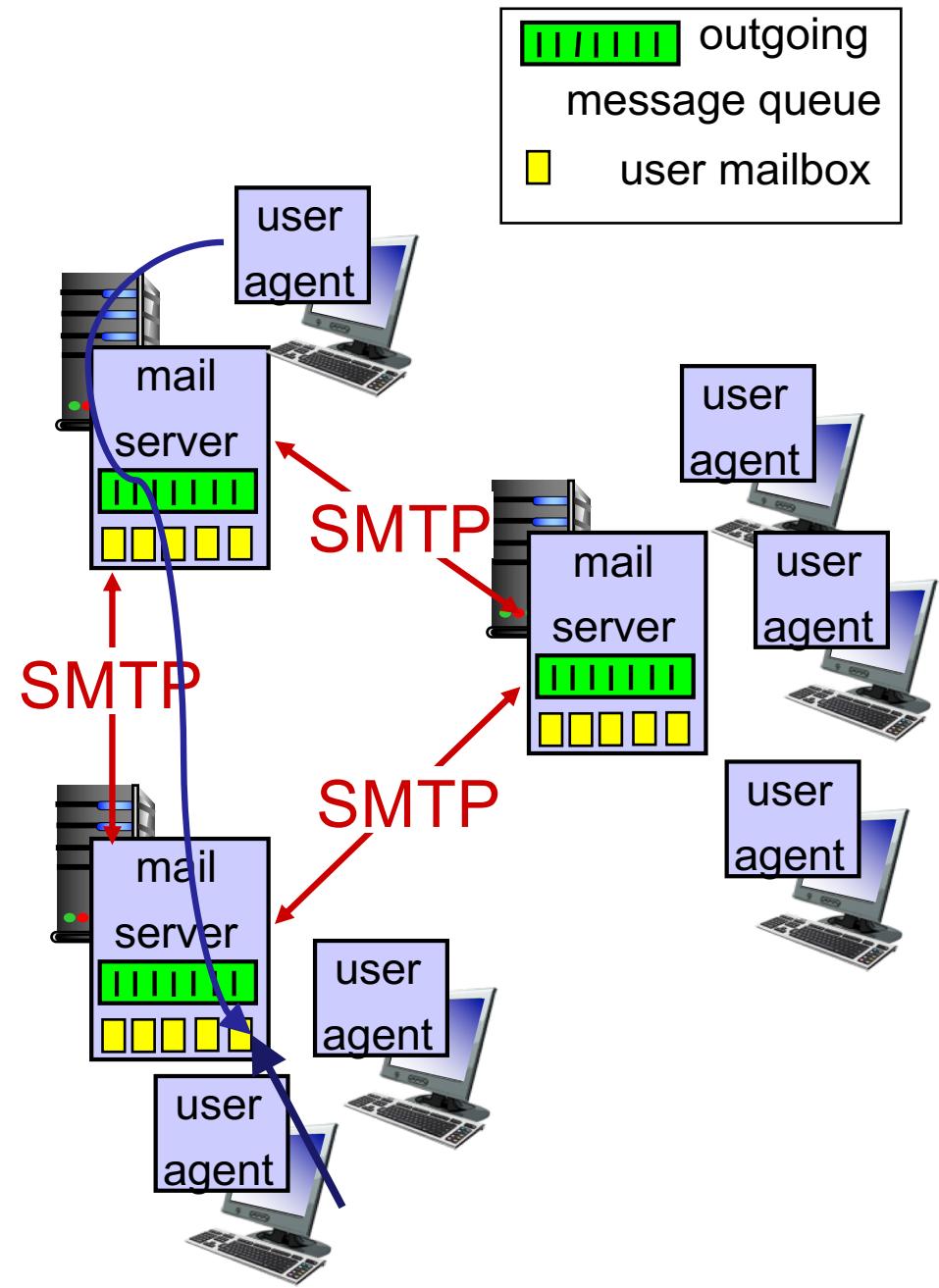
Electronic mail

Three major components:

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

User Agent

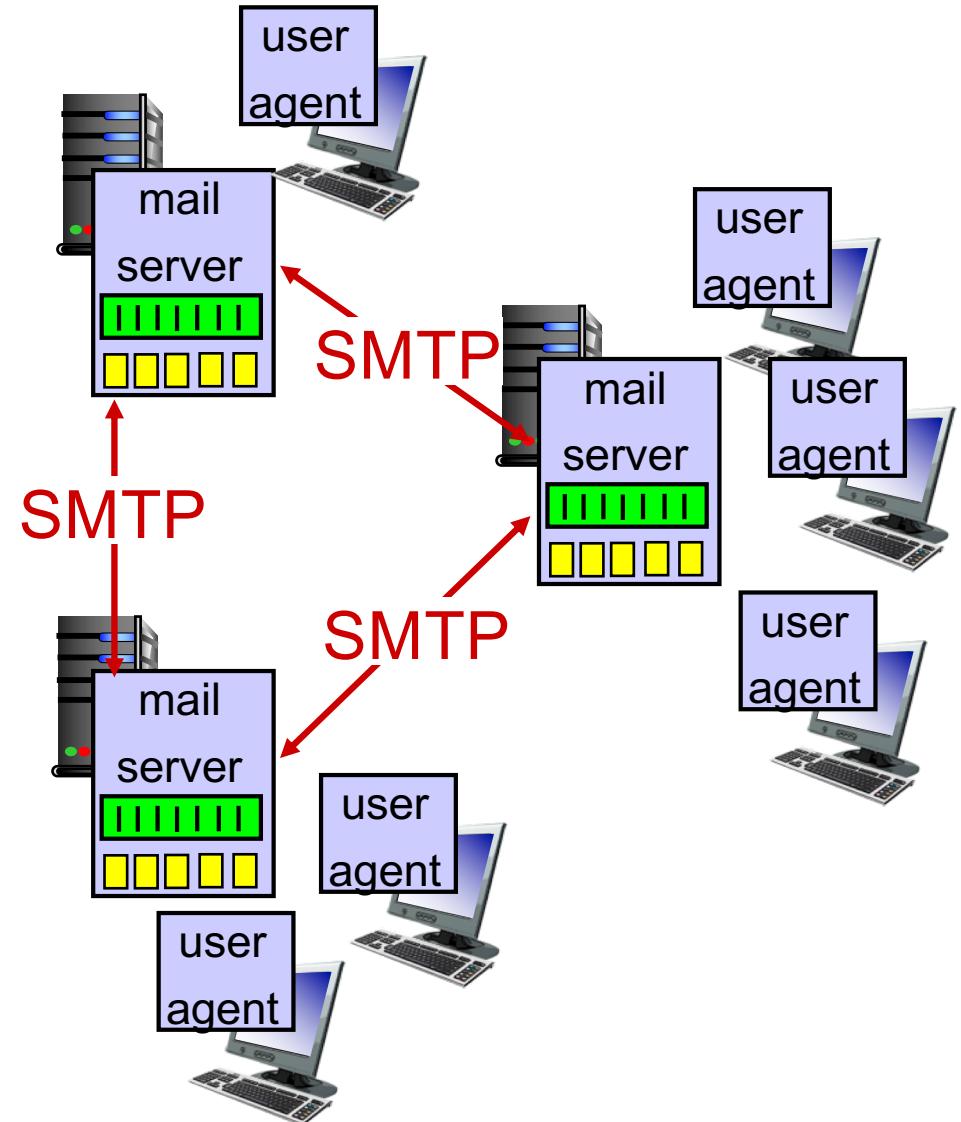
- ❖ a.k.a. “mail reader”
- ❖ composing, editing, reading mail messages
- ❖ e.g., Outlook, Thunderbird, iPhone mail client
- ❖ outgoing, incoming messages stored on server



Electronic mail: mail servers

mail servers:

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

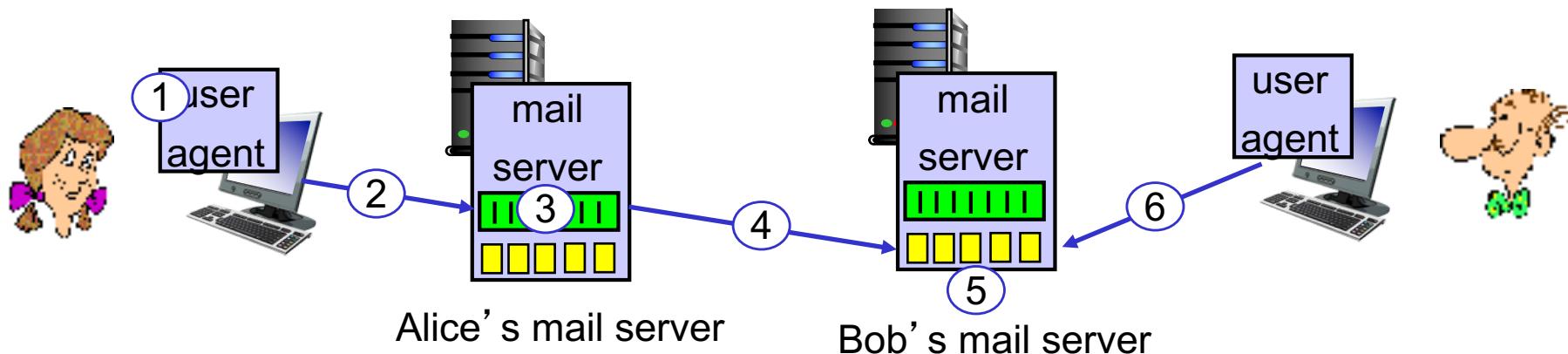


Electronic Mail: SMTP [RFC 2821]

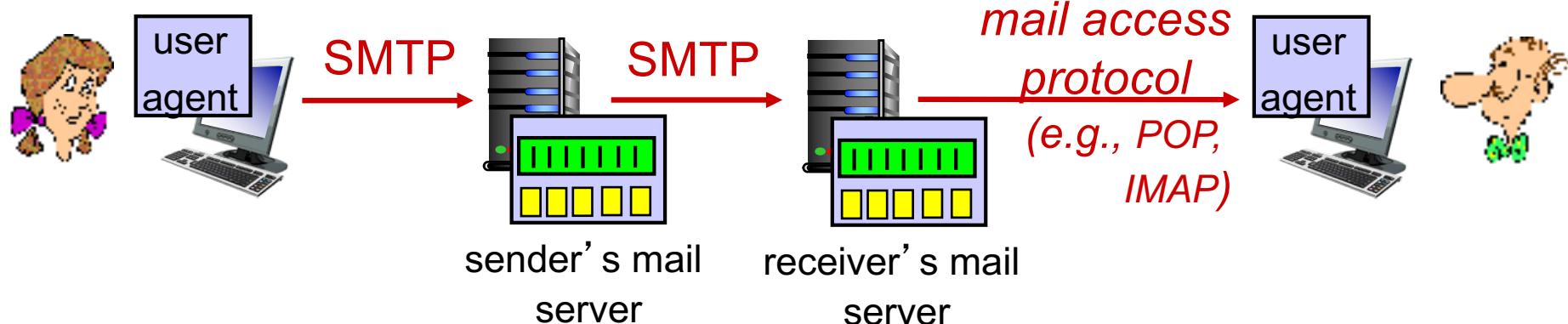
- ❖ uses TCP to reliably transfer email message from client to server, port 25
- ❖ direct transfer: sending server to receiving server
- ❖ three phases of transfer
 - handshaking (greeting)
 - transfer of messages
 - closure
- ❖ command/response interaction (like HTTP, FTP)
 - **commands:** ASCII text
 - **response:** status code and phrase
- ❖ messages must be in 7-bit ASCII

Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message “to”
bob@someschool.edu
- 2) Alice’s UA sends message to her mail server; message placed in message queue
- 3) client side of SMTP opens TCP connection with Bob’s mail server
- 4) SMTP client sends Alice’s message over the TCP connection
- 5) Bob’s mail server places the message in Bob’s mailbox
- 6) Bob invokes his user agent to read message



Mail access protocols



- ❖ **SMTP:** delivery/storage to receiver's server
- ❖ mail access protocol: retrieval from server
 - **POP:** Post Office Protocol [RFC 1939]: authorization, download
 - **IMAP:** Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
 - **HTTP(S):** Gmail, Yahoo! Mail, etc.

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?

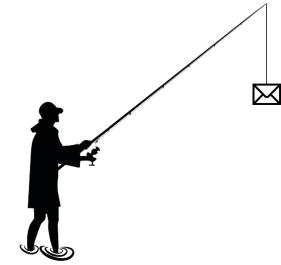
C: .

S: 250 Message accepted for delivery

C: QUIT

S: 221 hamburger.edu closing connection

Phishing



- ❖ Spear phishing
 - Phishing attempts directed at specific individuals or companies
 - Attackers may gather personal information (social engineering) about their targets to increase their probability of success
 - Most popular and accounts for over 90% of attacks
- ❖ Clone phishing
 - A type of phishing attack whereby a legitimate, and previously delivered email containing an attachment or link has had its content and recipient address(es) taken and used to create an almost identical or cloned email.
 - The attachment or link within the email is replaced with a malicious version and then sent from an email address spoofed to appear to come from the original sender.



SMTP: final words

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

comparison with HTTP:

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

Mail message format

SMTP: protocol for
exchanging email msgs

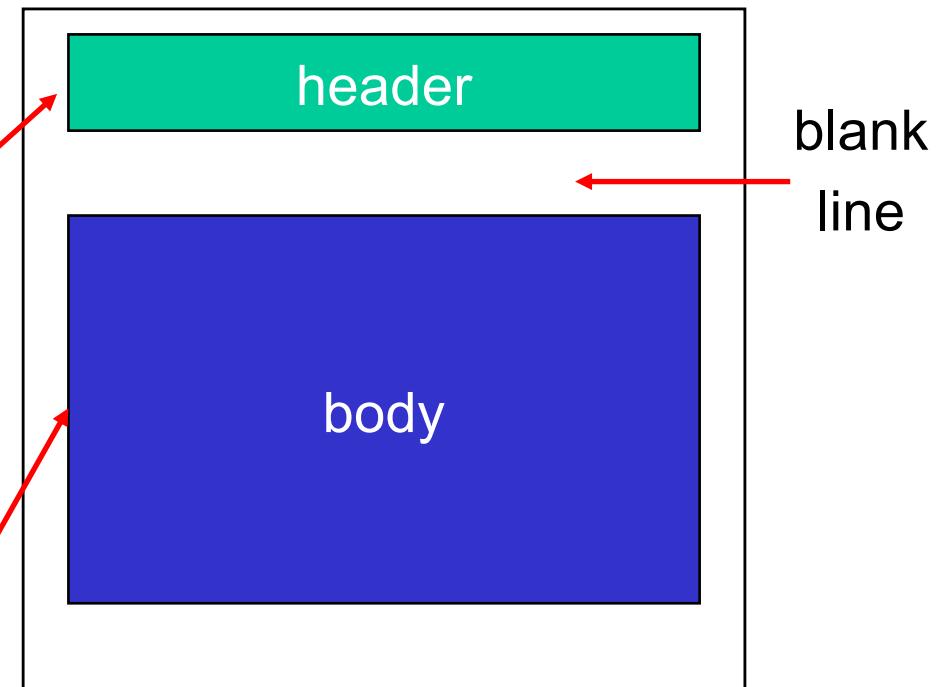
RFC 5322 (822,2822):
standard for text message
format (Internet Message
Format, IMF):

- ❖ header lines, e.g.

