4. Application Layer

2019 Fall

Yusung Kim
yskim525@skku.edu

Goal

- Understanding conceptual, implementation aspects of **network application protocols**
  - transport service models
  - communication models (client-server vs. P2P)
- Learning important application-level protocols
  - HTTP, DNS, web caching, CDN, P2P
- Experiencing of network application developments
  - socket API

Some network apps

- Web
- E-mail
- On-line games
- P2P file sharing
- Instant messaging
- Search (e.g. Google)
- Voice over IP (e.g. Skype)
- Real-time video conferencing
- Social networking (e.g. Facebook)
- Stored video streaming (e.g. YouTube)
- …

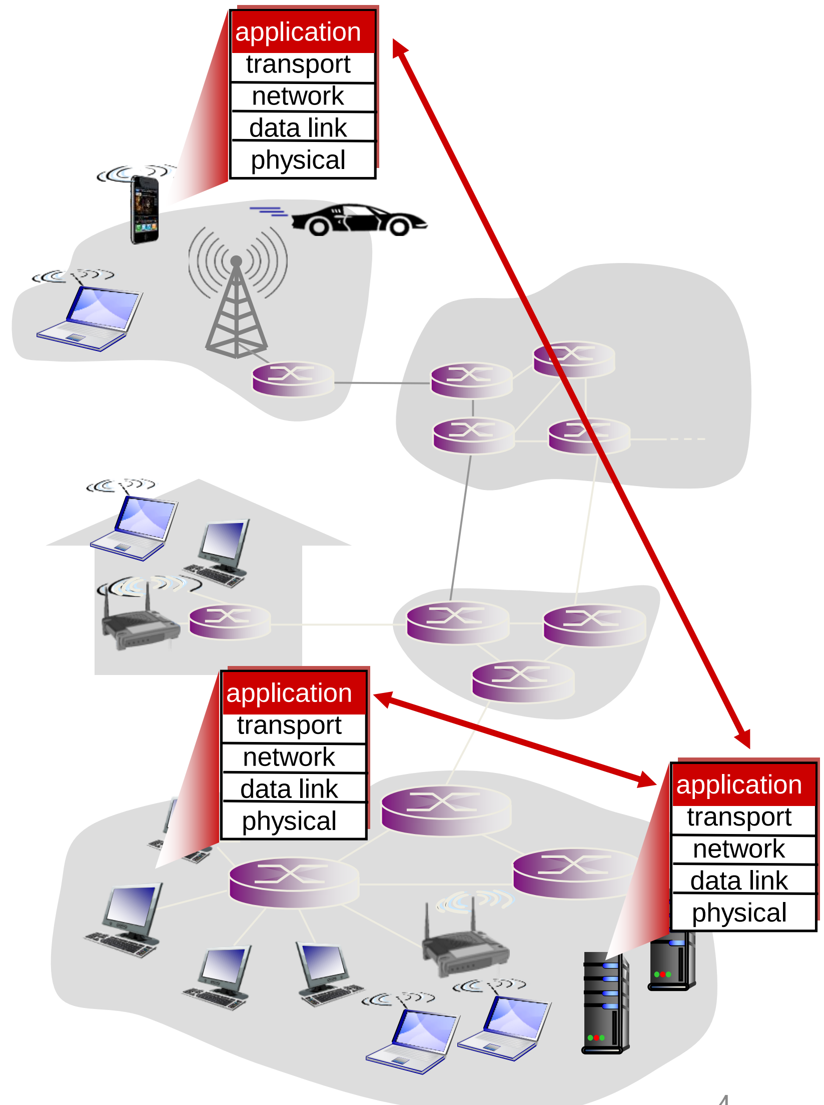
Creating a network app

Write programs that:

- run on (different) *end systems*
- communicate over network
- e.g. web server communicates with browser

No need to write programs for network-core devices

- network-core devices do not run user applications
- it has facilitated the rapid app development



Application architectures

- To design network applications, we need a broad architectural plan.

- Predominant architectural paradigm of applications:
  - **Client-server** architecture
  - **Peer-to-peer** (P2P) architecture

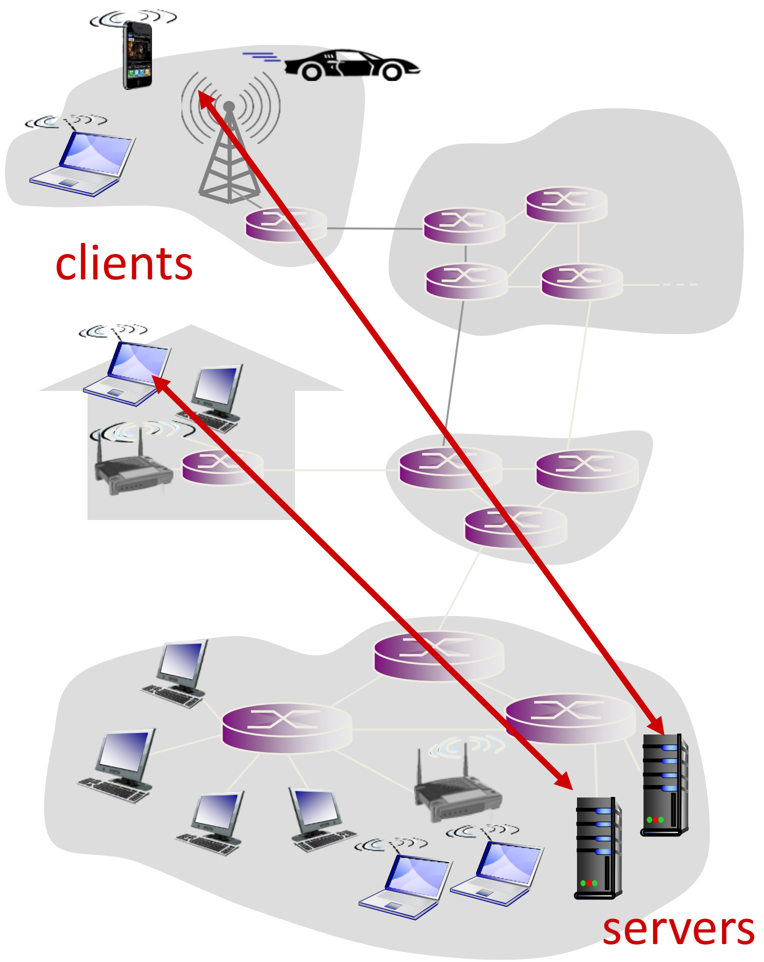
Client-server architecture

Servers:

- always-on host
- permanent IP address
- data centers for scaling

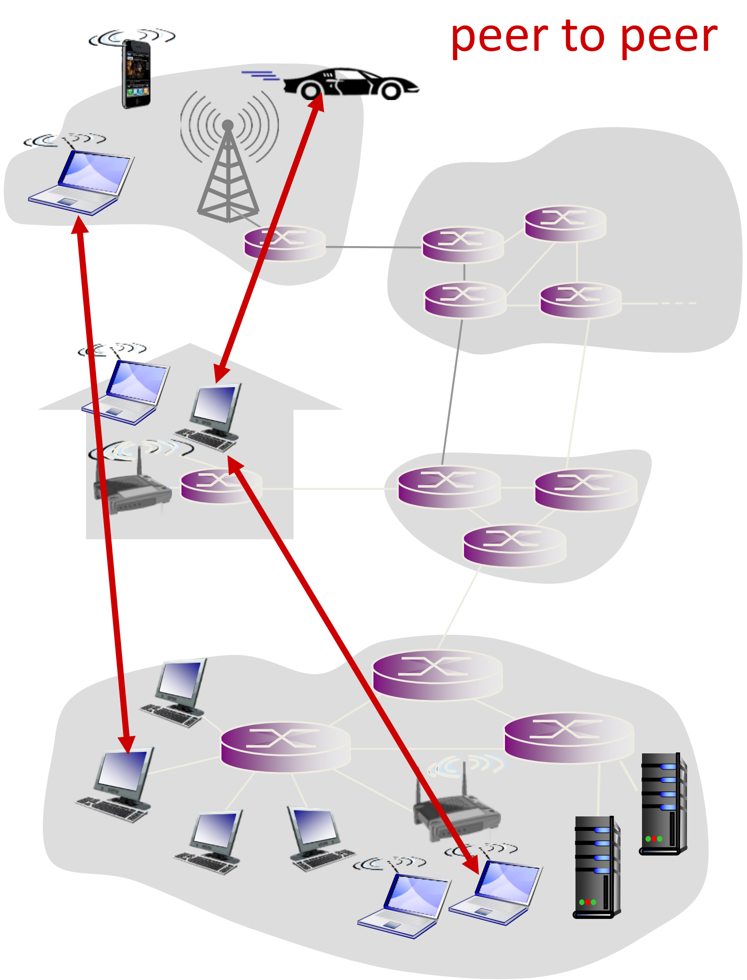
Clients:

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



P2P architecture

- *Not* always-on
- Peers directly communicate
- Peers request service from other peers, provide service in return to other peers
  - ***self scalability***: new peers bring service capacity, as well as service demands
- Peers are intermittently connected and change IP addresses
  - requires complex management



Processes communication

process: program running within a host

- Within same host, two processes communicate using **inter-process communication** (defined by OS)
- Processes in different hosts communicate by exchanging **messages**

client to server model

client process that initiates communication

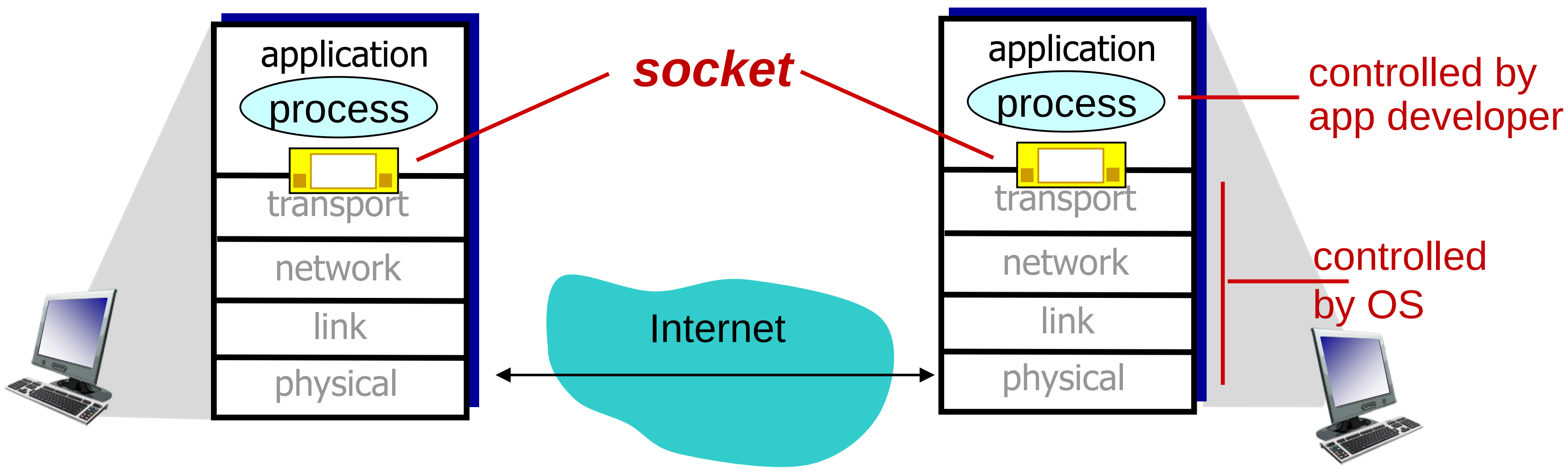
server process that waits to be contacted

peer-to-peer model

Applications with **P2P** architectures need both client & server processes

Sockets

- The interface between process and computer network
- Process sends/receives messages to/from its **socket.**
- Assume that there is a transportation infrastructure to forward the messages to the destination process.



