Topic 6 – Applications

- Infrastructure Services (DNS)
  - Now with added security…
- Traditional Applications (web)
  - Now with added QUIC
- P2P Networks
  - Every device serves

Client-server paradigm

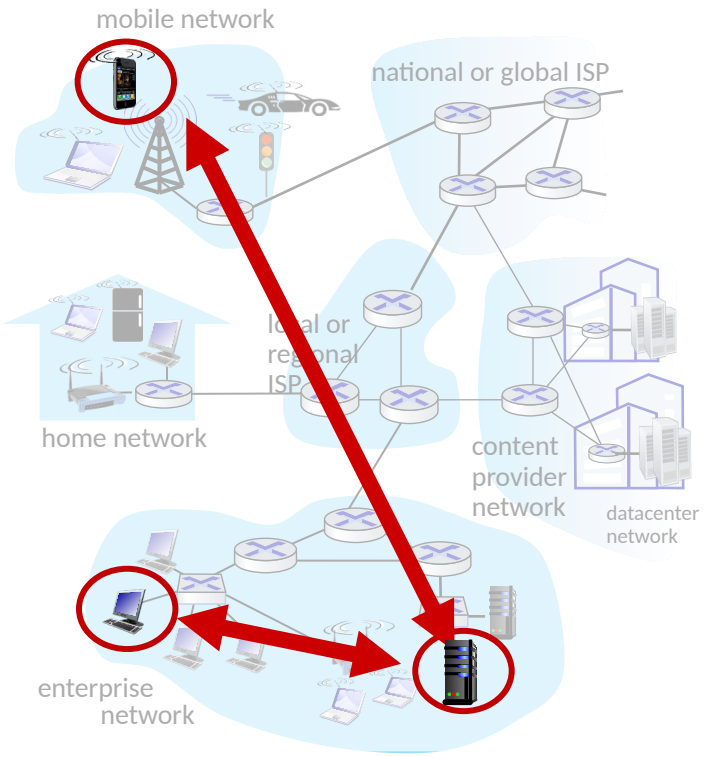
server:

- always-on host
- permanent IP address
- often in data centers, for scaling

clients:

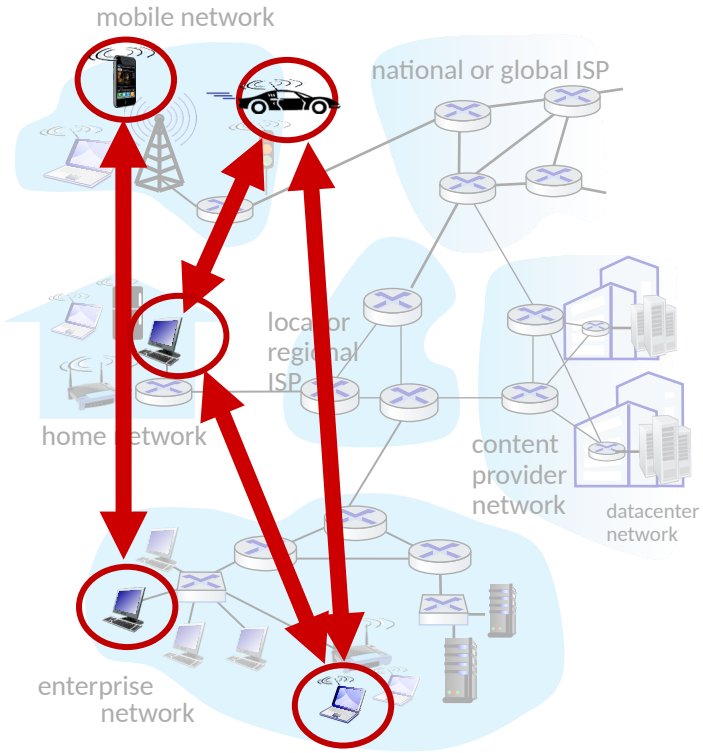
- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other

- examples: HTTP, IMAP, FTP



Peer-peer 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, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management
- example: P2P file sharing



An application-layer protocol defines:

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

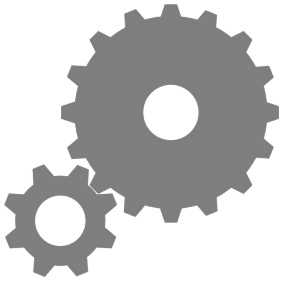
open protocols:

- defined in RFCs, everyone has access to protocol definition
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype, Zoom, ‘Call of Death’, ‘The Sims’ …

Just like Ib ‘Distributed Systems’ course !



Relationship between Names & Addresses

- Addresses can change underneath
  - Move www.bbc.co.uk to 212.58.246.92
  - Humans/apps should be unaffected

- Name could map to multiple IP addresses
  - www.bbc.co.uk to multiple replicas of the Web site
  - Enables
    - Load-balancing
    - Reducing latency by picking nearby servers

- Multiple names for the same address
  - E.g., aliases like www.bbc.co.uk and bbc.co.uk
  - Mnemonic stable name, and dynamic canonical name
    - Canonical name = actual name of host

DNS: Domain Name Resolver

people: many identifiers:

- NI no, name, passport no

Internet hosts, routers:

- IP address (32 bit or 128bit) - used for addressing datagrams
- "name", e.g., cam.ac.uk- used by humans

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

Domain Name System (DNS):

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

DNS: services, structure

DNS services:

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

Q: Why not centralize DNS?