- To:
 - From:
 - Subject:

*different from SMTP MAIL
FROM, RCPT TO:
commands!*

- ❖ Body: the “message”
 - ASCII characters only



Quiz: SMTP

Why do we have Sender's mail server?

- User agent can directly connect with recipient mail server without the need of sender's mail server? What's the catch?

Why do we have a separate Receiver's mail server?

- Can't the recipient run the mail server on own end system?

2. Application Layer: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

A nice overview: <https://webhostinggeeks.com/guides/dns/>

DNS: domain name system

people: many identifiers:

- TFN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., www.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System:

- ❖ *distributed database* implemented in hierarchy of many *name servers*
- ❖ *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
 - note: core Internet function, implemented as application-layer protocol
 - complexity at network’s “edge”

DNS: History

- ❖ Initially all host-address mappings were in a hosts.txt file (in /etc/hosts):
 - Maintained by the Stanford Research Institute (SRI)
 - Changes were submitted to SRI by email
 - New versions of hosts.txt periodically FTP'd from SRI
 - An administrator could pick names at their discretion
- ❖ As the Internet grew this system broke down:
 - SRI couldn't handle the load; names were not unique; hosts had inaccurate copies of hosts.txt
- ❖ The Domain Name System (DNS) was invented to fix this



Jon Postel

<http://www.wired.com/2012/10/joe-postel/>

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
 - Content Distribution Networks: use IP address of requesting host to find best suitable server
 - Example: closest, least-loaded, etc

why not centralize DNS?

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

A: *doesn't scale!*

Goals

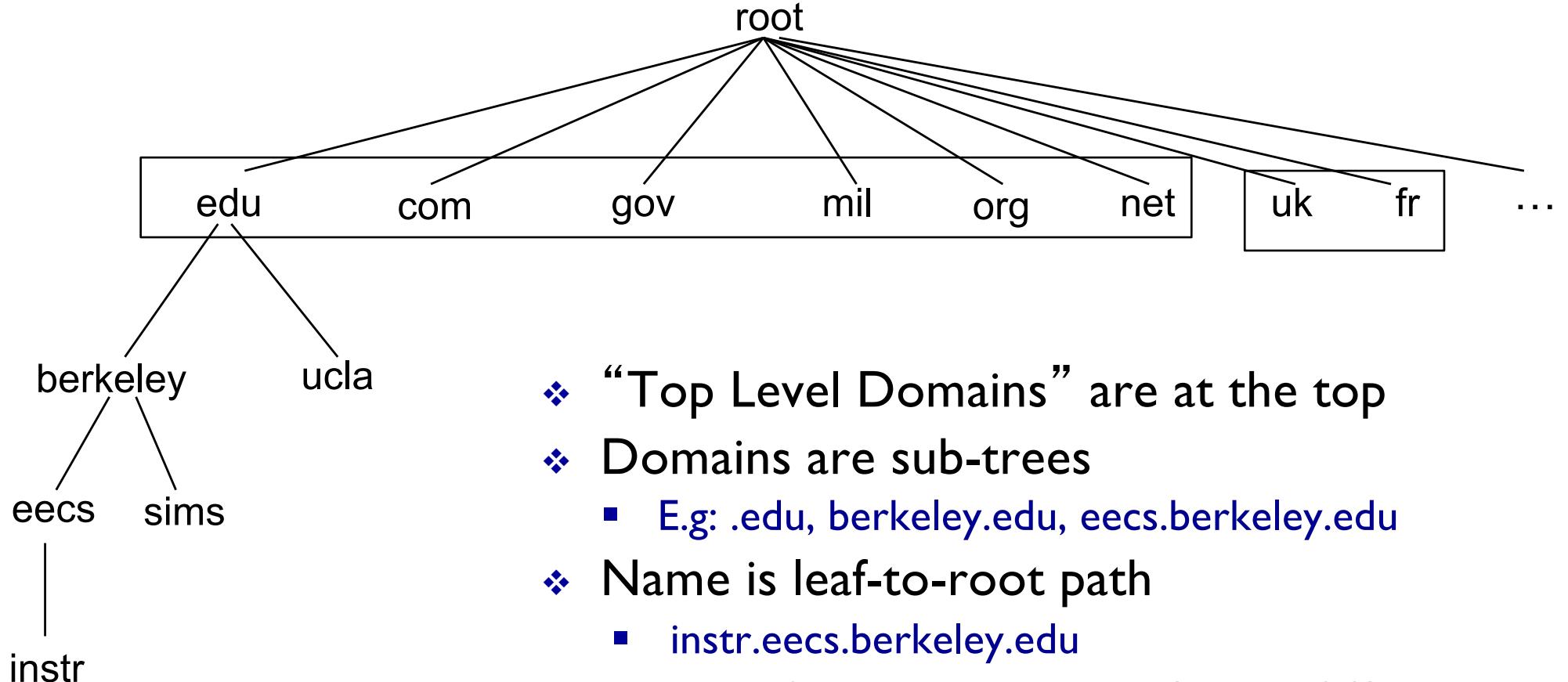
- ❖ No naming conflicts (uniqueness)
- ❖ Scalable
 - many names
 - (secondary) frequent updates
- ❖ Distributed, autonomous administration
 - Ability to update my own (machines') names
 - Don't have to track everybody's updates
- ❖ Highly available
- ❖ Lookups should be fast

Key idea: Hierarchy

Three intertwined hierarchies

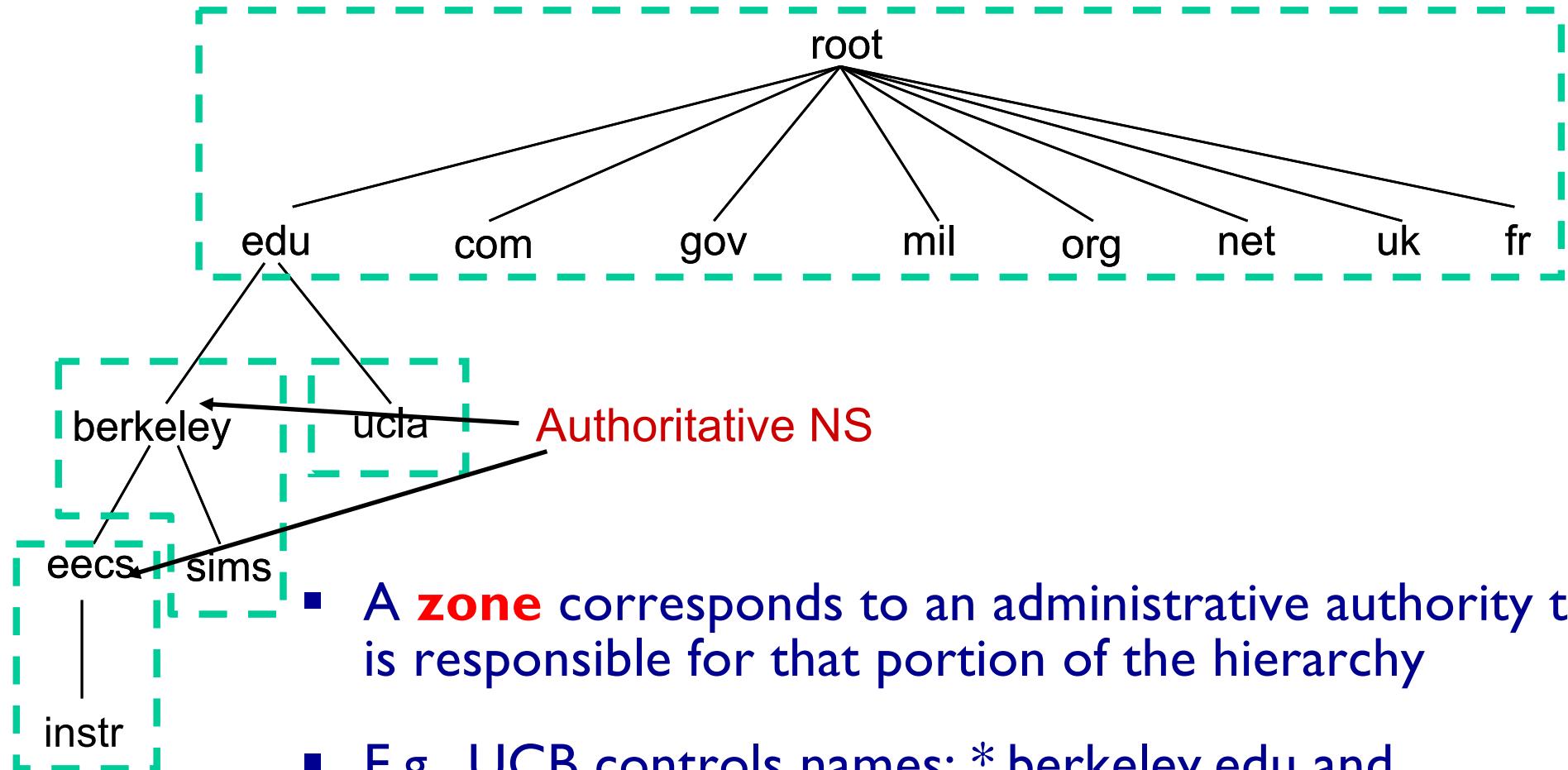
- Hierarchical namespace
 - As opposed to original flat namespace
- Hierarchically administered
 - As opposed to centralised
- (Distributed) hierarchy of servers
 - As opposed to centralised storage

Hierarchical Namespace



- ❖ “Top Level Domains” are at the top
- ❖ Domains are sub-trees
 - E.g: .edu, berkeley.edu, eecs.berkeley.edu
- ❖ Name is leaf-to-root path
 - instr.eecs.berkeley.edu
- ❖ Depth of tree is arbitrary (limit 128)
- ❖ Name collisions trivially avoided
 - each domain is responsible

Hierarchical Administration



- ❖ E.g., EECS controls names: *.eeecs.berkeley.edu

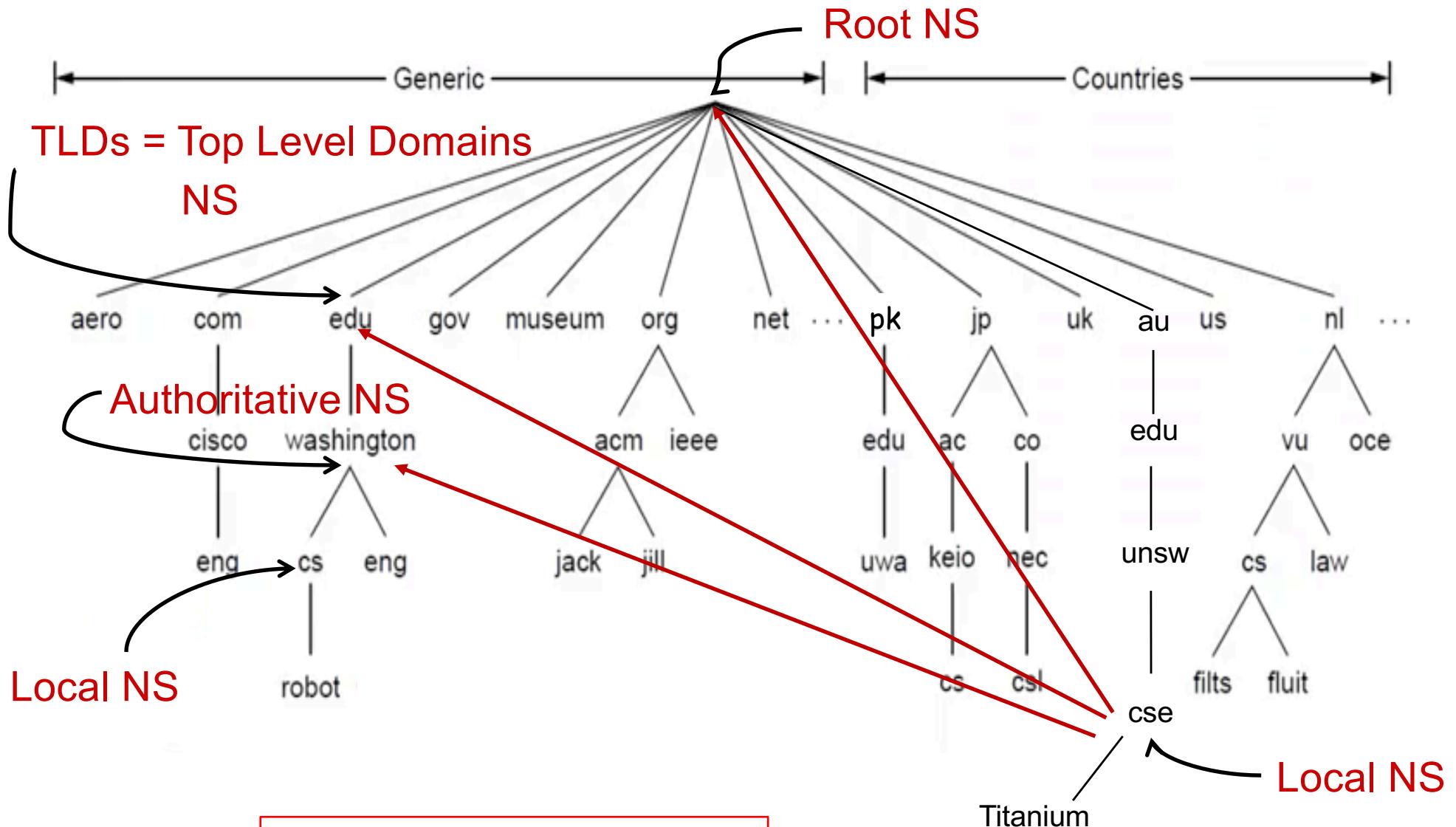
Server Hierarchy

- ❖ Top of hierarchy: Root servers
 - Location hardwired into other servers
- ❖ Next Level: Top-level domain (TLD) servers
 - .com, .edu, etc.
 - Managed professionally
- ❖ Bottom Level: **Authoritative** DNS servers
 - Actually store the name-to-address mapping
 - Maintained by the corresponding administrative authority

