Lecture 5

Application Layer Part II

Start Transport Layer

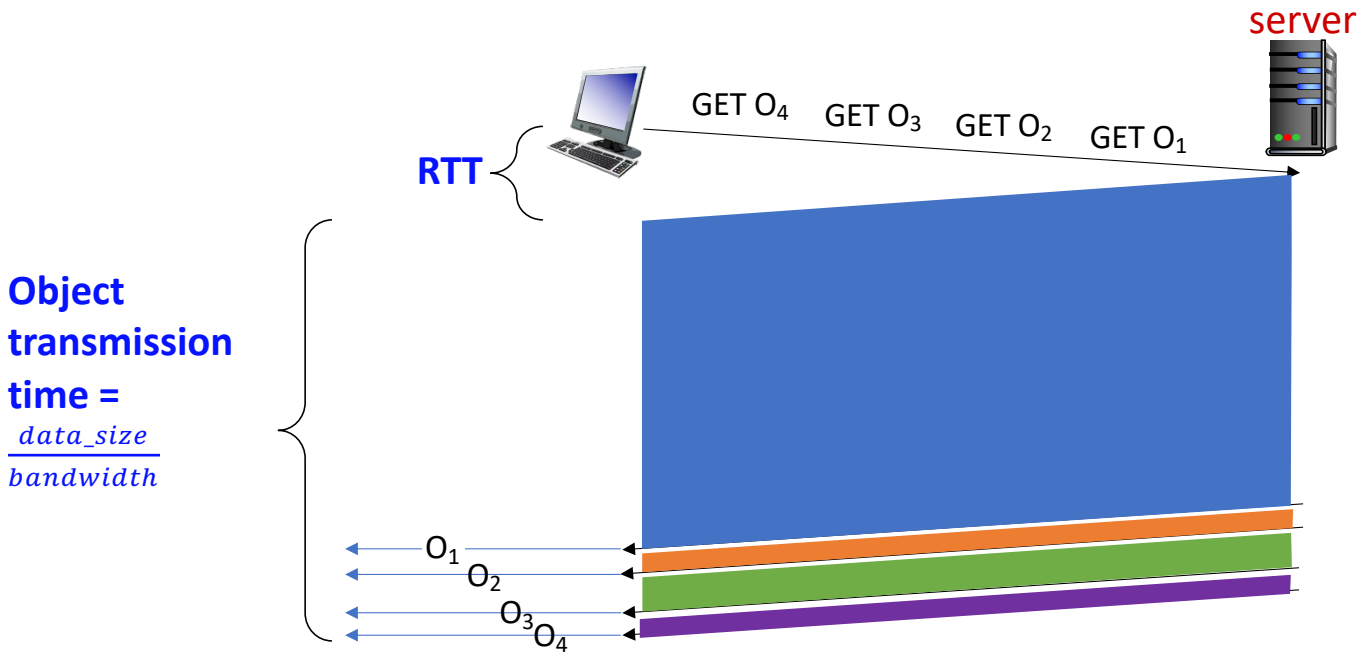
Revisit HTTP Performance

Objectives

- Understand how **caching** and **replication** are used to improve web performance

Recall: HTTP Performance

- In the **best case** (connection already established, pipelined requests, no queuing or processing delay), response time is about $RTT + \frac{data\_size}{bandwidth}$



Improving Performance: Caching

- To improve response times for frequently accessed data, we can **cache** (temporarily store) results we've already retrieved
  - Where to store? Simple option is for the **browser (client)** to do it
- Server doesn't need to re-send if we already have the latest copy of a resource

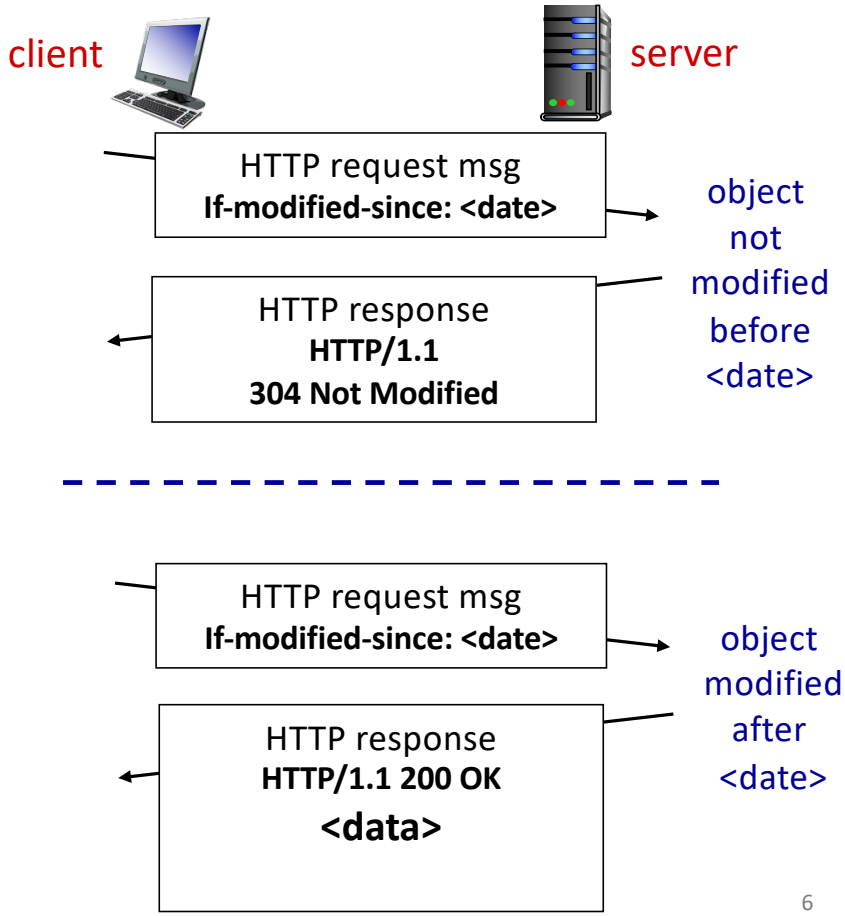
Caching Mechanism: Conditional GET

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

- *client:* specify date of cached copy in HTTP request
  If-modified-since: <date>
- *server:* response contains no object if cached copy is up-to-date:
  HTTP/1.1 304 Not Modified

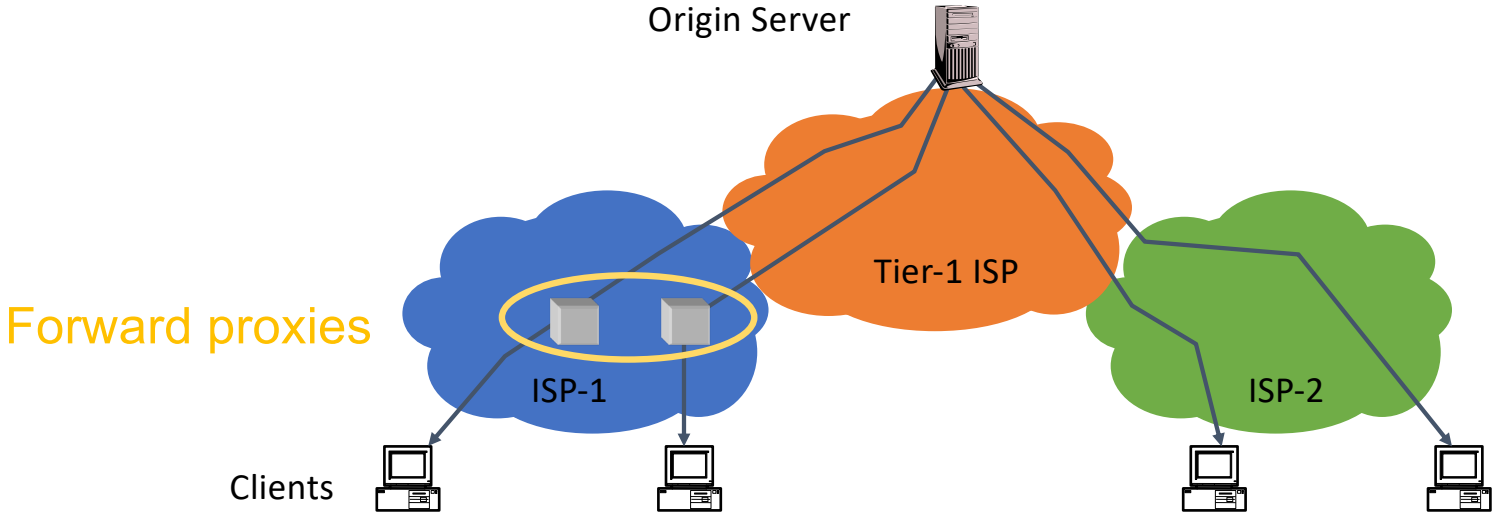
Outcomes:

- **no object transmission delay** – reduce **data size**
- **lower link utilization –** increase free **bandwidth** for other requests
- can also **use cached copy without contacting server** – very fast (**eliminates RTT**), but risks stale data



Caching beyond the browser: Proxy Servers

- **ISPs (or institutions)** can deploy proxy servers (web caches) in their networks
  - Increases potential for cache hits compared with browser-only cache (many users access the same content)
  - Reduces **network traffic** for ISP
  - Improves performance for users