Addressing processes

- To receive messages, process must have ***identifier***

- Host device has unique 32-bit **IP address**
  e.g. 115.145.129.40

- ***Identifier*** includes both **IP address** and **port number** associated with process on host.
  - Web server process : port number 80
  - E-mail server process : port number 25

Application layer protocol

- Web vs. HTTP
- The Web is a client-server application.
- HTTP [RFC 2616] is the Web's application-layer protocol.
- The Web application consists of many components;
  - HTML standard, browsers, servers, and HTTP.
  - HTTP is **only one piece** of the Web application.
- A browser developer should follow the rules of the HTTP RFC because all Web servers also follow the rules.

App-layer protocol defines

types of messages exchanged

- e.g., request, response

message syntax:

- what fields in messages & how fields are delineated

message semantics

- meaning of each field

rules for when and how processes send & respond to messages

open protocols:

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

proprietary protocols:

- e.g., Skype

What transport service does an app need?

Reliability

- many apps require 100% reliable data transfer
- other apps (e.g., audio, video) can tolerate some loss

Throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") use as much throughput as they can get

Timing

- some apps (e.g., VoIP, interactive games) require low delay/jitter to be "effective"

Security

- encryption, data integrity, …

Transport service requirements

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100 msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100 msec
text messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control:* sender will not overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security

UDP service:

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

Q: Why is there a UDP?

Internet apps and transport protocols

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

Web and HTTP

Web and HTTP

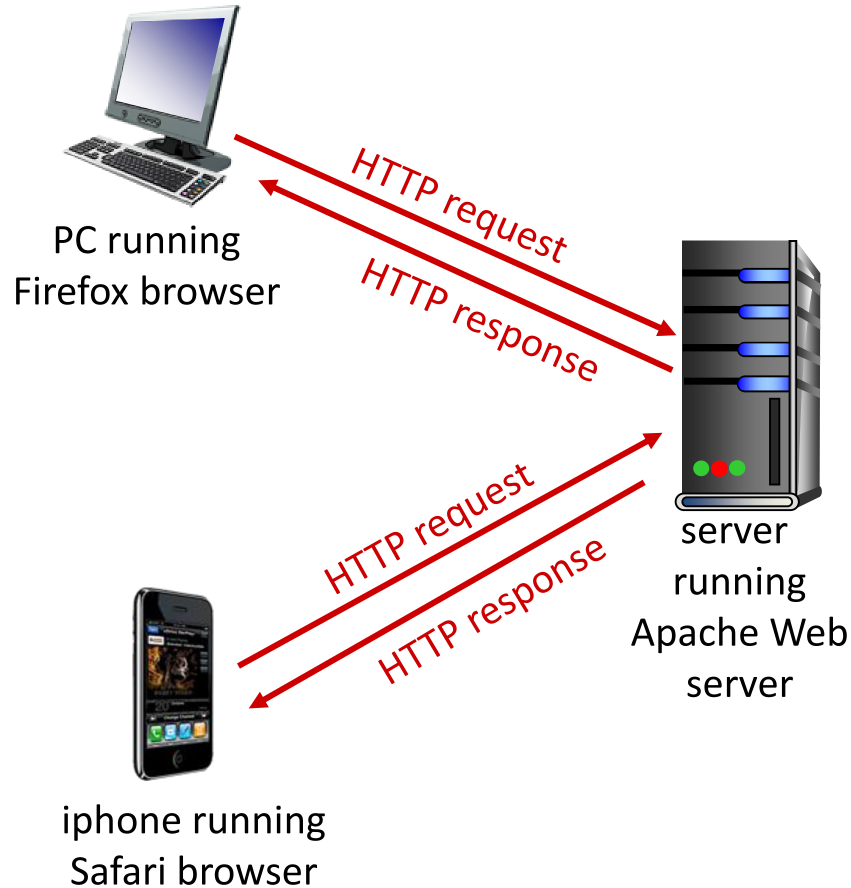
First, a review…

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

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

Hypertext Transfer Protocol (HTTP)

- Application layer protocol for the Web
- Client/server model
  - *client:* browser that requests and receives, and "displays" Web pages
  - *server:* Web server that sends objects in response to requests



HTTP Overview

uses TCP:

- Client initiates TCP connection to server
- Server accepts TCP connection from client
- Messages are exchanged between server and client
- TCP connection closed

HTTP is "stateless"

- Server maintains no information about past client requests

Tips

Protocols that maintain "state" are complex!

- past history (state) must be recorded.
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections

non-persistent HTTP

- At most one object sent over TCP connection
  - connection then closed
- Downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

Non-persistent HTTP

suppose user enters URL:

www.someSchool.edu/someDepartment/home.index (text and 10 jpeg image links)

1a. HTTP client initiates TCP connection to HTTP server at **www.someSchool.edu** on port 80

1b. HTTP server at host **www.someSchool.edu** waiting for TCP connection at port 80. "accepts" the connection, and notifying client

2. HTTP client sends HTTP ***request message*** 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 (cont.)

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

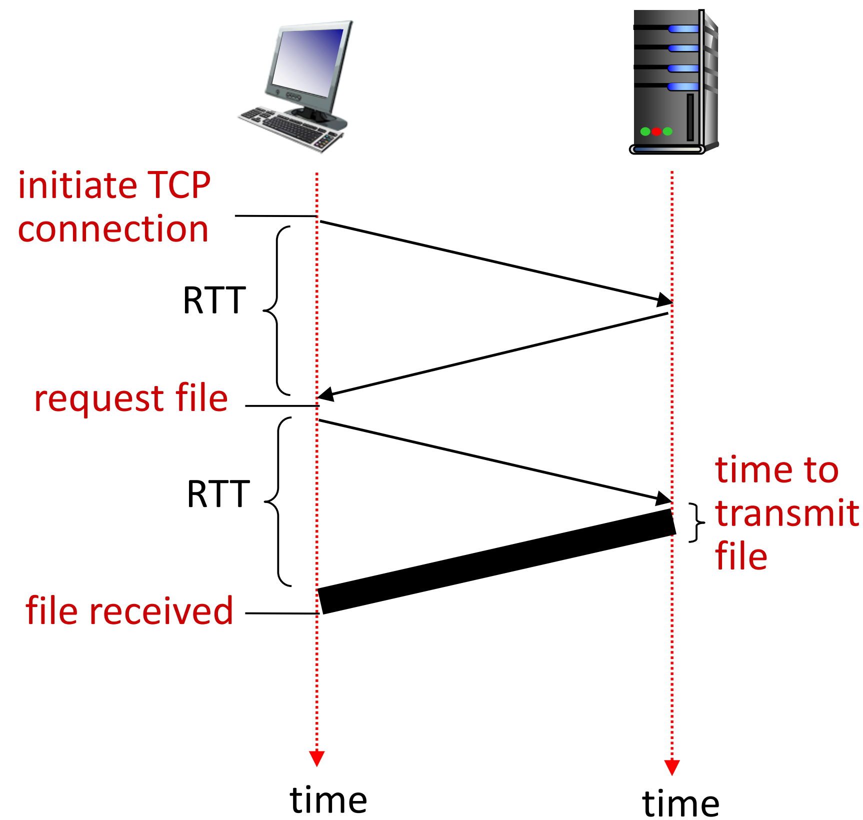
Non-persistent HTTP: response time

Round Trip Time (RTT)