Server Hierarchy

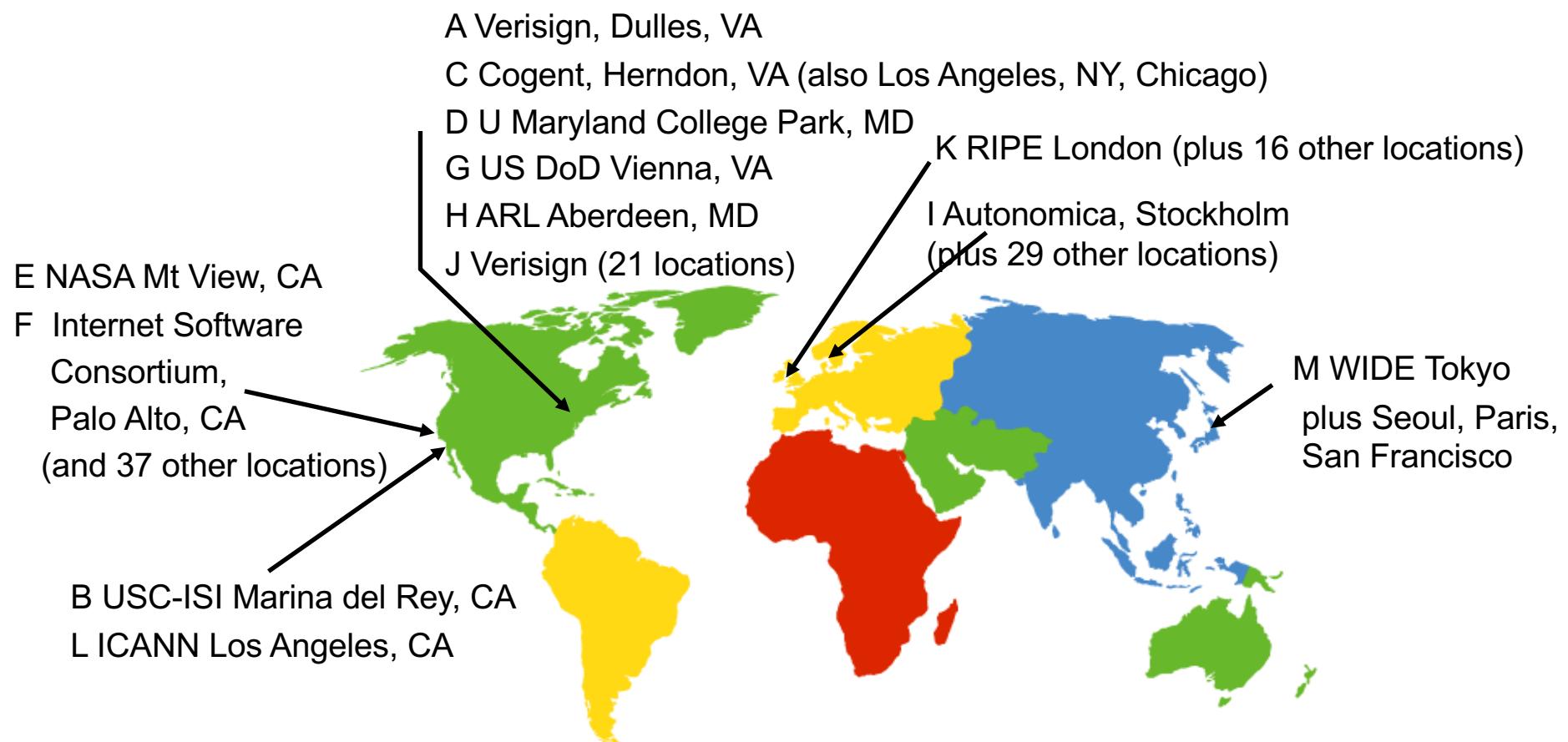
- ❖ Each server stores a (small!) subset of the total DNS database
- ❖ An authoritative DNS server stores “resource records” for all DNS names in the domain that it has authority for
- ❖ Each server needs to know other servers that are responsible for the other portions of the hierarchy
 - Every server knows the root
 - Root server knows about all top-level domains

DNS: a distributed, hierarchical database



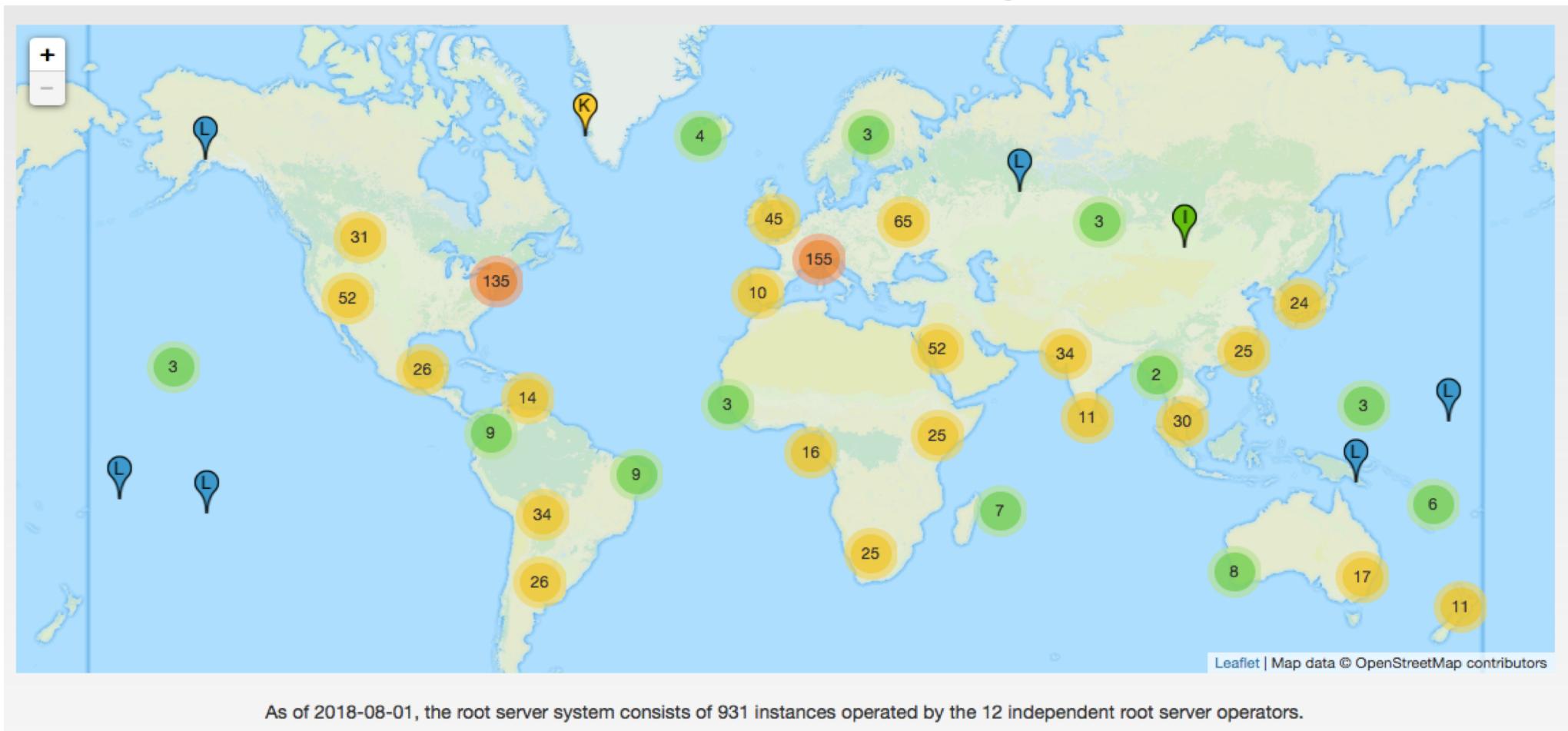
DNS Root Servers

- 13 root servers (labeled A-M; see <http://www.root-servers.org/>)
- Replicated via any-casting



Root Server health: <https://www.ultratools.com/tools/dnsRootServerSpeed>

DNS: root name servers



www.root-servers.org



TLD, authoritative servers

top-level domain (TLD) servers:

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

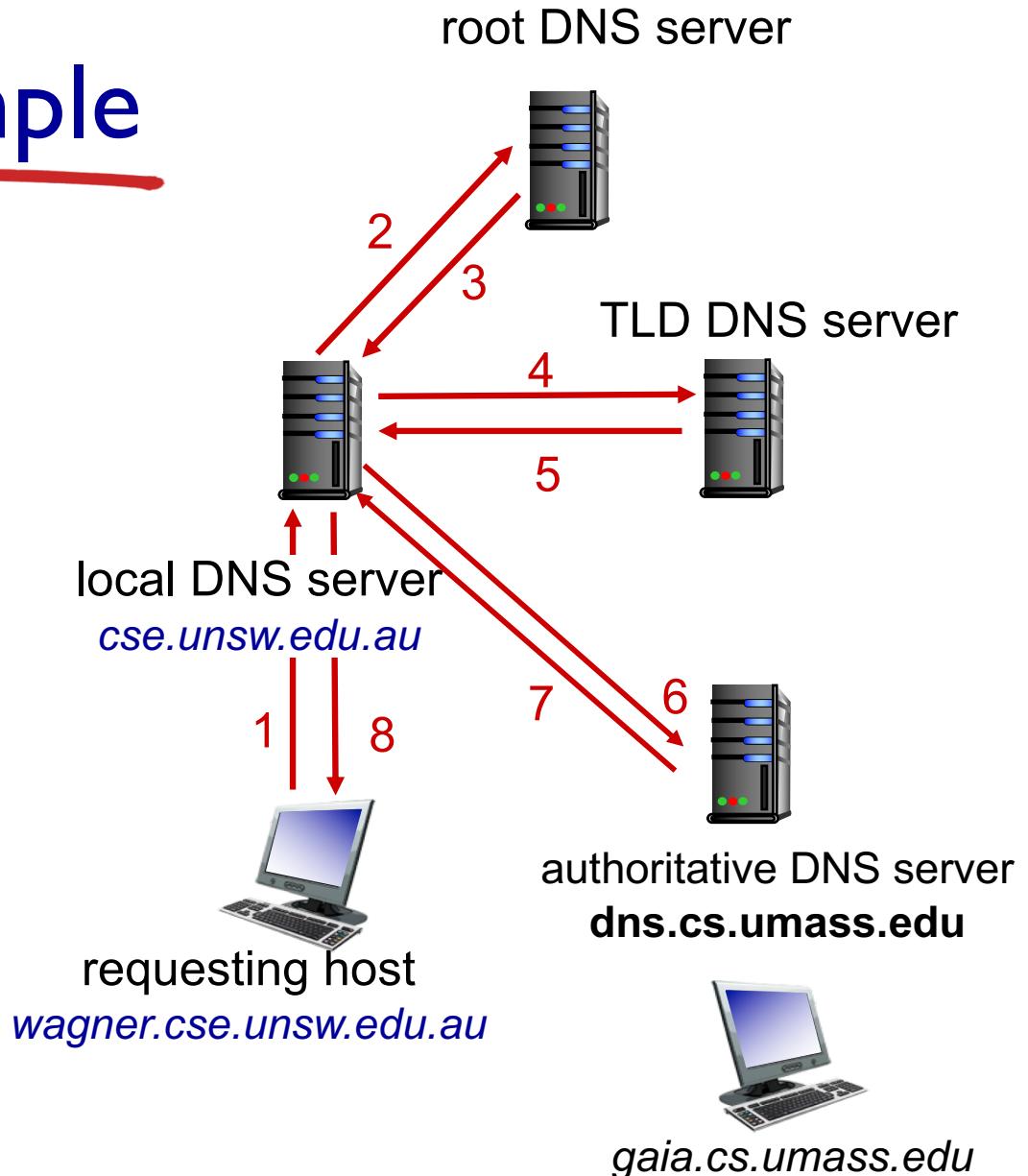
- ❖ does not strictly belong to hierarchy
- ❖ each ISP (residential ISP, company, university) has one
 - also called “default name server”
- ❖ Hosts configured with local DNS server address (e.g.,
`/etc/resolv.conf`) or learn server via a host configuration protocol (e.g., DHCP)
- ❖ Client application
 - Obtain DNS name (e.g., from URL)
 - Do `gethostbyname()` to trigger DNS request to its local DNS server
- ❖ when host makes DNS query, query is sent to its local DNS server
 - has local cache of recent name-to-address translation pairs (but may be out of date!)
 - acts as proxy, forwards query into hierarchy

DNS name resolution example

- ❖ host at `wagner.cse.unsw.edu.au` wants IP address for `gaia.cs.umass.edu`

iterated query:

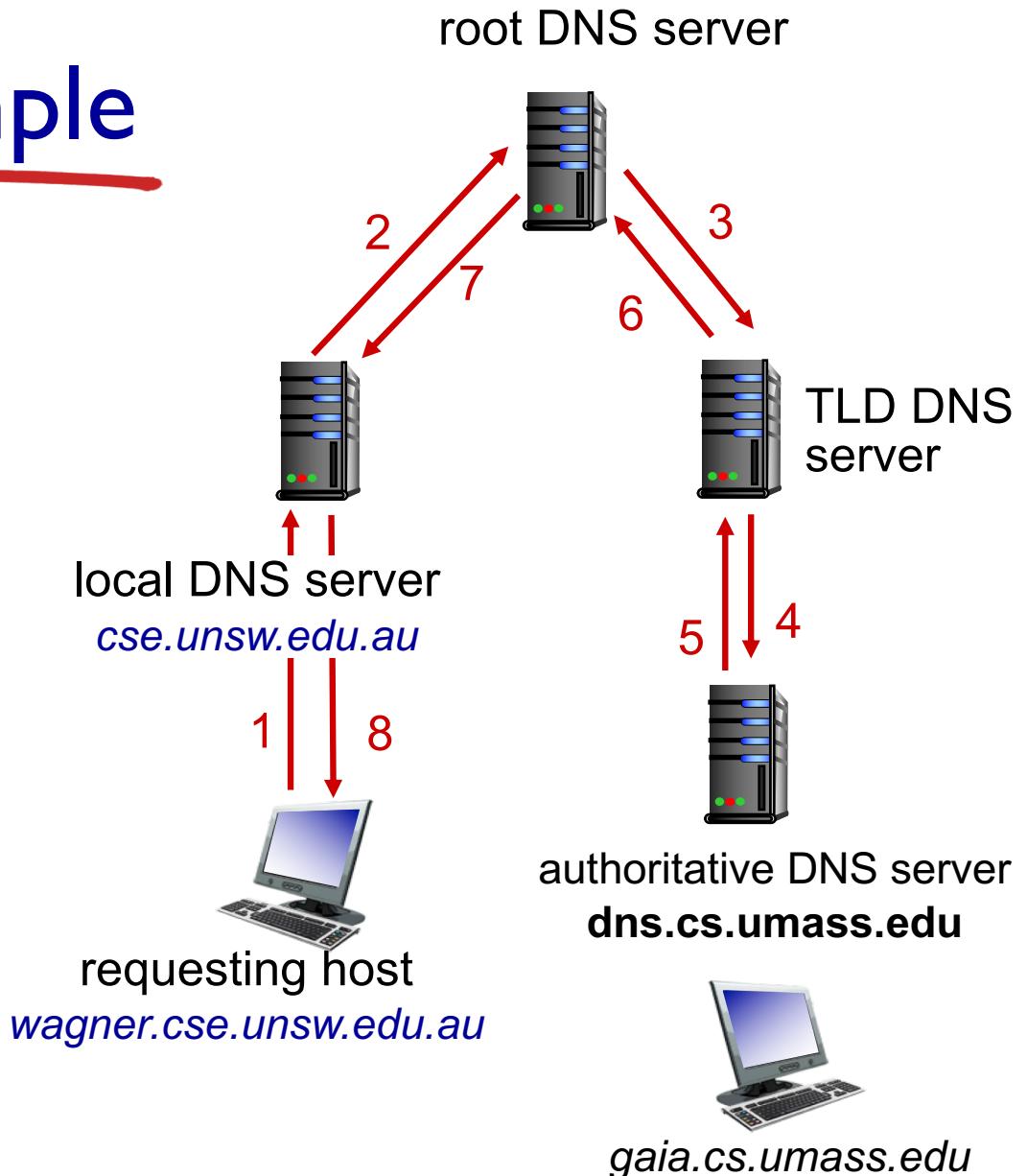
- ❖ contacted server replies with name of server to contact
- ❖ “I don’t know this name, but ask this server”



DNS name resolution example

recursive query:

- ❖ puts burden of name resolution on contacted name server



DNS: caching, updating records

- ❖ once (any) name server learns mapping, it *caches* mapping
 - cache entries timeout (disappear) after some time (TTL)
 - TLD servers typically cached in local name servers
 - thus root name servers not often visited
- ❖ Subsequent requests need not burden DNS
- ❖ cached entries may be *out-of-date* (best effort name-to-address translation!)
 - if name host changes IP address, may not be known Internet-wide until all TTLs expire