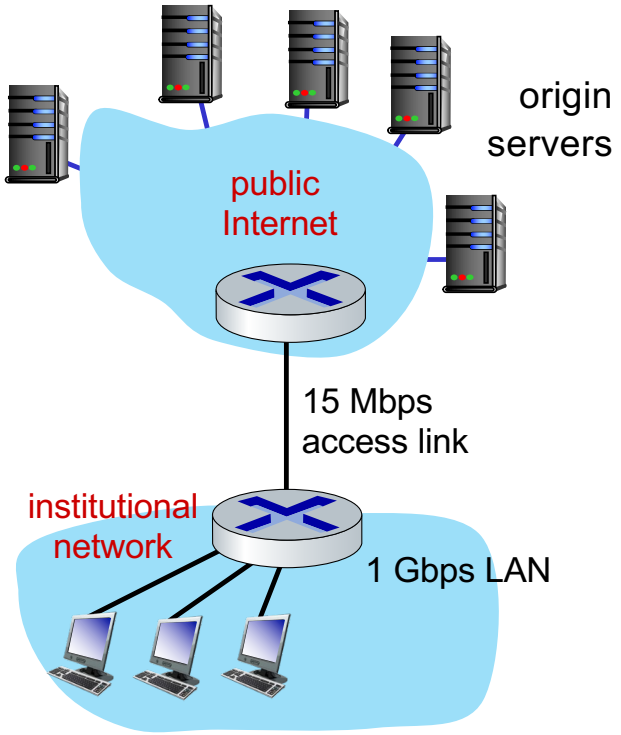
Forward Proxy: Motivation

Scenario:

- access link rate: 15 Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 1 Mbits
- Average request rate from browsers to origin servers: 15/sec
  - average data rate to browsers: 15 Mbps

Performance:

- LAN utilization: .015
- access link utilization = 1.0
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + **unbounded delay** + usecs



Option 1: Upgrade Access Link

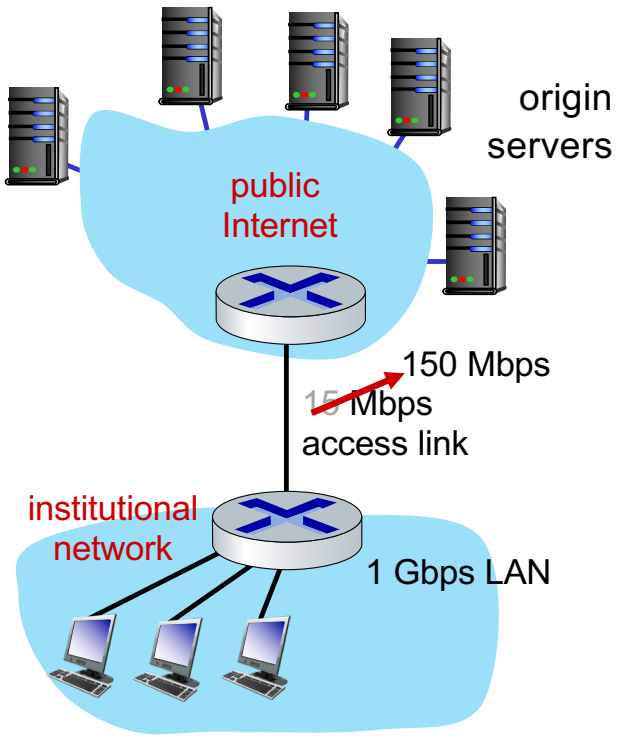
Scenario:

- access link rate: ~~15~~ 150 Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 1 Mbits
- Average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 15 Mbps

Performance:

- LAN utilization: .015
- access link utilization = ~~1.0~~ .1
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + ~~unbounded delay~~ + usecs

Cost: faster access link (expensive!) msecs



Option 2: Install a Proxy

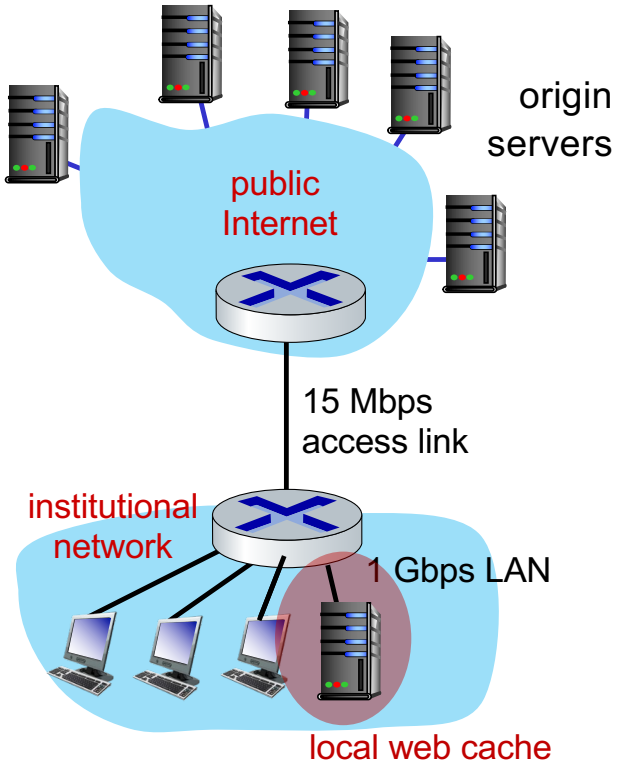
Scenario:

- access link rate: 15 Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 1 Mbits
- Average request rate from browsers to origin servers: 15/sec
  - average data rate to browsers: 15 Mbps

Performance:

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

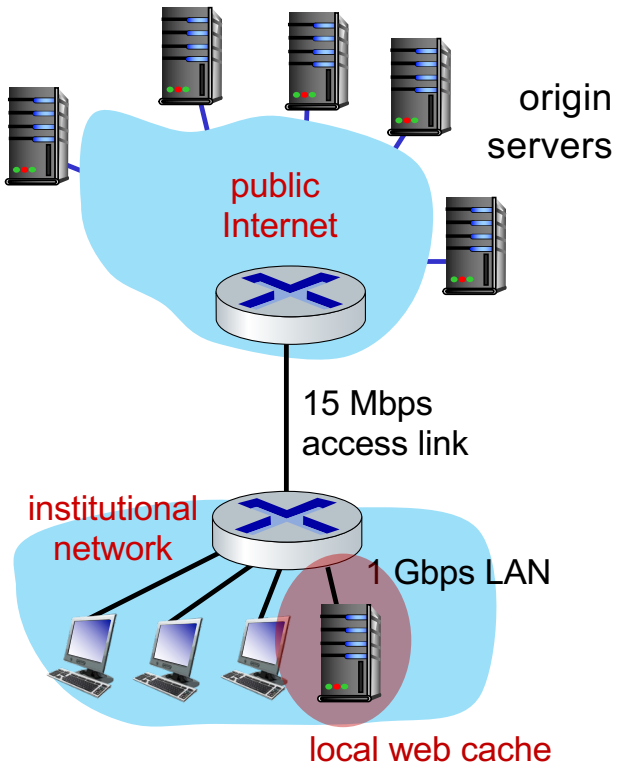
Cost: web cache (cheap!)



Option 2: Install a Proxy

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 * 15 Mbps = 9 Mbps
- utilization = 9/15 = .6
- 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 150 Mbps link (and cheaper too!)

Conditional GET

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

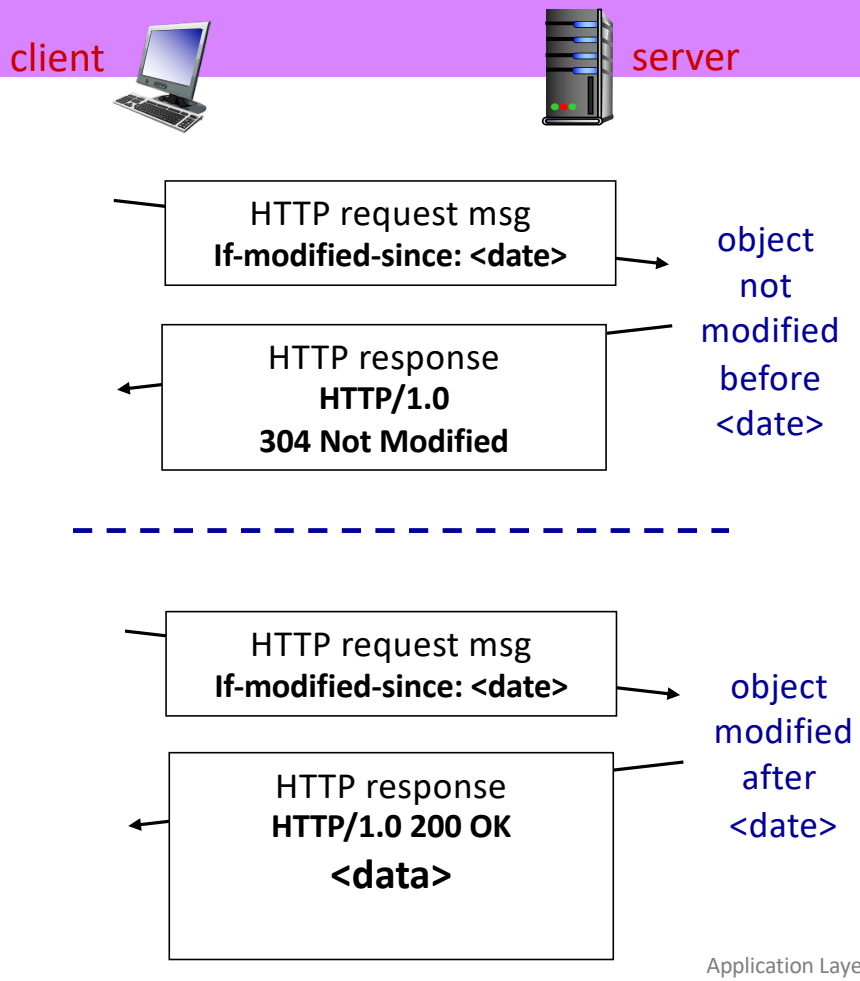
- no object transmission delay
- lower link utilization