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

A: doesn't scale!

- Comcast DNS servers alone: 770B DNS queries/day
- Akamai DNS servers alone: 2.6T DNS queries/day

Thinking about the DNS

humongous distributed database:

- ~ billion records, each simple

handles many *trillions* of queries/day:

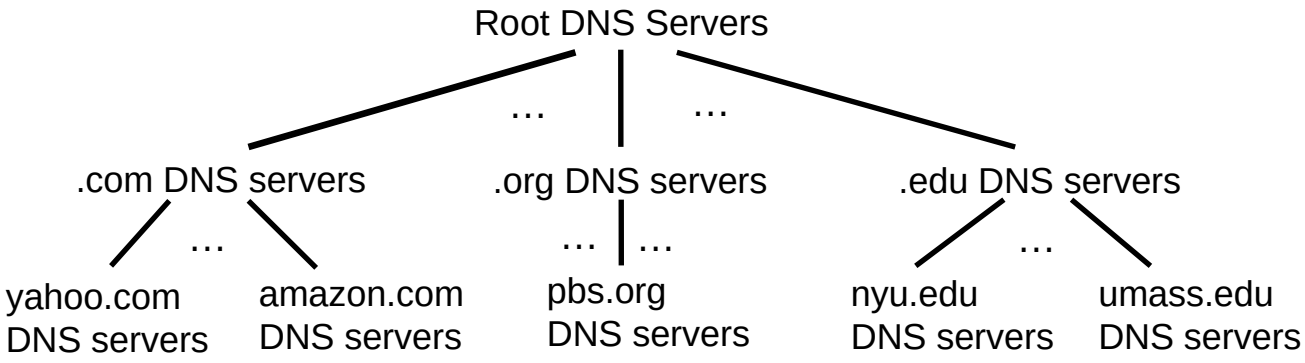
- *many* more reads than writes
- *performance matters:* almost every Internet transaction interacts with DNS - msecs count!

organizationally, physically decentralized:

- millions of different organizations responsible for their records

"bulletproof": reliability, security

DNS: a distributed, hierarchical database

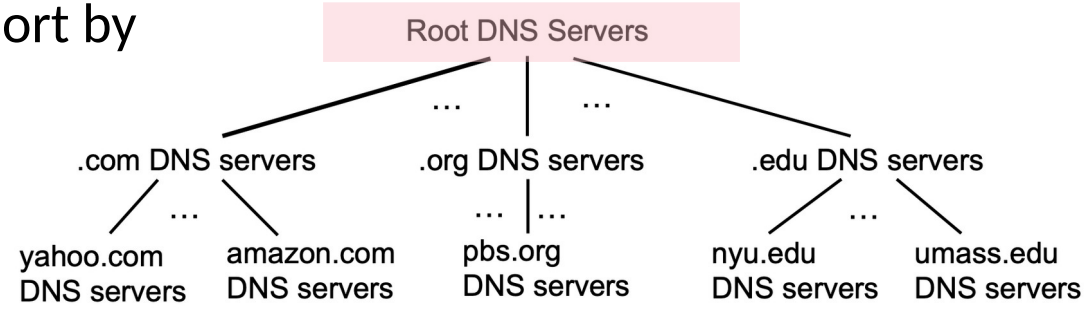


Client wants IP address for www.amazon.com; 1st approximation:

- 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

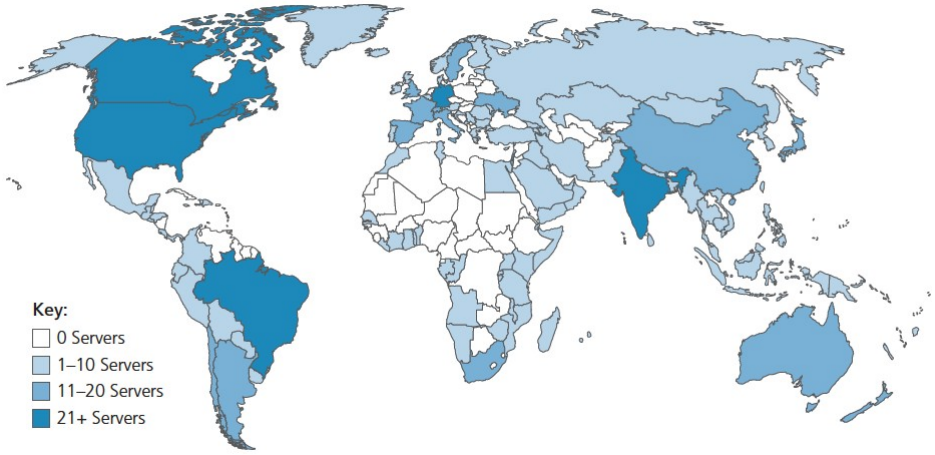
- official, contact-of-last-resort by name servers that cannot resolve name