time for a small packet to travel from client to server and back

HTTP response time:

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



Persistent HTTP

non-persistent HTTP issues:

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

persistent HTTP:

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over **open connection**
- Reducing one RTT for each referenced object

HTTP request message

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

carriage return character

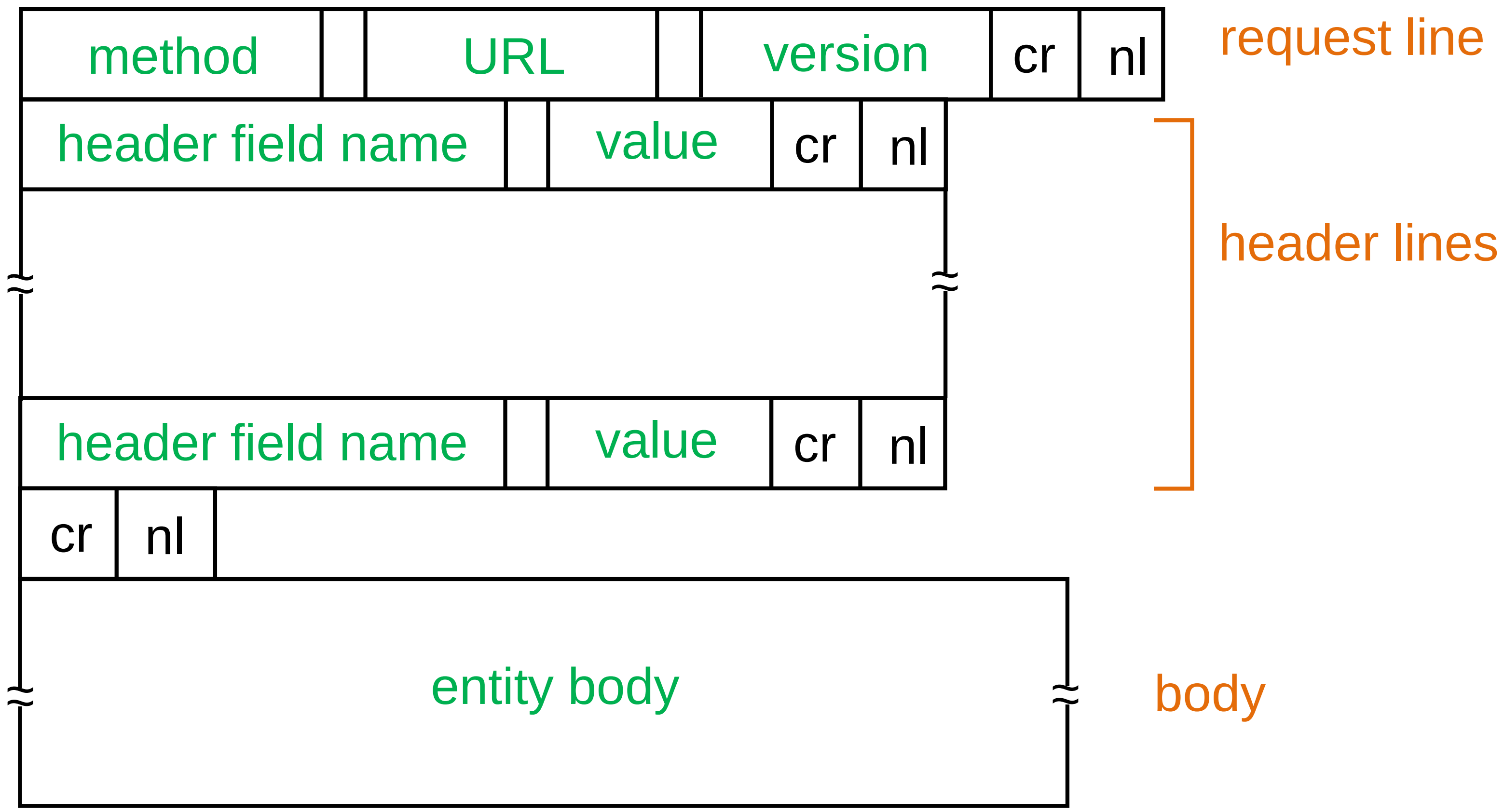
newline character

request line (GET, POST, HEAD commands)

header lines

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

HTTP request message: general format



Uploading form input

POST method:

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

URL method:

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

```
www.somesite.com?id=yskim525&pw=XXXXX
```

HTTP response message

status line (protocol status code)

HTTP/1.1 200 OK\r\n
Date: Sun, 16 Sep 2014 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Aug 2014 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...

header lines

data, e.g., requested HTML file

HTTP response status codes

- Status code appears in 1st line in response message.
- Some sample codes:

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

- requested object moved, new location specified later in this msg

400 Bad Request

- request msg not understood by server

403 Forbidden

- You don't have permission to access

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

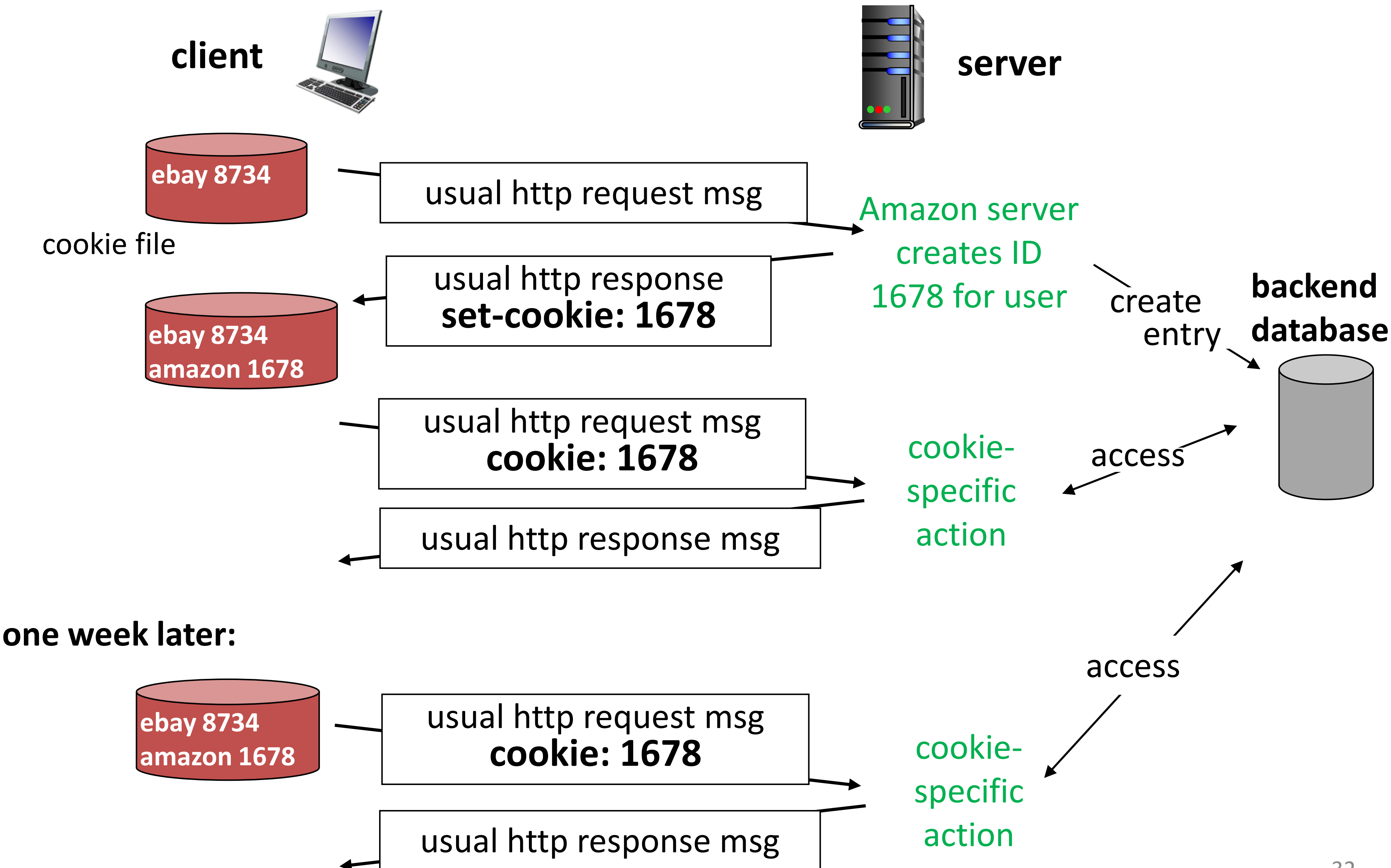
User-server state: cookies

Many Web sites use cookies

four components:

- Cookie header line of HTTP *response* message
- Cookie header line in next HTTP *request* message
- Cookie file kept on user's host, managed by user's browser
- Back-end database at Web site

Cookies: keeping “state” (cont.)



Cookies (continued)

what cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

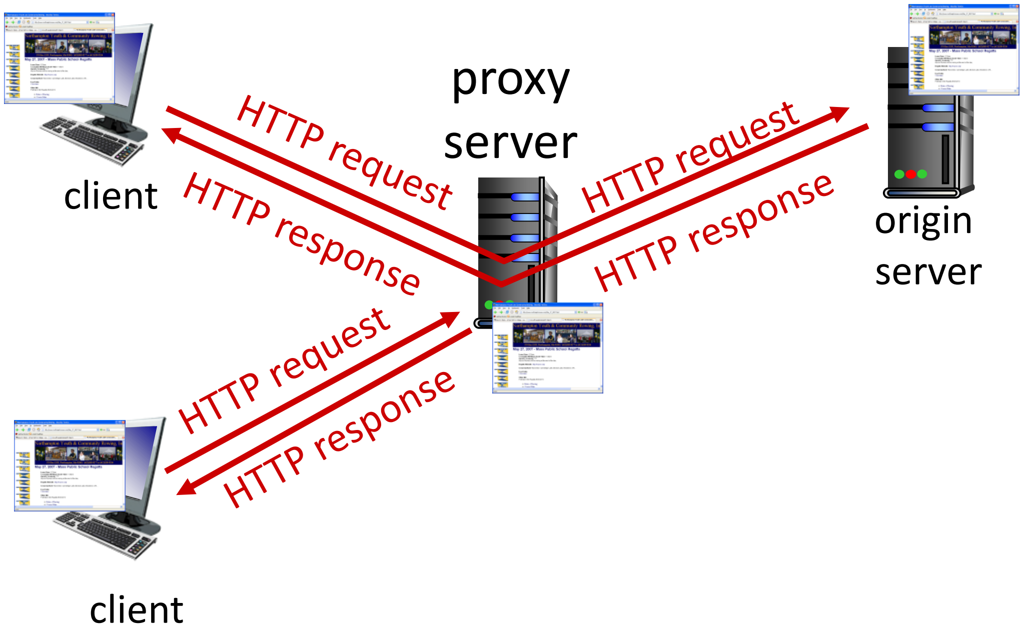
aside

cookies permit sites to learn a lot about you, and may invade your privacy

Web caches (proxy server)

goal: satisfy client request without involving origin server

- User sets browser: Web accesses via cache
- Browser sends all HTTP requests to cache
  - **If object in cache:** cache returns object
  - **Else:** cache retrieves object from origin server, then returns object to client