- *cache:* specify date of cached copy in HTTP request

  If-modified-since: <date>

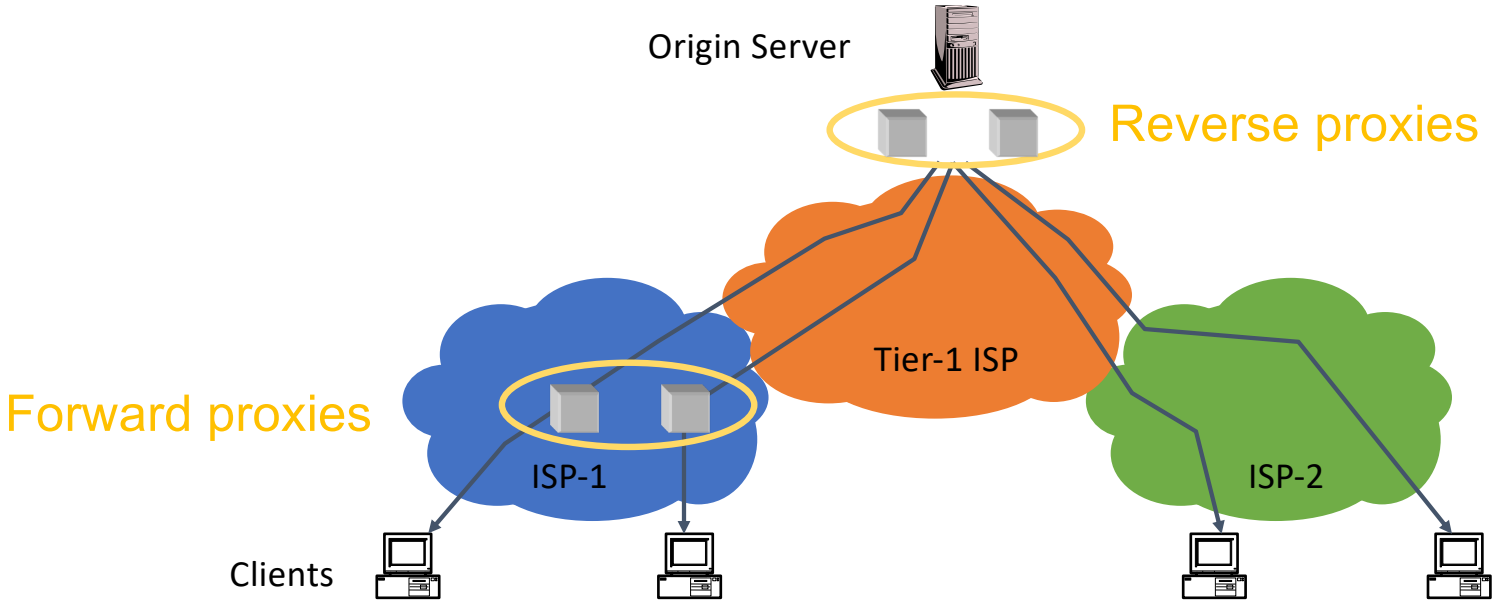
- *server:* response contains no object if cached copy is up-to-date:

  HTTP/1.0 304 Not Modified



Caching beyond the browser: Proxy Servers

- **Content providers** can also deploy proxy servers in their networks
  - Reduces **server load** for content provider
  - Improves performance for users (reduce queuing at server processing level)

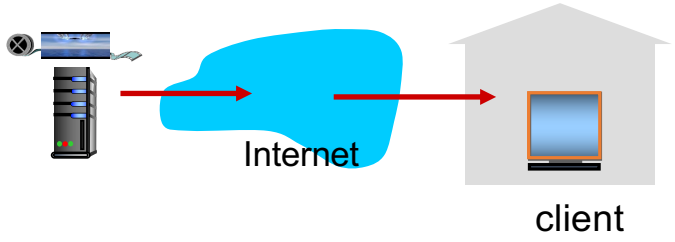


Video Streaming and CDNs: context

- video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1B YouTube users, ~75M Netflix users
- challenge: scale - how to reach ~1B 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

Streaming multimedia: DASH

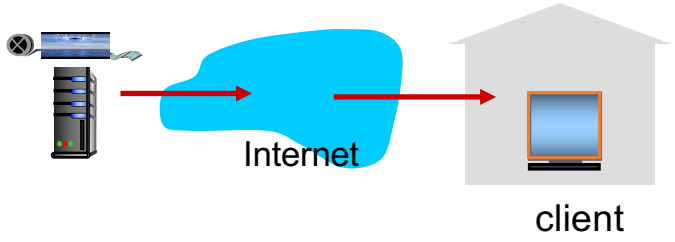
- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *server:*
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file:* provides URLs for different chunks



- *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

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



Streaming video = encoding + DASH + playout buffering

Content Distribution Networks (CDNs)

- Caching and replication by **content providers**
  - May be private CDN built by content provider (e.g. Google, Netflix)
  - May be third-party CDN that serves many content providers (e.g. Akamai, Limelight)
- Aim to **get content closer to users** – servers distributed across the globe and placed in or close to access ISPs
  - Reduce propagation delay
  - Reduce number of links traversed (reduce chance of encountering congested "bottleneck link")
  - Reduce backbone traffic (and lower content provider cost)
  - Improve **fault tolerance**
- Can combine **reactive caching** and **proactive replication**
  - Pull: Cache content requested by clients
  - Push: Proactively replicate content expected to have high access rates

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*

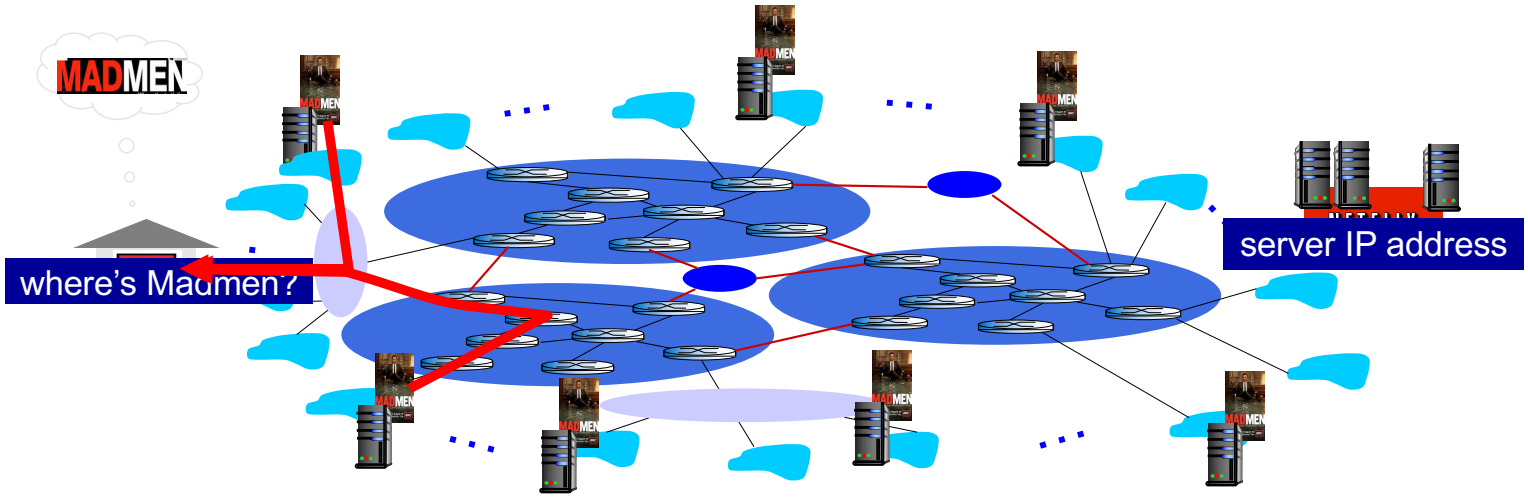
Content distribution networks

- *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, 1700 locations
  - *bring home:* smaller number (10's) of larger clusters in (points of presence) 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



Content distribution networks (CDNs)

Internet host-host communication as a service

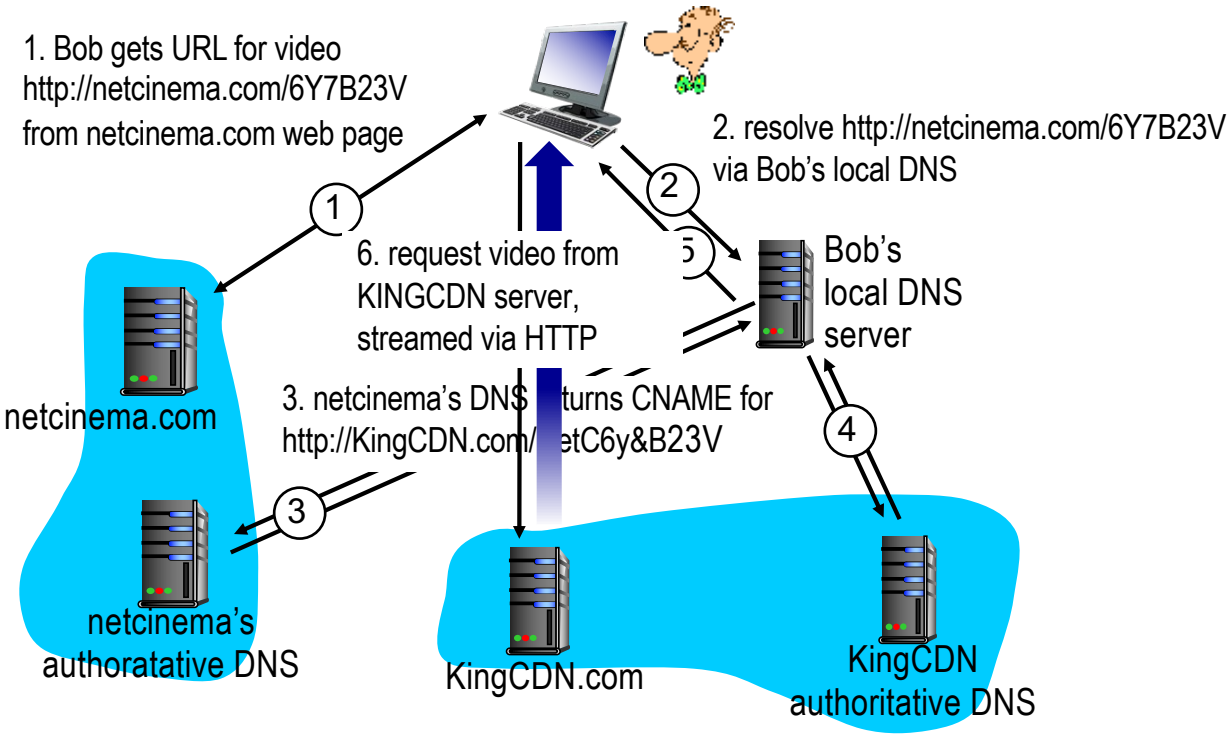
OTT challenges: coping with a congested Internet