DNS: root name servers

- official, contact-of-last-resort by name servers that can not resolve name
- *incredibly important* Internet function
  - Internet couldn't function without it!
  - DNSSEC – provides security (authentication, message integrity)
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain

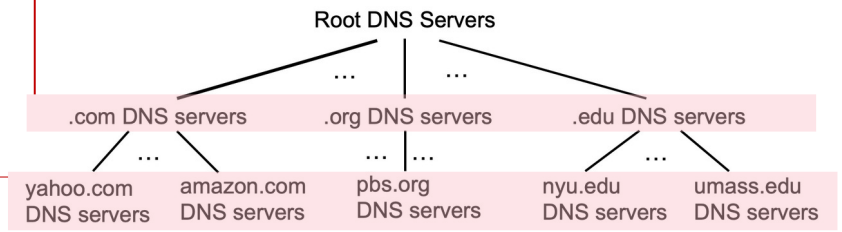
13 logical root name "servers" worldwide each "server" replicated many times (~200 servers in US)



Top-Level Domain, and authoritative servers

Top-Level Domain (TLD) servers:

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



authoritative DNS servers:

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

Using DNS

- Two components
  - DNS servers spread out across the globe
  - Resolver software on each host

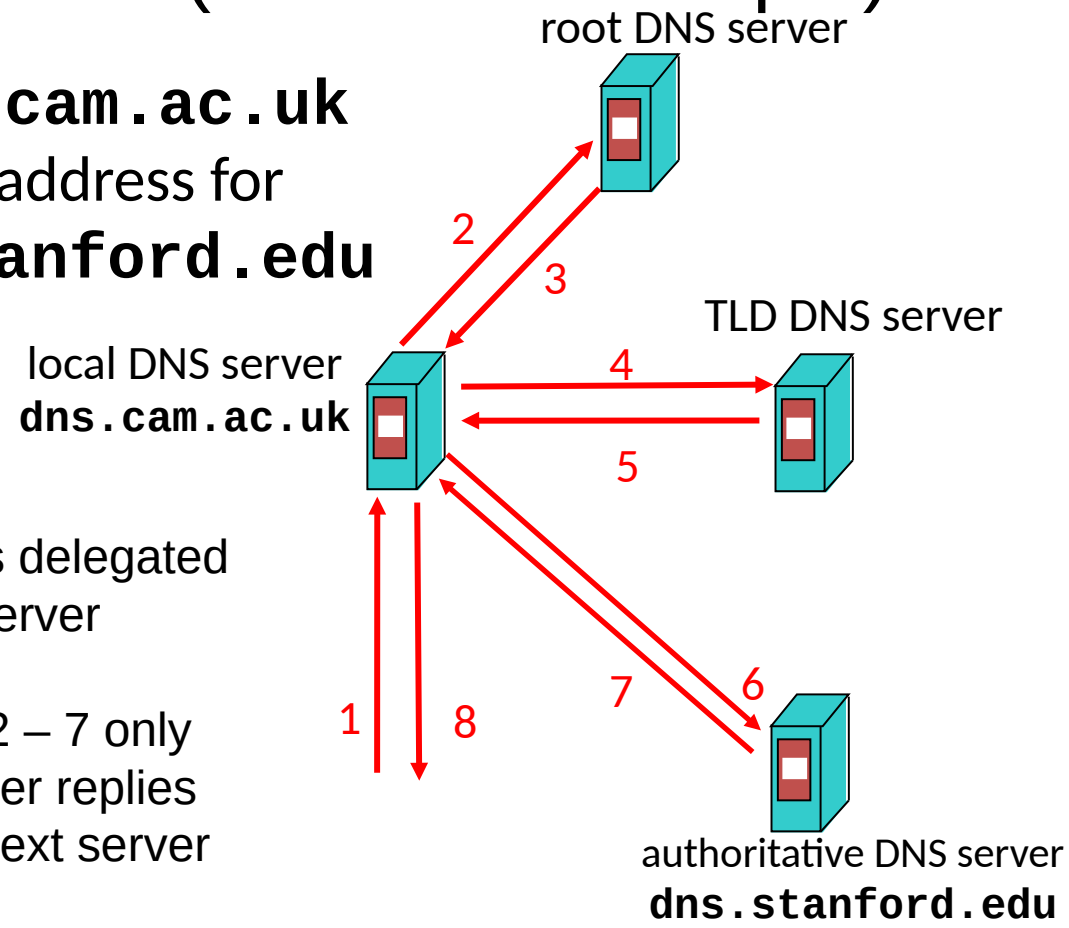
- Client application
  - Extract server name (e.g., from the URL)
  - Invoke O/S gethostbyname() to trigger resolver code

Local DNS name servers

- when host makes DNS query, it is sent to its *local* DNS server
  - Local DNS server returns reply, answering:
    - from its local cache of recent name-to-address translation pairs (possibly out of date!)
    - forwarding request into DNS hierarchy for resolution
  - each ISP has local DNS name server; to find yours:
    - MacOS: `% scutil --dns`
    - Windows: `>ipconfig /all`
    - `Linux: resolvectl status`
- local DNS server doesn't strictly belong to hierarchy, acting as they do, on behalf of other hosts.

