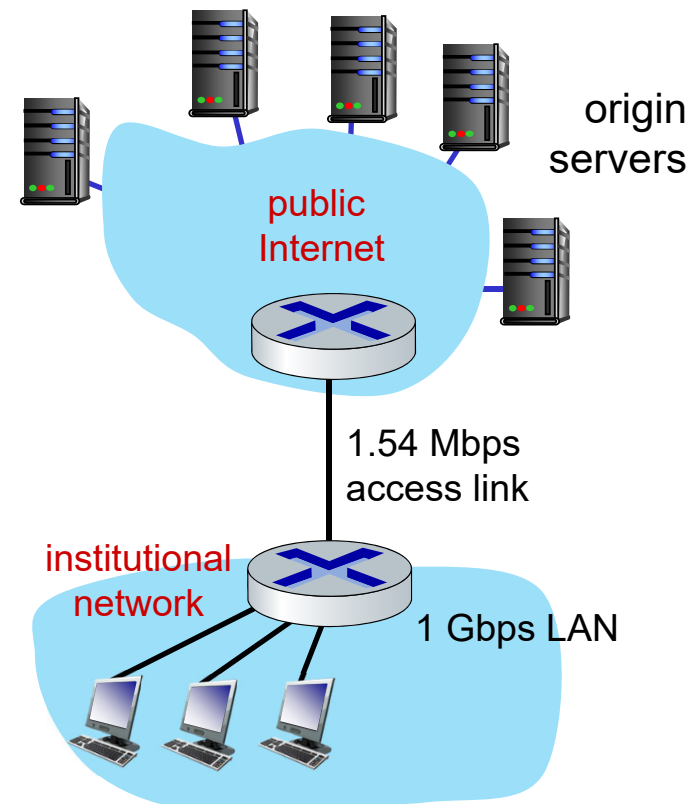# Caching example

Scenario:

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

Performance:

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

problem: large delays at high utilization!

origin servers

public Internet

1.54 Mbps access link

institutional network

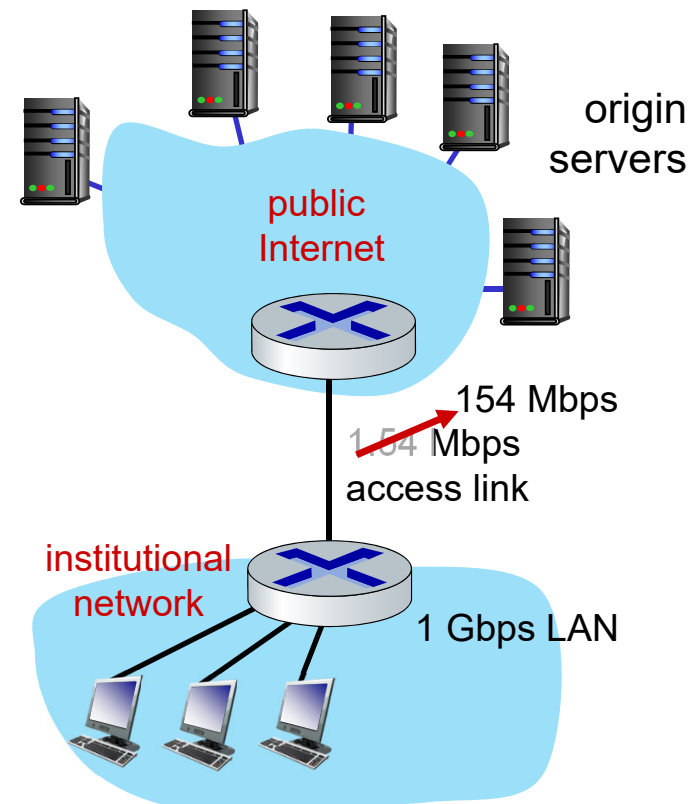1 Gbps LAN

Caching example: buy a faster access link

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

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

Cost: faster access link (expensive!)

origin servers

public Internet

~~1.54 Mbps~~ → 154 Mbps access link

institutional network

1 Gbps LAN

Caching example: install a web cache

Scenario:

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

Performance:

- LAN utilization: .?
- access link utilization = ?
- average end-end delay = ?

How to compute link utilization, delay?

Cost: web cache (cheap!)

origin servers

public Internet

1.54 Mbps access link

institutional network
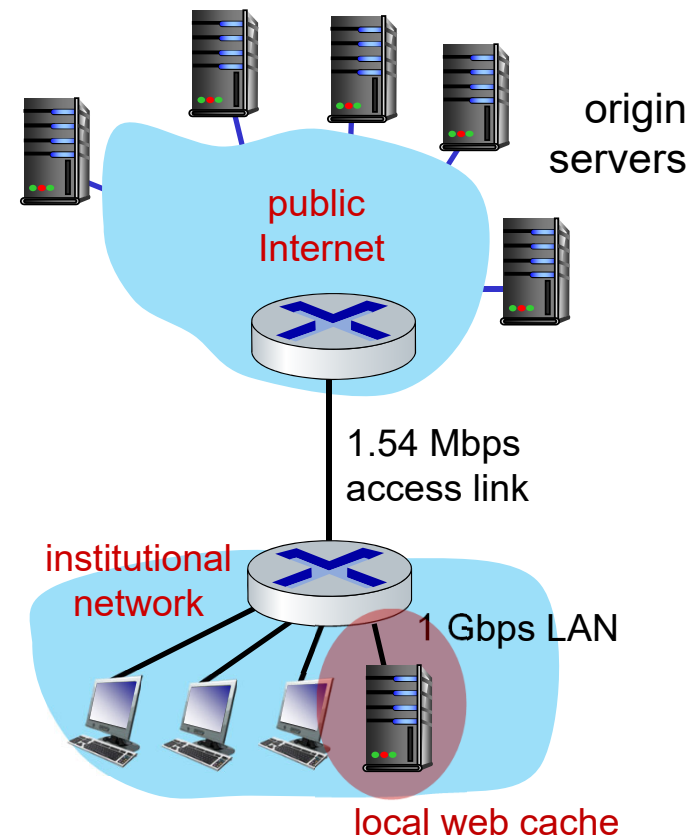
1 Gbps LAN

local web cache
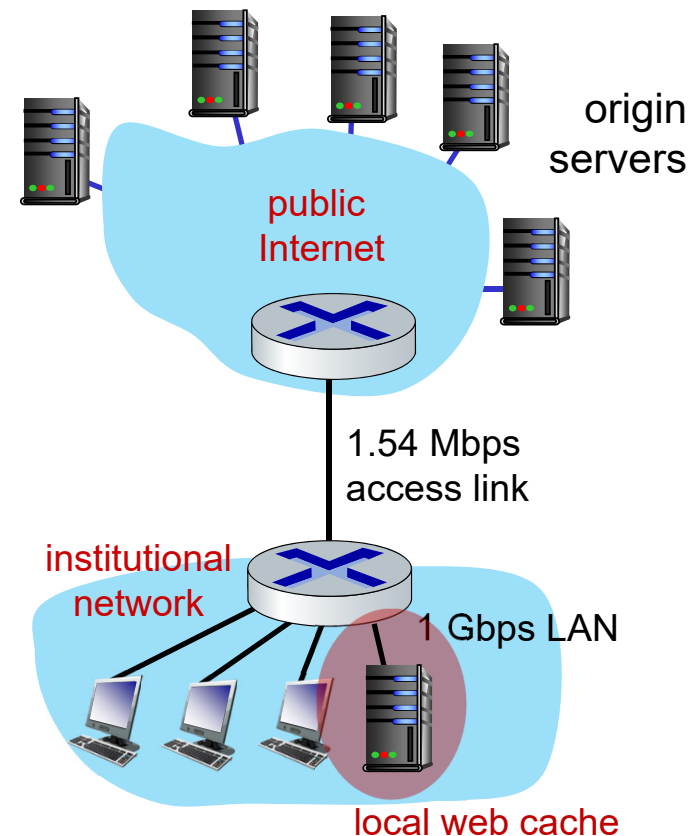
Caching example: install a web cache