- from which CDN node to retrieve content?
- viewer behavior in presence of congestion?
- what content to place in which CDN node?

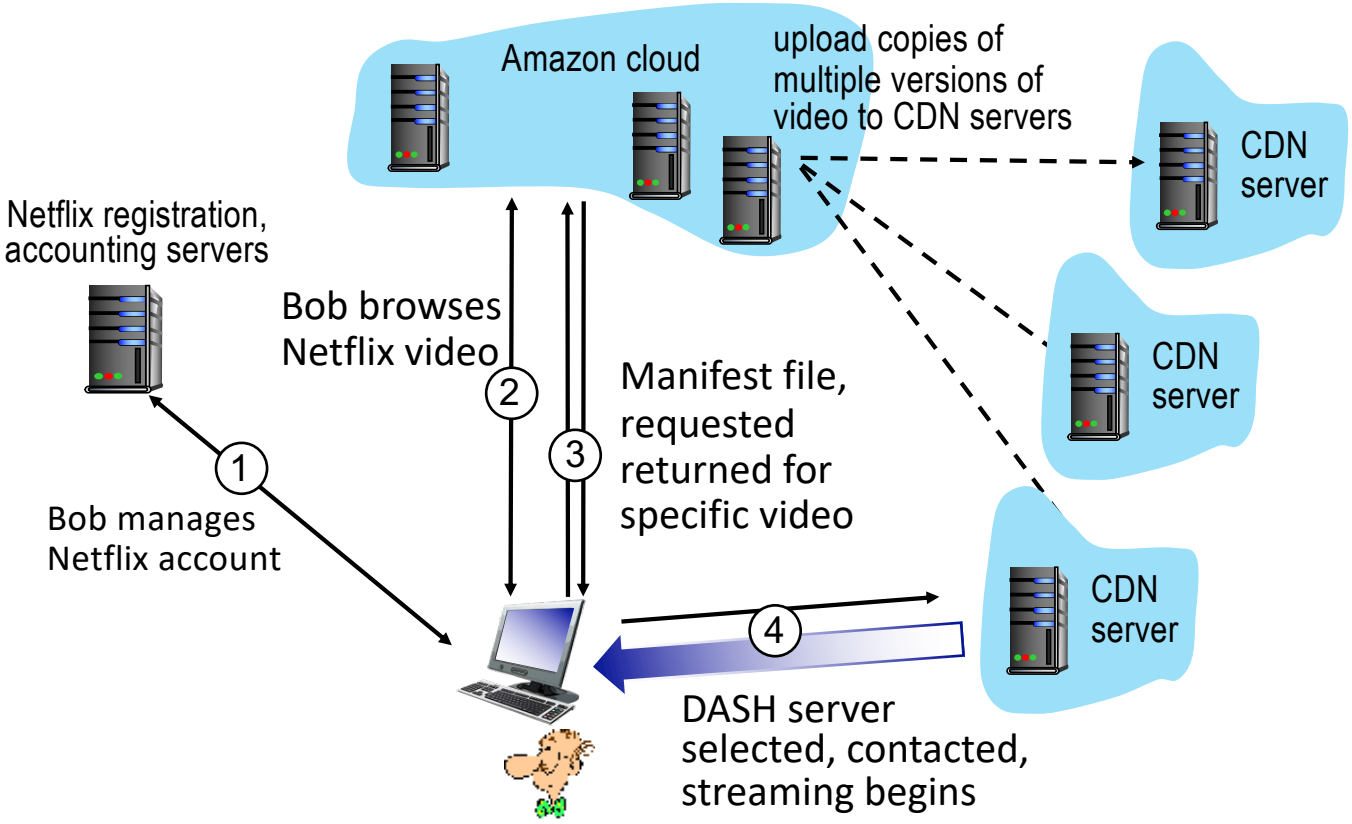
CDN content access: a closer look

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

- video stored in CDN at http://KingCDN.com/NetC6y&B23V



Case study: Netflix



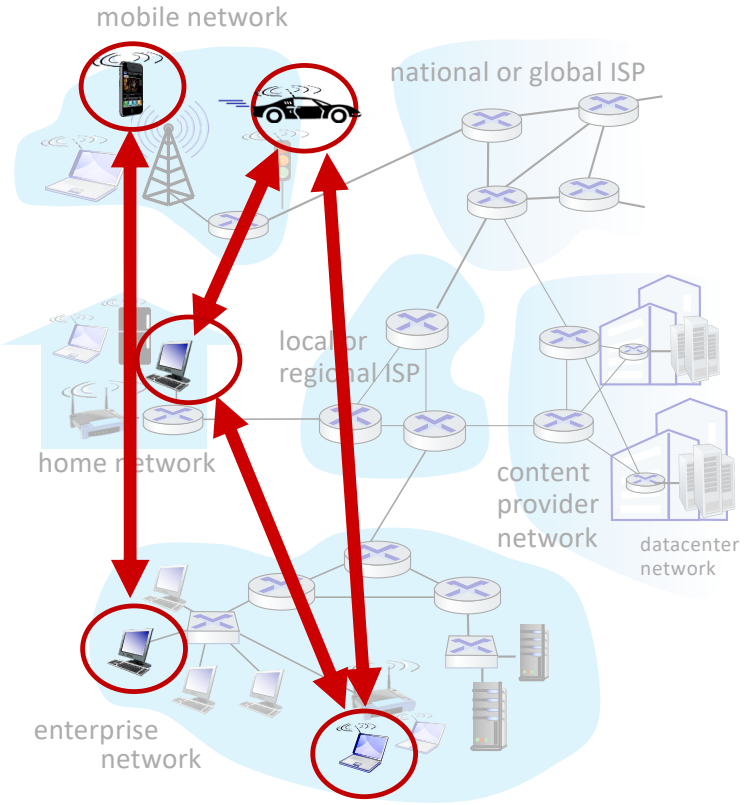
Summary

- Caching and replication improve web performance
  - Move content closer to clients
    - Reduce RTT (assuming content served directly from cache/replica)
    - Reduce chance to encounter bottleneck link
  - Avoid transferring the same data over the same part of the network multiple times
    - Reduce overall network traffic

Lecture 5, Part 2: Quick look at P2P

Peer-to-peer (P2P) architecture

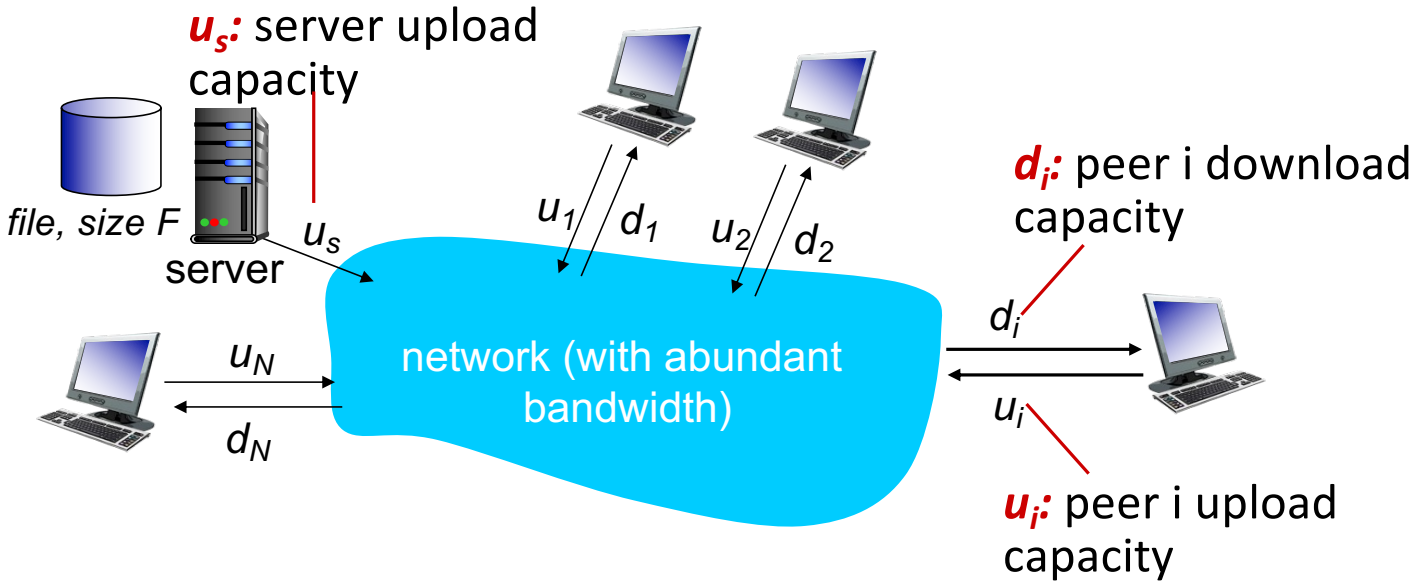
- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, and new service demands
- peers are intermittently connected and change IP addresses
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