DNS records

DNS: distributed db 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

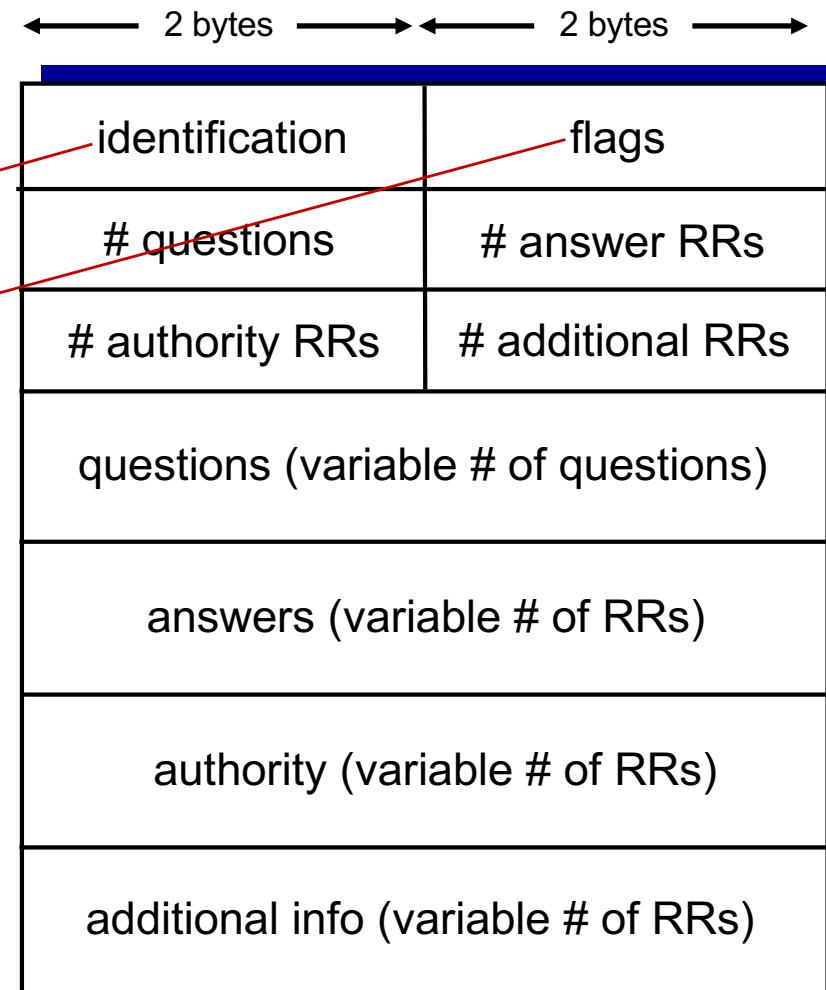
- **value** is name of mailserver
associated with **name**

DNS protocol, messages

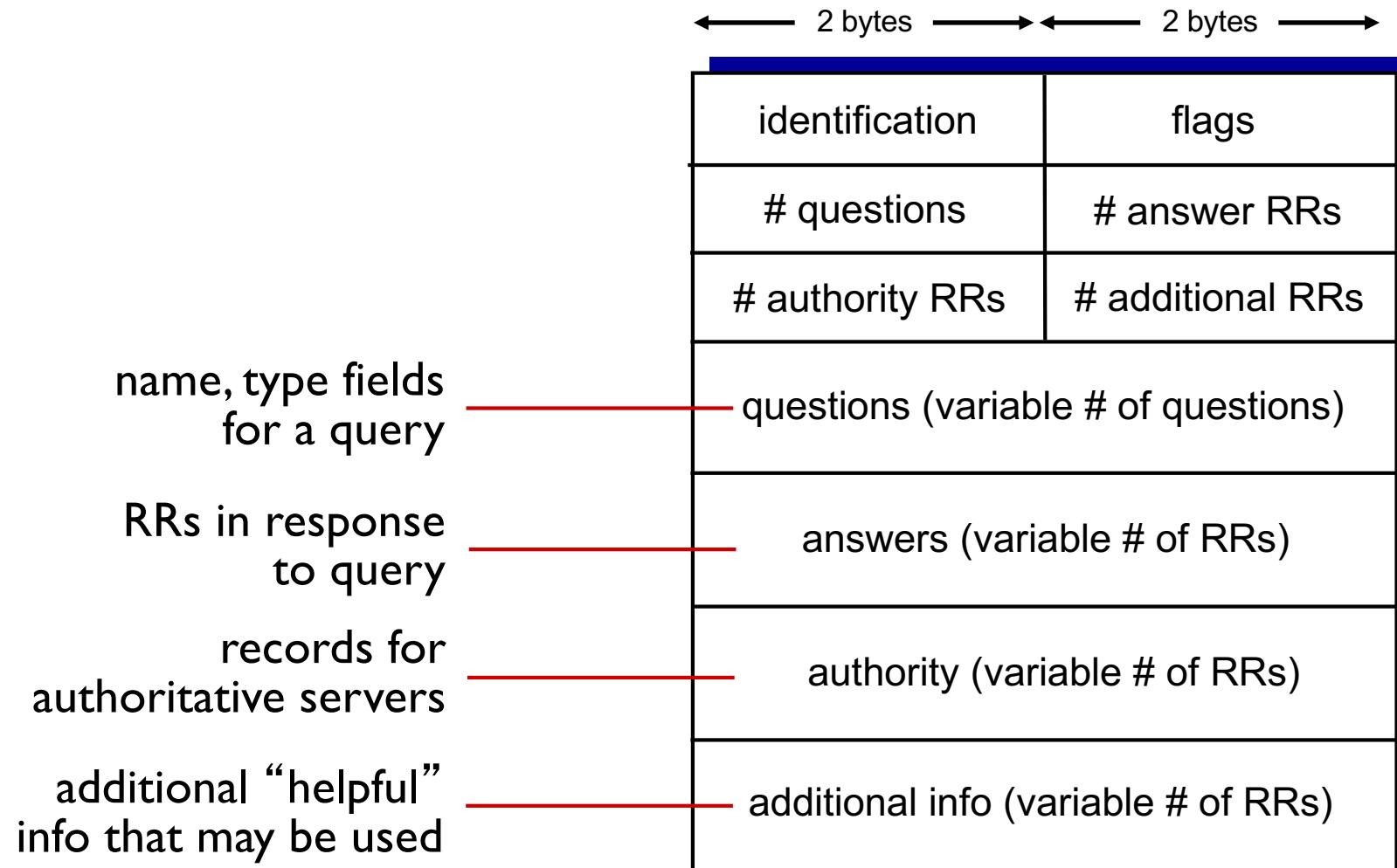
- ❖ *query* and *reply* messages, both with same *message format*

msg header

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



DNS protocol, messages



An Example

Try this out
yourself. Part of
one of the lab

```
bash-3.2$ dig www.oxford.ac.uk

; <>> DiG 9.8.3-P1 <>> www.oxford.ac.uk
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35102
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 4, ADDITIONAL: 5

;; QUESTION SECTION:
;www.oxford.ac.uk.           IN      A

;; ANSWER SECTION:
www.oxford.ac.uk.        300     IN      A      129.67.242.154
www.oxford.ac.uk.        300     IN      A      129.67.242.155

;; AUTHORITY SECTION:
oxford.ac.uk.            86399   IN      NS      dns2.ox.ac.uk.
oxford.ac.uk.            86399   IN      NS      dns1.ox.ac.uk.
oxford.ac.uk.            86399   IN      NS      ns2.ja.net.
oxford.ac.uk.            86399   IN      NS      dns0.ox.ac.uk.

;; ADDITIONAL SECTION:
ns2.ja.net.              33560   IN      A      193.63.105.17
ns2.ja.net.              33560   IN      AAAA    2001:630:0:45::11
dns0.ox.ac.uk.          48090   IN      A      129.67.1.190
dns1.ox.ac.uk.          86399   IN      A      129.67.1.191
dns2.ox.ac.uk.          54339   IN      A      163.1.2.190

;; Query time: 589 msec
;; SERVER: 129.94.172.11#53(129.94.172.11)
;; WHEN: Thu Mar  9 17:53:52 2017
;; MSG SIZE  rcvd: 242
```

Inserting records into DNS

- ❖ example: new startup “Network Utopia”
- ❖ register name networkutopia.com at *DNS registrar* (e.g., Network Solutions)
 - provide names, IP addresses of authoritative name server (primary and secondary)
 - registrar inserts two RRs into .com TLD server:
(`networkutopia.com`, `dns1.networkutopia.com`, NS)
(`dns1.networkutopia.com`, `212.212.212.1`, A)
- ❖ create authoritative server type A record for `www.networkutopia.com`; type MX record for `networkutopia.com`
- ❖ Q: Where do you insert these type A and type MX records?

A: ??

Reliability

- ❖ DNS servers are **replicated** (primary/secondary)
 - Name service available if at least one replica is up
 - Queries can be load-balanced between replicas
- ❖ Usually, UDP used for queries
 - Need reliability: must implement this on top of UDP
 - Spec supports TCP too, but not always implemented
- ❖ Try alternate servers on timeout
 - **Exponential backoff** when retrying same server
- ❖ Same identifier for all queries
 - Don't care which server responds

DNS provides Indirection

- ❖ Addresses can **change** underneath
 - Move www.cnn.com to 4.125.91.21
 - Humans/Apps should be unaffected
- ❖ Name could map to **multiple** IP addresses
 - Enables
 - Load-balancing
 - Reducing latency by picking nearby servers
- ❖ **Multiple names** for the same address
 - E.g., many services (mail, www, ftp) on same machine
 - E.g., aliases like www.cnn.com and cnn.com
- ❖ But, this flexibility applies only within domain!

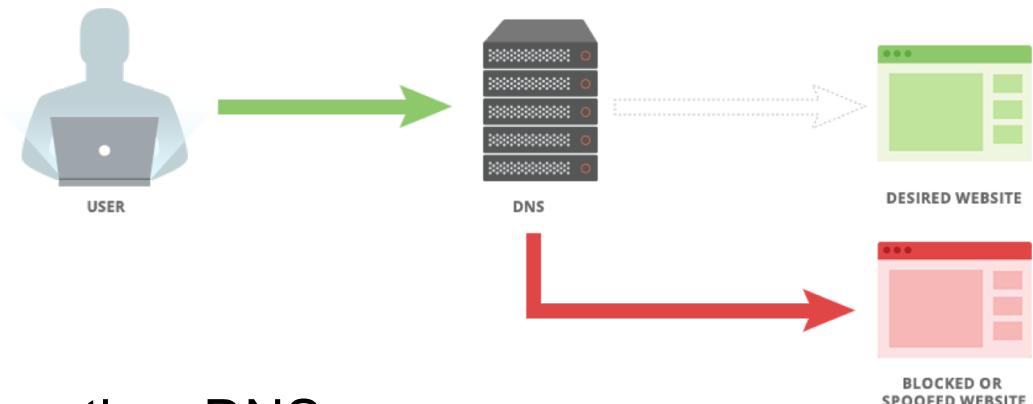
Reverse DNS

- ❖ IP address -> domain name
- ❖ Special PTR record type to store reverse DNS entries
- ❖ Where is reverse DNS used?
 - Troubleshooting tools such as traceroute and ping
 - “Received” trace header field in SMTP e-mail
 - SMTP servers for validating IP addresses of originating servers
 - Internet forums tracking users
 - System logging or monitoring tools
 - Used in load balancing servers/content distribution to determine location of requester



Do you trust your DNS server?

- ❖ Censorship



https://wikileaks.org/wiki/Alternative_DNS

- ❖ Logging

- IP address, websites visited, geolocation data and more
- E.g., Google DNS:

<https://developers.google.com/speed/public-dns/privacy>

Attacking DNS



DDoS attacks

- ❖ Bombard root servers with traffic
 - Not successful to date
 - Traffic Filtering
 - Local DNS servers cache IPs of TLD servers, allowing root server to be bypassed
- ❖ Bombard TLD servers
 - Potentially more dangerous

Redirect attacks

- ❖ Man-in-middle
 - Intercept queries
- ❖ DNS poisoning
 - Send bogus replies to DNS server, which caches

Exploit DNS for DDoS

- ❖ Send queries with spoofed source address: target IP
- ❖ Requires amplification

Want to dig deeper?

<http://www.networkworld.com/article/2886283/security0/top-10-dns-attacks-likely-to-infiltrate-your-network.html>



Schneier on Security

Blog

Newsletter

Books

Essays

News

Talks

Academic

About Me

[Blog >](#)

IoT Attack Against a University Network

Verizon's *Data Brief Digest 2017* describes [an attack](#) against an unnamed university by attackers who hacked a variety of IoT devices and had them spam network targets and slow them down:

Analysis of the university firewall identified over 5,000 devices making hundreds of Domain Name Service (DNS) look-ups every 15 minutes, slowing the institution's entire network and restricting access to the majority of internet services.

In this instance, all of the DNS requests were attempting to look up seafood restaurants -- and it wasn't because thousands of students all had an overwhelming urge to eat fish -- but because devices on the network had been instructed to repeatedly carry out this request.

"We identified that this was coming from their IoT network, their vending machines and their light sensors were actually looking for seafood domains; 5,000 discreet systems and they were nearly all in the IoT infrastructure," says Laurance Dine, managing principal of investigative response at Verizon.

The actual Verizon document doesn't appear to be available online yet, but there is an advance version that only discusses the incident above, available [here](#).

Detailed Report at - http://www.verizonenterprise.com/resources/reports/rp_data-breach-digest-2017-sneak-peek_xg_en.pdf

DNS Cache Poisoning



- ❖ Suppose you are a bad guy and you control the name server for drevil.com. Your name server receives a request to resolve www.drevil.com. and you respond as follows:

;; QUESTION SECTION:

;www.drevil.com. IN A

;; ANSWER SECTION:

www.drevil.com 300 IN A 129.45.212.42

;; AUTHORITY SECTION:

drevil.com 86400 IN NS dns1.drevil.com.

drevil.com 86400 IN NS google.com

A drevil.com machine, **not** google.com

;; ADDITIONAL SECTION:

google.com 600 IN A 129.45.212.222

- ❖ Solution: Do not allow DNS servers to cache IP address mappings unless they are from authoritative name servers

Dig deeper?