How Does Resolution Happen?
(**Iterative** example)

Host at **`cl.cam.ac.uk`**
wants IP address for
`www.stanford.edu`



iterated query:

- Host enquiry is delegated to local DNS server
- Consider transactions 2 – 7 only
- contacted server replies with name of next server to contact
- "I don't know this name, but ask this server"

DNS Caching

- Performing all these queries takes time
  - And all this before actual communication takes place
  - E.g., 1-second latency before starting Web download
- Caching greatly reduces overhead
  - The top-level servers very rarely change
  - Popular sites (e.g., www.bbc.co.uk) visited often
  - Local DNS servers have regularly-used information cached
- How DNS caching works
  - DNS servers will cache responses to queries
  - Responses include a "time to live" (TTL) field
  - Server deletes cached entry after TTL expires
  - Cached entries may be *out-of-date*
    - if named host changes IP address, may not be known Internet-wide until all TTLs expire!
    - *best-effort name-to-address translation!*

Reliability

- DNS servers are replicated
  - Name service available if at least one replica is up
  - Queries can be load-balanced between replicas
- Anycast provides reliability for ROOT servers
- Usually, UDP is used for queries
  - Need reliability: must implement this on top of UDP
  - DNS spec. supports TCP too, but not always available
- Try alternate servers on timeout
  - Exponential backoff when retrying same server
- Same identifier for all queries
  - Don’t care which server responds

DNS records

DNS: distributed database storing resource records (RR)

RR format: (`name, value, type, ttl`)

type=A

- `name` is hostname
- `value` is IP address

type=NS

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

type=CNAME

- `name` is alias name for some "canonical" (the real) name
- www.ibm.com is really servereast.backup2.ibm.com
- `value` is canonical name