File distribution: client-server vs P2P

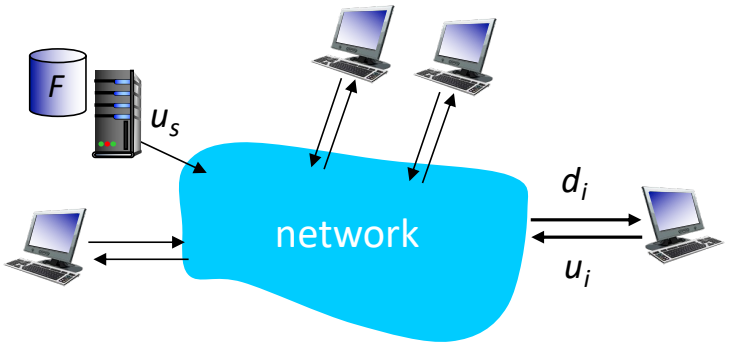
Q: how much time to distribute file (size *F*) from one server to *N peers*?

- peer upload/download capacity is limited resource



File distribution time: client-server

- *server transmission:* must sequentially send (upload) *N* file copies:
  - time to send one copy: F/u_s
  - time to send *N* copies: NF/u_s
- *client:* each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min}



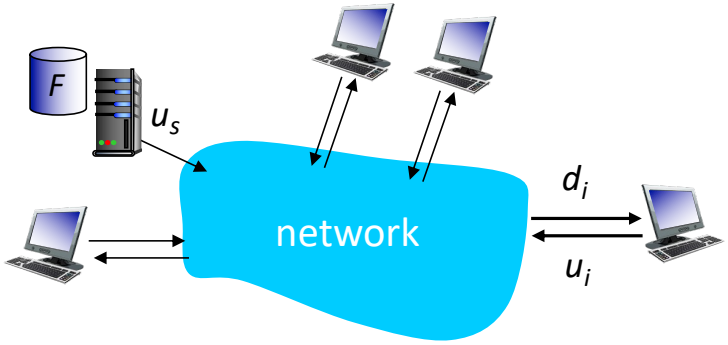
time to distribute F to N clients using client-server approach

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

increases linearly in N

File distribution time: P2P

- *server transmission:* must upload at least one copy:
  - time to 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 $u_s + \Sigma u_i$



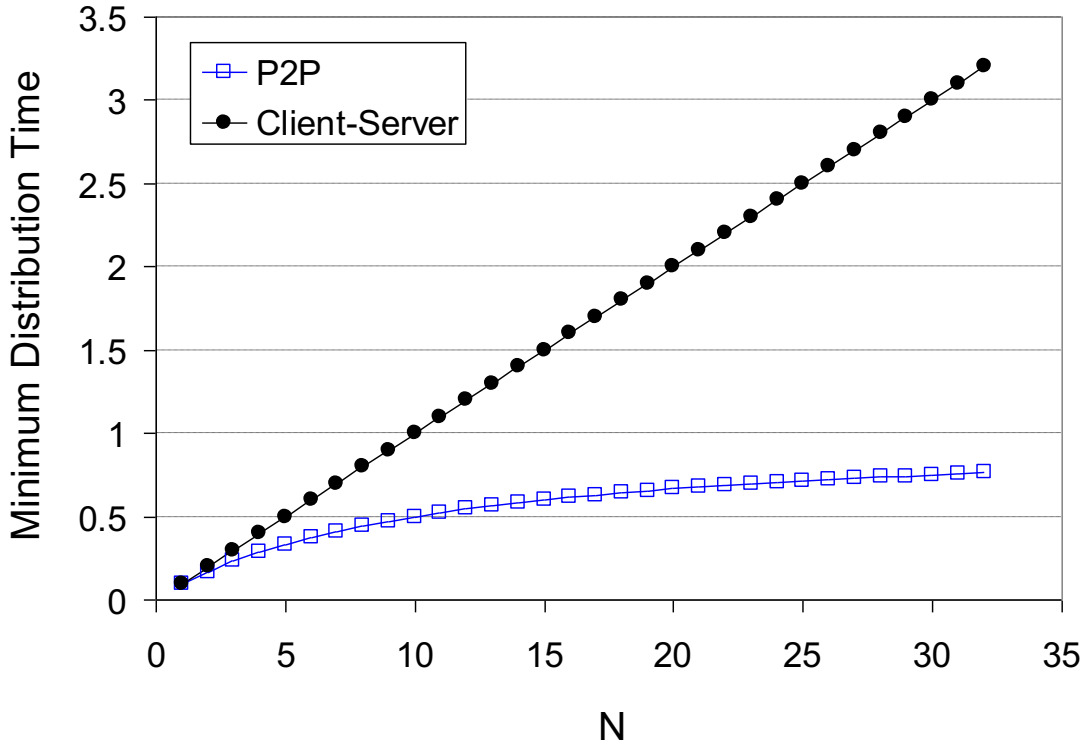
time to distribute F to N clients using P2P approach

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

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, $u_s = 10u$, $d_{min} \geq u_s$



Lecture 5, Part 3: DNS

Partially adapted from Kurose & Ross slides:
http://gaia.cs.umass.edu/kurose_ross/ppt.htm

Partially adapted from JHU Computer Networks course:
https://github.com/xinjin/course-net

Objectives

- Understand how **DNS (Domain Name Service)** is used to map host names to IP addresses

Internet names & addresses

- **IP addresses**: e.g. 136.142.156.132
  - Router-usable labels for machines
  - Conforms to network structure (the "where")
- **Hostnames**: e.g. dins.pitt.edu
  - Human-usable labels for machines
  - Conforms to organizational structure (the "who")
- The **Domain Name System (DNS)** is how we map from one to the other
  - A directory service

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

Why do we need DNS?

- Convenience
  - Much easier to remember www.dins.pitt.edu than 136.142.156.132
- Flexibility – Decoupling names from addresses allows:
  - Seamless address changes
    - e.g. move www.dins.pitt.edu to new machine with IP 136.142.156.133
  - Multiple names per IP address
    - different webpages on same machine (e.g. www.facultydiversity.pitt.edu hosted on same machine as www.dins.pitt.edu)
    - different services on same machine (mail, web)
    - aliasing, e.g. ewi-vip-20.cssd.pitt.edu = www.dins.pitt.edu = dins.pitt.edu
  - Multiple IP addresses per name
    - Load distribution, Redirection to CDN infrastructure
    - Mail server aliasing (mail and web servers on different machines can share hostname)

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
- 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

DNS: Design Goals

- Fix the problems of hosts.txt solution…

- Scalable
- Easy to maintain
- Highly available
- Fast lookups

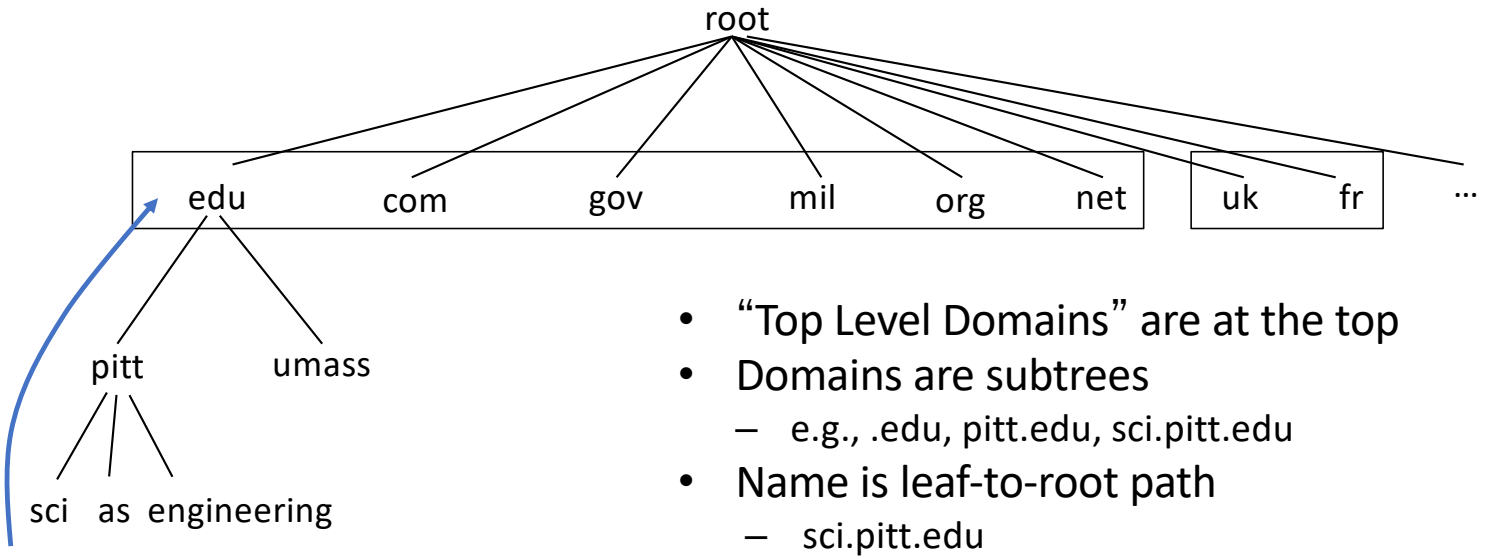
DNS: Overview

- Distributed database of hostname-IP address mappings
  - Partitioned across many **name servers**
- Organized hierarchically
  - Simplifies maintenance