More about Web caching

- Cache acts as both **client** and **server**
- Typically cache is installed by ISP (university, company, residential ISP)
- **Reduce response time** for client request
- **Reduce traffic** on an institution's access link
- Web caching enables "poor" content providers to **effectively deliver content**

If access link is the bottleneck

assumptions:

- avg object size: 10 Mbits
- avg request rate: 12/sec
- public delay : 1 sec
- access link rate: 100 Mbps

LAN utilization:

12/s x 10 Mbits / 1 Gbps = 0.12

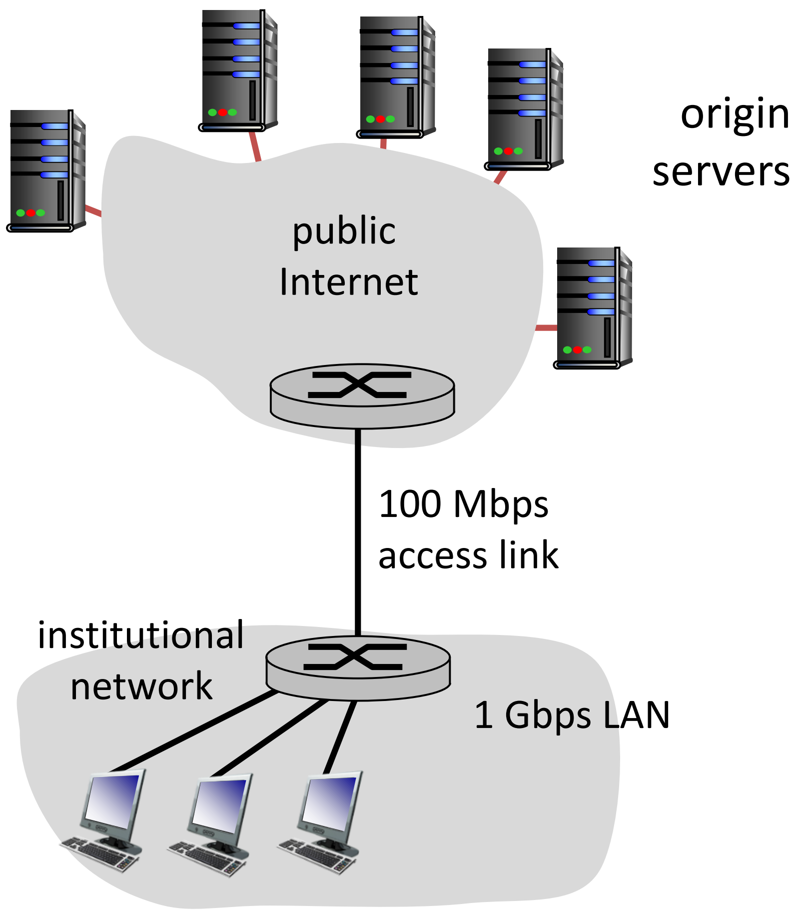
Access link utilization:

12/s x 10 Mbits/100 Mbps = 1.2

Total (approximately) delay:

public + access + LAN

= 1 sec + minutes? + μ secs



If upgrading faster access link

assumptions:

- avg object size: 10 Mbits
- avg request rate: 12/sec
- public delay : 1 sec
- access link rate: ~~100 Mbps~~ 1 Gbps

LAN utilization:

12/s x 10 Mbits / 1 Gbps = 0.12

Access link utilization:

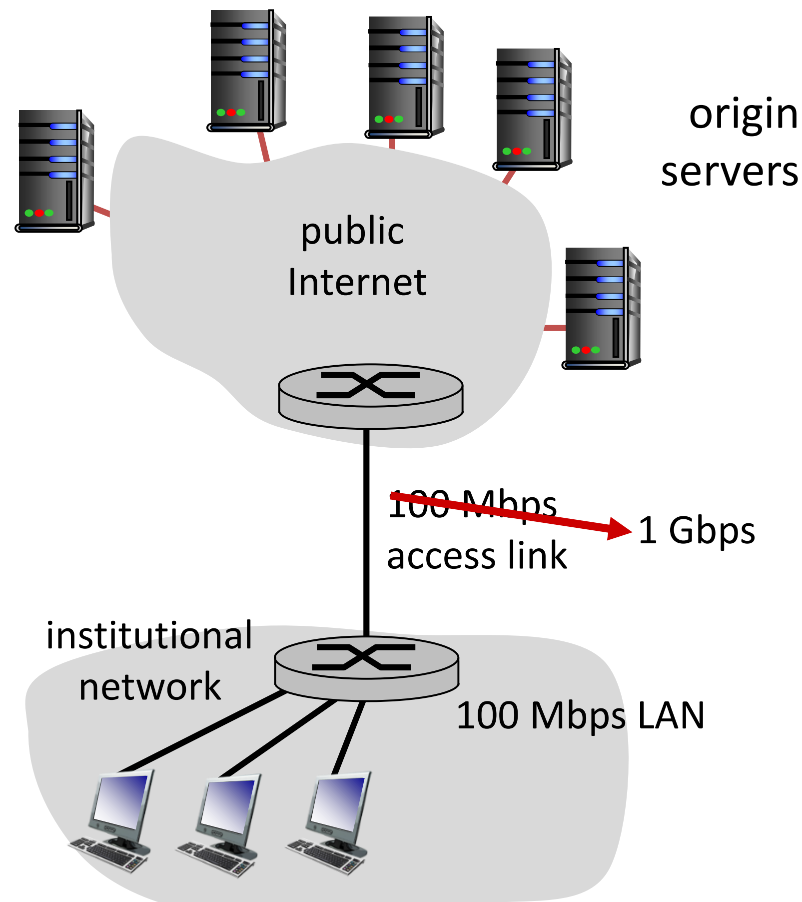
12/s x 10 Mbits / 1 Gbps = ~~1.2~~ 0.12

Total (approximately) delay:

public + access + LAN

= 1 sec + ~~minutes~~ + μ secs

μ secs



Cost: upgrading access link speed (not cheap!)

