

# Computer Networks and Applications

COMP 3331/COMP 9331

Week 3

**Application Layer (DNS, P2P, Video Streaming and CDN)**

**Reading Guide: Chapter 2, Sections 2.4 -2.7**

## 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](http://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

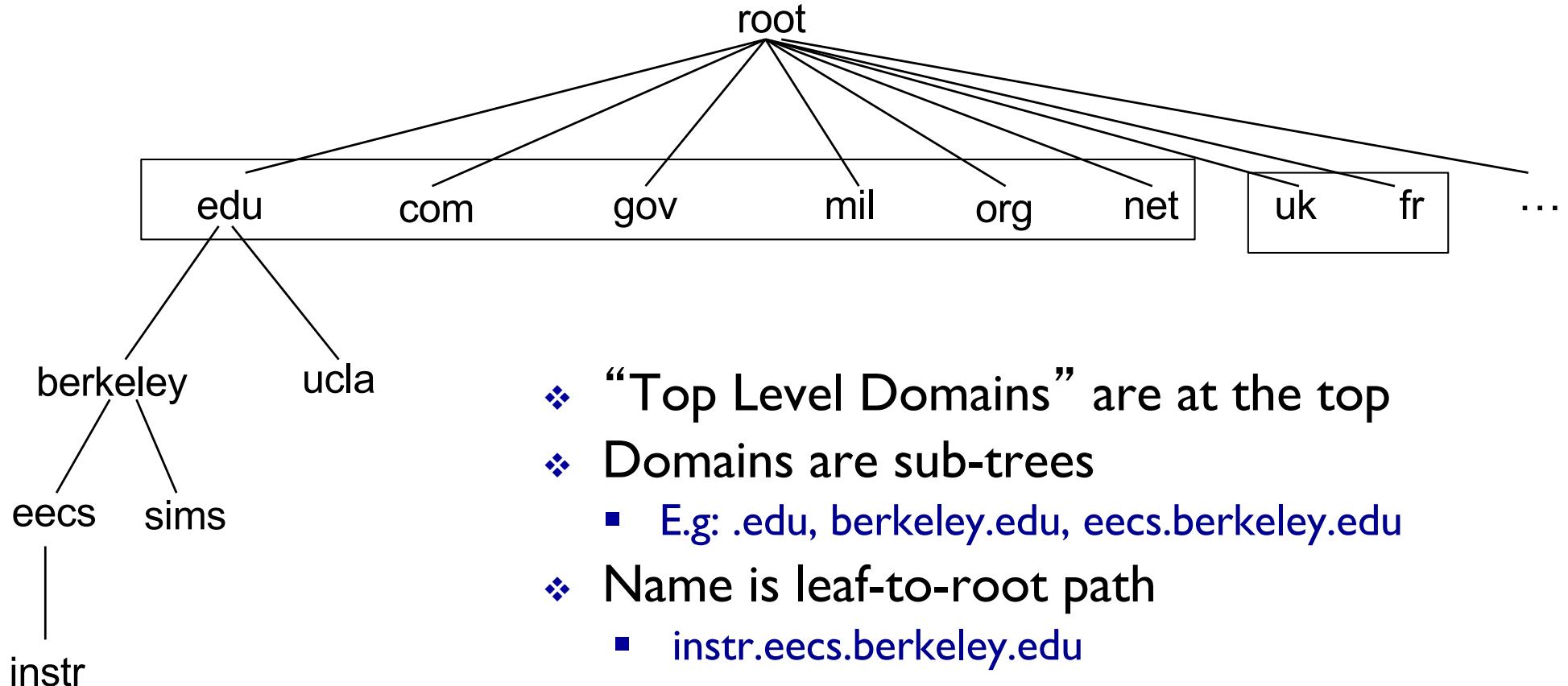
- ❖ No naming conflicts (uniqueness)
- ❖ Scalable
  - many names
  - (secondary) frequent updates
- ❖ Distributed, autonomous administration
  - Ability to update my own (machines') names
  - Don't have to track everybody's updates
- ❖ Highly available
- ❖ Lookups should be fast

# Key idea: Hierarchy

Three intertwined hierarchies

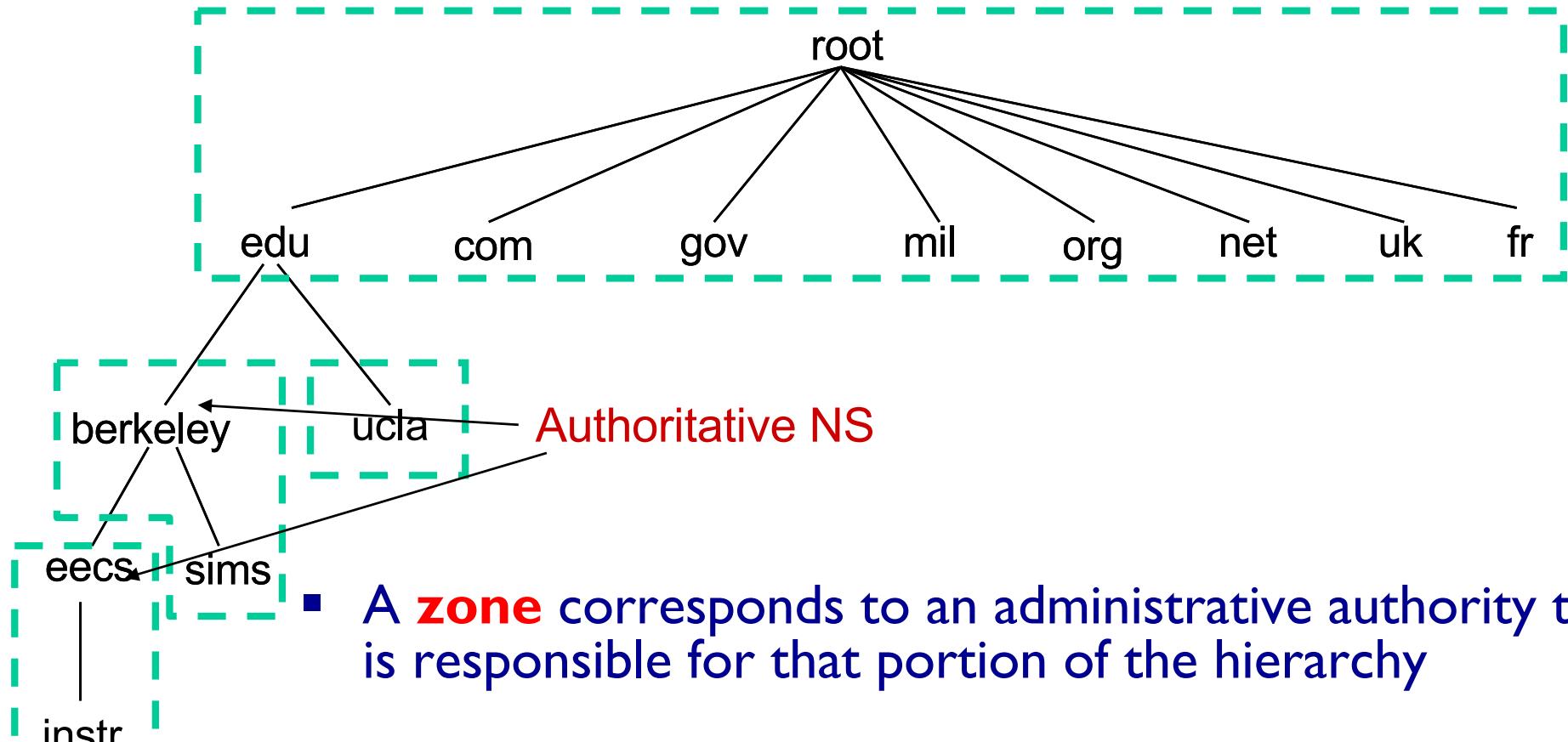
- Hierarchical namespace
  - As opposed to original flat namespace
- Hierarchically administered
  - As opposed to centralised
- (Distributed) hierarchy of servers
  - As opposed to centralised storage