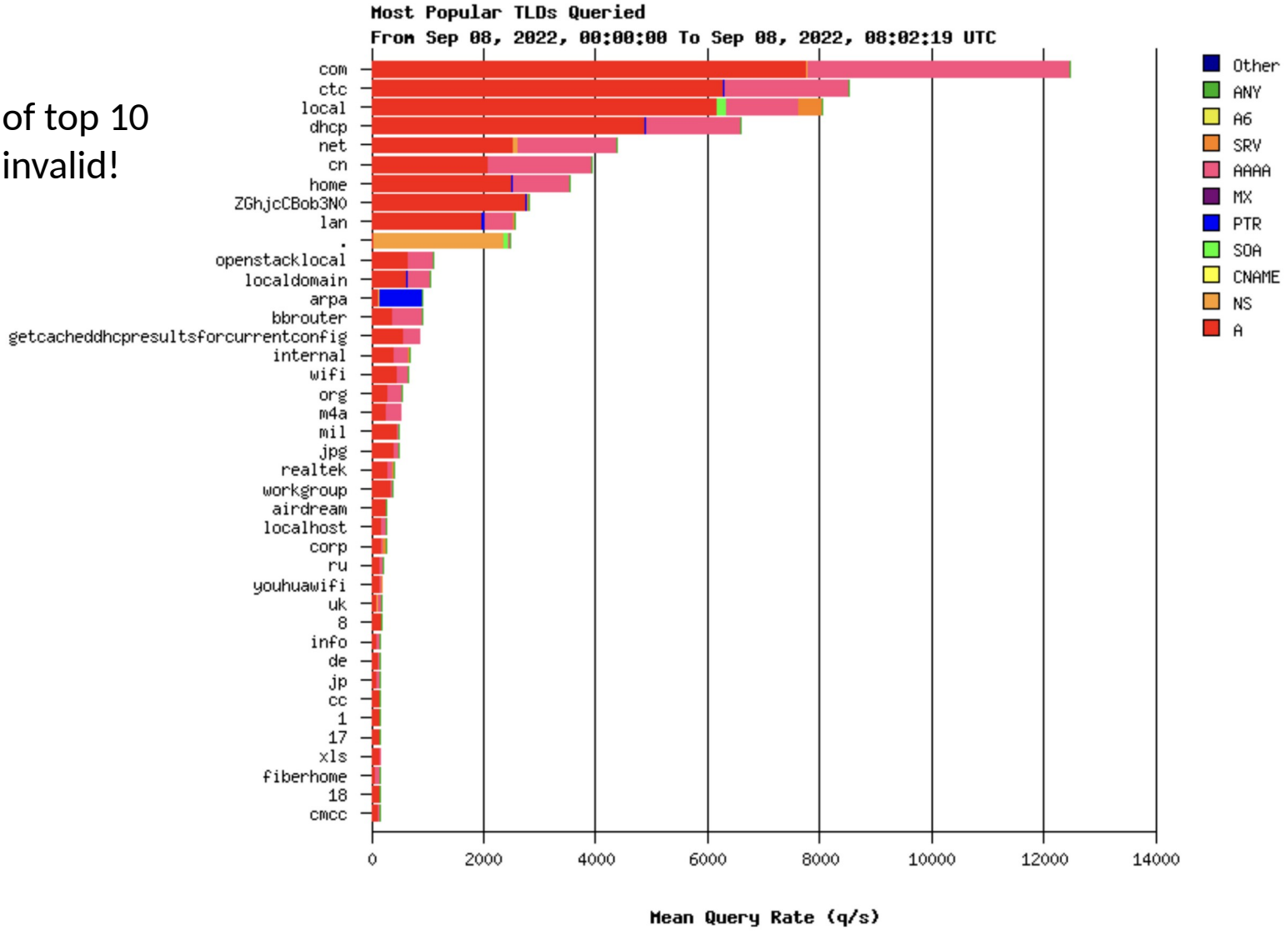
type=MX

- `value` is name of SMTP mail server associated with `name`

Most popular TLD

At least WORKGROUP is no longer here!
It was the top invalid TLD for years…

7 of top 10 invalid!



On osx "*host –av www.cl.cam.ac.uk*

```
 % host -va www.cl.cam.ac.uk
Trying "www.cl.cam.ac.uk"
Trying "www.cl.cam.ac.uk"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25214
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 23, AUTHORITY: 0, ADDITIONAL: 0


;; QUESTION SECTION:
:www.cl.cam.ac.uk. IN ANY


;; ANSWER SECTION:
www.cl.cam.ac.uk. 1200 IN RRSIG NSEC 5 5 1200 20230317214336 20230215204336 31575 cl.cam.ac.uk.
h2JCZHfF9m+jqvHQR6z67LDC3g1xKoCRDmrss+LVrAXJqWsi+d8+/Gio /U07C2SXKGa3NXj5ByNvqH2HJs6Loc1emEEeiPlSYqkOUSGTLRQhlEqy
bHEfeCPV4hJy/NSuRvcxZCqpgEDSbF5K0JzqI6dnPCoOoMFdSrA4n9OT kDQ=

www.cl.cam.ac.uk. 1200 IN NSEC www-443-120.cl.cam.ac.uk. A PTR TXT AAAA SSHFP RRSIG NSEC CAA

www.cl.cam.ac.uk. 21600 IN RRSIG SSHFP 5 5 21600 20230306104604 20230204102237 31575 cl.cam.ac.uk.
kfCKxAD9cyLJDj/UEJl7Sr8b55yH8dxfYc+BF9tgcqbReo2GNLQelOZN rB5JAhoewZ9HxlASO5rzCX1BR9AP0H+Rk7BNbDp8rvnO9G8PWGQcpKHm
BJWb9nj2a/zi360WCUGH/u8GlPxw0L7b2P460DYxE4wDmL3jNjvw61Ca Y3g=

www.cl.cam.ac.uk. 21600 IN SSHFP 3 2 B7E1DF5B943C481A263307EDEE23F0719858CDF516F2482B54A4B248 0118CAE9

www.cl.cam.ac.uk. 21600 IN SSHFP 2 1 6FB539DBE0E273B56327E619BB1814DB4CE810D8

www.cl.cam.ac.uk. 21600 IN SSHFP 4 2 293F122F4B4970C42B767898C5505C3EB038E0C5BB432EF3CAF33C03 1FA792FA

www.cl.cam.ac.uk. 21600 IN SSHFP 4 1 5E0CBE0730922925C446DF1B2DCE336AA7565122

www.cl.cam.ac.uk. 21600 IN SSHFP 1 1 FF45237BF493102CF7478AE0A96DE773FB4877C4

www.cl.cam.ac.uk. 21600 IN SSHFP 1 2 2953D9172EC850D2A46FA0245DFFAE978EE31B3BED233DED77BC937B 115952D7

www.cl.cam.ac.uk. 21600 IN SSHFP 3 1 FD276CF12A0B909533ABFA5931622950308AF099

www.cl.cam.ac.uk. 21600 IN SSHFP 2 2 403A5EE7B8ADD3E16B5973874E54CDFAC82268CC63B4CFD90E74DDC6 4E2EDF6F

www.cl.cam.ac.uk. 21600 IN RRSIG AAAA 5 5 21600 20230316213444 20230214203939 31575 cl.cam.ac.uk.
eVJM0NwnGPVC9y+96IJq48feYCDxTlEZ66fcH83aO2VFXoCblJkLUCoK e0TeobR+mnLad0XJFUocfjKorIV6s1CNzG90nmV1+dxQD1VBxQzBrV9A
k+JqokUQbkvb0UsV4UIWUvRav0M1GccXS5Nxzl/HDITMyVXMZx/Citlr lgU=

www.cl.cam.ac.uk. 21600 IN AAAA 2a05:b400:110::80:14

www.cl.cam.ac.uk. 21600 IN RRSIG TXT 5 5 21600 20230321232700 20230219230148 31575 cl.cam.ac.uk.
Tjlztn2dsdir5wGAkuPVIv/0V/BBIDFC3K8x7nNnml9dRov/ncRLWFvA 9XsxFNO2Oei7eyt6pelFstpVwny6F9nMs+xAFYDiX0PpcJ4pZMNADQgs
BNXhR2XS0IknnuqUuWPlH1WTFZqBd27gIsQ0F+79Atj3MHQ5hBZlCS0n EoE=

www.cl.cam.ac.uk. 21600 IN TXT "pseudo IP address for cl.cam.ac.uk departmental WWW server (IPv4 and IPv6)"

www.cl.cam.ac.uk. 21600 IN RRSIG PTR 5 5 21600 20230314163841 20230212160205 31575 cl.cam.ac.uk.
gceJqmj4zzcCsdYuU2ymxuMABblZO6VDIp8J4seDqDt09924Aycpl2P9 5wlDds02a0JGTqbnnDiydKEgOA6fJHNMCEbAQr6GVjl/Eg+YLWH8YwLx
Ou9q7FgmZyXLk67wC9ji17VZ3V9Co5Kd2kWuDOvp7Xb5OeyPKAll//tL BDk=

www.cl.cam.ac.uk. 21600 IN PTR svr-www-00.cl.cam.ac.uk.

www.cl.cam.ac.uk. 21600 IN RRSIG A 5 5 21600 20230314163841 20230212160205 31575 cl.cam.ac.uk.
j8k+Q8L2C7zzHSvpWpg+1t5WPk9IeTZ9GOvw/0v1pbYVXJfeHuNm9ERM Ff/hEkZm21ooFIBtrbTe/m5b+kBBm2OETDbbGP+na3/DEQyWFp0sHe6j
S2ZS9KeOXENJk92eA+/dBAWe0vFvTpXrZ/thp61ctqm9b3mR8AbnuNgu uHs=

www.cl.cam.ac.uk. 21600 IN A 128.232.0.20
```