DNS Cache Poisoning Test

<https://www.grc.com/dns/dns.htm>



DNSSEC: DNS Security Extensions,
<http://www.dnssec.net>

Computer Networks and Applications

COMP 3331/COMP 9331

Week 4

P2P + CDN

Reading Guide: Chapter 2, 2.5, 2.6, 2.7

Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

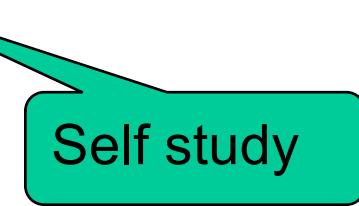
- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP



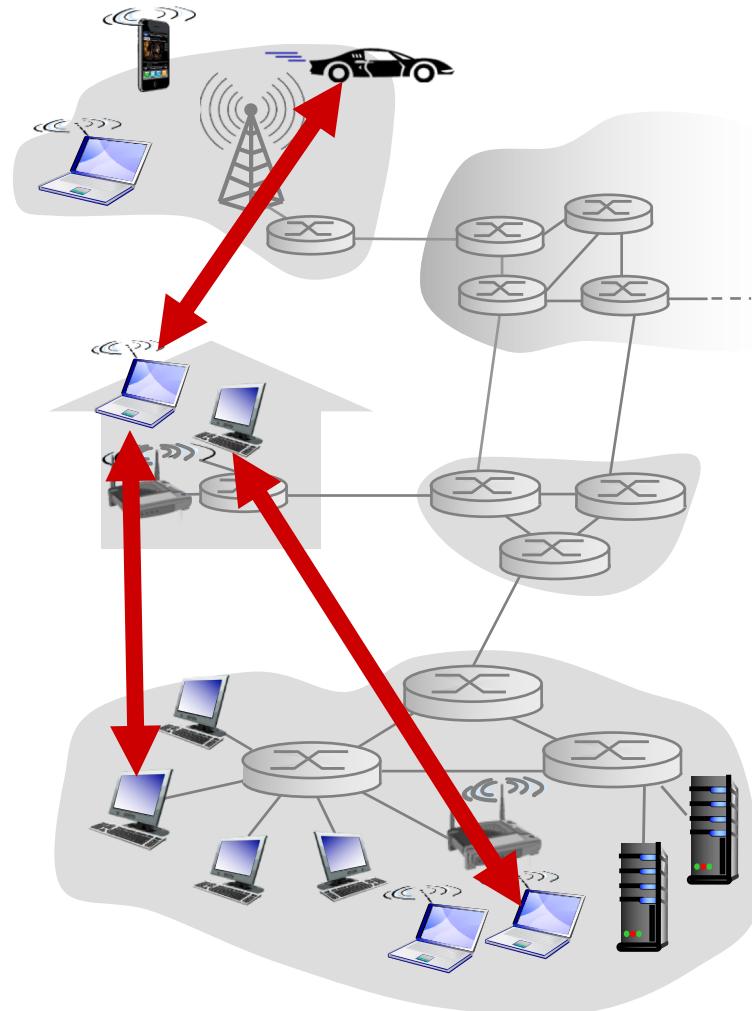
Self study

Pure P2P architecture

- ❖ no always-on server
- ❖ arbitrary end systems directly communicate
- ❖ peers are intermittently connected and change IP addresses

examples:

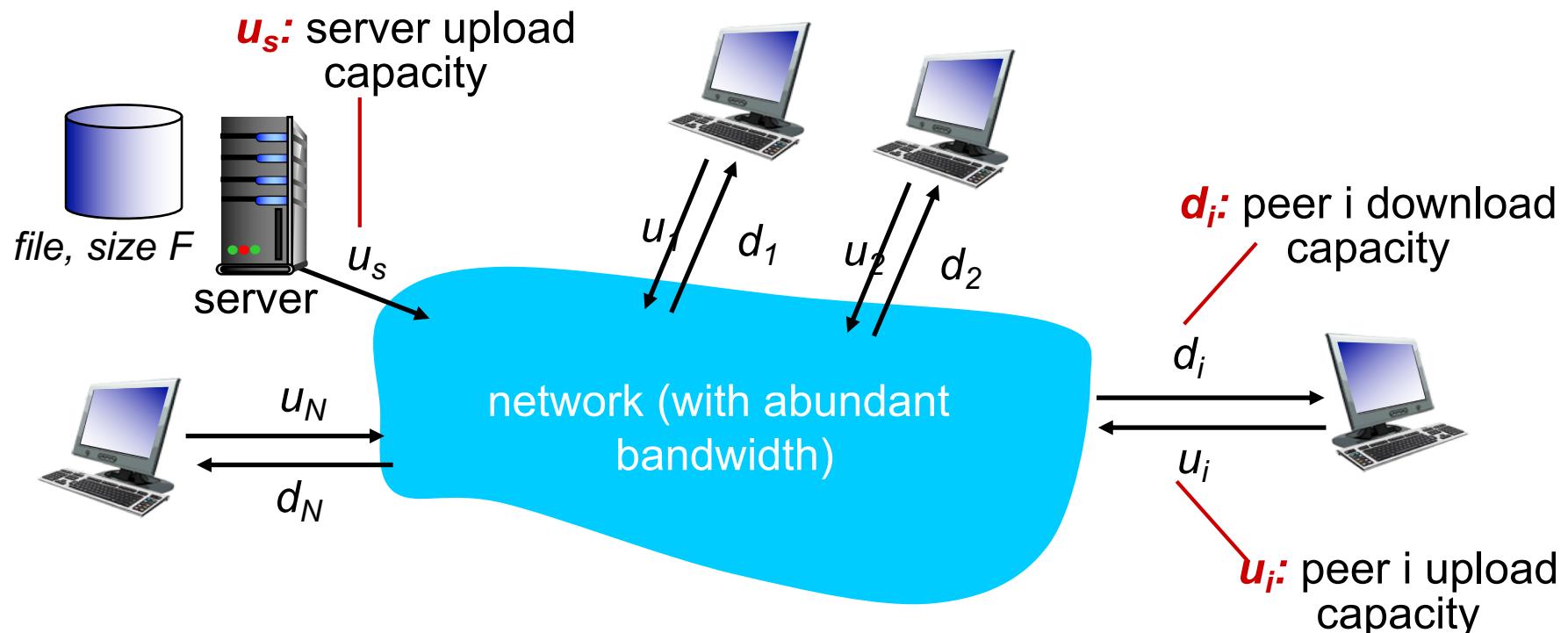
- file distribution
(BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



File distribution: client-server vs P2P

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

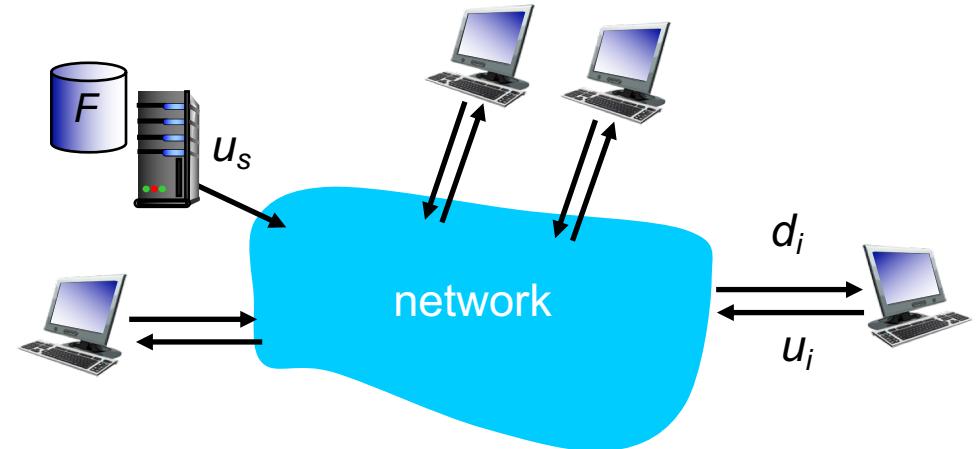
- peer upload/download capacity is limited resource



File distribution time: client-server

- ❖ **server transmission:** must 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
 - client download time: F/d_{\min}

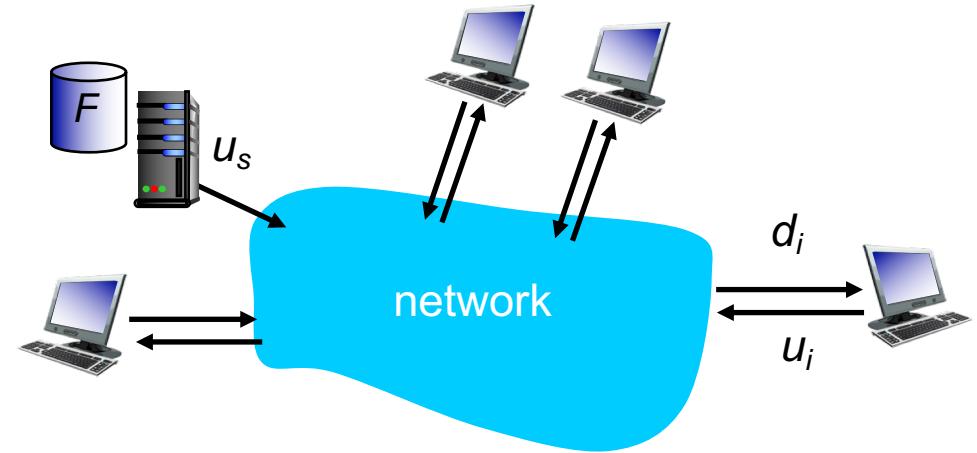
*time to distribute F
to N clients using
client-server approach*

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

increases linearly in N

File distribution time: P2P

- ❖ **server transmission:** must upload at least one copy
 - time to send one copy: F/u_s
- ❖ **client:** each client must download file copy
 - client download time: F/d_{\min}
- ❖ **clients:** as aggregate must download NF bits
 - max upload rate (limiting max download rate) is $u_s + \sum u_i$



time to distribute F
to N clients using
P2P approach

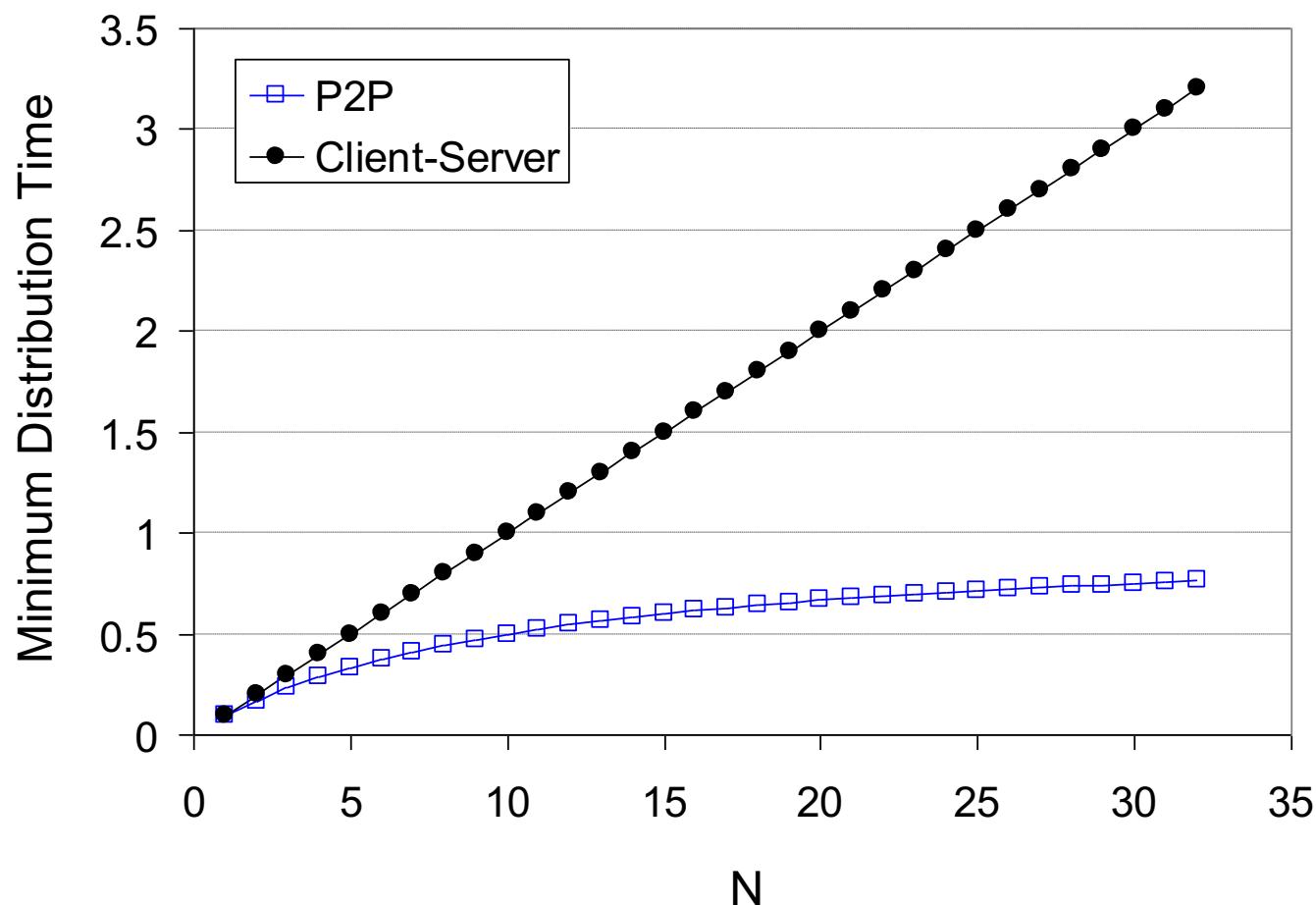
$$D_{P2P} \geq \max\{F/u_s, F/d_{\min}, NF/(u_s + \sum_{i=1}^N u_i)\}$$