If installing a local cache

assumptions:

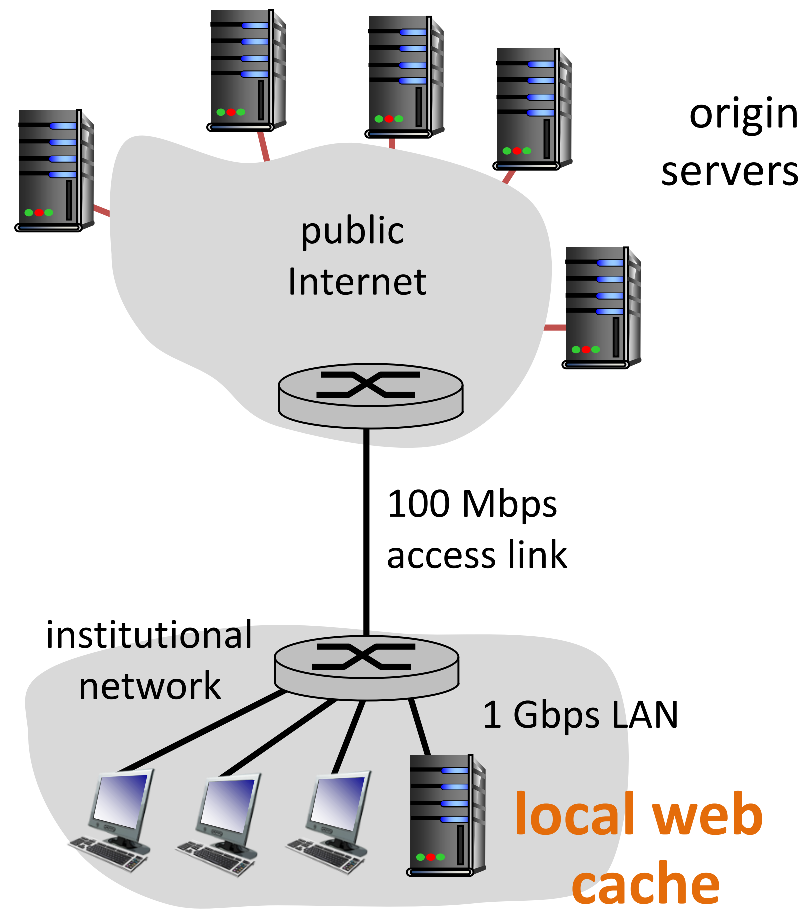
- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache
  - 60% requests satisfied at origin

Access link utilization:

- 60% of requests use access link

Total (approximately) delay:

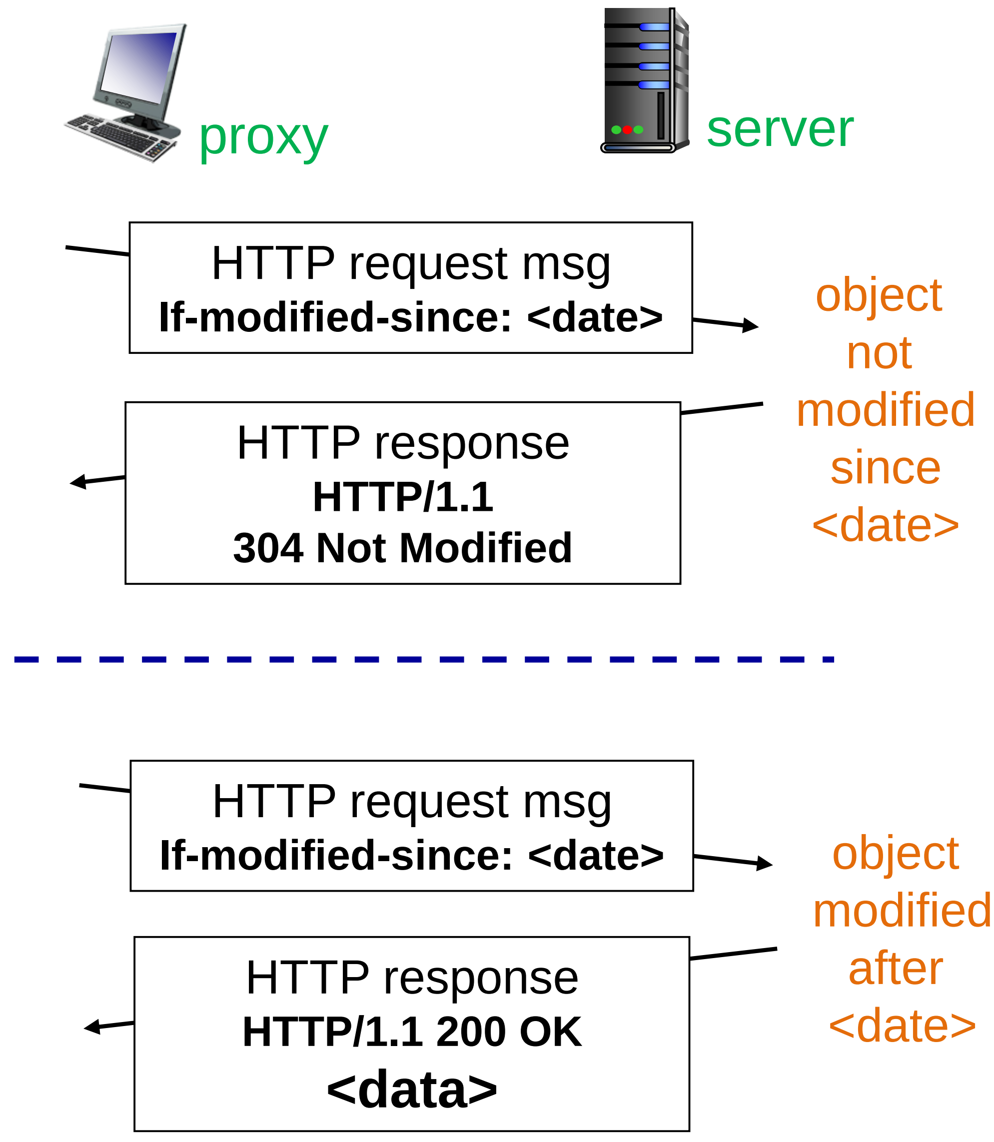
= 0.6 x (public + access + LAN) + 0.4 x LAN
= 0.6 x (1 sec + μ secs) + 0.4 x μ secs
= around 0.6 secs



Cost: better than the link upgrade and cheaper!