URL and URI

`https://www.cl.cam.ac.uk/users/djg11/marmite.jpg`

Definition:

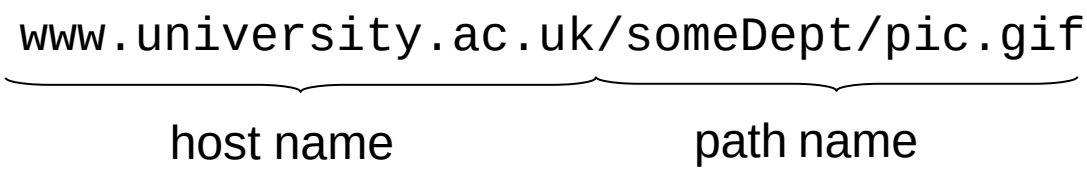
- *A URI identifies a resource by name or location*
  - *e.g., urn:isbn:02342342*
- *A URL specifies a location, usually starting with a protocol like https:// …*
- *All URIs contain a colon.*
- *URLs sit in a subset of the URI space.*

Fun fact: my email address used to be djg@uk.ac.cam.cl

World Wide Web and HTTP

Basics…

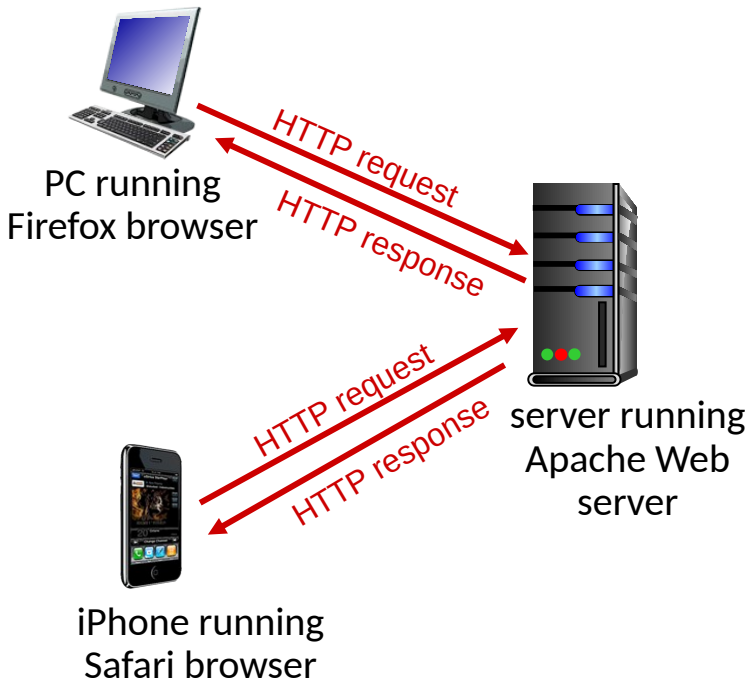
- web page consists of *objects,* each of which can be stored on different Web servers
- object can be HTML file, JPEG image, Java applet, audio file, or anything really …
- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,



HTTP overview

HTTP: hypertext transfer protocol

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



HTTP overview (continued)

HTTP uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains *no* information about past client requests

Reminder: Distributed Systems are Hard!

protocols that maintain "state" are complex!

- If past history (state) must be maintained
- when server/client crashes, their views of "state" may be inconsistent; must be reconciled.

HTTP connections: two types

Non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed.

Downloading multiple objects required multiple connections.

Persistent HTTP

- TCP connection opened to a server
- multiple objects conveyd over *single* TCP connection between client and that server
- TCP connection closed.