- Provides a critical service for the Internet, but is implemented as an application level protocol (end-to-end argument)
  - Runs over UDP on port 53, client-server architecture

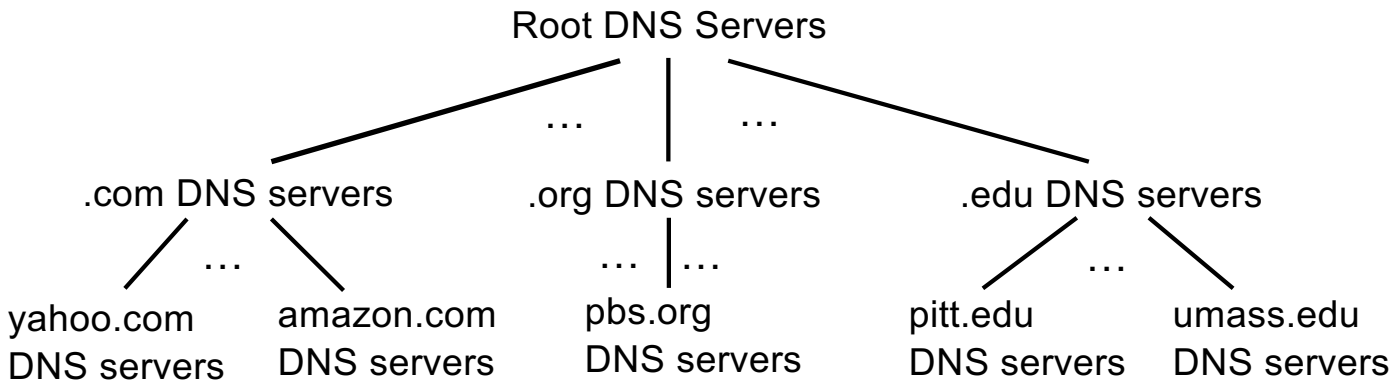
DNS: Hierarchical Organization

- Namespace is organized hierarchically



- "Top Level Domains" are at the top
- Domains are subtrees
  - e.g., .edu, pitt.edu, sci.pitt.edu
- Name is leaf-to-root path
  - sci.pitt.edu
- Depth of tree is arbitrary (limit 128)
- Name collisions trivially avoided
  - Each domain is responsible

DNS: Hierarchical Organization



Root

Top Level Domain

Authoritative

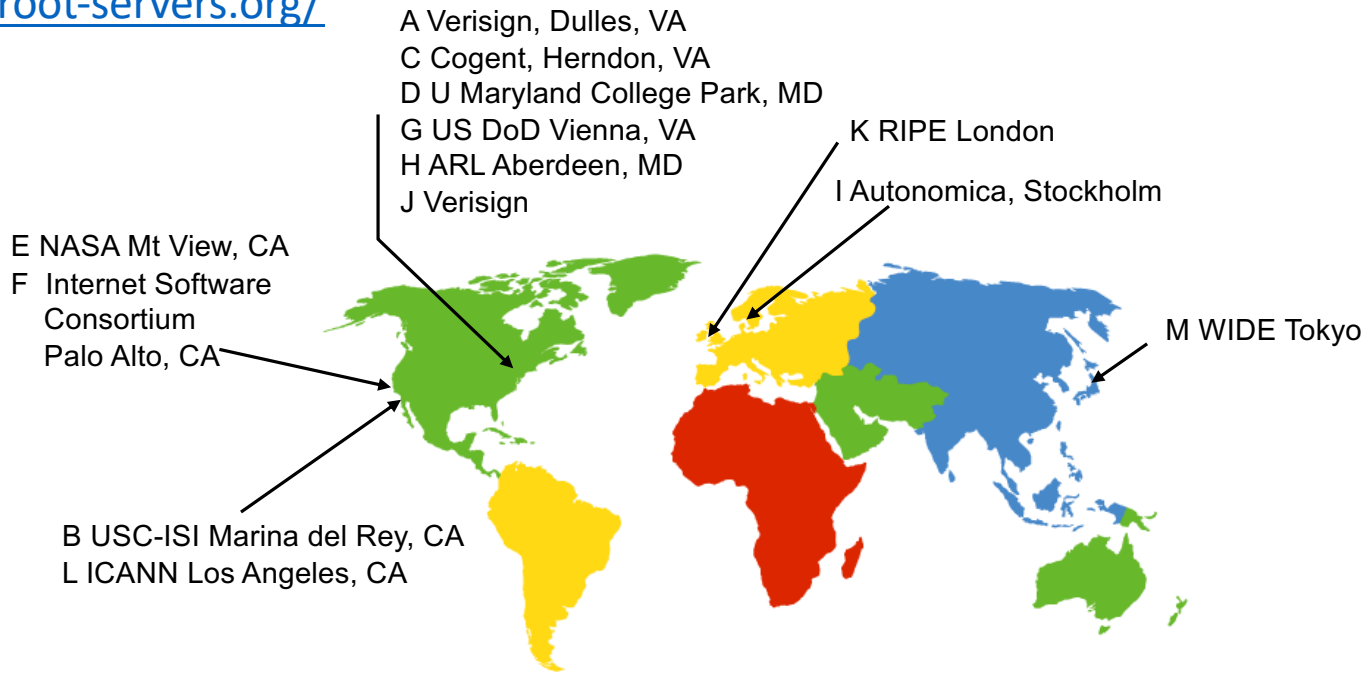
Servers are also organized hierarchically

Client wants IP address for www.pitt.edu; 1st approximation:

- client queries root server to find .edu DNS server
- client queries .edu DNS server to get pitt.edu DNS server
- client queries pitt.edu DNS server to get IP address for www.pitt.edu

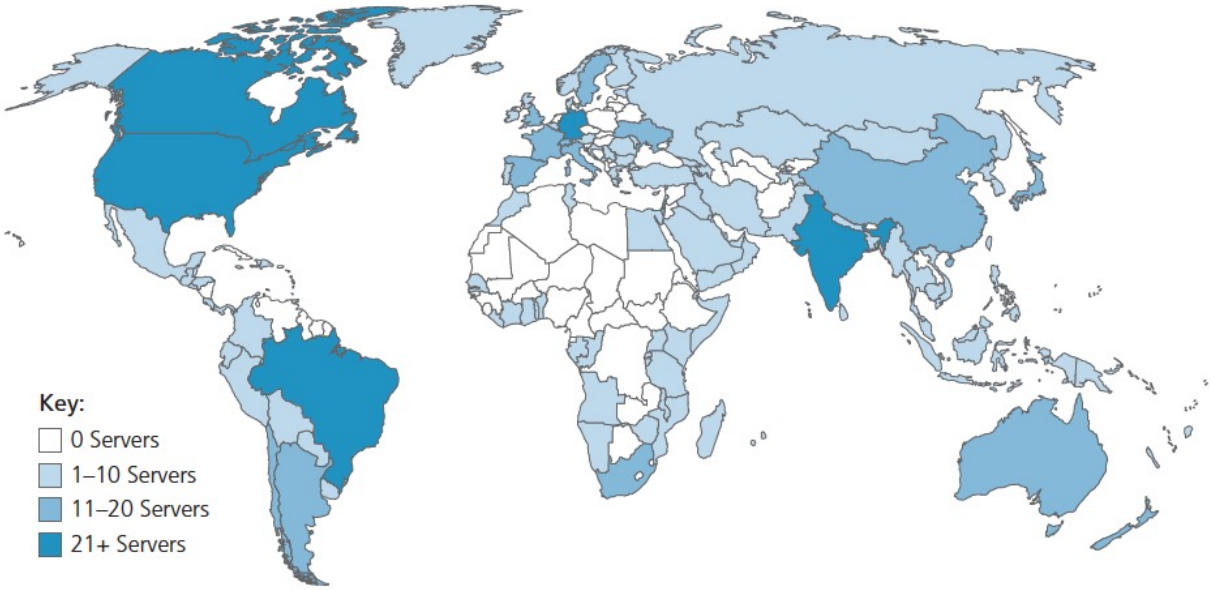
Root Name Servers

- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain
- 13 logical root name "servers" worldwide each "server" replicated many times (over 1000 total instances)
  - https://root-servers.org/



Root Name Servers

- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain
- 13 logical root name "servers" worldwide each "server" replicated many times (over 1000 total instances)
  - https://root-servers.org/



Top-Level Domain Servers

Top-Level Domain (TLD) servers:

- responsible for .com, .org, .net, .edu, .aero, .jobs, .museums, and all top-level country domains, e.g.: .cn, .uk, .fr, .ca, .jp
- Verisign Global Registry Services: authoritative registry for .com, .net TLD
  - After acquisition of Network Solutions, which originally served this role
- Educause: .edu TLD