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

- suppose cache hit rate is 0.4: 40% requests satisfied at cache, 60% requests satisfied at origin
- access link: 60% of requests use access link
- data rate to browsers over access link
  = 0.6 * 1.50 Mbps = .9 Mbps
- utilization = 0.9/1.54 = .58

- average end-end delay
  = 0.6 * (delay from origin servers)
       + 0.4 * (delay when satisfied at cache)
  = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs

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

origin servers

public Internet

1.54 Mbps access link

institutional network
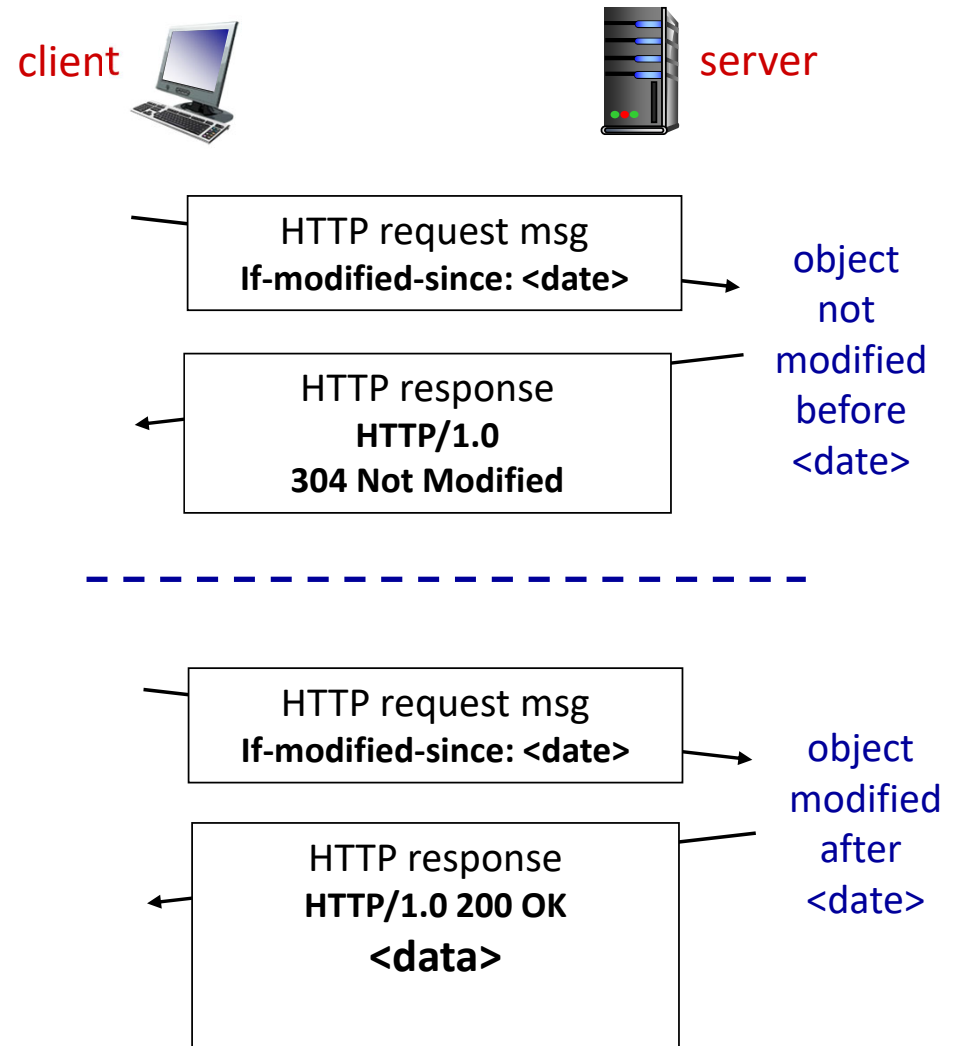
1 Gbps LAN

local web cache

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>

object not modified before <date>

HTTP response
**HTTP/1.0
304 Not Modified**

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

object modified after <date>

HTTP response
**HTTP/1.0 200 OK
<data>**

HTTP/2

Key goal: decreased delay in multi-object HTTP requests

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

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (head-of-line (HOL) blocking) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission
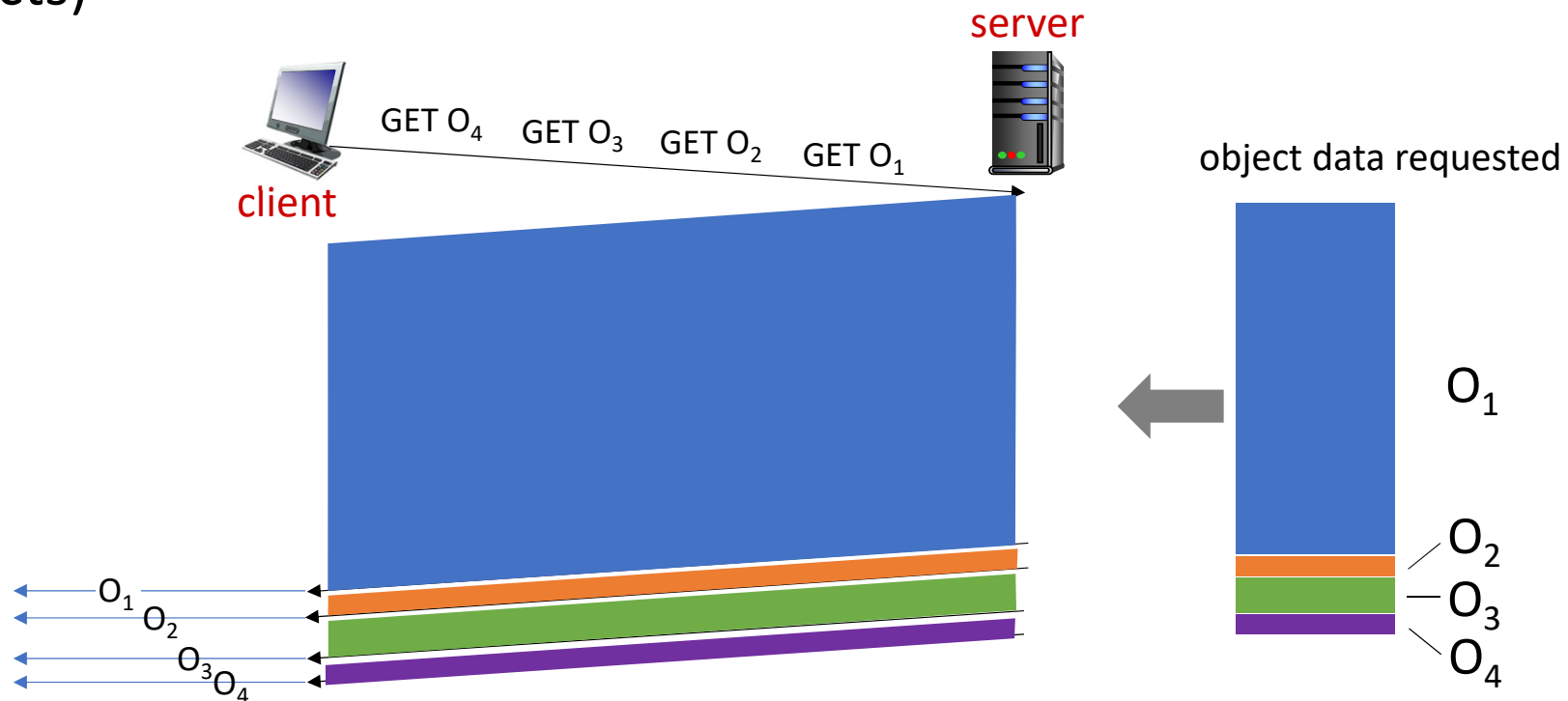
HTTP/2