Non-persistent HTTP: example

User enters URL: **`www.university.ac.uk/someDepartment/home.index`**
(containing text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.university.ac.uk on port 80

1b. HTTP server at host www.university.ac.uk waiting for TCP connection at port 80 "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Non-persistent HTTP: example (cont.)

User enters URL: **`www.university.ac.uk/someDepartment/home.index`**
(containing text, references to 10 jpeg images)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

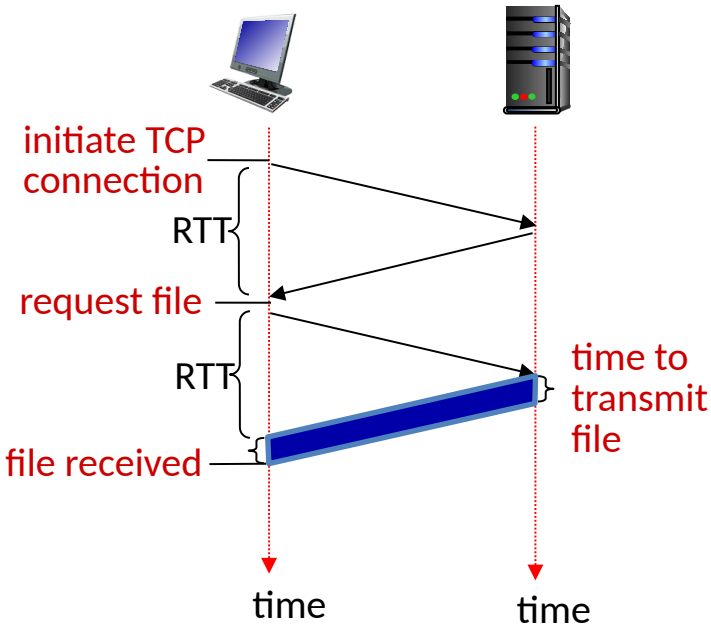
time

Non-persistent HTTP: response time

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

HTTP response time (per object):

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- object/file transmission time



Non-persistent HTTP response time = 2RTT+ file transmission time

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open multiple parallel TCP connections to fetch referenced objects in parallel

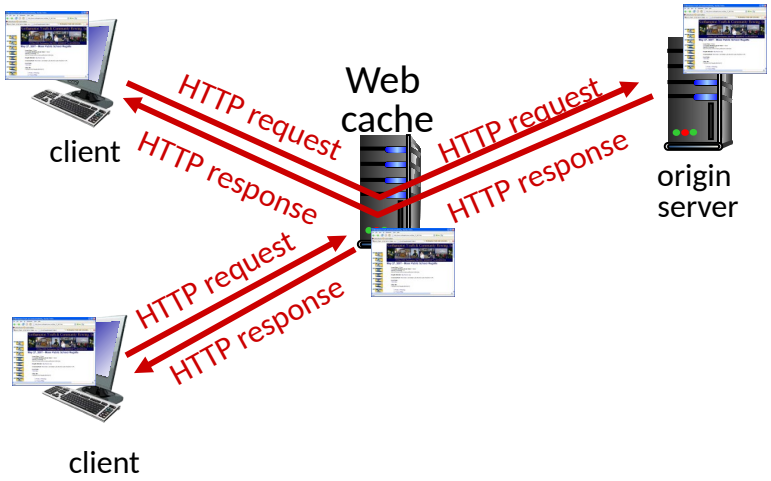
Persistent HTTP (HTTP1.1):

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects (cutting response time in half)

Web caches

Goal: satisfy client requests without involving origin server

- user configures browser to point to a (local) *Web cache*
- browser sends all HTTP requests to cache
  - *if* object in cache: cache returns object to client
  - *else* cache requests object from origin server, caches received object, then returns object to client



Web caches (aka proxy servers)

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

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

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

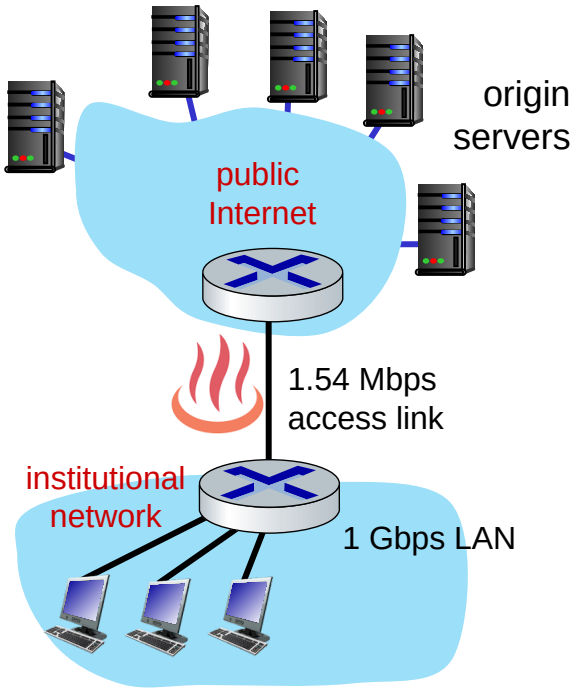
Why Web caching?