Authoritative DNS Servers

Authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Local DNS name servers

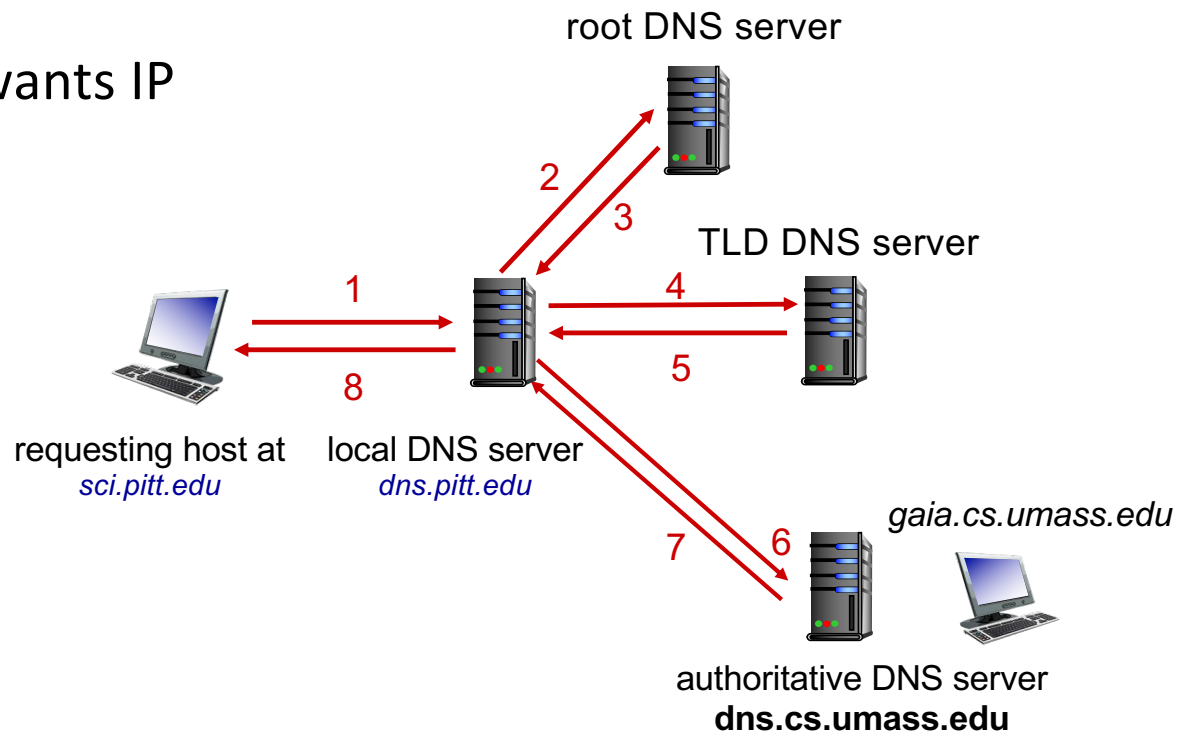
- **Not part of hierarchy**
  - Not “in charge” of any part of the name space
- Acts as **proxy**: when host makes DNS query, query is sent to its local DNS server
  - Local DNS server forwards query into hierarchy
- each ISP (residential ISP, company, university) has one
  - also called “default name server”

DNS name resolution: iterated query

Example: host at sci.pitt.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"
- In this example, all requests other than 1 are iterated

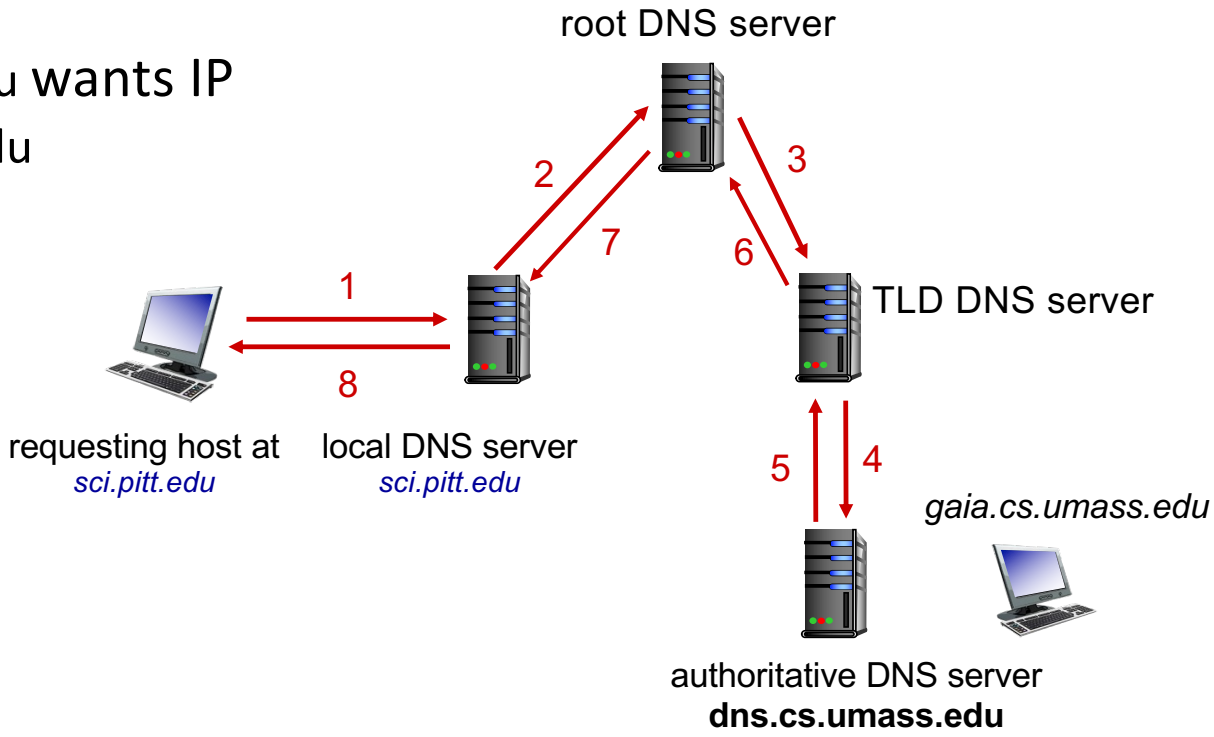


DNS name resolution: recursive query

Example: host at sci.pitt.edu wants IP address for gaia.cs.umass.edu

Recursive query:

- puts burden of name resolution on contacted name server
- All the queries can be recursive (not common)
- Usually the query from the host to the local server is recursive and all the rest iterative (like previous example)



DNS Caching

- **How DNS caching works**
  - DNS servers (at every level) cache responses to queries
  - Responses include a "time to live" (TTL) field
  - Server deletes cached entry after TTL expires
- **Caching can greatly reduce overhead**
  - The top-level servers very rarely change
  - Popular sites visited often
  - Local DNS server often has the information cached
- **But, cached entries may be *out-of-date*** (best-effort name-to-address translation)
  - if host changes its IP address, may not be known Internet-wide until all TTLs expire

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

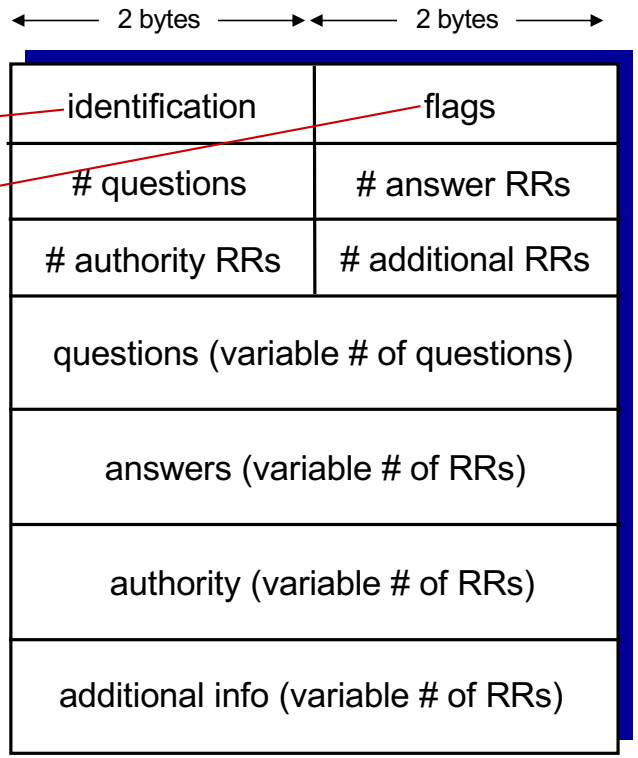
- `value` is name of mailserver associated with `name`