Key goal: decreased delay in multi-object HTTP requests

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

- methods, status codes, most header fields unchanged from HTTP 1.1
- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
- *push* unrequested objects to client
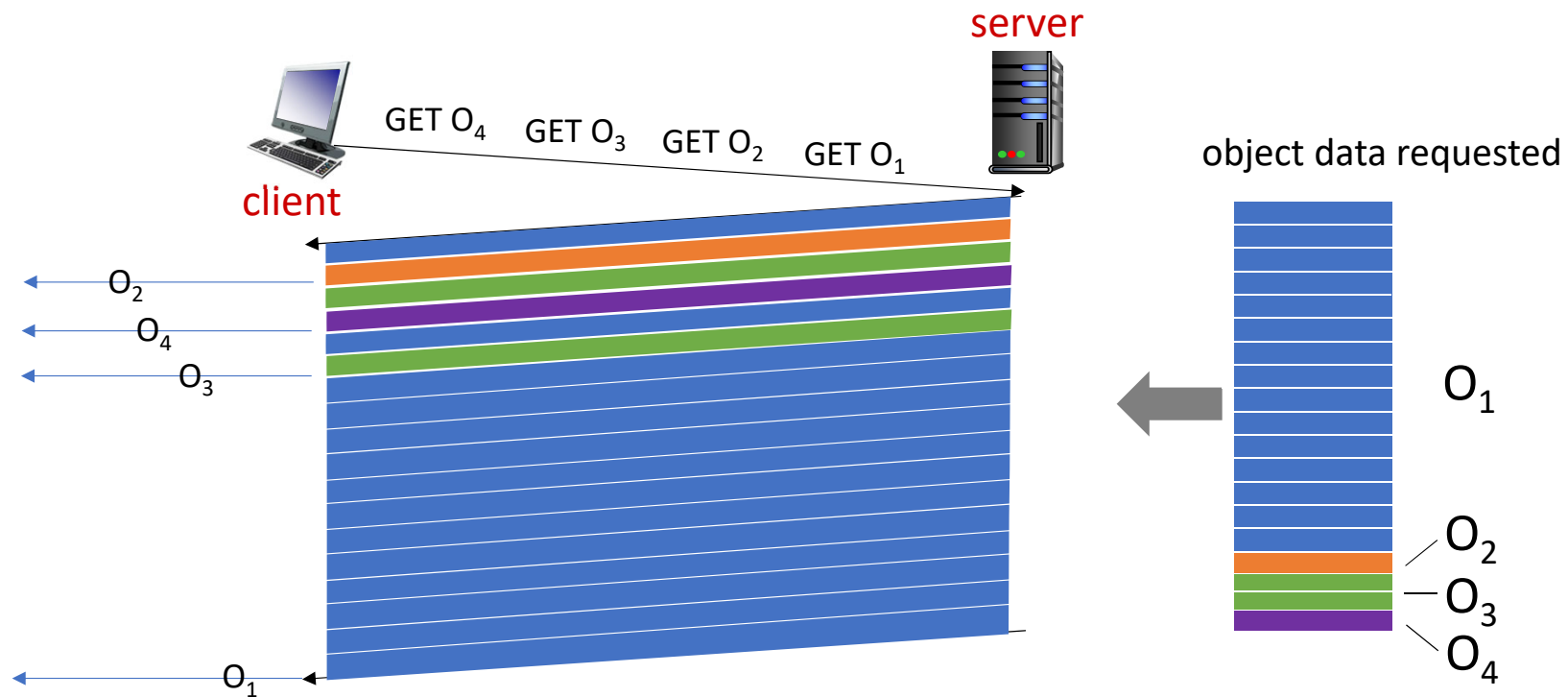- divide objects into frames, schedule frames to mitigate HOL blocking

HTTP/2: mitigating HOL blocking

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

objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

HTTP/2: mitigating HOL blocking

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

O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

HTTP/2 to HTTP/3

Key goal: decreased delay in multi-object HTTP requests

HTTP/2 over single TCP connection means:

- recovery from packet loss still stalls all object transmissions
  - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput

- no security over vanilla TCP connection

- HTTP/3: adds security , per object error- and congestion-control (more pipelining) over UDP
  - more on HTTP/3 in transport layer

Application layer: overview

- Principles of network applications

- Web and HTTP

- **E-mail, SMTP, IMAP**

- The Domain Name System DNS

- P2P applications

- video streaming and content distribution networks

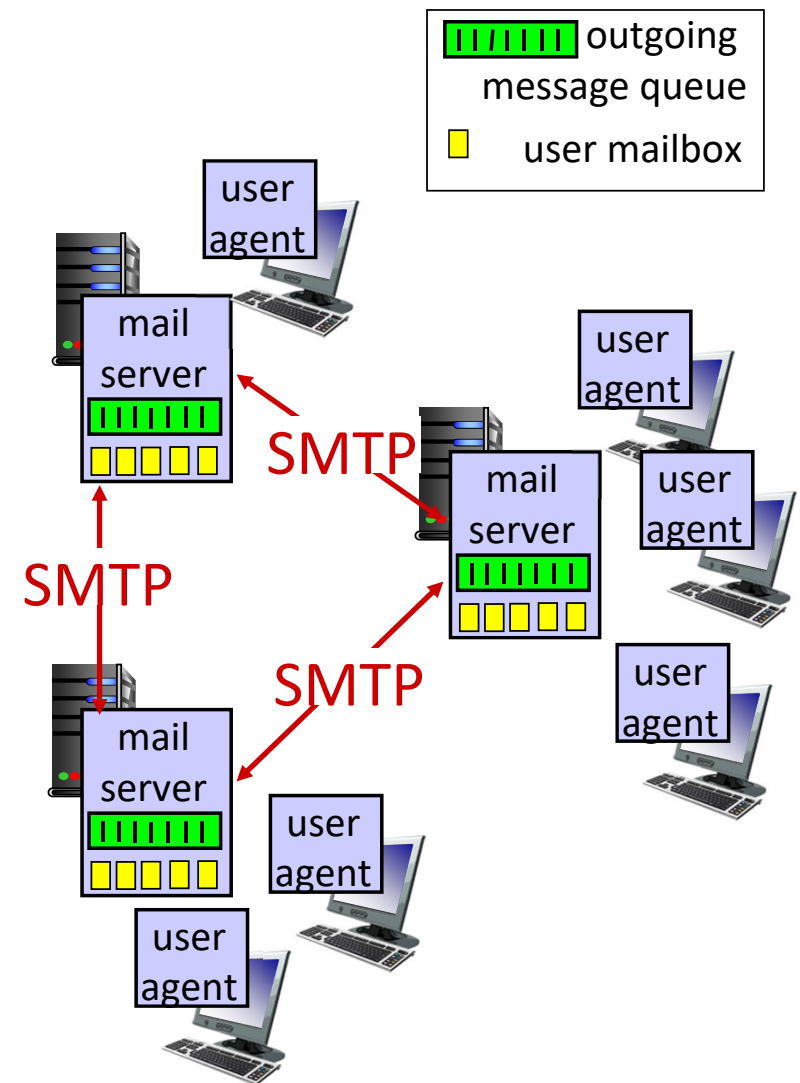- socket programming with UDP and TCP

E-mail

Three major components:

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

User Agent

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

outgoing message queue

user mailbox

E-mail: mail servers

mail servers:

▪ *mailbox* contains incoming messages for user

▪ *message queue* of outgoing (to be sent) mail messages

▪ *SMTP protocol* between mail servers to send email messages
  • client: sending mail server
 • "server": receiving mail server
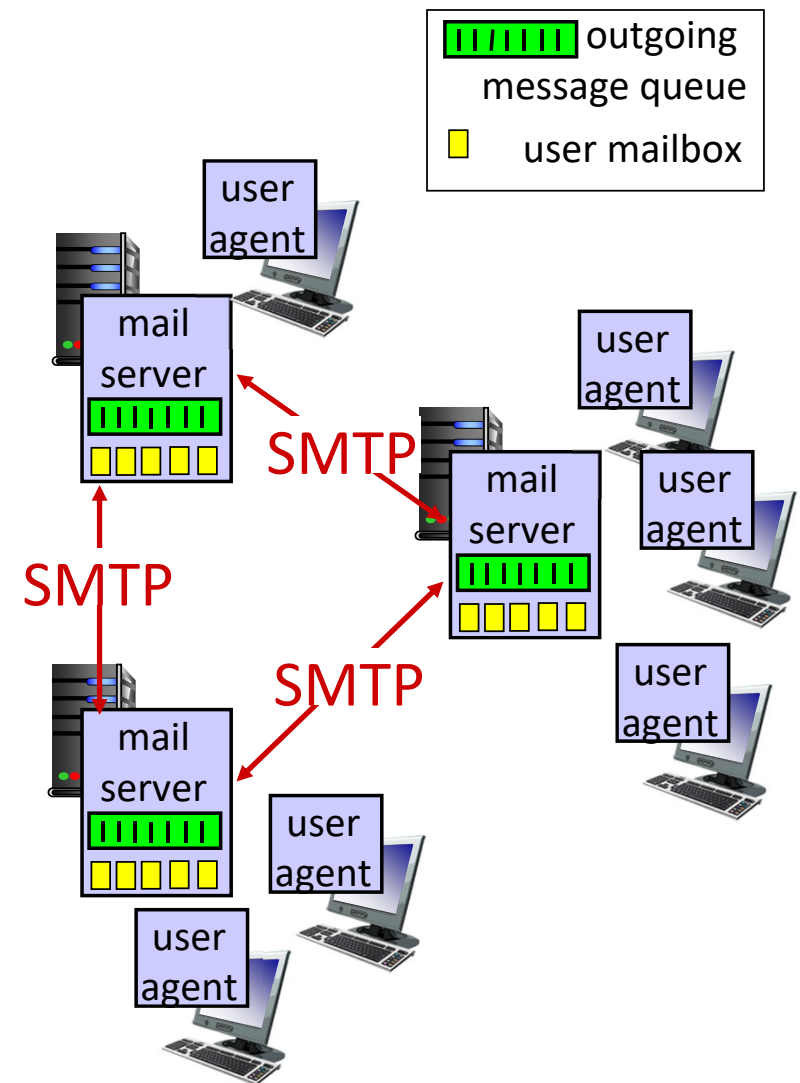
E-mail: the RFC (5321)

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

Scenario: Alice sends e-mail to Bob

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

2) Alice's UA sends message to her mail server; 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

Alice's mail server   Bob's mail server
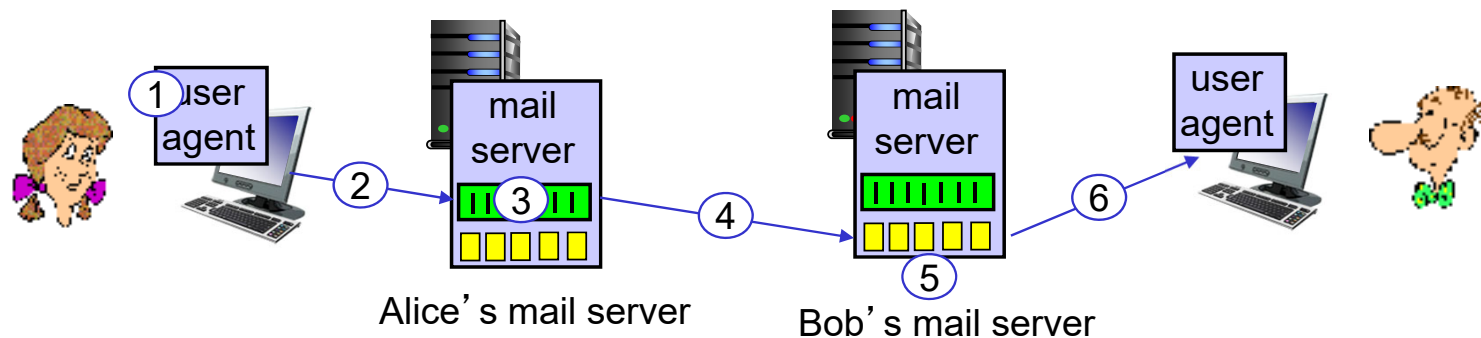
Sample SMTP interaction

```
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
```

Try SMTP interaction for yourself:

telnet <servername> 25

- see 220 reply from server

- enter HELO, MAIL FROM:, RCPT TO:, DATA, QUIT commands

above lets you send email without using e-mail client (reader)

Note: this will only work if <servername> allows telnet connections to port 25 (this is becoming increasingly rare because of security concerns)

SMTP: closing observations

comparison with HTTP:

- HTTP: pull

- SMTP: push

- both have ASCII command/response interaction, status codes

- HTTP: each object encapsulated in its own response message

- SMTP: multiple objects sent in multipart message

- 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

# Mail message format

SMTP: protocol for exchanging e-mail messages, defined in RFC 531 (like HTTP)

RFC 822 defines *syntax* for e-mail message itself (like HTML)

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  these lines, within the body of the email message area different from SMTP MAIL FROM:, RCPT TO: commands!

- Body: the "message" , ASCII characters only

header

blank line

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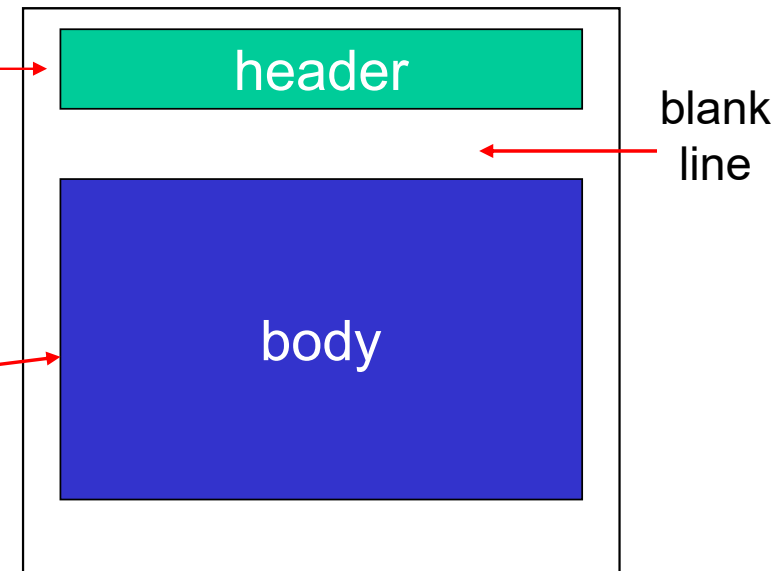
Mail access protocols
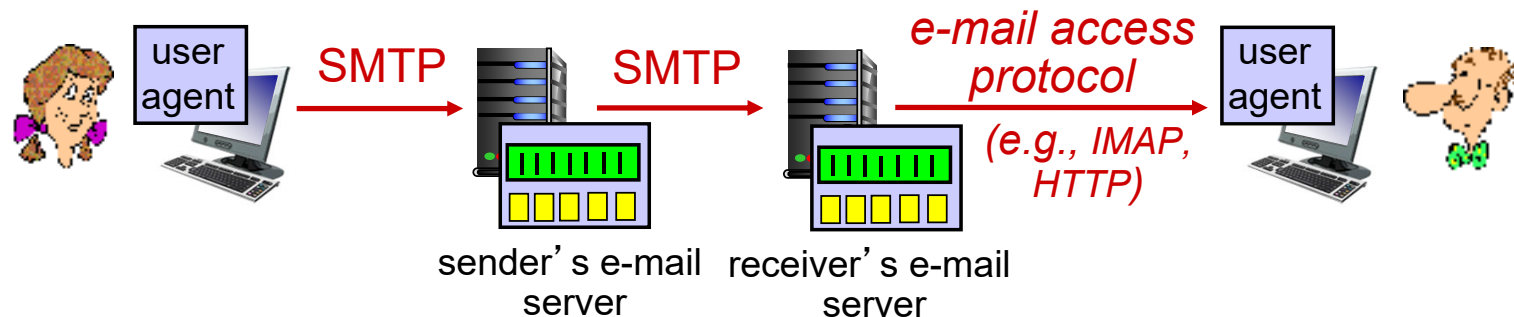
- SMTP: delivery/storage of e-mail messages to receiver's server