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, $u_s = 10u$

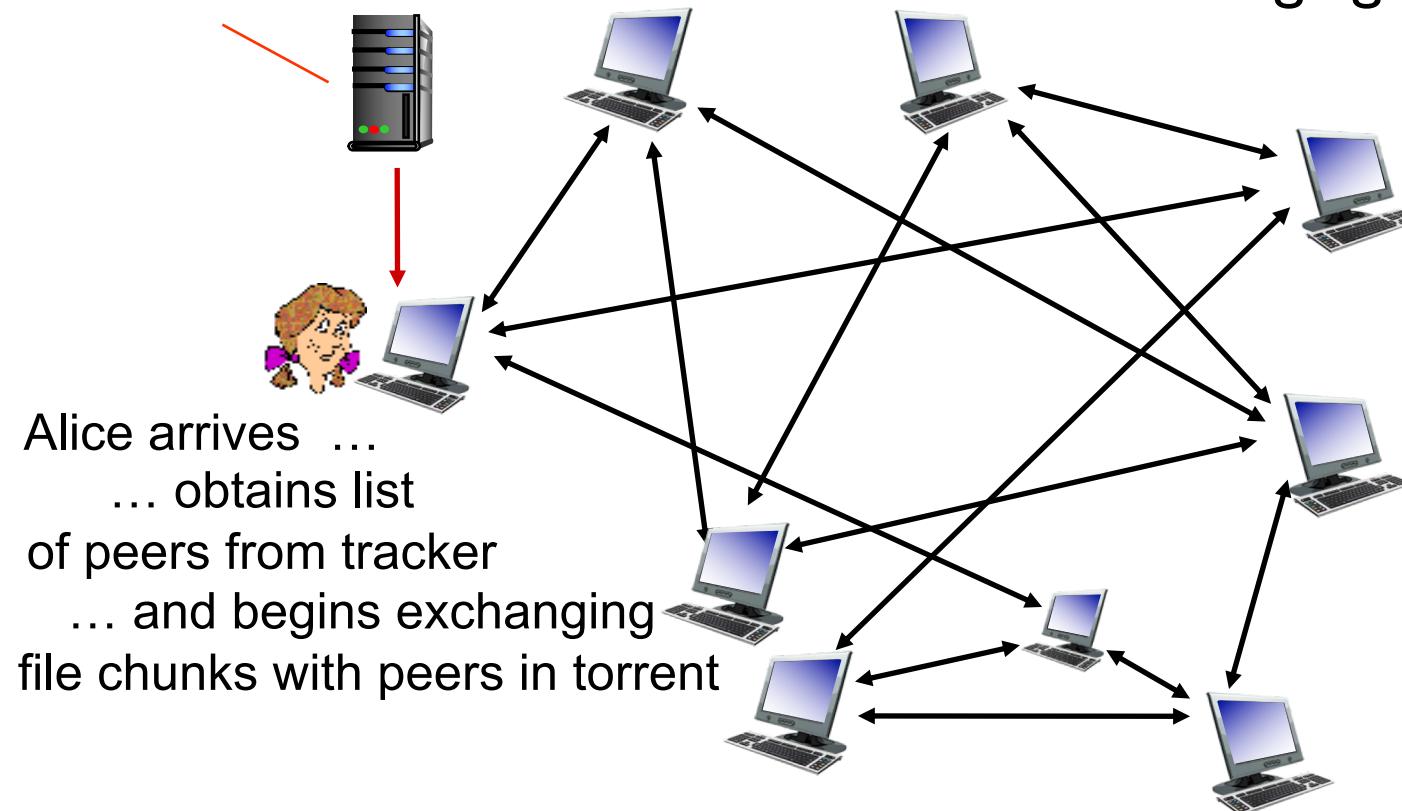


P2P file distribution: BitTorrent

- ❖ file divided into 256KB chunks
- ❖ peers in torrent send/receive file chunks

tracker: tracks peers
participating in torrent

torrent: group of peers
exchanging chunks of a file



.torrent files

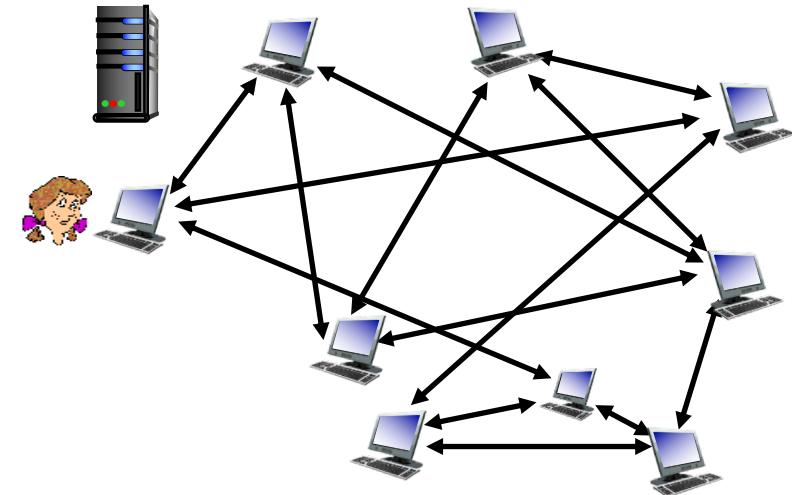
- ❖ Contains address of trackers for the file
 - Where can I find other peers?
- ❖ Contain a list of file chunks and their cryptographic hashes
 - This ensures that chunks are not modified

Title	Trackers
House of Cards Season 4	Tracker1-url
Walking Dead Season 6	Tracker2-url
Game of Thrones Season 8	Tracker2-url, Tracker3-url

P2P file distribution: BitTorrent

- ❖ peer joining torrent:
 - has no chunks, but will accumulate them over time from other peers
 - registers with tracker to get list of peers, connects to subset of peers ("neighbours")

- ❖ while downloading, peer uploads chunks to other peers
- ❖ peer may change peers with whom it exchanges chunks
 - ❖ *churn*: peers may come and go
- ❖ once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent



BitTorrent: requesting, sending file chunks

requesting chunks:

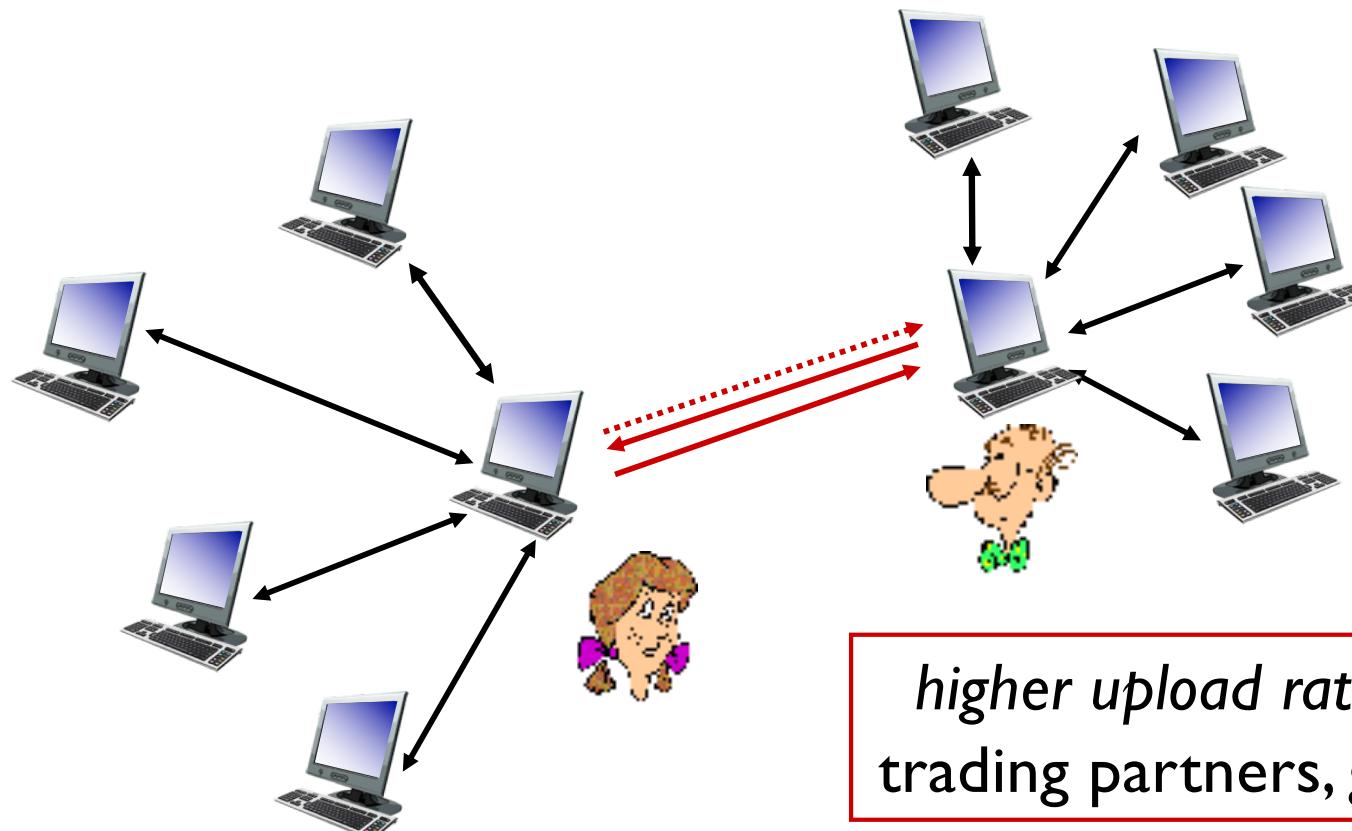
- ❖ at any given time, different peers have different subsets of file chunks
- ❖ periodically, Alice asks each peer for list of chunks that they have
- ❖ Alice requests missing chunks from peers, rarest first
- ❖ **Q:** Why rarest first?

sending chunks: tit-for-tat

- ❖ Alice sends chunks to those four peers currently sending her chunks *at highest rate*
 - other peers are choked by Alice (do not receive chunks from her)
 - re-evaluate top 4 every 10 secs
- ❖ every 30 secs: randomly select another peer, starts sending chunks
 - “optimistically unchoke” this peer
 - newly chosen peer may join top 4

BitTorrent: tit-for-tat

- (1) Alice “optimistically unchoke” Bob
- (2) Alice becomes one of Bob’s top-four providers; Bob reciprocates
- (3) Bob becomes one of Alice’s top-four providers



higher upload rate: find better trading partners, get file faster !

Quiz: Free-riding



- ❖ Suppose Todd joins a BitTorrent torrent, but he does not want to upload any data to any other peers. Todd claims that he can receive a complete copy of the file that is shared by the swarm. Is Todd's claim possible? Why or Why not?

Getting rid of the server/tracker

- ❖ Distribute the tracker information using a Distributed Hash Table (DHT)
- ❖ A DHT is a lookup structure
 - Maps keys to an arbitrary value
 - Works a lot like, well hash table

Content available in 6th Edition of the textbook Section 2.6.2

Hash table - review

- ❖ (key,value) pairs
- ❖ Centralised hash table – all (key,value) pairs on 1 node
- ❖ Distributed hash tables – each node has a “section” of (key,value) pairs

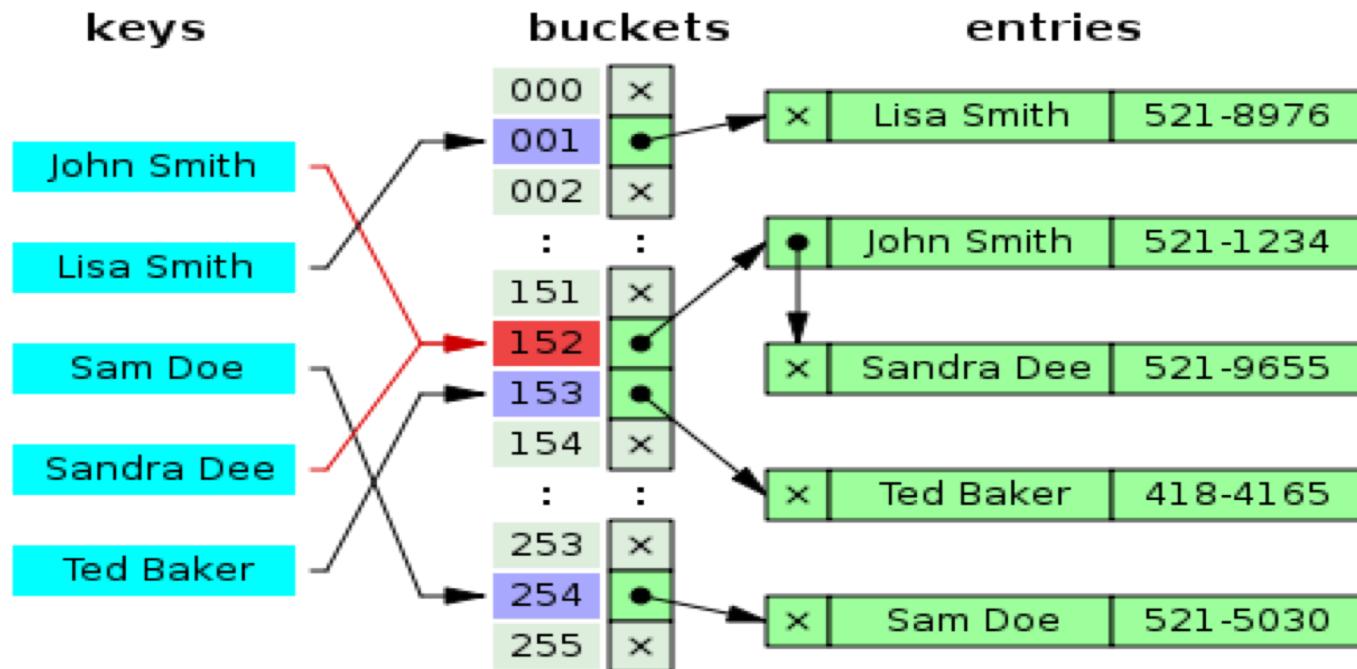


Figure src: http://en.wikipedia.org/wiki/Hash_table

Distributed Hash Table (DHT)

- ❖ DHT: a *distributed P2P database*
- ❖ database has (key, value) pairs; examples:
 - key: TFN number; value: human name
 - key: file name; value: BT tracker(s)
- ❖ Distribute the (key, value) pairs over the (millions of peers)
- ❖ a peer **queries** DHT with key
 - DHT returns values that match the key
- ❖ peers can also **insert** (key, value) pairs

Q: how to assign keys to peers?

- ❖ basic idea:
 - convert each key to an integer
 - Assign integer to each peer
 - put (key,value) pair in the peer that is **closest** to the key

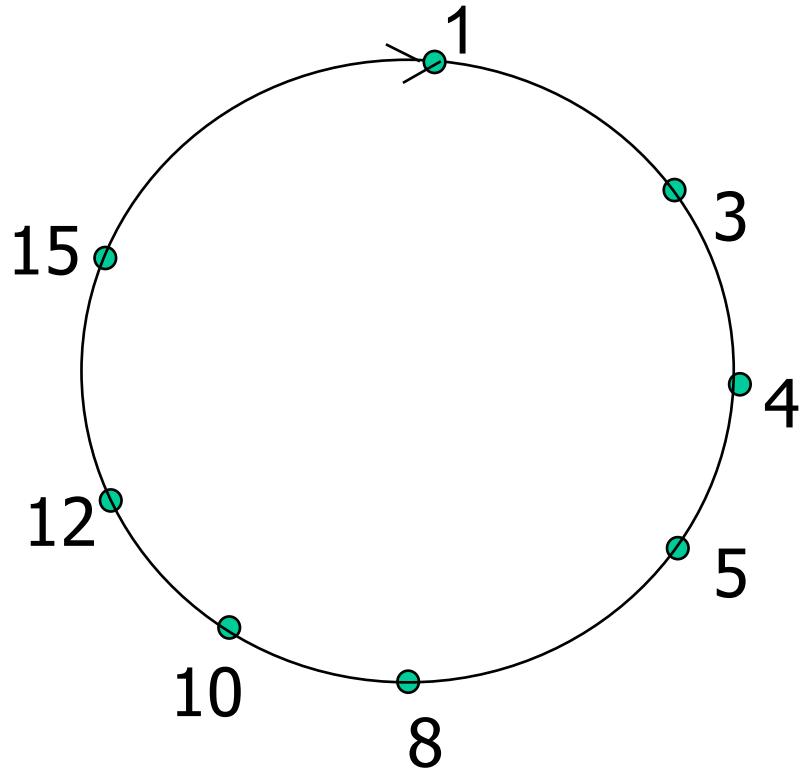
DHT identifiers: Consistent Hashing

- ❖ assign integer identifier to each peer in range $[0, 2^n - 1]$ for some n -bit hash function
 - E.g., node ID is hash of its IP address
- ❖ require each key to be an integer in same range
- ❖ to get integer key, hash original key
 - e.g., key = **hash**("House of Cards Season 4")
 - this is why it's referred to as a *distributed "hash" table*

Assign keys to peers

- ❖ rule: assign key to the peer that has the *closest* ID.
- ❖ common convention: closest is the *immediate successor* of the key.
- ❖ e.g., $n=4$; all peers & key identifiers are in the range [0-15], peers: 1,3,4,5,8,10,12,14;
 - key = 13, then successor peer = 14
 - key = 15, then successor peer = 1