# 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: \*.eecs.berkeley.edu

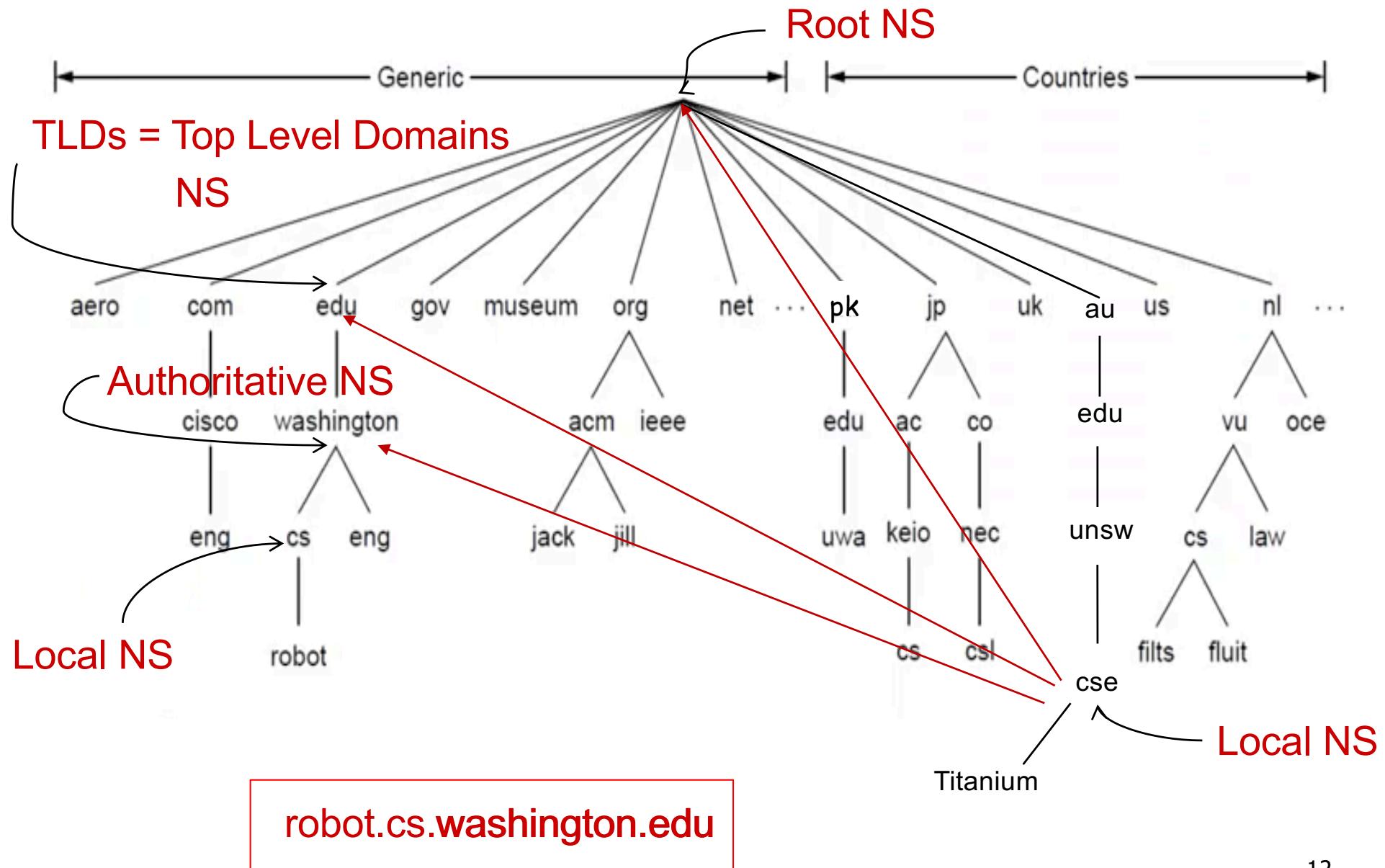
# Server Hierarchy

- ❖ 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



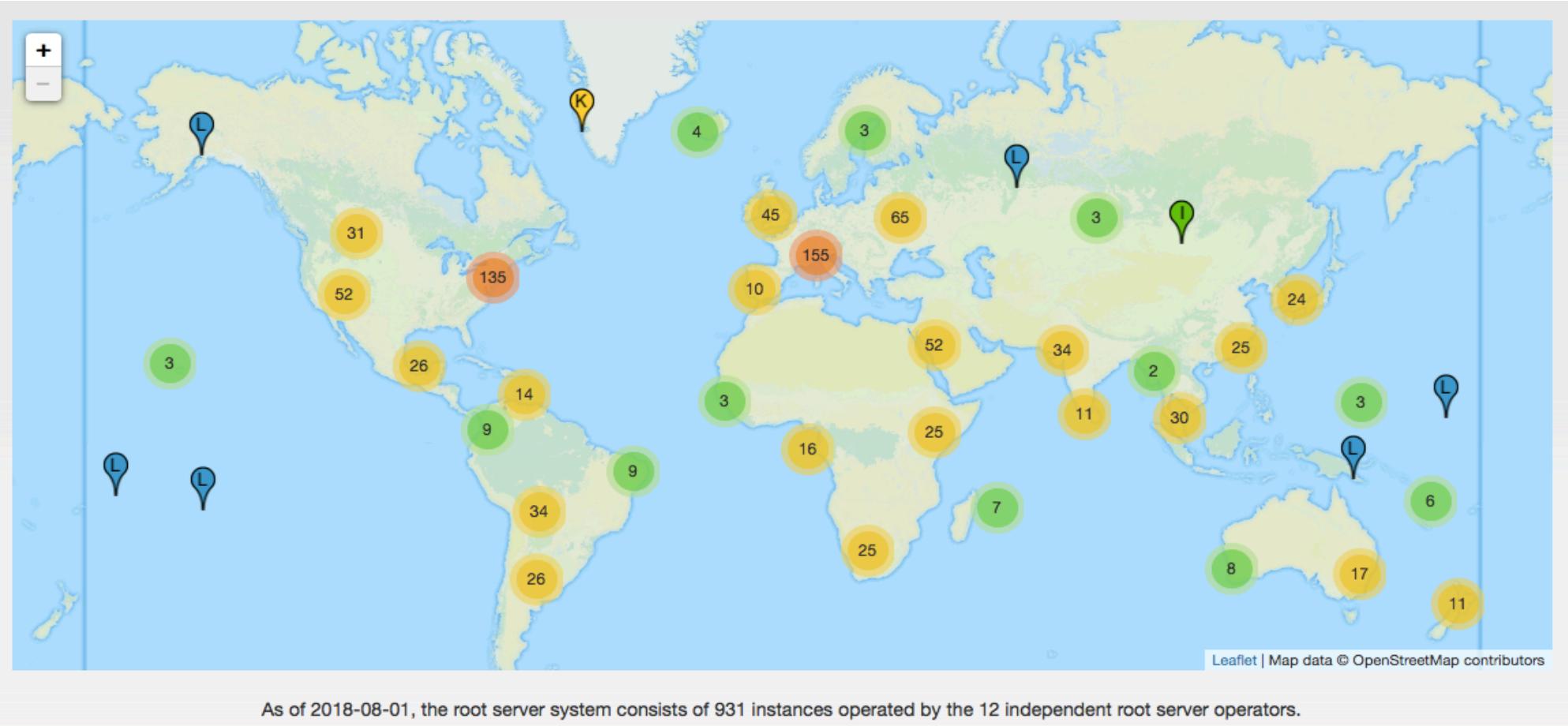
# DNS Root Servers

- 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](http://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

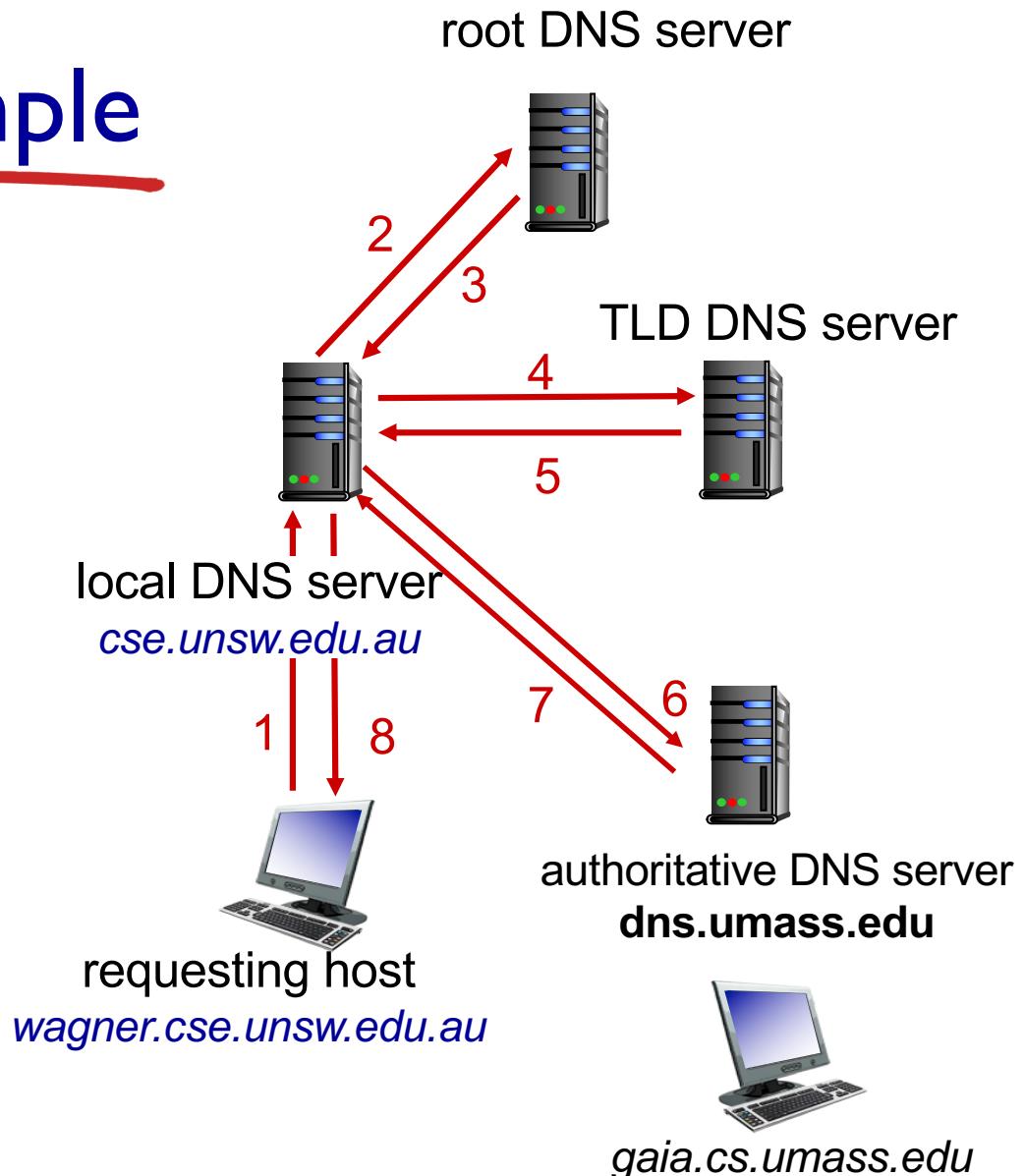
- ❖ does not strictly belong to hierarchy
- ❖ each ISP (residential ISP, company, university) has one
  - also called “default name server”
- ❖ Hosts configured with local DNS server address (e.g., /etc/resolv.conf) or learn server via a host configuration protocol (e.g., DHCP)
- ❖ Client application
  - Obtain DNS name (e.g., from URL)
  - Do `getaddrinfo()` 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:*

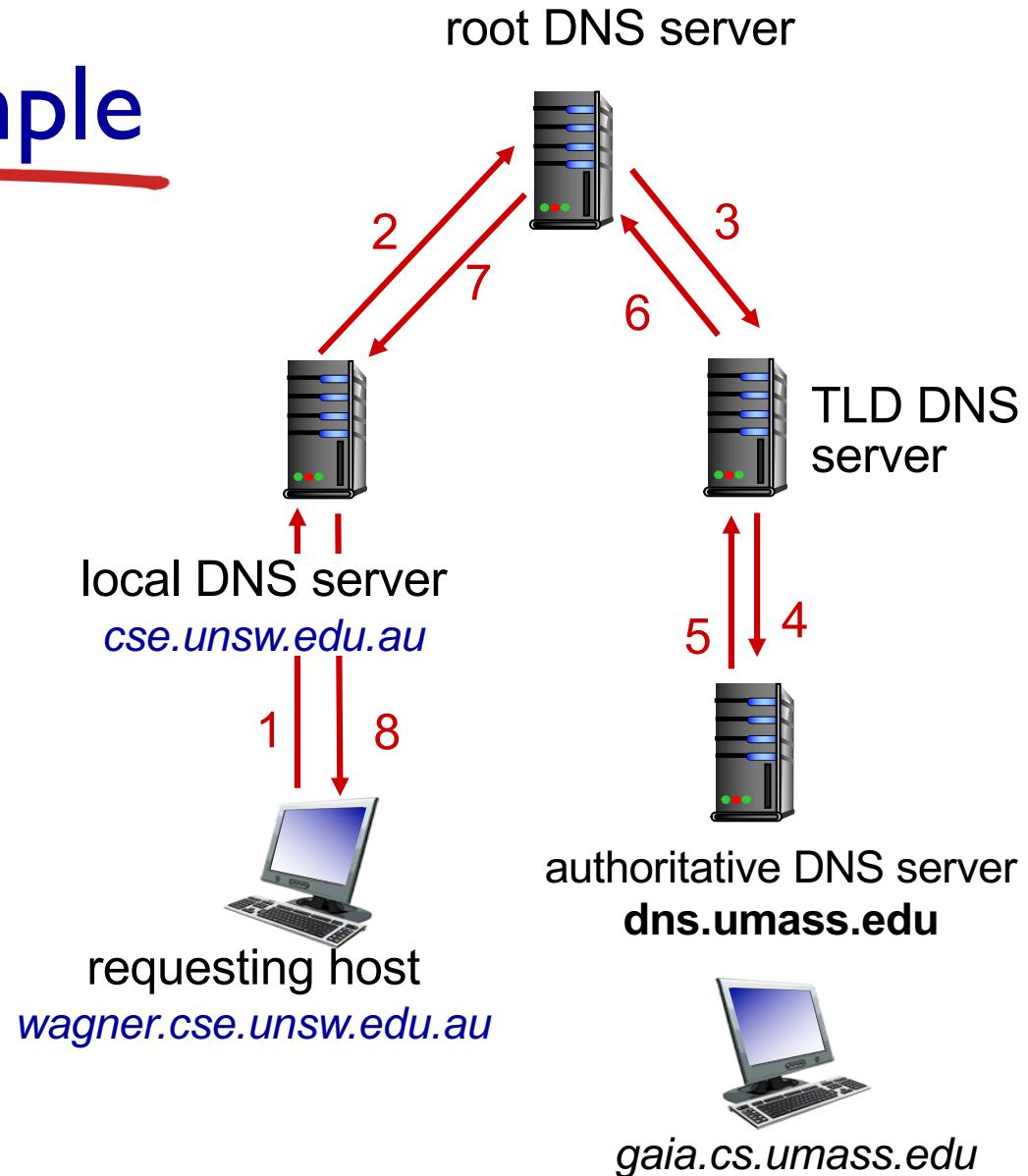
- ❖ contacted server replies with name of server to contact
- ❖ “I don’t know this name, but ask this server”