- mail access protocol: retrieval from server
  - IMAP: Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server

- HTTP: gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of STMP (to send), IMAP (or POP) to retrieve e-mail messages

Application Layer: Overview

- Principles of network applications

- Web and HTTP

- E-mail, SMTP, IMAP

- **The Domain Name System DNS**

- P2P applications

- video streaming and content distribution networks

- socket programming with UDP and TCP

DNS: Domain Name System

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

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., cs.umass.edu - used by humans

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

Domain Name System:
- *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: services, structure

DNS services

- hostname to IP address translation

- host aliasing
  - canonical, alias names

- mail server aliasing

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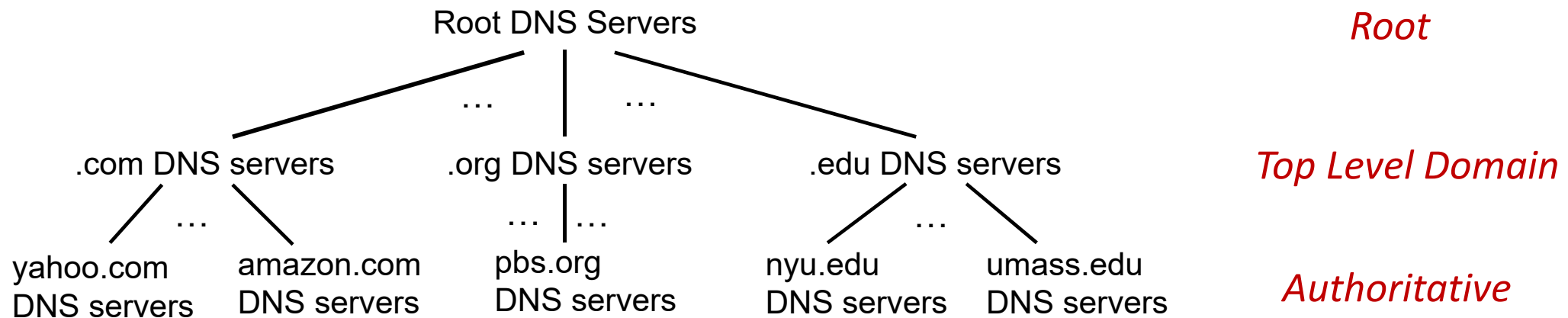 - replicated Web servers: many IP addresses correspond to one name

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 per day

DNS: a distributed, hierarchical database

Root DNS Servers   *Root*

.com DNS servers   .org DNS servers   .edu DNS servers   *Top Level Domain*

yahoo.com   amazon.com   pbs.org   nyu.edu   umass.edu   *Authoritative*
DNS servers   DNS servers   DNS servers   DNS servers   DNS servers

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

- *incredibly important* Internet function

  - Internet couldn't function without it!

  - DNSSEC – provides security (authentication and message integrity)

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

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

Key:
- ☐ 0 Servers
- ☐ 1–10 Servers
- ☐ 11–20 Servers
- ☐ 21+ Servers
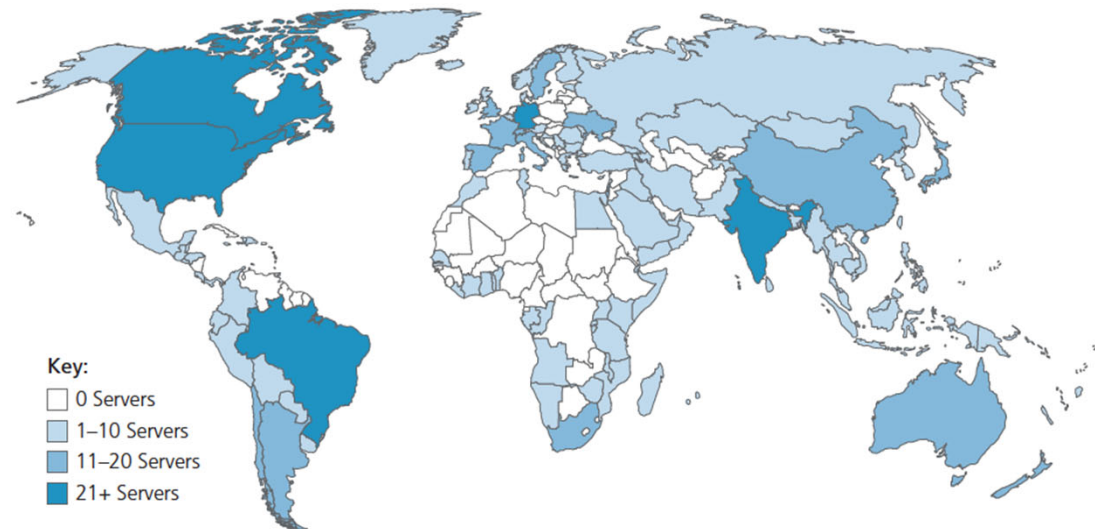
TLD: authoritative servers

Top-Level Domain (TLD) servers:

- responsible for .com, .org, .net, .edu, .aero, .jobs, .museums, and all top-level country domains, e.g.: .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
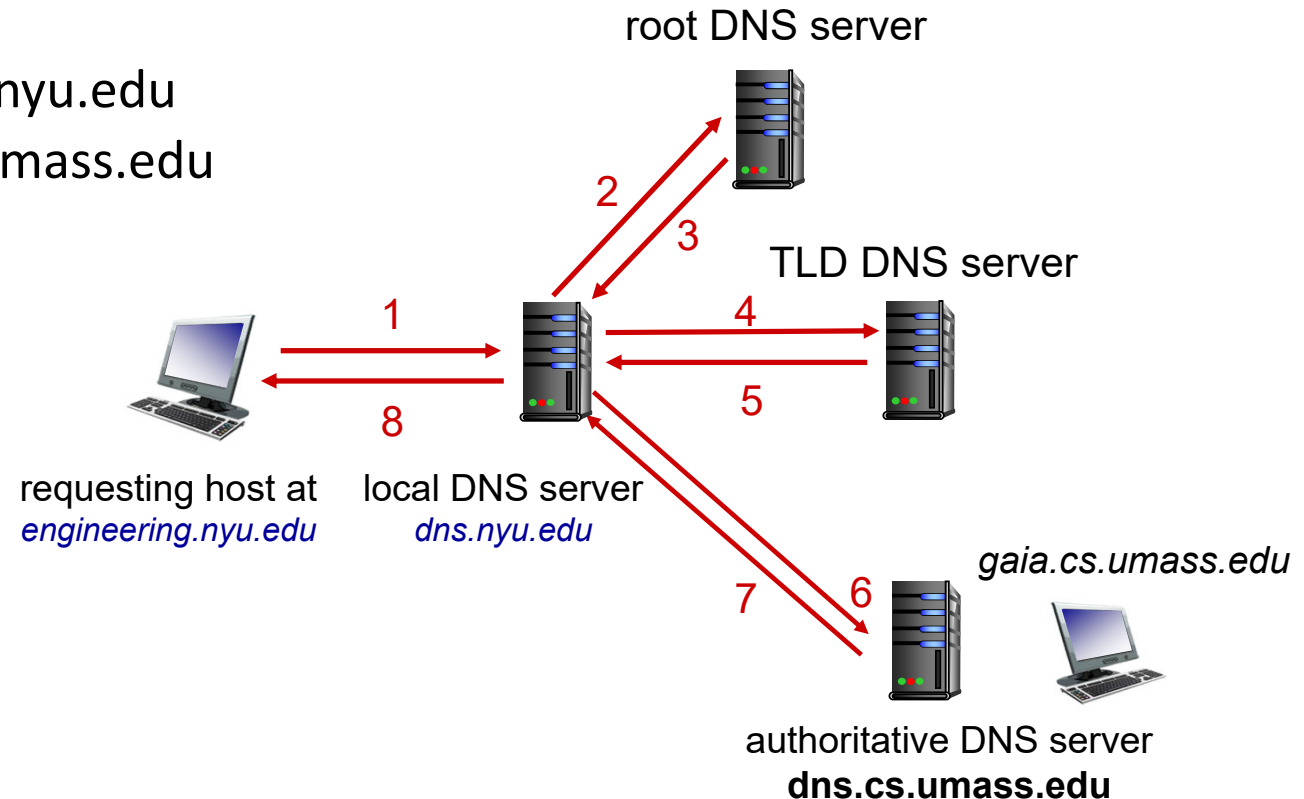
Local DNS name servers