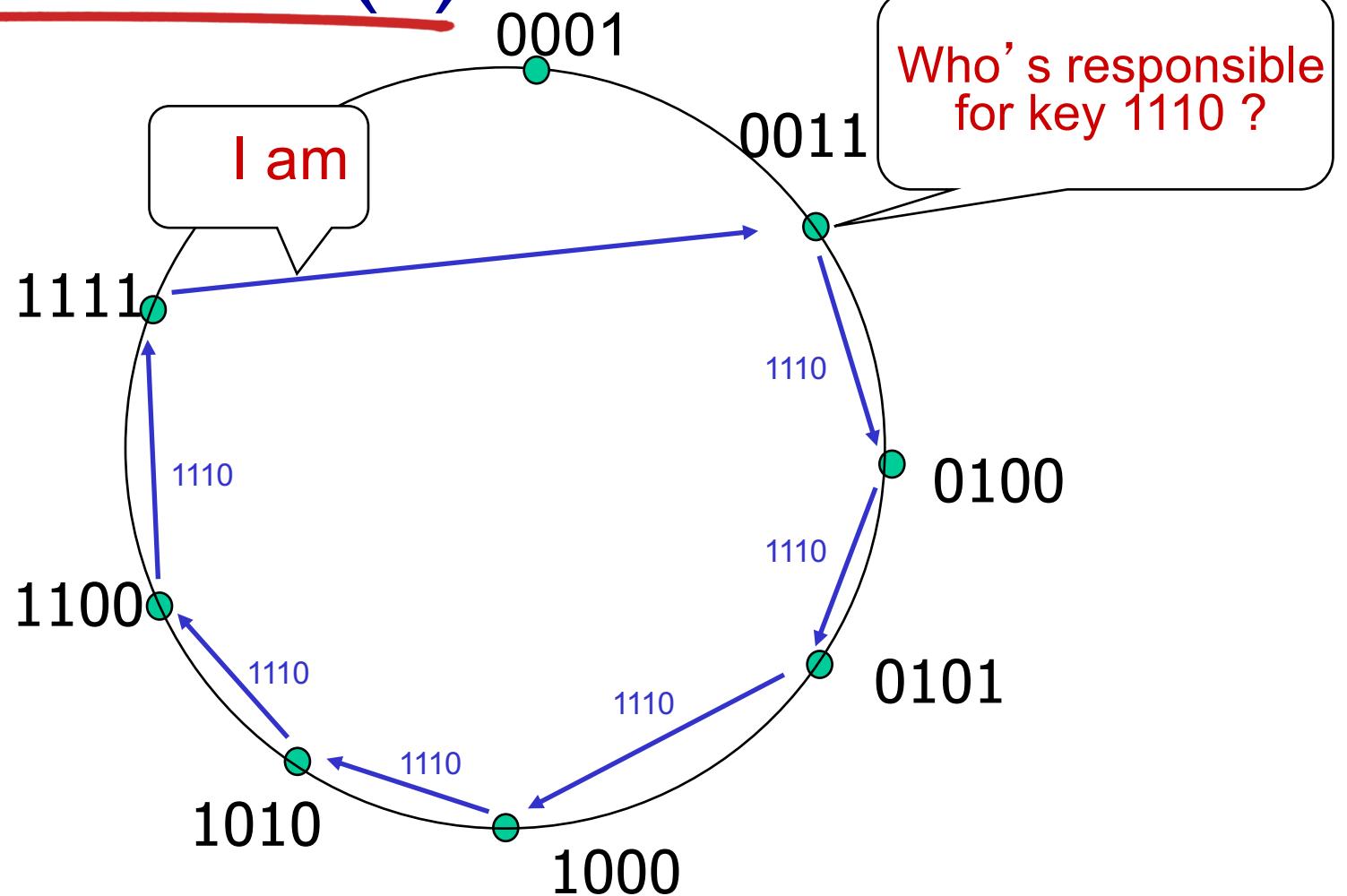
Circular DHT (I)



- ❖ each peer *only* aware of immediate successor and predecessor.
- ❖ “overlay network”

Circular DHT (2)

Define closest as closest successor



Worst case all peers probed, N messages, on average $N/2$

Mesh overlay (each peer tracks all other $N-1$ peers) only one message is sent per query

Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

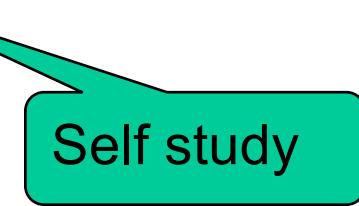
- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 **video streaming and content distribution networks (CDNs)**

2.7 socket programming with UDP and TCP



Self study

Video Streaming and CDNs: context

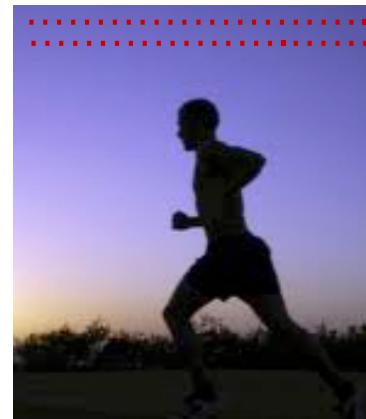
- video traffic: major consumer of Internet bandwidth
 - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
 - ~1.8B YouTube users, ~140M Netflix users
- challenge: scale - how to reach ~2B users?
 - single mega-video server won't work (why?)
- challenge: heterogeneity
 - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure



Multimedia: video

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

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



frame i

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

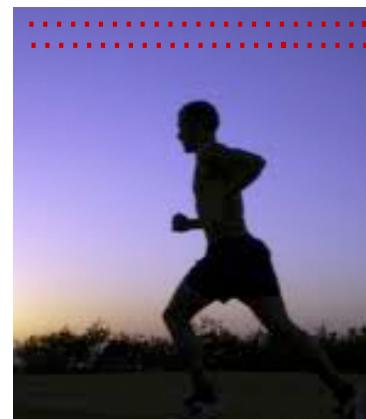


frame $i+1$

Multimedia: video

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

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



frame i

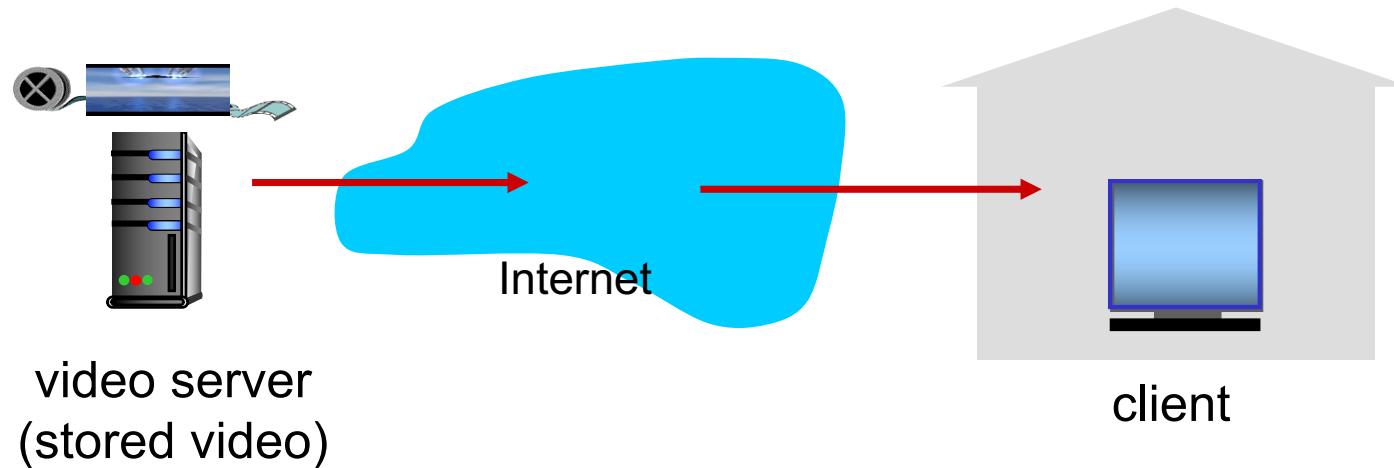
temporal coding example:
instead of sending complete frame at $i+1$,
send only differences from frame i



frame $i+1$

Streaming stored video:

simple scenario:

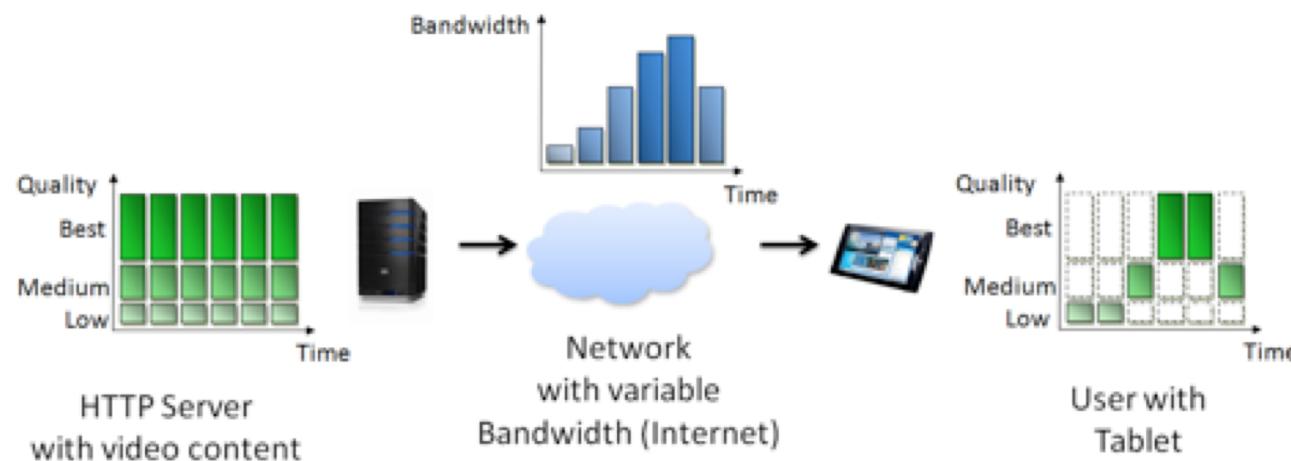


Streaming multimedia: DASH

- ❖ *DASH: Dynamic, Adaptive Streaming over HTTP*
- ❖ *server:*
 - divides video file into multiple chunks
 - each chunk stored, encoded at different rates
 - *manifest file:* provides URLs for different chunks
- ❖ *client:*
 - periodically measures server-to-client bandwidth
 - consulting manifest, requests one chunk at a time
 - chooses maximum coding rate sustainable given current bandwidth
 - can choose different coding rates at different points in time (depending on available bandwidth at time)

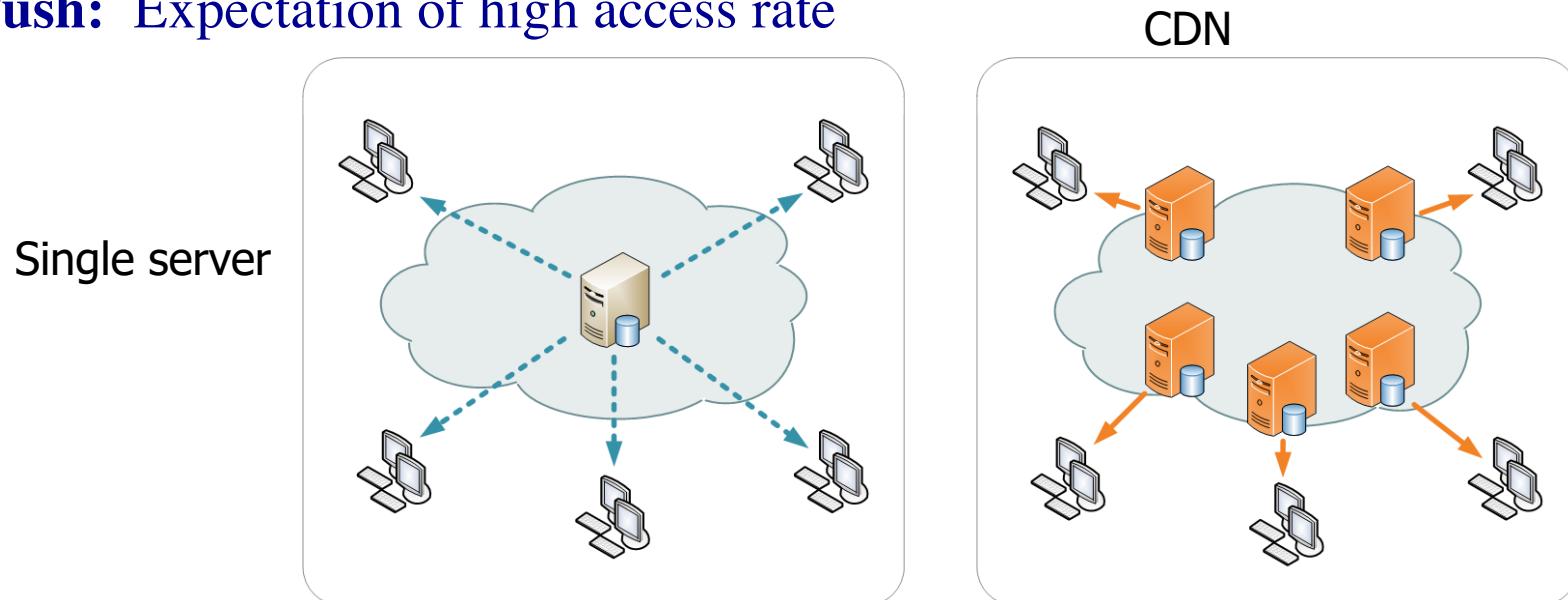
Streaming multimedia: DASH

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



Content distribution networks

- ❖ Caching and replication as a service (amortise cost of infrastructure)
- ❖ Goal: bring content close to the user
- ❖ Large-scale distributed storage infrastructure (usually) administered by one entity
 - *e.g.*, Akamai has servers in 20,000+ locations
- ❖ Combination of (pull) caching and (push) replication
 - **Pull:** Direct result of clients' requests
 - **Push:** Expectation of high access rate



An example

```
bash-3.2$ dig www.mit.edu

; <>> DiG 9.8.3-P1 <>> www.mit.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 27387
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 9, ADDITIONAL: 9

;; QUESTION SECTION:
;www.mit.edu.           IN      A

;; ANSWER SECTION:
www.mit.edu.          1800    IN      CNAME   www.mit.edu.edgekey.net,
www.mit.edu.edgekey.net. 60      IN      CNAME   e9566.dscb.akamaiedge.net,
e9566.dscb.akamaiedge.net. 20    IN      A       23.77.150.125

;; AUTHORITY SECTION:
dscb.akamaiedge.net. 681     IN      NS      n4dscb.akamaiedge.net,
dscb.akamaiedge.net. 681     IN      NS      n5dscb.akamaiedge.net,
dscb.akamaiedge.net. 681     IN      NS      a0dscb.akamaiedge.net,
dscb.akamaiedge.net. 681     IN      NS      n6dscb.akamaiedge.net,
dscb.akamaiedge.net. 681     IN      NS      n1dscb.akamaiedge.net,
dscb.akamaiedge.net. 681     IN      NS      n3dscb.akamaiedge.net,
dscb.akamaiedge.net. 681     IN      NS      n0dscb.akamaiedge.net,
dscb.akamaiedge.net. 681     IN      NS      n7dscb.akamaiedge.net,
dscb.akamaiedge.net. 681     IN      NS      n2dscb.akamaiedge.net,

;; ADDITIONAL SECTION:
a0dscb.akamaiedge.net. 7144   IN      AAAA   2600:1480:e800::c0
n0dscb.akamaiedge.net. 3048   IN      A       88.221.81.193
n1dscb.akamaiedge.net. 2752   IN      A       88.221.81.194
n2dscb.akamaiedge.net. 1380   IN      A       104.72.70.167
n3dscb.akamaiedge.net. 3048   IN      A       88.221.81.195
n4dscb.akamaiedge.net. 2810   IN      A       104.71.131.100
n5dscb.akamaiedge.net. 1326   IN      A       104.72.70.166
n6dscb.akamaiedge.net. 49     IN      A       104.72.70.174
n7dscb.akamaiedge.net. 2554   IN      A       104.72.70.175

;; Query time: 246 msec
;; SERVER: 129.94.172.11#53(129.94.172.11)
;; WHEN: Thu Mar  9 18:04:37 2017
;; MSG SIZE  rcvd: 463
```

Many well-known sites
are hosted by CDNs. A
simple way to check
using dig is shown here.