DNS protocol messages

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

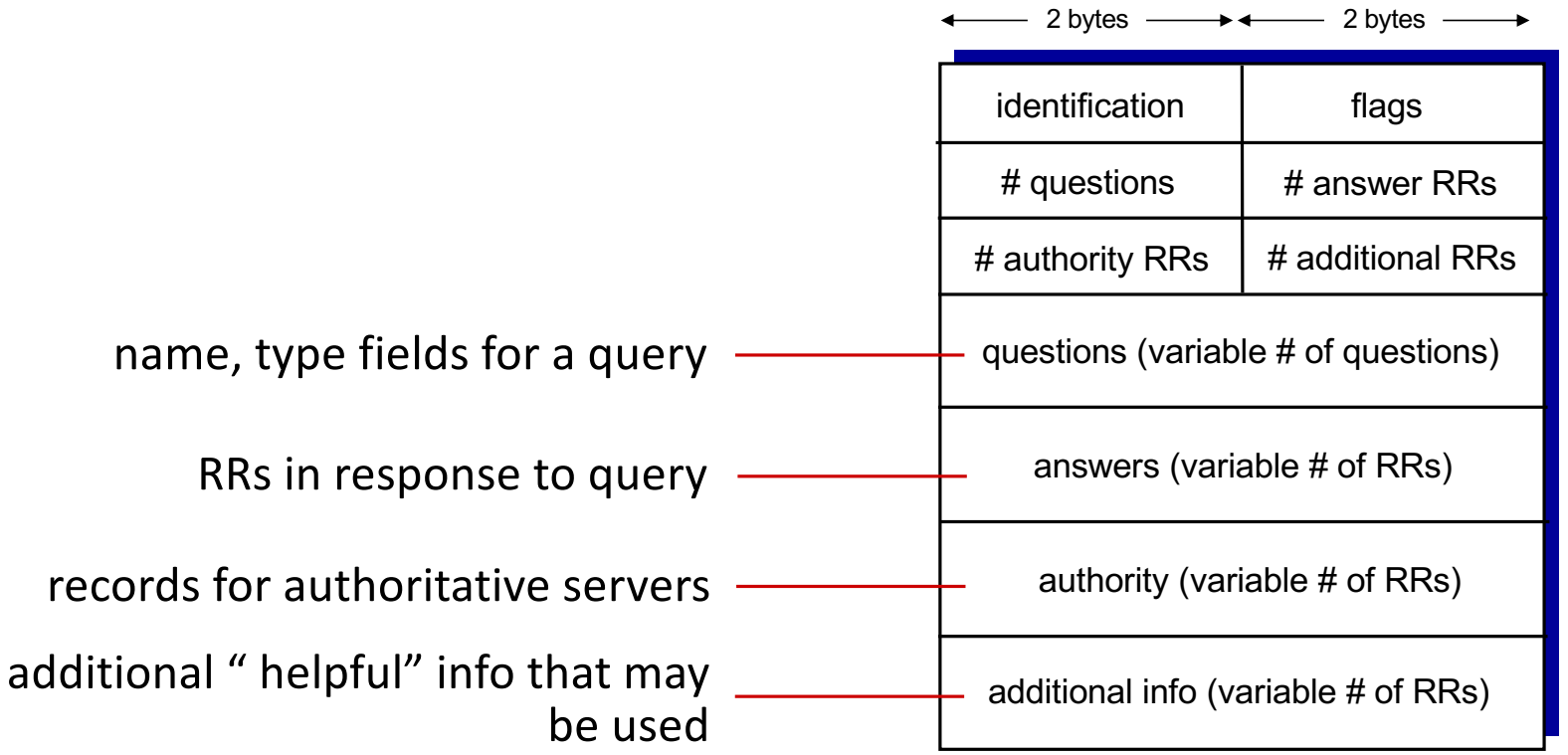
message header:

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



DNS protocol messages

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



Adding New Records to DNS

Example: new startup "Network Utopia"

- register name networkutopia.com at *DNS registrar* (e.g., Network Solutions, GoDaddy)
  - 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 (if it's also a mail server)

Summary

- DNS translates hostnames to IP addresses
  - provides convenience and flexibility
- Critical Internet service, implemented as application-level protocol
- Scalability and ease of maintenance achieved through hierarchical organization

Electronic mail (Email)

Two components:

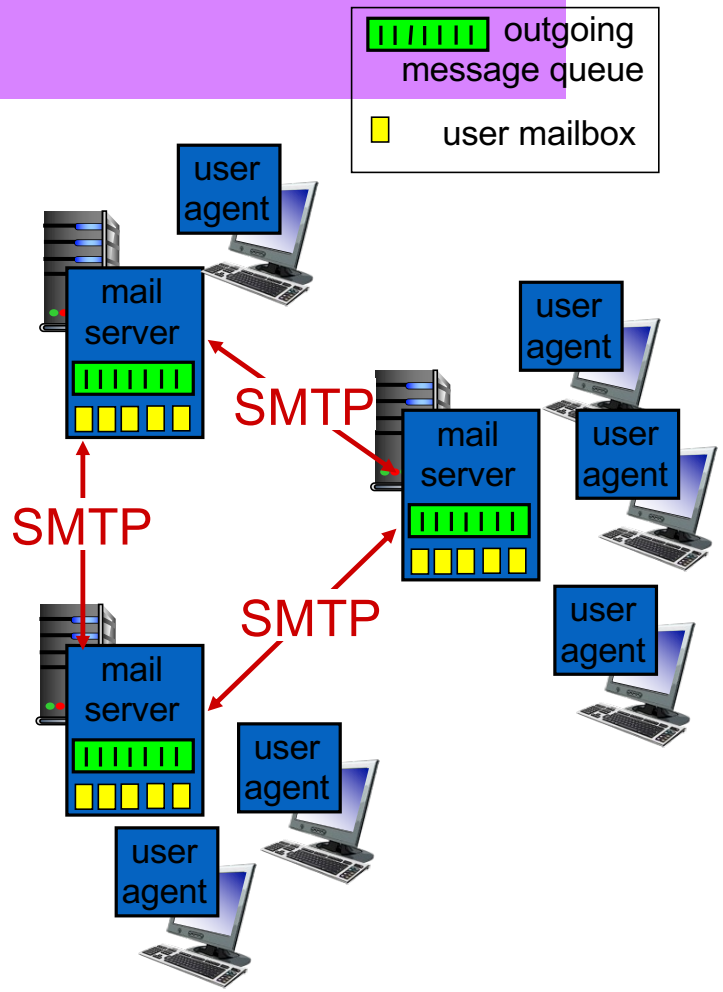
- user agents
- mail servers

User Agent: a.k.a. "mail reader"

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

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages



How is an email sent out? Scenario: Alice sends message to Bob

1) Alice uses UA to compose message "to" `bob@someschool.edu`

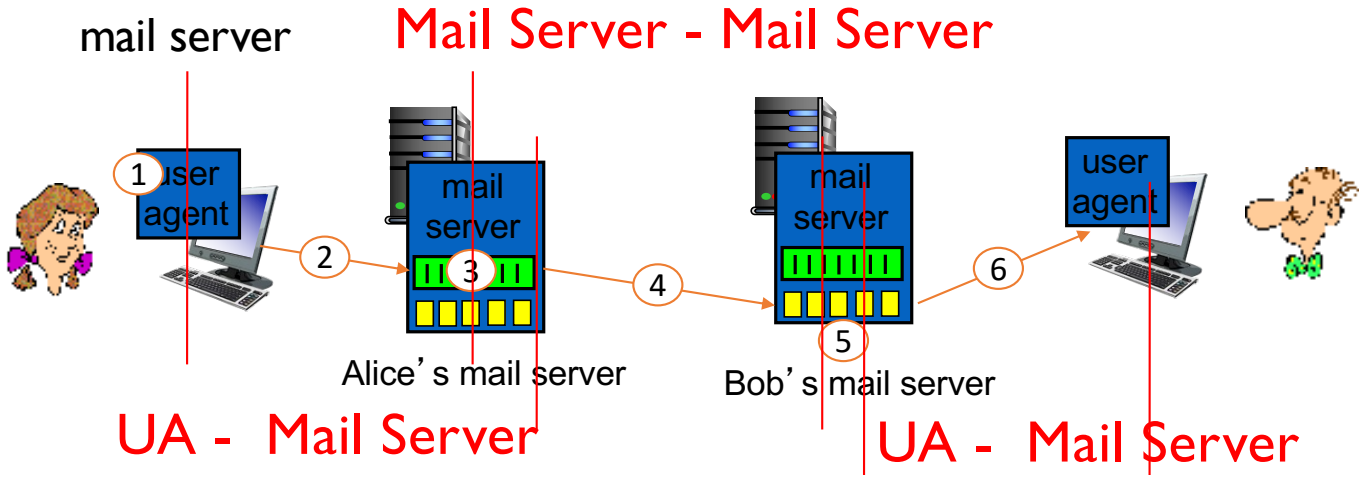
2) Alice's UA sends message to her mail server; message placed in message queue

3) Alice's mail server opens TCP connection with Bob's mail server

4) Alice's mail server sends Alice's message over the TCP connection

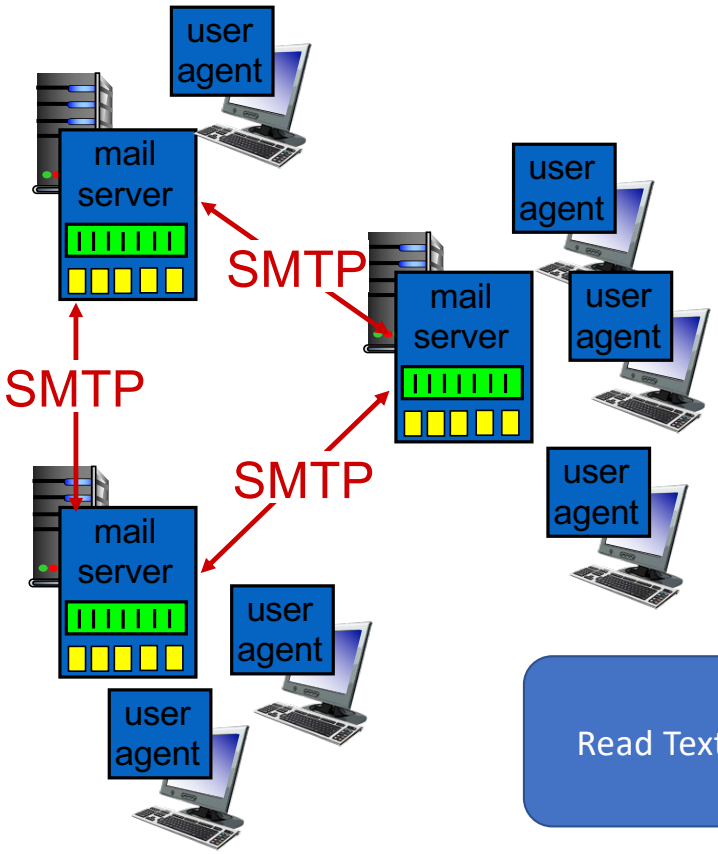
5) Bob's mail server places the message in Bob's mailbox

6) Bob invokes his user agent to read message



Simple Mail Transfer protocol: SMTP

- *SMTP protocol* between mail servers to send email messages
  - "client": sending mail server
  - "server": receiving mail server



Read Textbook!