- does not strictly belong to hierarchy

- each ISP (residential ISP, company, university) has one
  - also called "default name 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: iterated query

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

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

root DNS server

2

3

TLD DNS server

1

4

8

5

requesting host at
engineering.nyu.edu

local DNS server
dns.nyu.edu

gaia.cs.umass.edu
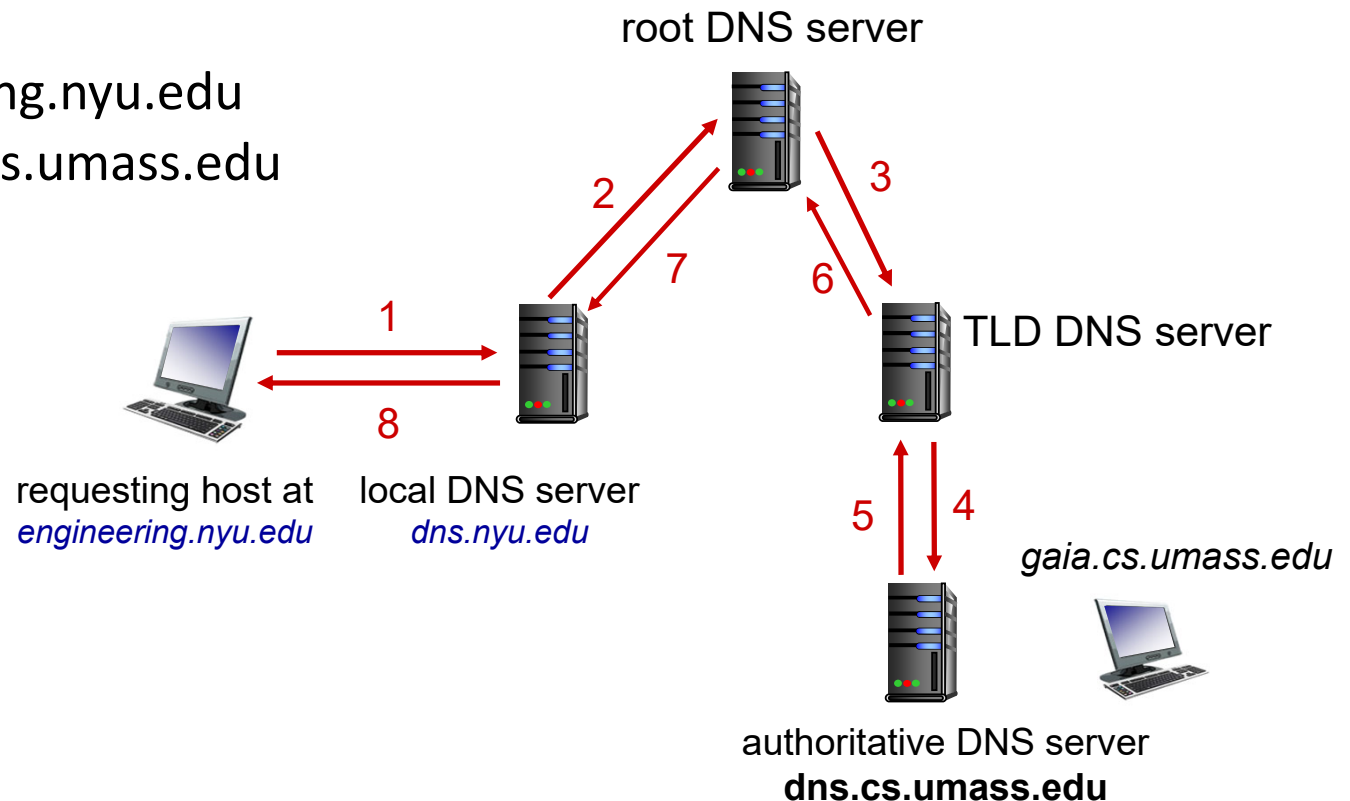
7

6

authoritative DNS server
dns.cs.umass.edu

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?

root DNS server

2

3

7

6

1

TLD DNS server

8

5

4

requesting host at
engineering.nyu.edu

local DNS server
dns.nyu.edu

gaia.cs.umass.edu

authoritative DNS server
dns.cs.umass.edu

Caching, Updating DNS 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
- 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!
- update/notify mechanisms proposed IETF standard
  - RFC 2136

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
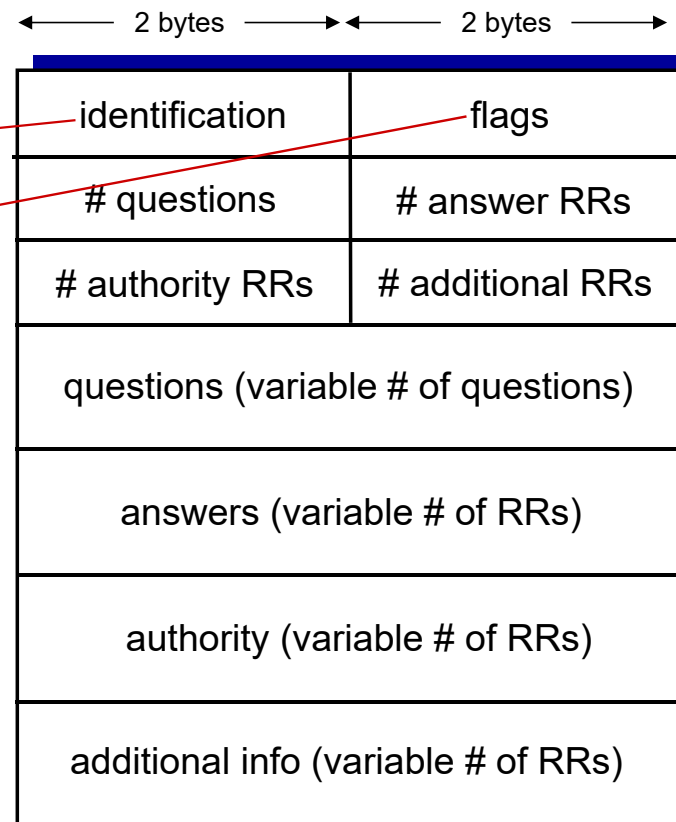- value is canonical name

type=MX
- value is name of mailserver associated with name

DNS protocol messages

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

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

|← 2 bytes →|← 2 bytes →|

identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs

questions (variable # of questions)

answers (variable # of RRs)

authority (variable # of RRs)

additional info (variable # of RRs)