- reduce response time for client request
  - cache is closer to client
- reduce traffic on an institution's access link
- Internet is dense with caches
  - enables "poor" content providers to more effectively deliver content

Caching example

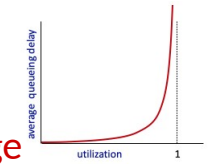
Scenario:

- 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
  - avg data rate to browsers: 1.50 Mbps



Performance:

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



problem: large queueing delays at high utilization!

Option 1: buy a faster access link

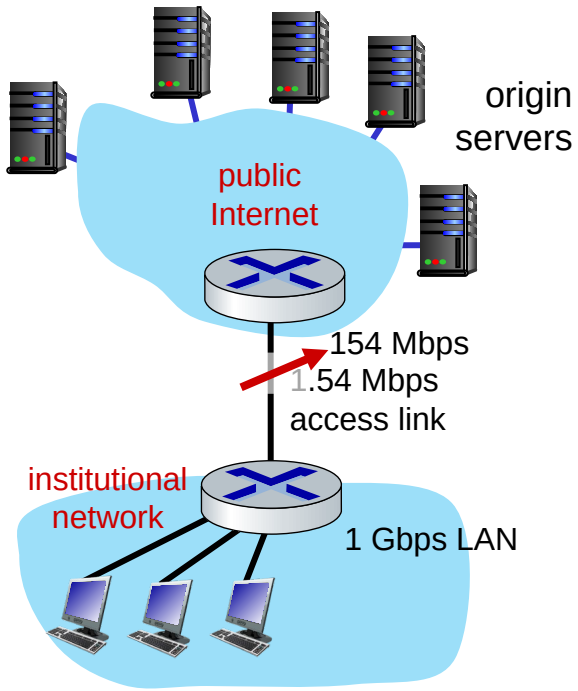
Scenario:

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

Performance:

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

Cost: faster access link (expensive!)



Option 2: install a web cache

Scenario:

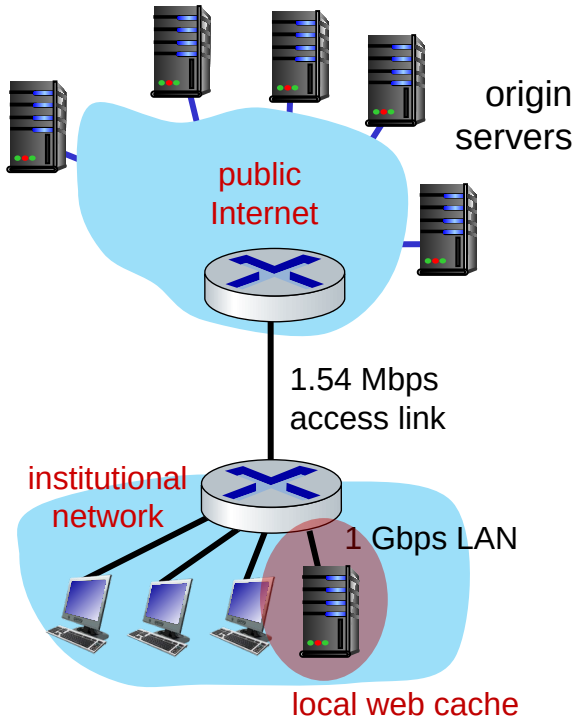
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Cost: web cache (cheap!)

Performance:

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

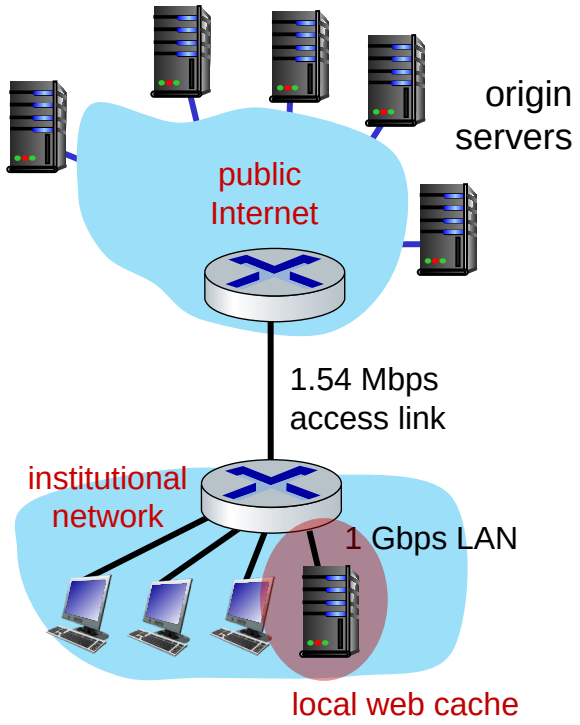
How to compute link utilization, delay?



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

suppose cache hit rate is 0.4:

- 40% requests served by cache, with low (msec) delay
- 60% requests satisfied at origin
  - rate to browsers over access link
    = 0.6 * 1.50 Mbps = .9 Mbps
  - access link utilization = 0.9/1.54 = .58 means low (msec) queueing delay at access link
- 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!)