# 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

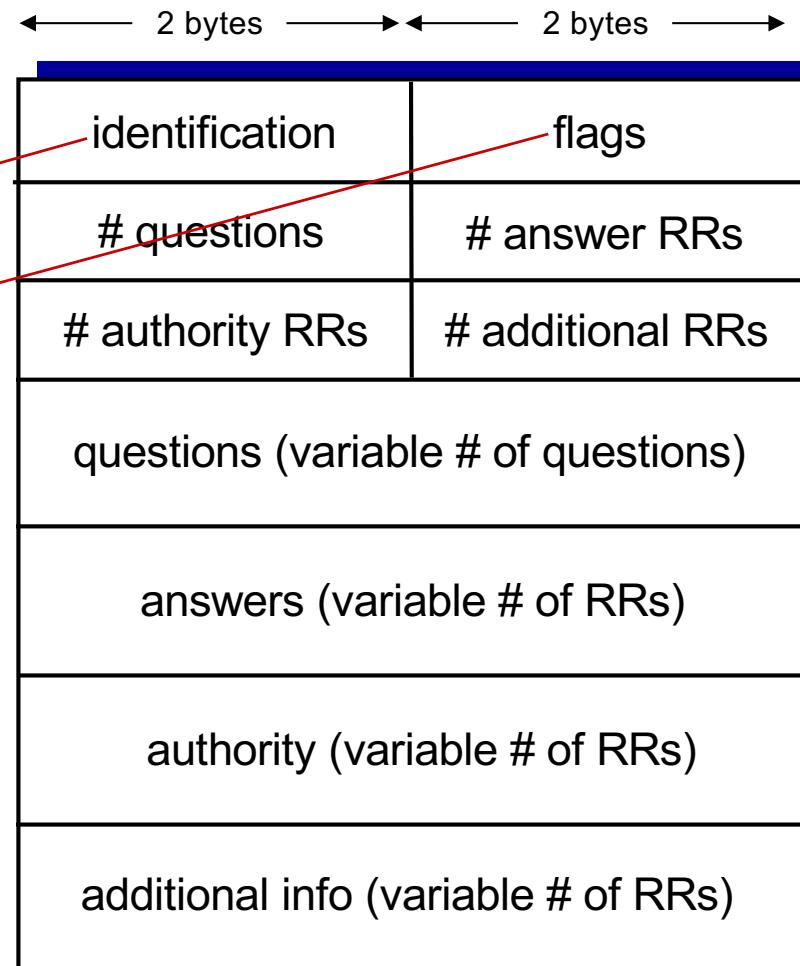
- **value** is name of mailserver  
associated with **name**

# DNS protocol, messages

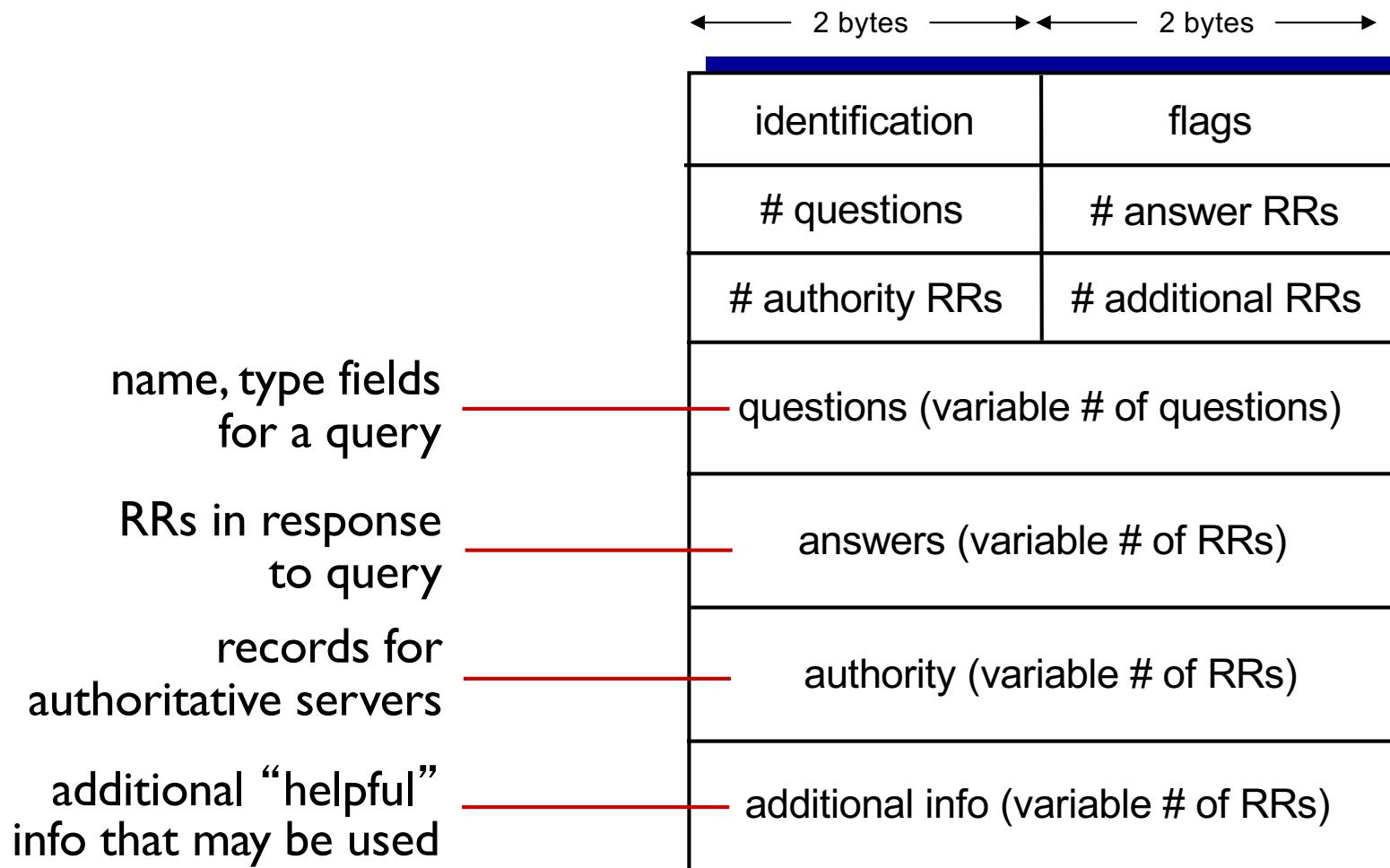
- ❖ *query* and *reply* messages, both with same *message format*

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!

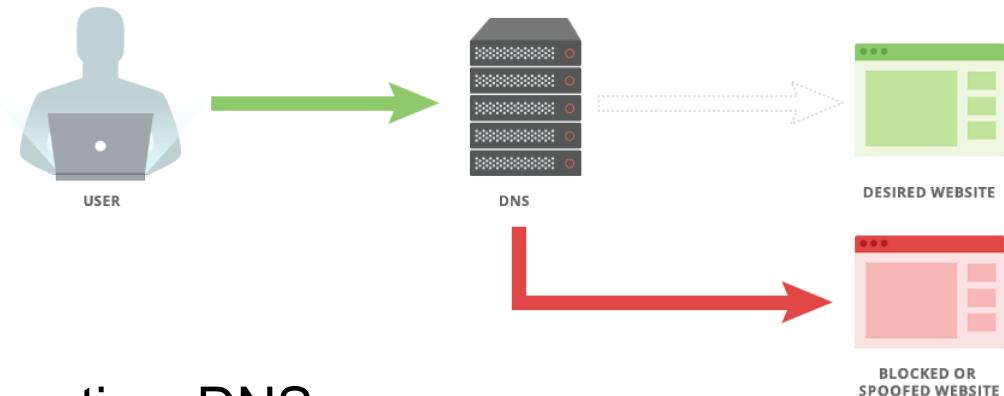
# Reverse DNS



- ❖ 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](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](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 it responds 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

;; ADDITIONAL SECTION:

google.com 600 IN A 129.45.212.222

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

- ❖ 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,

<https://www.dnssec.net>

DNS over HTTPS (DoH):

[https://developers.cloudflare.com/1.1.1.1/dns-over-  
https/](https://developers.cloudflare.com/1.1.1.1/dns-over-https/)

# 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)

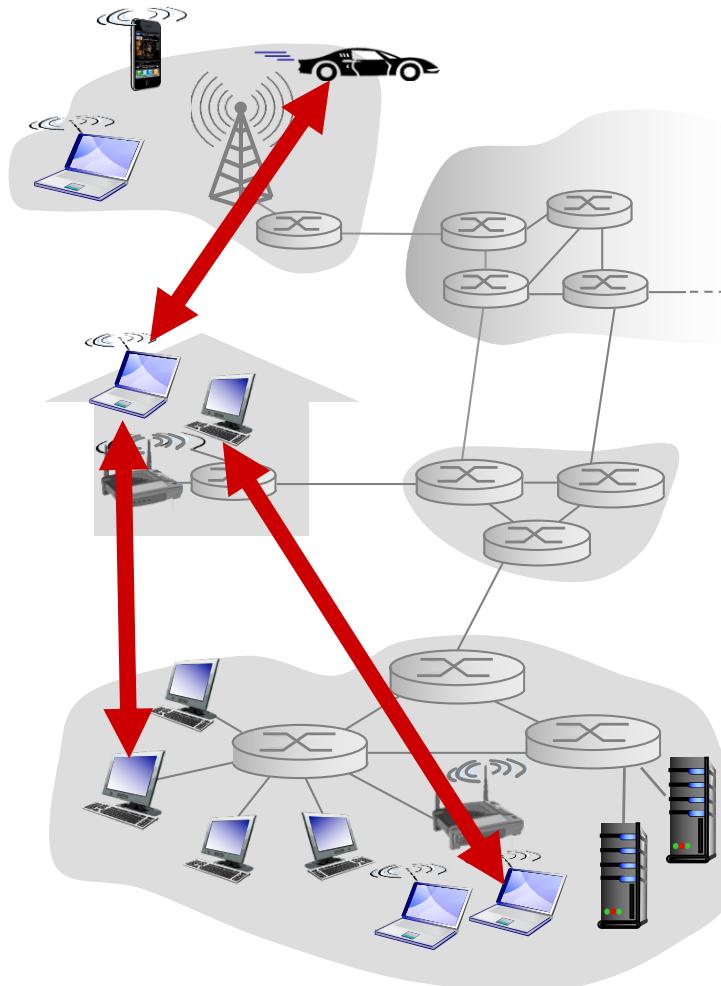
2.7 socket programming with UDP and TCP

# Pure P2P architecture

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

## *examples:*

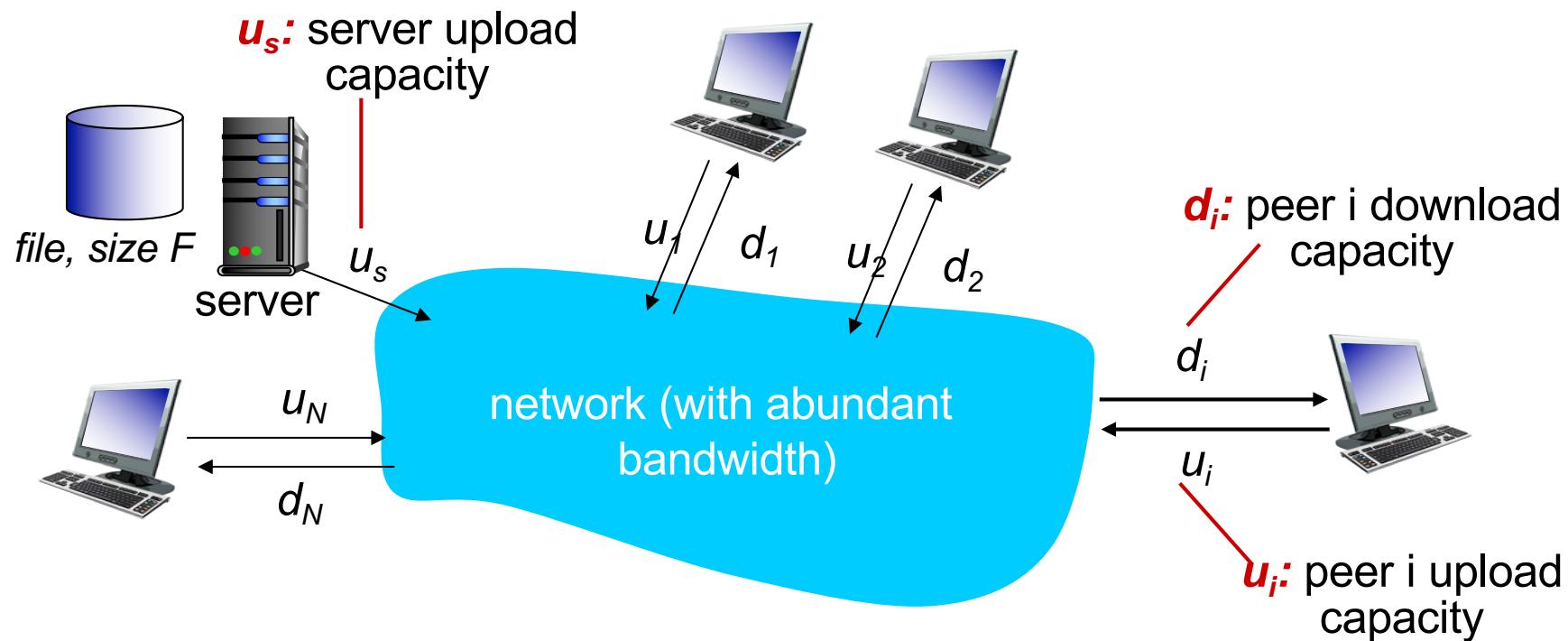
- file distribution  
(BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)
- Cryptocurrency  
(BitCoin)