DNS protocol messages

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

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

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

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

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

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

Inserting records into 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

DNS security

DDoS attacks

- bombard root servers with traffic
  - not successful to date
  - traffic filtering
  - local DNS servers cache IPs of TLD servers, allowing root server bypass
- bombard TLD servers
  - potentially more dangerous

Redirect attacks

- man-in-middle
  - intercept DNS queries
- DNS poisoning
  - send bogus relies to DNS server, which caches

Exploit DNS for DDoS

- send queries with spoofed source address: target IP
- requires amplification
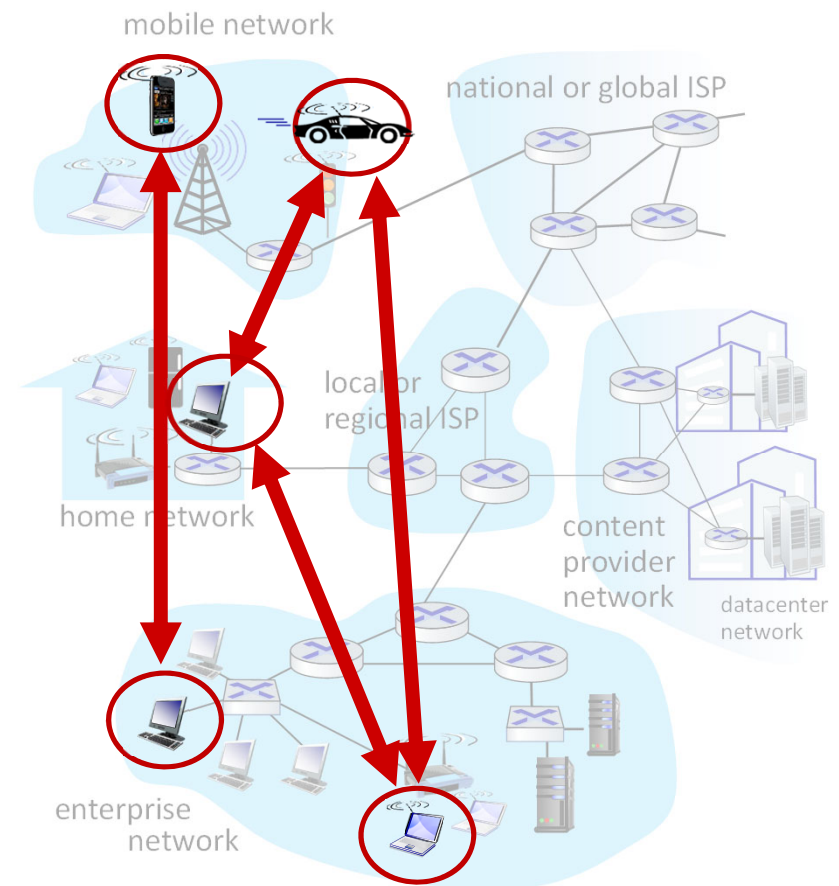
DNSSEC
[RFC 4033]

Application Layer: Overview

- Principles of network applications

- Web and HTTP

- E-mail, SMTP, IMAP

- The Domain Name System DNS

- **P2P applications**

- video streaming and content distribution networks

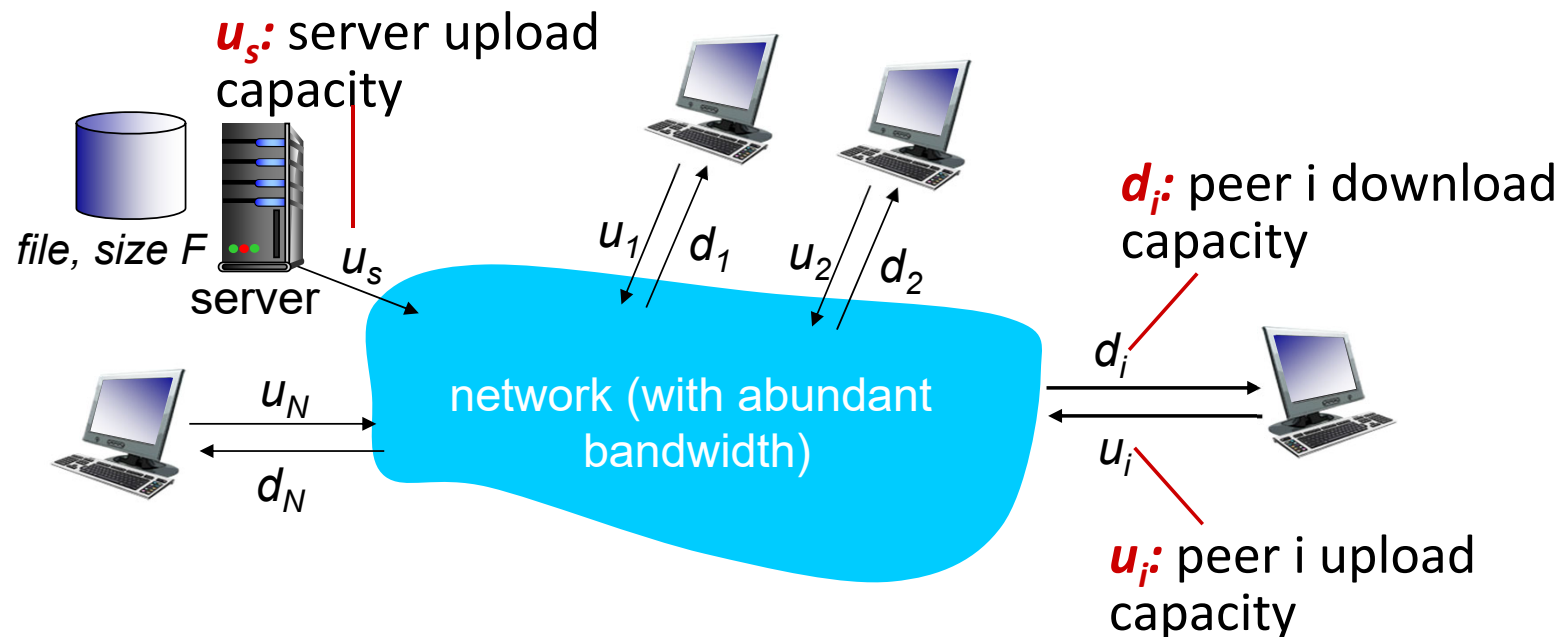- socket programming with UDP and TCP

Peer-to-peer (P2P) architecture

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

mobile network

national or global ISP

local or regional ISP

home network

content provider network

datacenter network

enterprise network

File distribution: client-server vs P2P

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

- peer upload/download capacity is limited resource

u_s: server upload capacity

file, size F

server

u_s

u_1 d_1 u_2 d_2

network (with abundant bandwidth)

u_N

d_N

d_i: peer i download capacity

d_i

u_i

u_i: peer i upload capacity

File distribution time: client-server

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

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



u_s

d_i

network

u_i
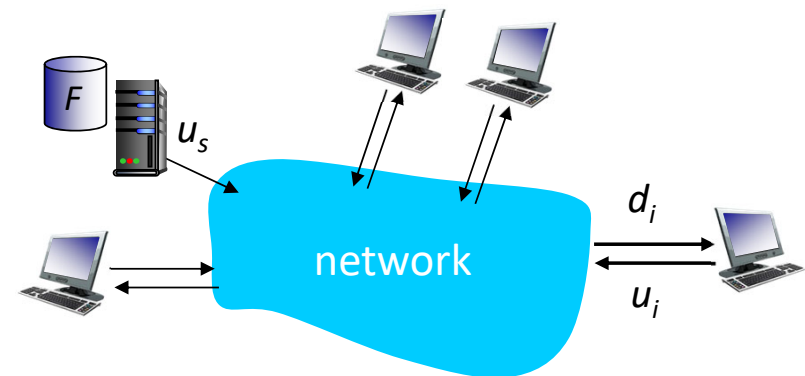
time to distribute F to N clients using client-server approach
$$D_{c-s} \geq max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