Content distribution networks

- *challenge*: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?
- *option 1*: single, large “mega-server”
 - single point of failure
 - point of network congestion
 - long path to distant clients
 - multiple copies of video sent over outgoing link

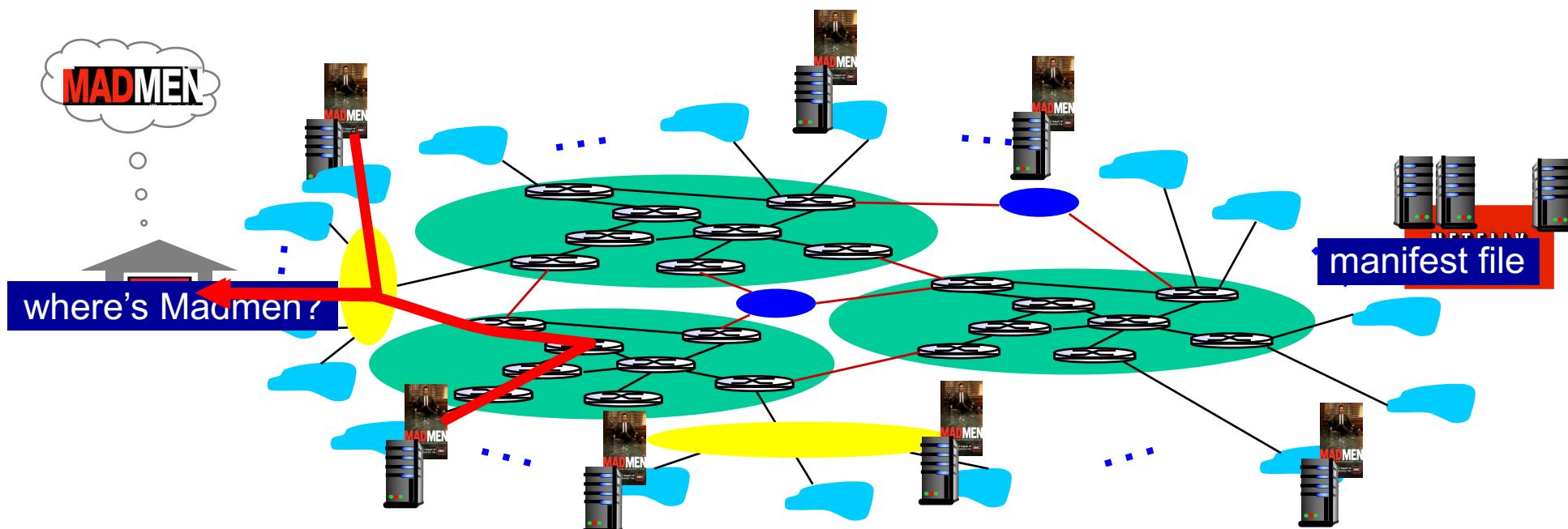
....quite simply: this solution *doesn't scale*

Content distribution networks

- ❖ *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
 - used by Akamai, thousands of locations
 - *bring home*: smaller number (10's) of larger clusters in POPs near (but not within) access networks
 - used by Limelight

Content Distribution Networks (CDNs)

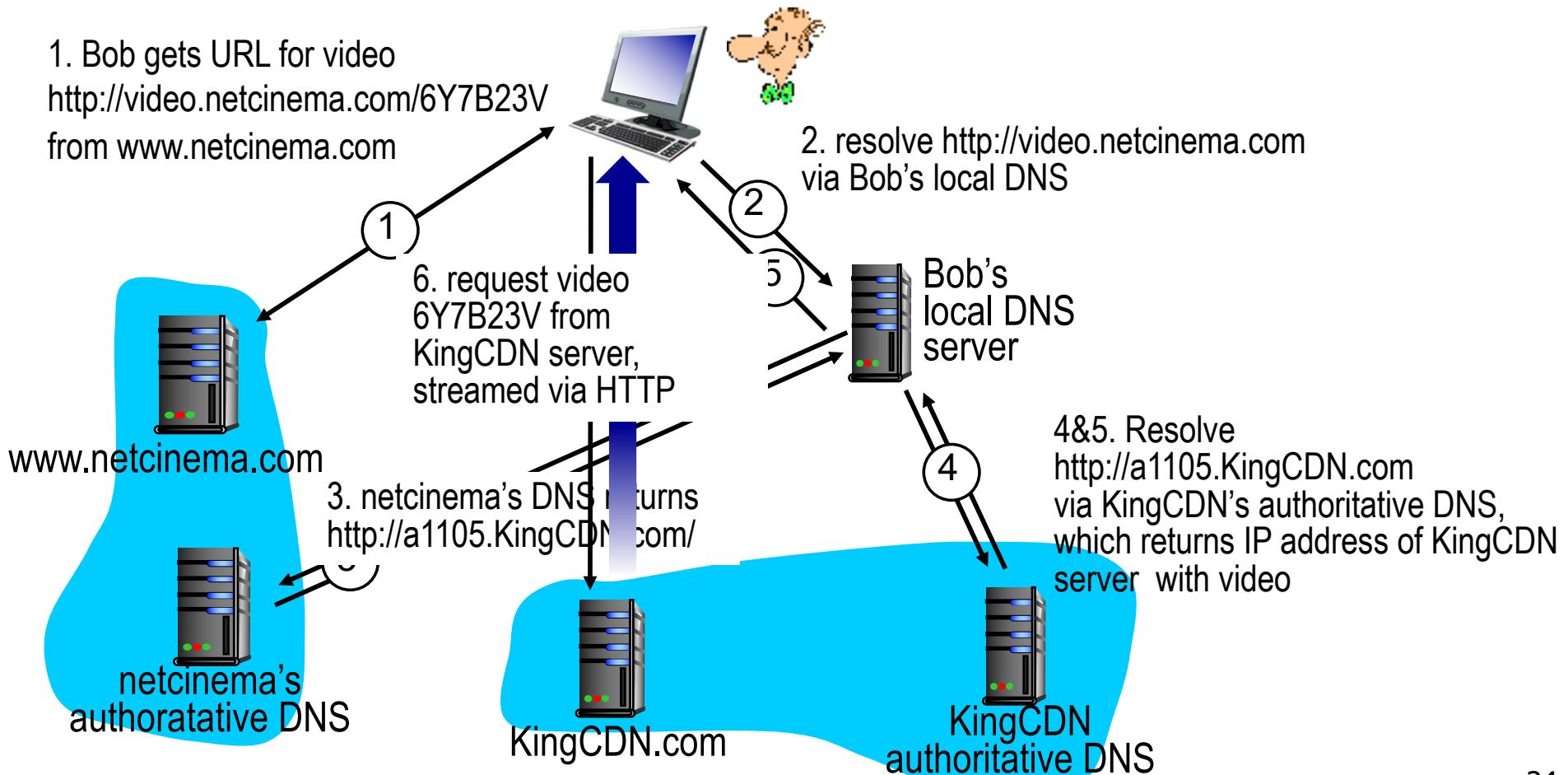
- CDN: stores copies of content at CDN nodes
 - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
 - directed to nearby copy, retrieves content
 - may choose different copy if network path congested



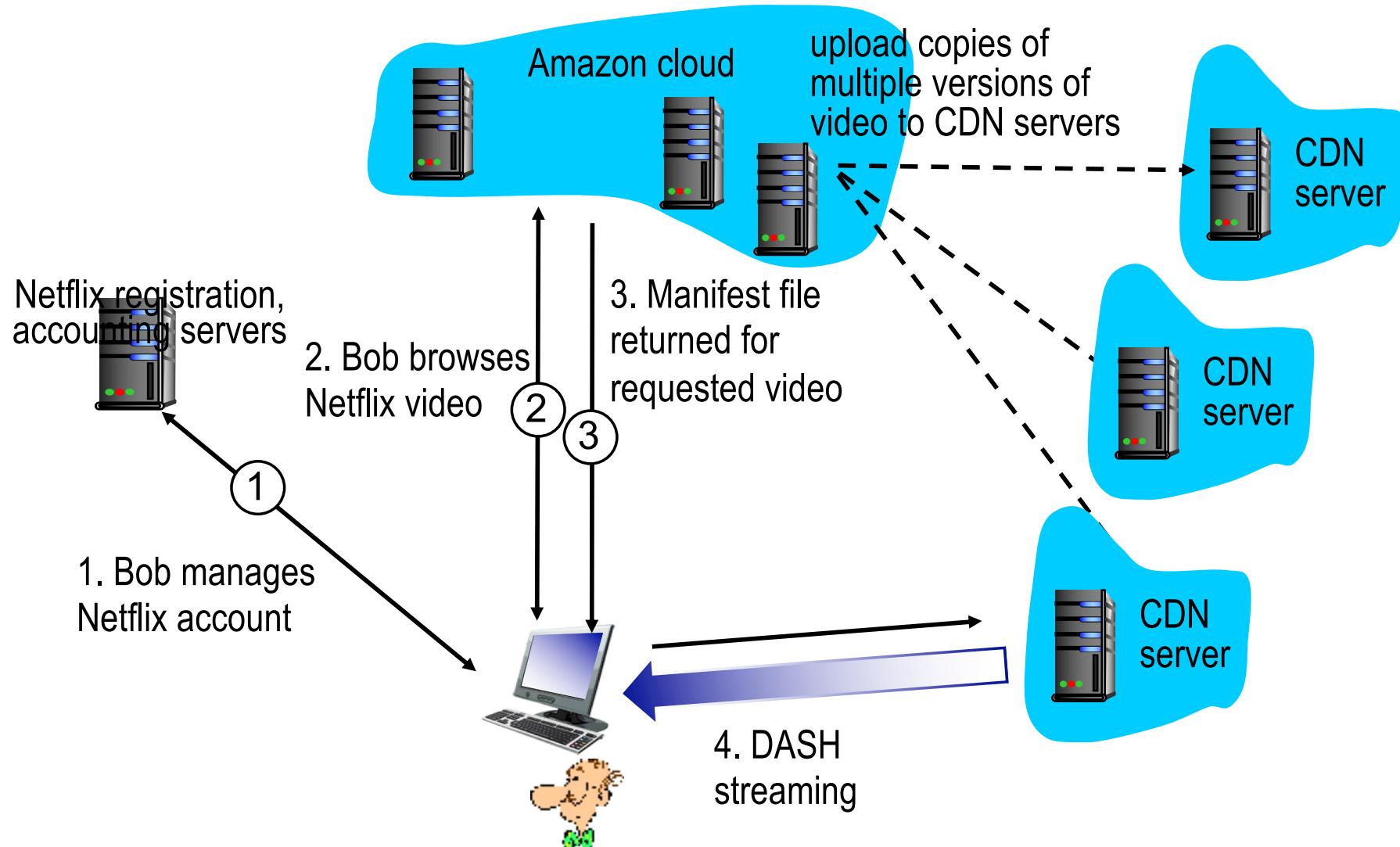
CDN content access: a closer look

Bob (client) requests video <http://video.netcinema.com/6Y7B23V>

- video stored in CDN at managed by KingCDN.com

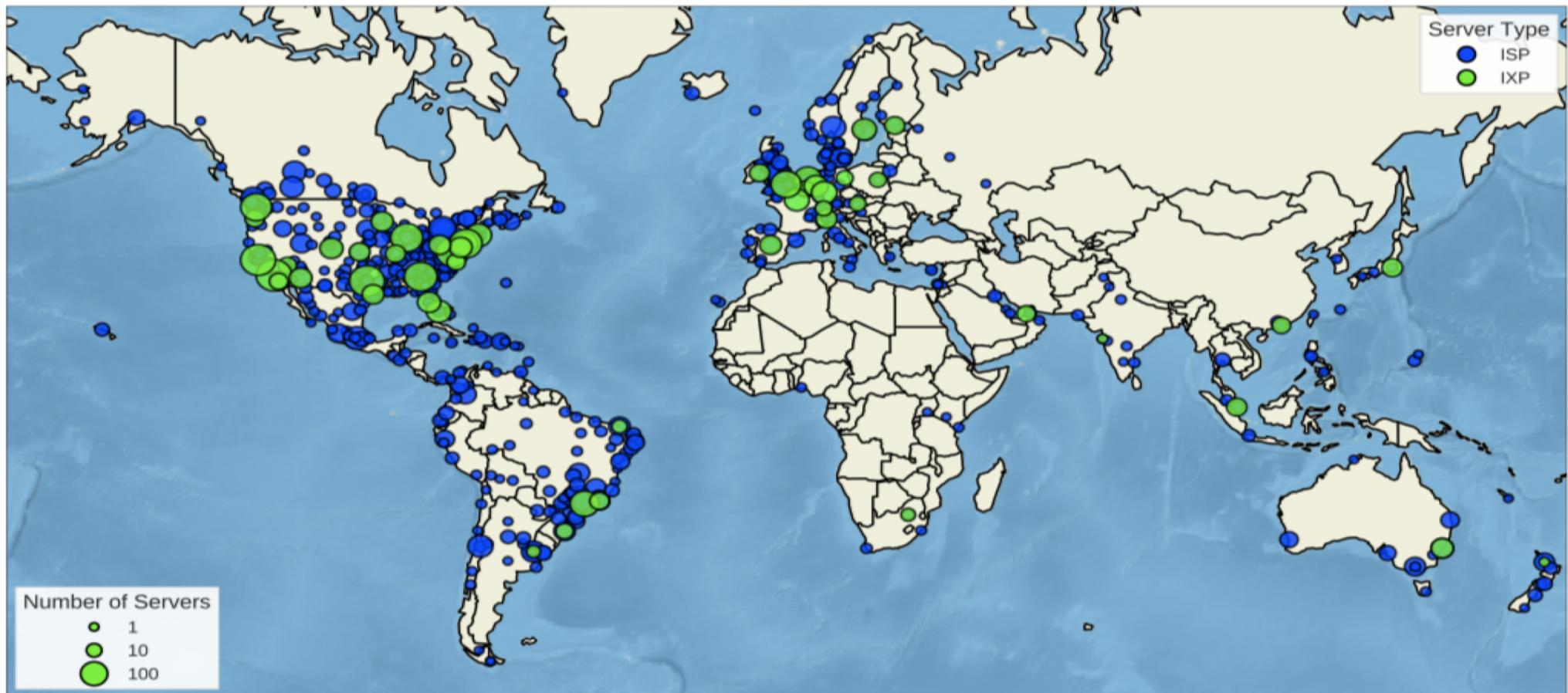


Case study: Netflix



Uses Push caching (during offpeak)
Preference to "deep inside" followed by "bring home"

NetFlix servers (snap shot from Jan 2018)



Researchers from Queen Mary University of London (QMUL) traced server names that are sent to a user's computer every time they play content on Netflix to find the location of the 8492 servers (4152 ISP, 4340 IXP). They have been found to be scattered across 578 locations around the world.



Quiz: CDN

- ❖ The role of the CDN provider's authoritative DNS name server in a content distribution network, simply described, is:
 - a) to provide an alias address for each browser access to the “origin server” of a CDN website
 - b) to map the query for each CDN object to the CDN server closest to the requestor (browser)
 - c) to provide a mechanism for CDN “origin servers” to provide paths for clients (browsers)
 - d) none of the above, CDN networks do not use DNS