# File distribution: client-server vs P2P

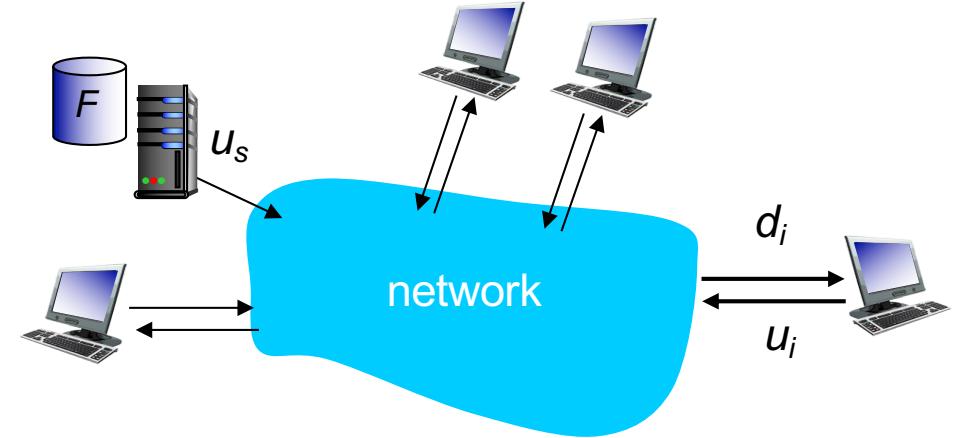
Question: how much time to distribute file (size  $F$ ) from one server to  $N$  peers?

- 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:  $F/d_{\min}$

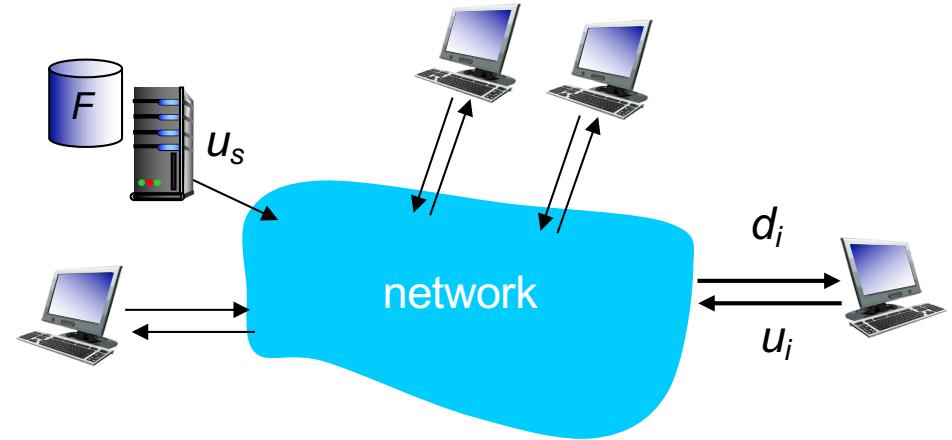
*time to distribute  $F$   
to  $N$  clients using  
client-server approach*

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

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  $u_s + \sum u_i$



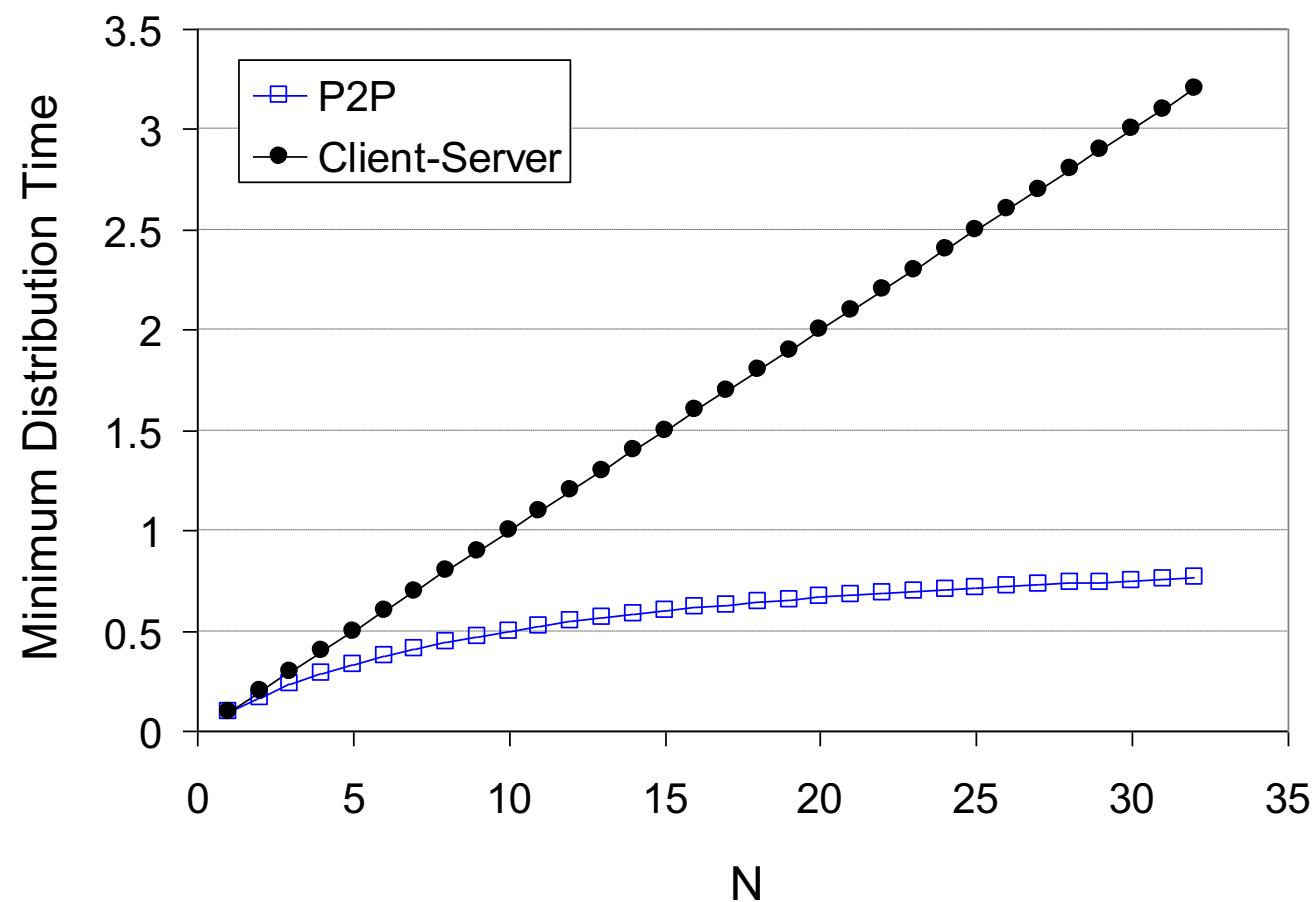
time to distribute  $F$   
to  $N$  clients using  
P2P approach

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

increases linearly in  $N$  ...  
... but so does this, as each peer brings service capacity

# Client-server vs. P2P: example

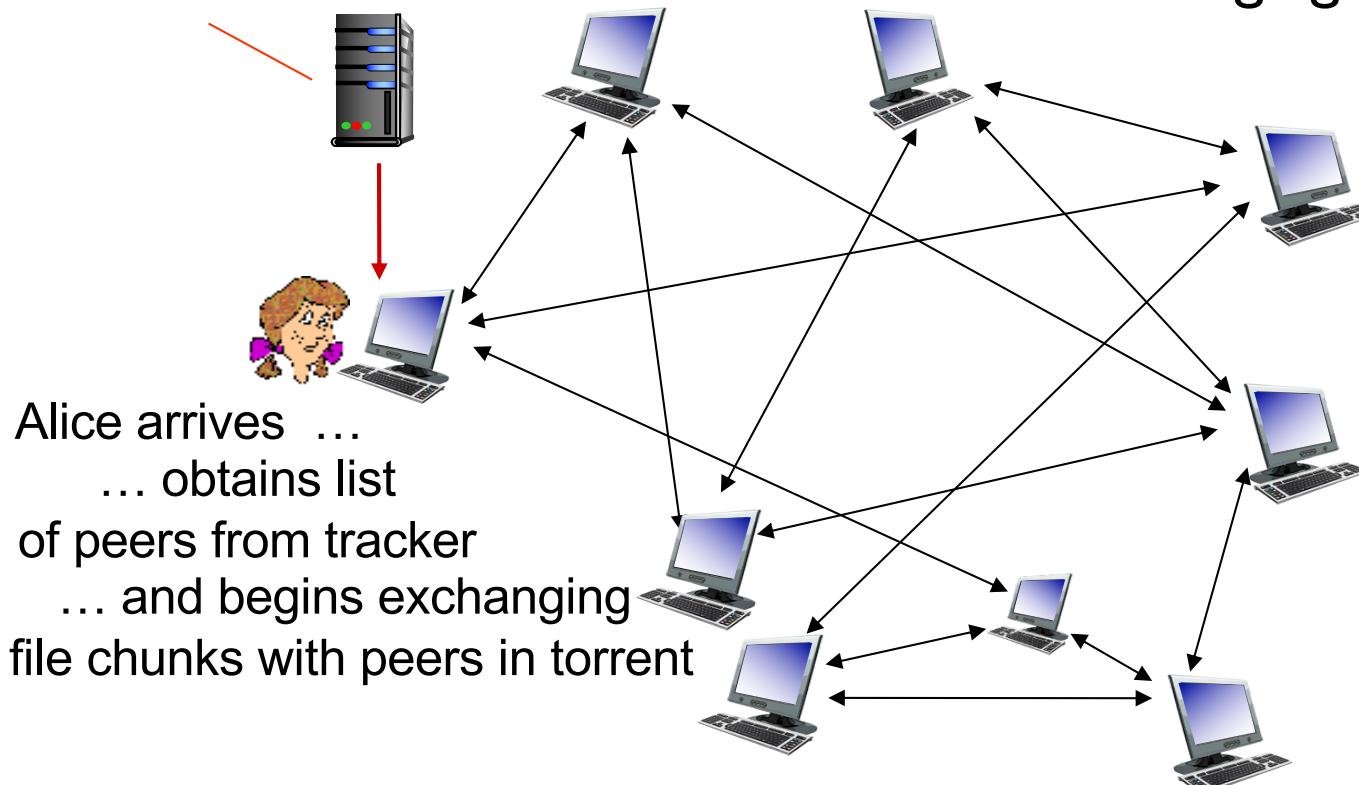
client upload rate =  $u$ ,  $F/u = 1$  hour,  $u_s = 10u$



# 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**

House of Cards Season 4

Walking Dead Season 6

Game of Thrones Season 8

**Trackers**

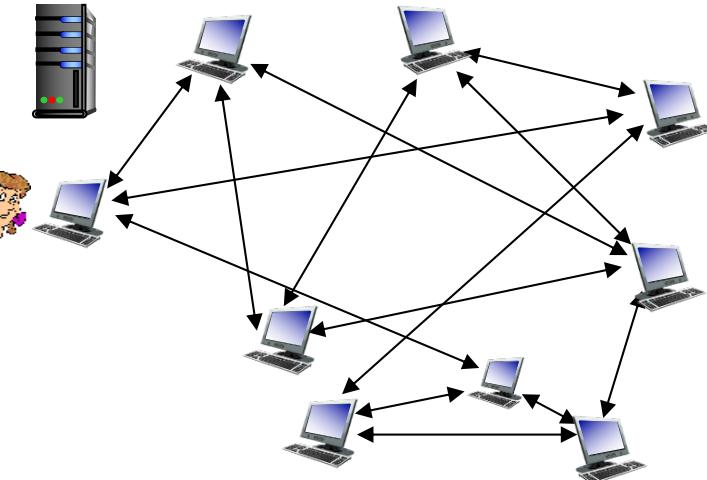
Tracker1-url

Tracker2-url

Tracker2-url, Tracker3-url

# P2P file distribution: BitTorrent

- ❖ peer joining torrent:
  - has no chunks, but will accumulate them over time 
  - 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:*