File distribution time: P2P

- *server transmission:* must upload at least one copy:
  - time to send one copy: F/u_s

- *client:* each client must download file copy
  - min client download time: F/d_{min}

- *clients:* as aggregate must download NF bits
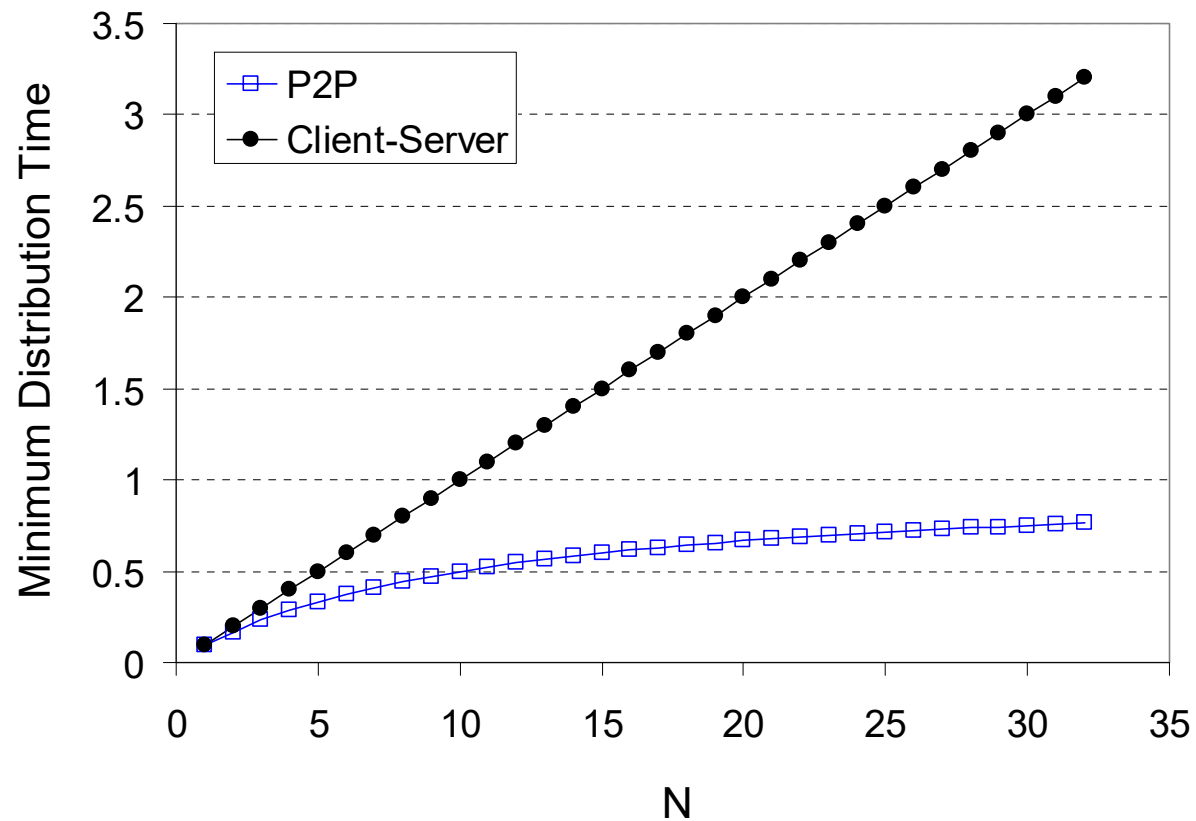 - max upload rate (limiting max download rate) is $u_s + \Sigma u_i$
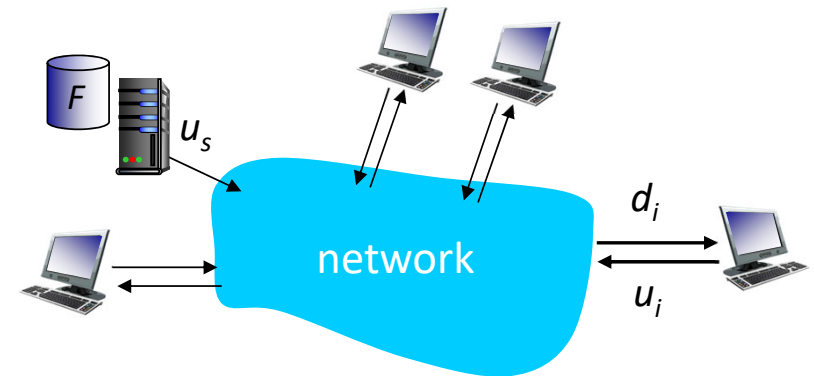
time to distribute F to N clients using P2P approach

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

increases linearly in *N* …
… but so does this, as each peer brings service capacity

Client-server vs. P2P: example

client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$
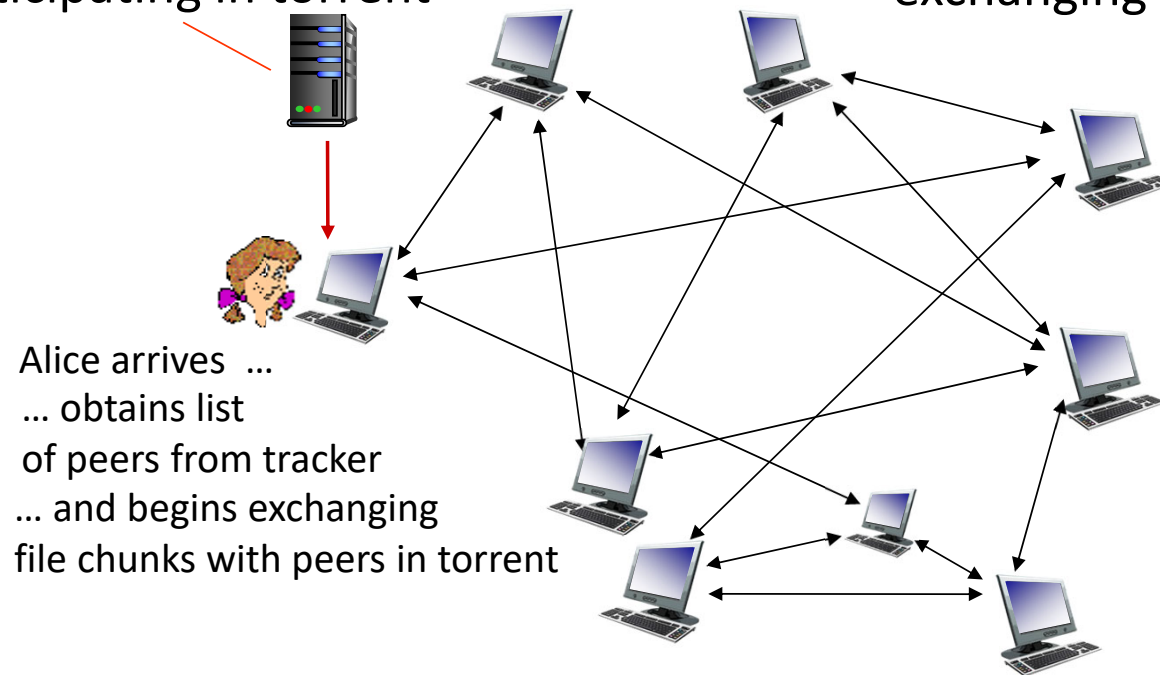
P2P file distribution: BitTorrent

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

tracker: tracks peers
participating in torrent

torrent: group of peers
exchanging chunks of a file

Alice arrives ...
... obtains list
of peers from tracker
... and begins exchanging
file chunks with peers in torrent

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 ("neighbors")
- 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