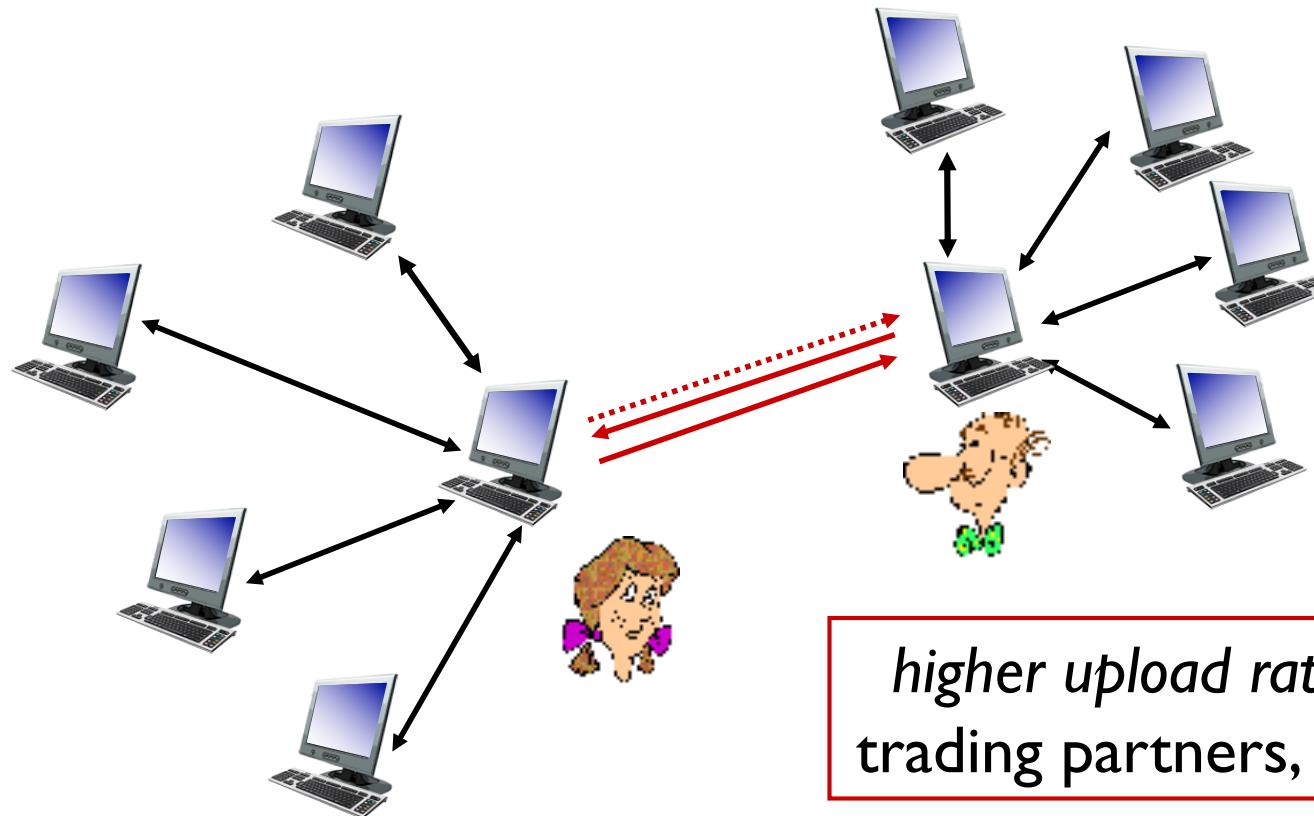
- ❖ at any given time, different peers have different subsets of file chunks
- ❖ periodically, Alice asks each peer for list of chunks that they have
- ❖ Alice requests missing chunks from peers, rarest first
- ❖ **Q:** Why rarest first?

## *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



# Getting rid of the server/tracker

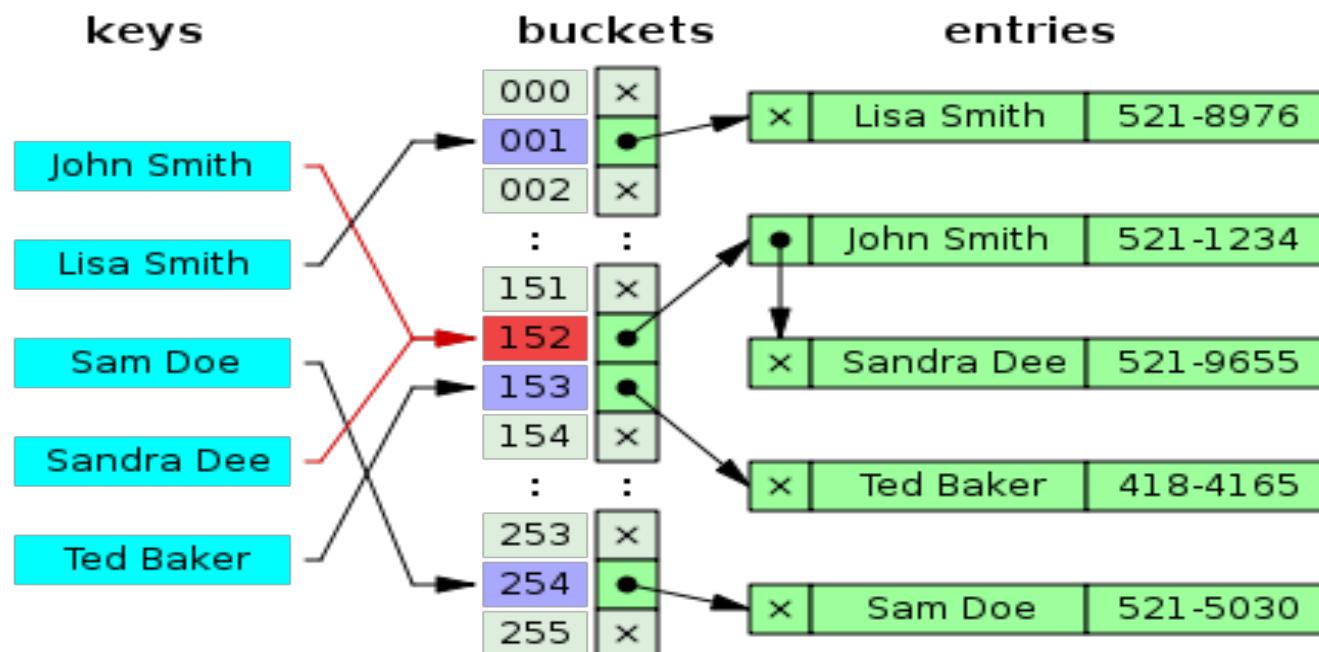
---

- ❖ 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 6<sup>th</sup> Edition of the textbook Section 2.6.2

# Hash table - review

- ❖ Database stores (key,value) pairs
- ❖ Centralised hash table – all (key,value) pairs stored on a single node
- ❖ Distributed hash table – each node has a “section” of (key,value) pairs, i.e., stores a fraction of the database



e.g., Hash(Lisa Smith) = 001

Figure src: [http://en.wikipedia.org/wiki/Hash\\_table](http://en.wikipedia.org/wiki/Hash_table)

# Distributed Hash Table (DHT)

- ❖ DHT: a *distributed P2P database*
- ❖ Database has (**key, value**) pairs; examples:
  - key: *movie name* (e.g., Led Zeppelin IV);
  - value: *IP address* the movie is stored (e.g., 128.17.123.38 )
- ❖ Distribute the (**key, value**) pairs over many peer nodes
- ❖ A peer **queries** DHT with key
  - DHT returns values that match the key
- ❖ Peers can also **insert** (**key, value**) pairs

# Challenges

- ❖ How do we assign (key, value) pairs to nodes?
  - i.e., what fraction of the database should be stored in which node?
- ❖ How do we find them again quickly?
- ❖ What happens if nodes join/leave?

# 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

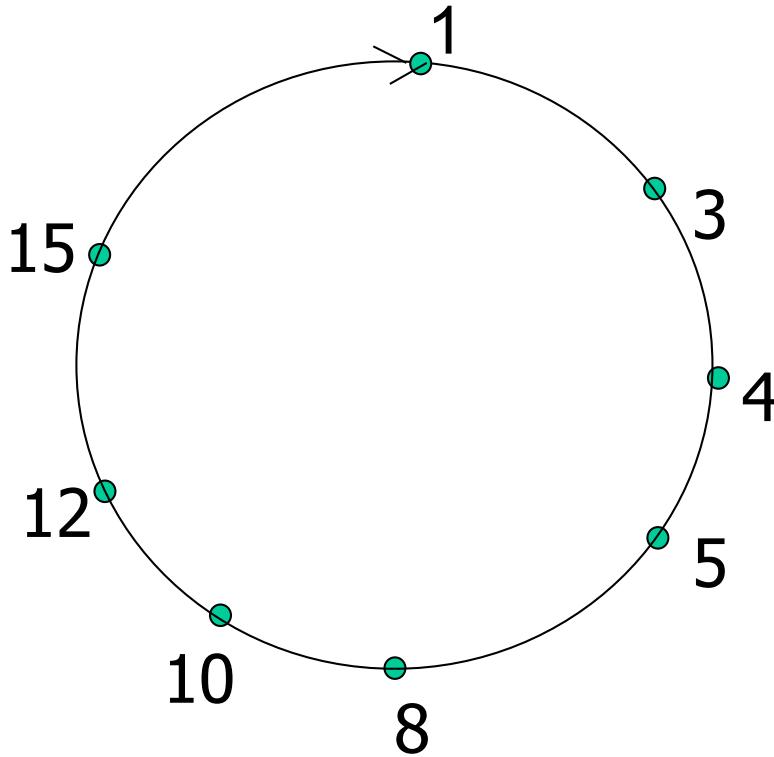
# DHT identifiers: Consistent Hashing

- ❖ 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

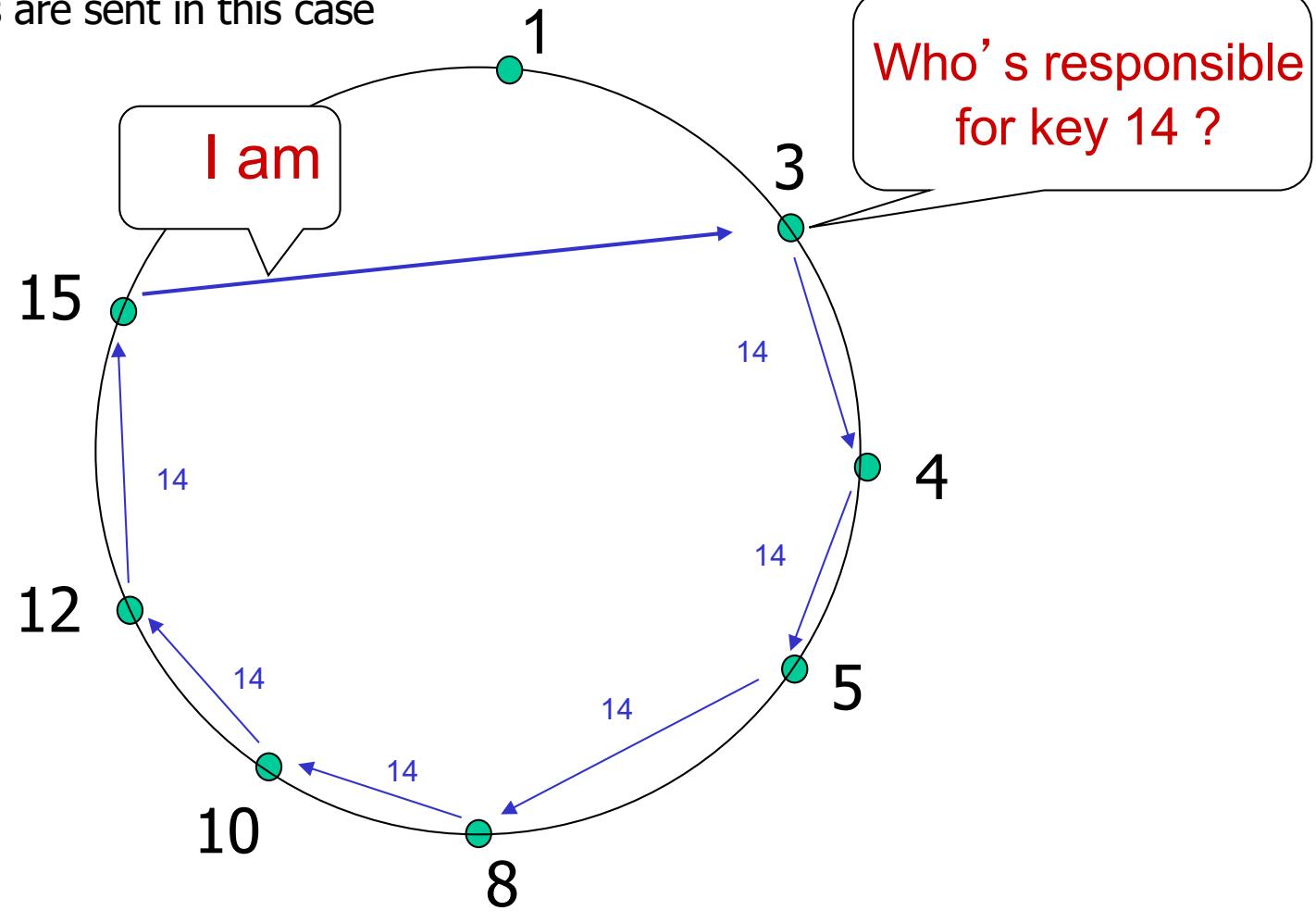
# Circular DHT (I)



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

# Circular DHT (2)

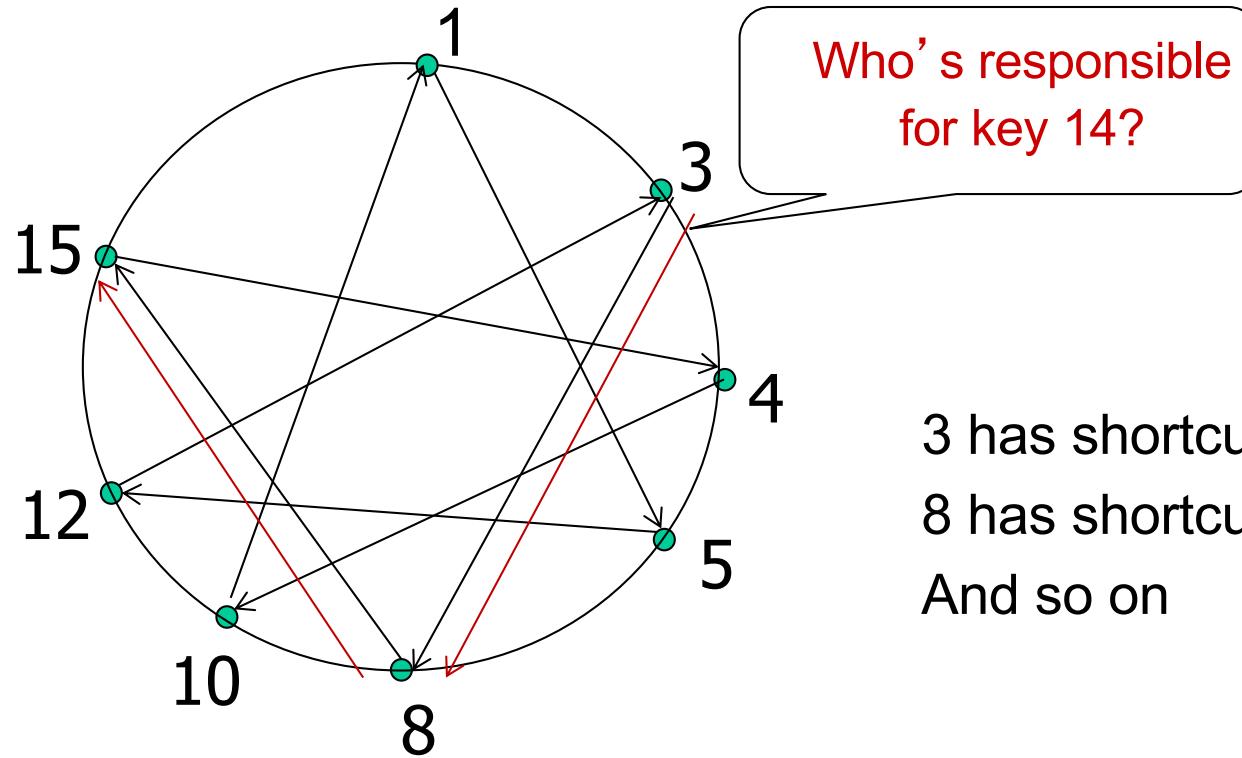
- Each peer maintains 2 neighbours
- 6 query messages are sent in this case



# Tradeoff: #of neighbours vs. #of messages

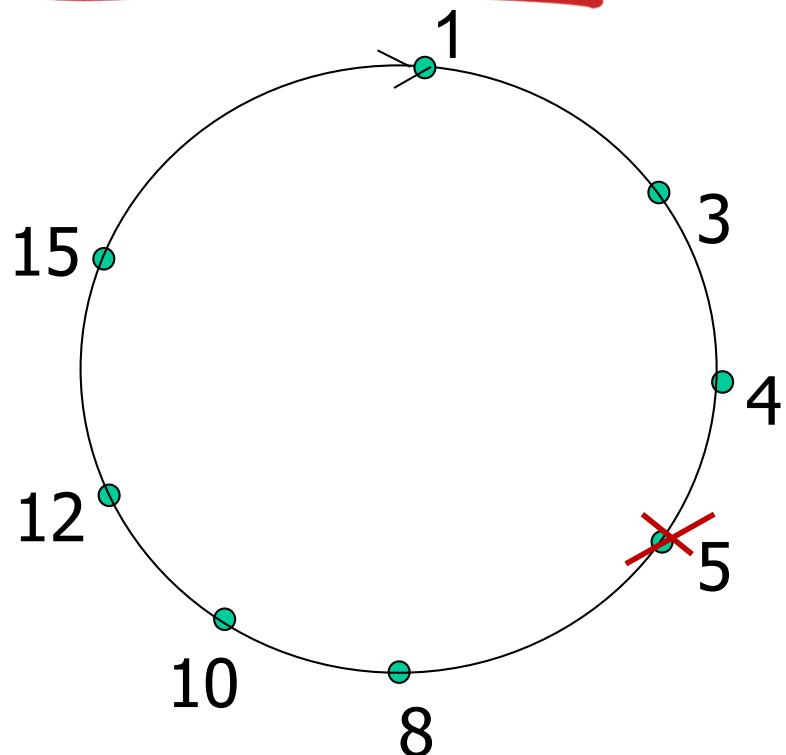
- The smaller the number of neighbours to maintain, the larger the number of messages have to be sent;
- Previous example: 2 neighbours, 6 messages
- Worst case:  $N$  messages
- Average:  $N/2$  messages

# Circular DHT with shortcuts



- ❖ each peer keeps track of IP addresses of predecessor, successor, and *shortcuts*
- ❖ reduced from 6 to 2 messages.
- ❖ possible to design shortcuts so  $O(\log N)$  neighbours,  $O(\log N)$  messages in query

# Peer churn



## handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its **two** successors
- ❖ each peer periodically pings its two successors to check aliveness
- ❖ if immediate successor leaves, choose next successor as new immediate successor

*example: peer 5 abruptly leaves*

- ❖ peer 4 detects peer 5 departure; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
- ❖ 8 makes 4 its new predecessor

# More DHT info

- ❖ How do nodes join?
- ❖ How does cryptographic hashing work?
- ❖ How much state does each node store?

Research Article

CHORD: A Scalable Peer-to-Peer Lookup Service for Internet Applications

# DHT: Applications

- ❖ File sharing and backup [CFS, Ivy, OceanStore, PAST, Pastiche ...]
- ❖ Web cache and replica [Squirrel, Croquet Media Player]
- ❖ Censor-resistant stores [Eternity]
- ❖ DB query and indexing [PIER, Place Lab, VPN Index]
- ❖ Event notification [Scribe]
- ❖ Naming systems [ChordDNS, Twine, INS, HIP]
- ❖ Distributed BitTorrent tracker [Kademlia, Vuze]
- ❖ Communication primitives [I3, ...]
- ❖ Host mobility [DTN Tetherless Architecture]

# 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

# 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  $i$

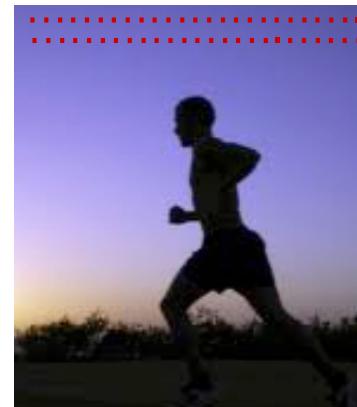


frame  $i+1$

# 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  $i$

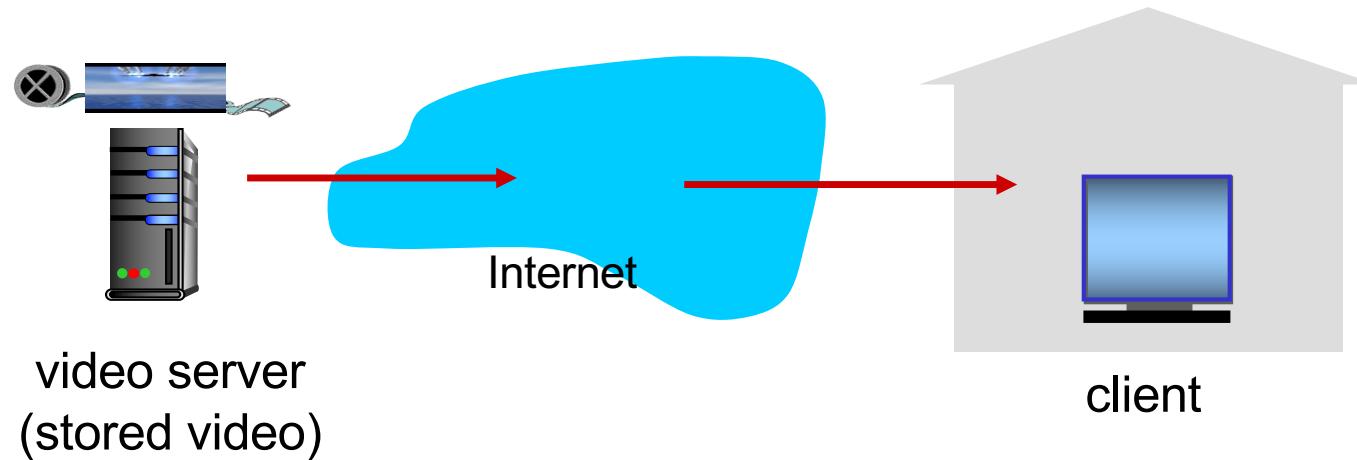
*temporal coding example:*  
instead of sending complete frame at  $i+1$ ,  
send only differences from frame  $i$



frame  $i+1$

# Streaming stored video:

simple scenario:

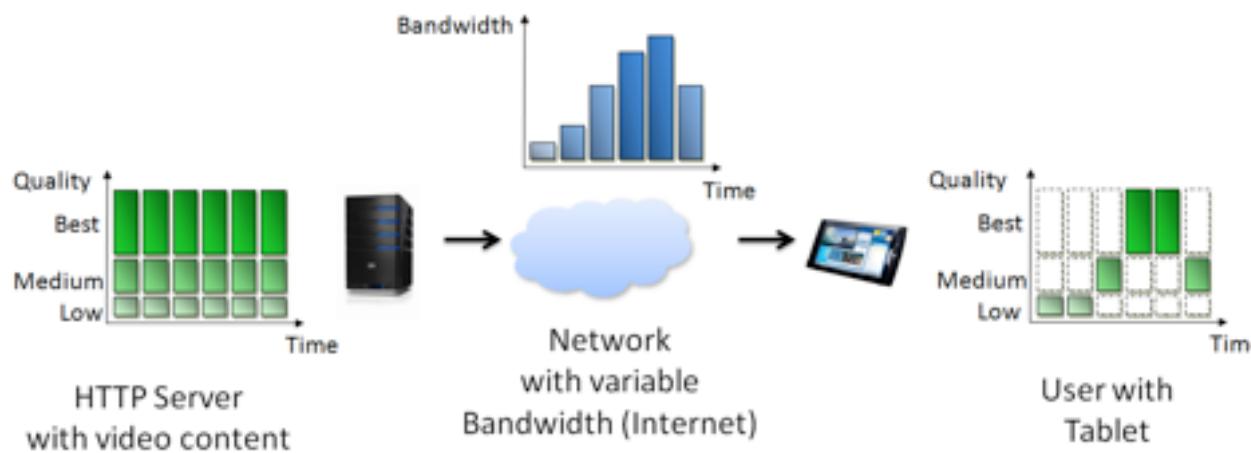


# Streaming multimedia: DASH

- ❖ *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 (CDNs)

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

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

# Content Distribution Networks (CDNs)

- ❖ *challenge*: how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?
- ❖ *option 2*: store/serve multiple copies of videos at multiple geographically distributed sites (*CDN*)
  - *enter deep*: push CDN servers deep into many access networks
    - close to users
    - 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

# 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.

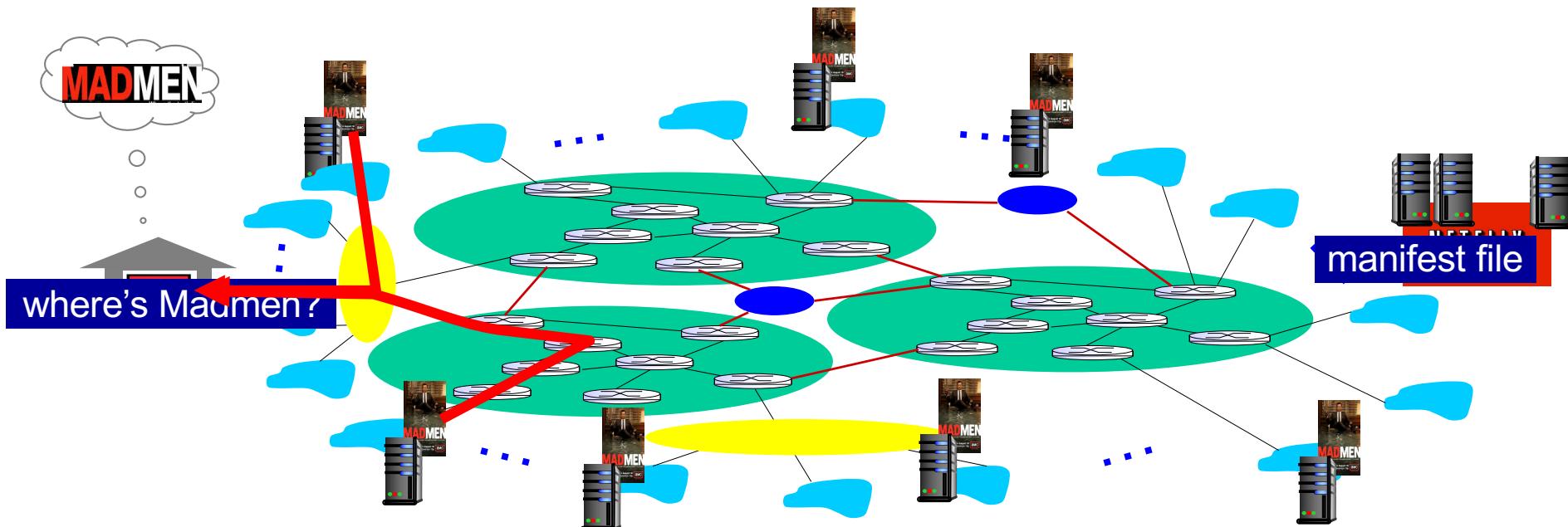
;; ADDITIONAL SECTION:
a0dscb.akamaiedge.net. 7144    IN      AAAA   2600:1480:e800::c0
n0dscb.akamaiedge.net. 3048    IN      A       88.221.81.193
n1dscb.akamaiedge.net. 2752    IN      A       88.221.81.194
n2dscb.akamaiedge.net. 1380    IN      A       104.72.70.167
n3dscb.akamaiedge.net. 3048    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 (CDNs)

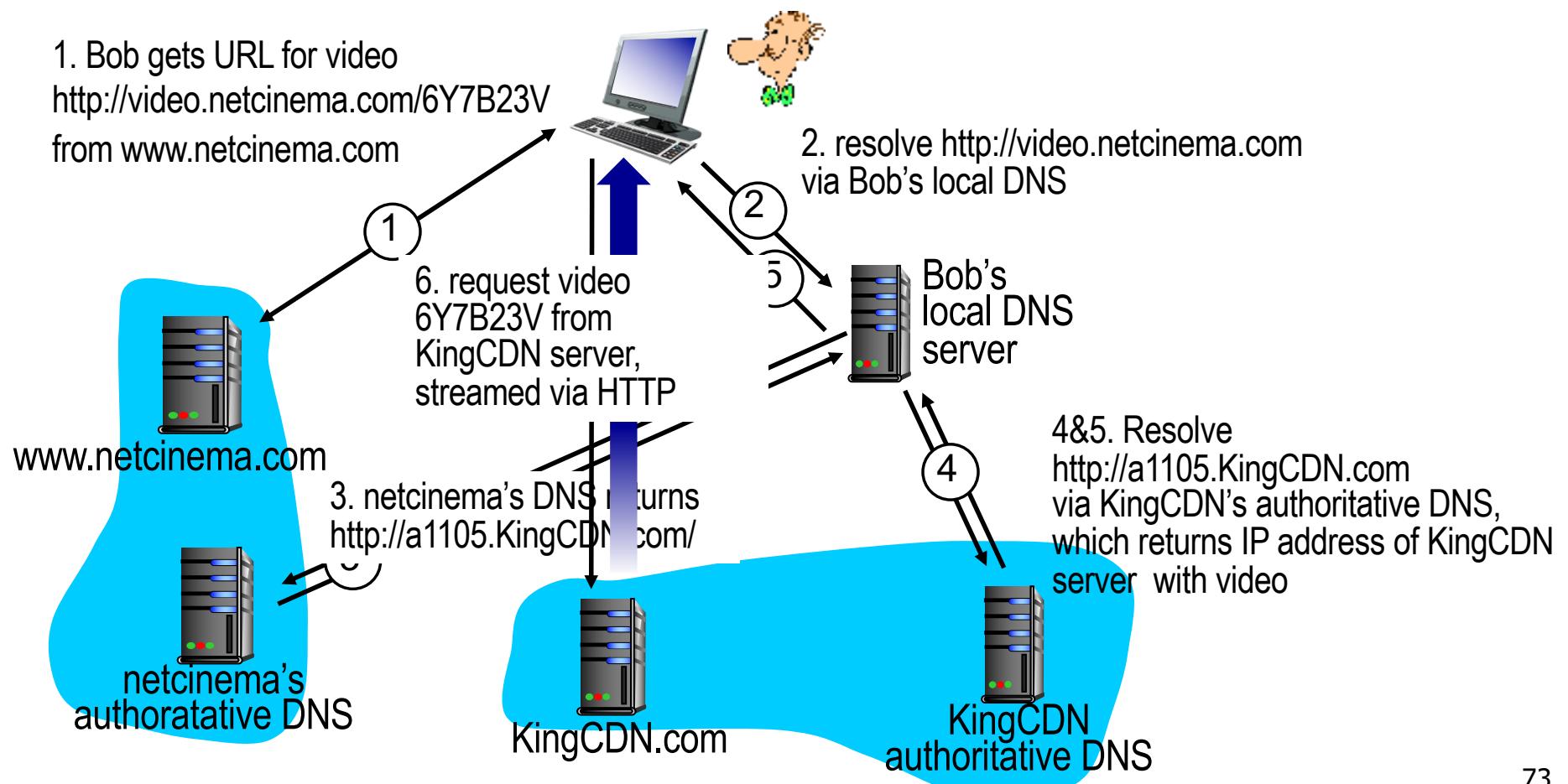
- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested



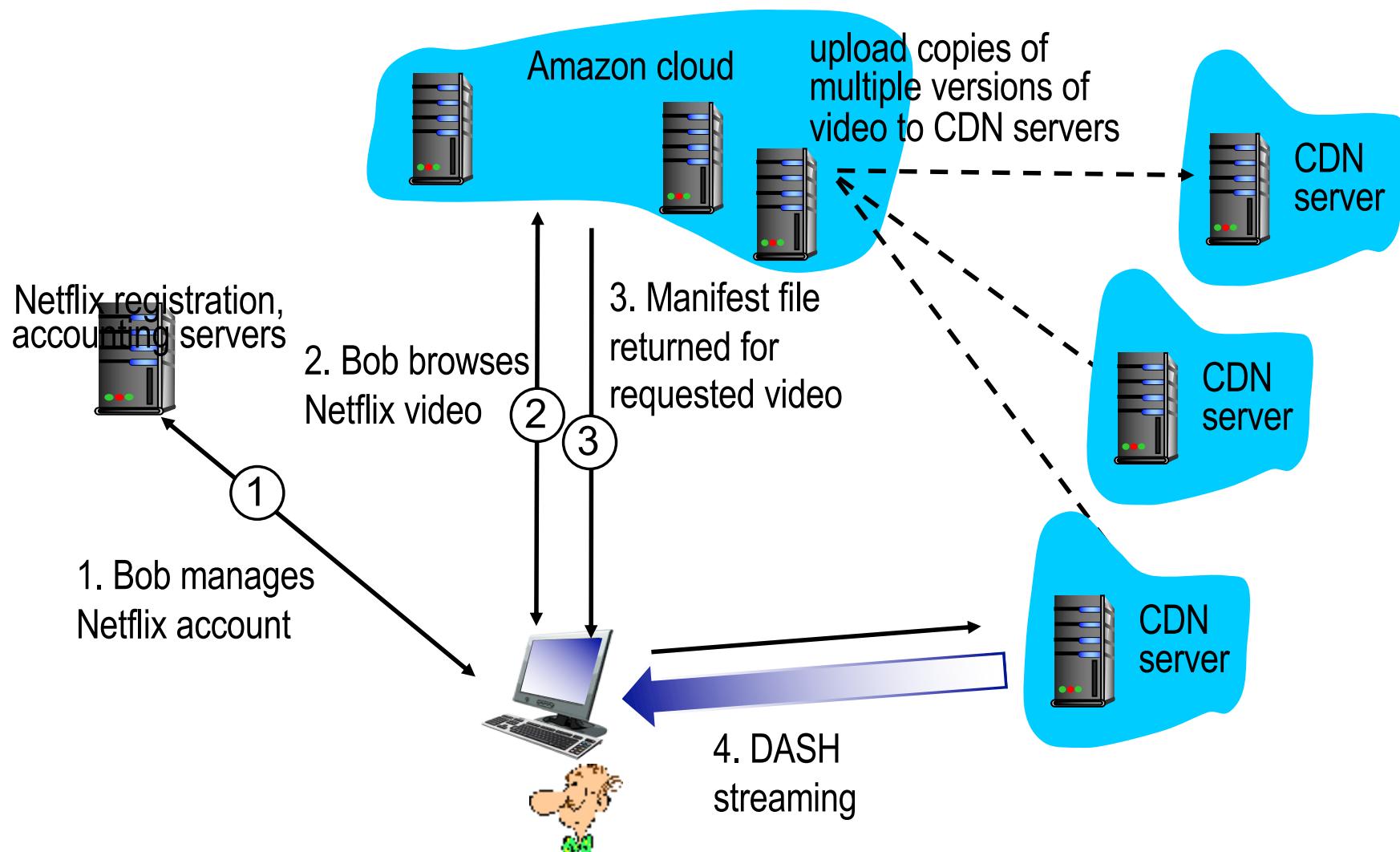
# CDN content access: a closer look

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

- video stored in CDN at managed by KingCDN.com

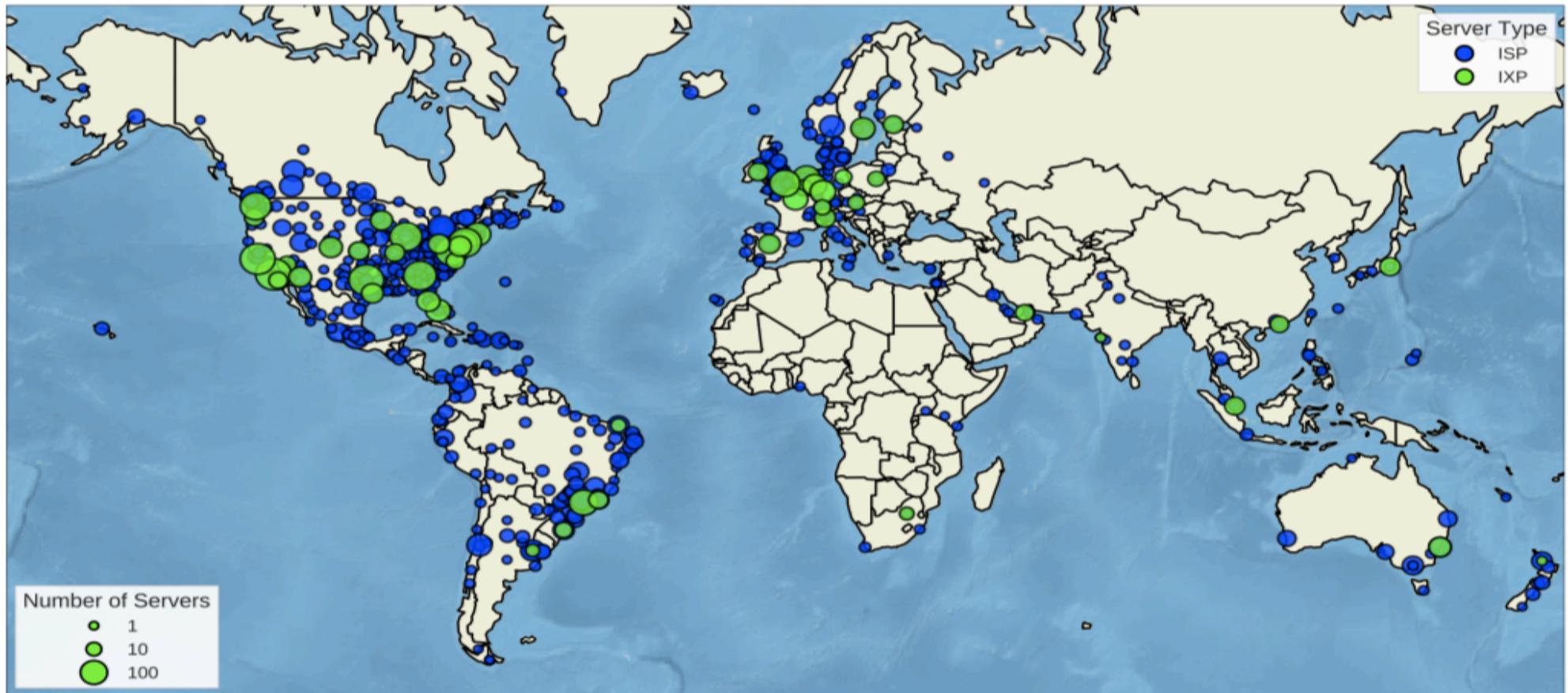


# Case study: Netflix



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.

## 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)

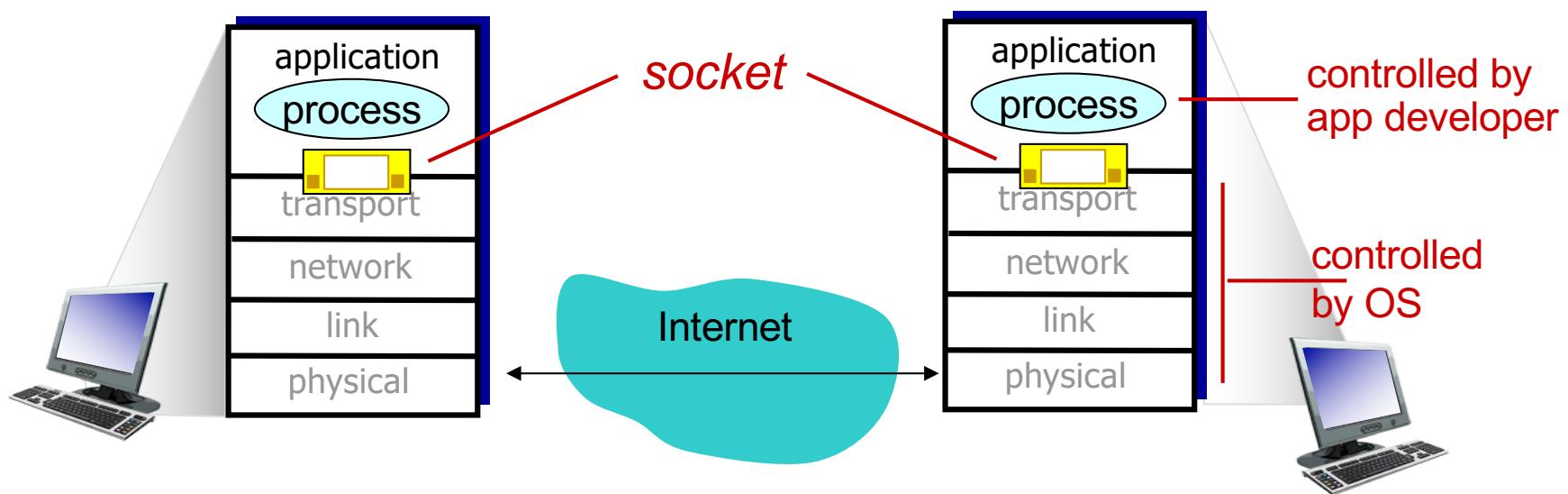
### 2.7 socket programming with UDP and TCP

Please see example code (C, Java, Python) on course website  
Labs 2 & 3 include a socket programming exercise

# Socket programming

**goal:** learn how to build client/server applications that communicate using sockets

**socket:** door between application process and end-end-transport protocol



# Socket programming with UDP

UDP: no “connection” between client & server

- ❖ no handshaking before sending data
- ❖ sender explicitly attaches IP destination address and port # to each packet
- ❖ rcvr extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- ❖ UDP provides *unreliable* transfer of groups of bytes (“datagrams”) between client and server

# Pseudo code UDP client

- ❖ Create socket
- ❖ Loop
  - (Send UDP datagram to known port of server)
  - (Receive UDP datagram as a response from server)
- ❖ Close socket

# Pseudo code UDP server

- ❖ Create socket
- ❖ Bind socket to a specific port where clients can contact you
- ❖ Loop
  - (Receive UDP datagram from client X)
  - (Send UDP datagram as reply to client X)
- ❖ Close socket

# Socket programming with TCP

## client must contact server

- ❖ server process must first be running
- ❖ server must have created socket (door) that welcomes client's contact

## client contacts server by:

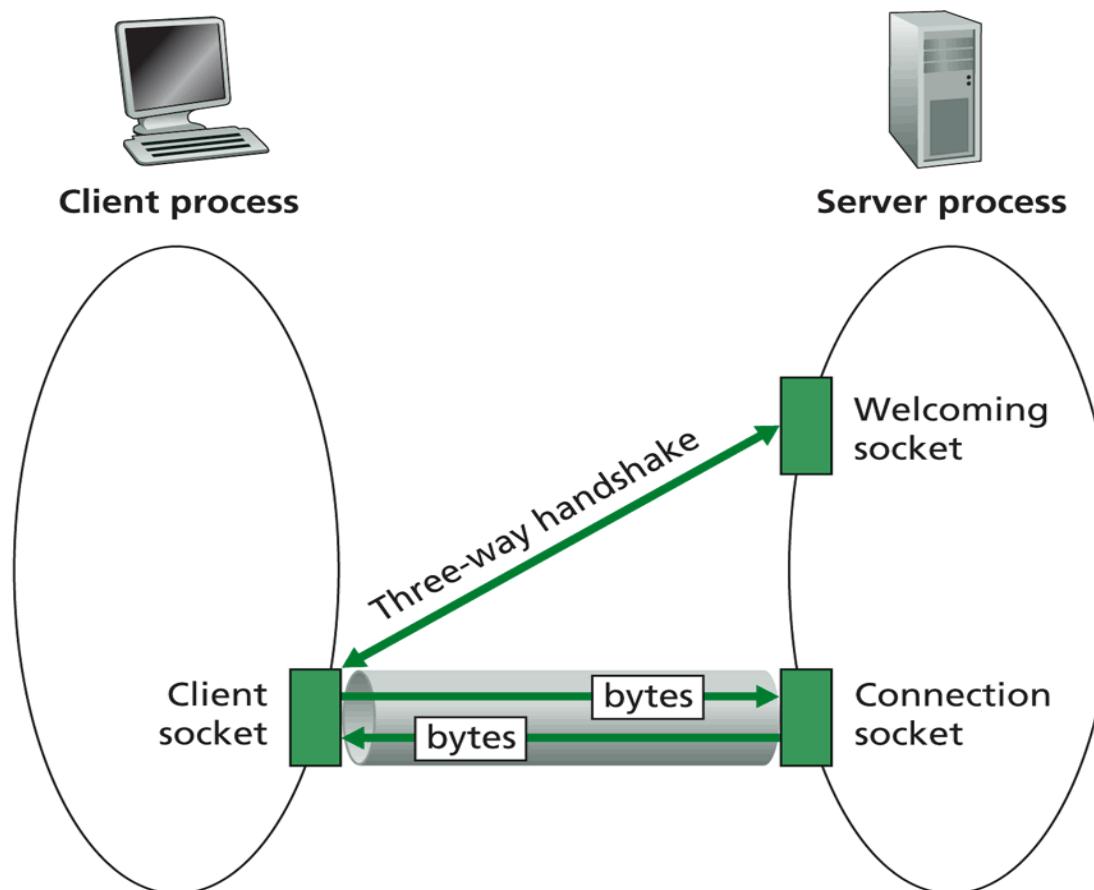
- ❖ Creating TCP socket, specifying IP address, port number of server process
- ❖ *when client creates socket:* client TCP establishes connection to server TCP

- ❖ when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more later)

## application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

# TCP Sockets



# Pseudo code TCP client

- ❖ Create socket (`ConnectionSocket`)
- ❖ Do an active connect specifying the IP address and port number of server
- ❖ Read and write data into `ConnectionSocket` to communicate with client
- ❖ Close `ConnectionSocket`

# Pseudo code TCP server

- ❖ Create socket (WelcomingSocket)
- ❖ Bind socket to a specific port where clients can contact you
- ❖ Register with the OS your willingness to listen on that socket for clients to contact you
- ❖ Loop
  - Accept new connection(ConnectionSocket)
  - Read and write data into ConnectionSocket to communicate with client
  - Close ConnectionSocket
- ❖ Close WelcomingSocket

# Queues

- ❖ While the server socket is busy, incoming connection requests are stored in a queue
- ❖ Once the queue fills up, further incoming connections are refused
- ❖ This is clearly a problem
  - Example: HTTP servers
- ❖ Solution
  - Concurrency

# Concurrent TCP Servers

- ❖ Benefit comes in ability to hand off interaction with a client to another process
- ❖ Parent process creates the WelcomingSocket and waits for clients to request connection
- ❖ When a connection request is received, fork off a child process to handle that connection so that the parent process can return to waiting for connections as soon as possible
- ❖ Multithreaded server: same idea, just spawn off another thread rather than a process