Conditional GET

- ***Goal:*** to check the current cached object is valid
- *Proxy:* specify date of cached copy in HTTP request
  If-modified-since: <date>
- *Server:* respond without object if cached copy is up-to-date:
  HTTP/1.0 304 Not Modified



Domain Name System

Domain name system (DNS)

Internet hosts, routers:

- "IP address" (32 bit) used for routing packets.
  e.g. 115.145.129.40
- "name",
  e.g., www.skku.edu
  used by humans

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

Domain Name System:

- ***distributed database*** implemented in **hierarchy** of many *name servers*
- ***application-layer*** *protocol:* hosts and name servers communicate to *resolve* names
  - core Internet function but implementation at app-layer
  - Simplicity at "core", and complexity at "edge"

DNS: services, structure

DNS services

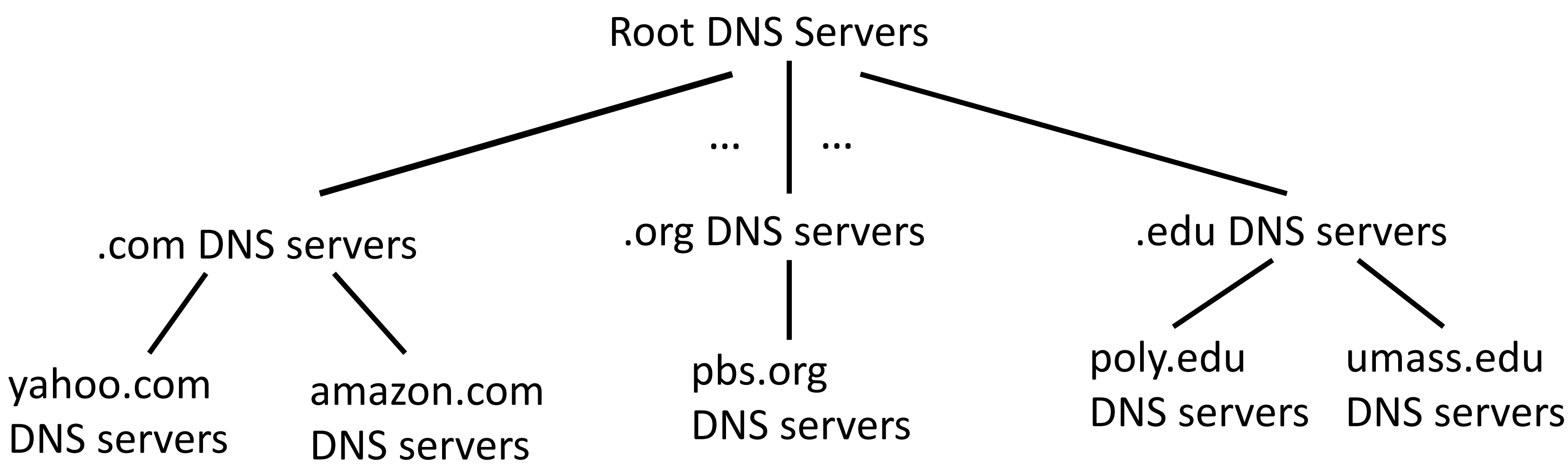
- hostname to IP address translation
- host aliasing
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

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

Answer: doesn't scale!

DNS: a distributed, hierarchical database

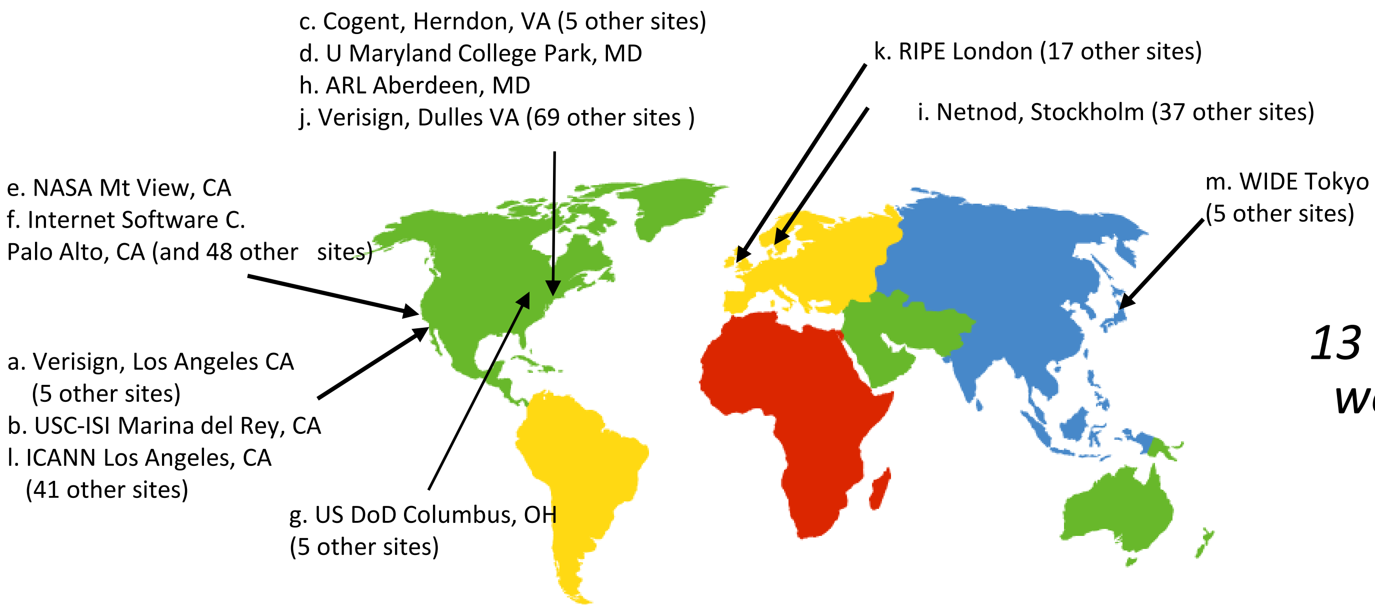


Client wants IP for www.amazon.com

- 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

- Root name server:
  - Name server for the root zone of the DNS in the Internet.
  - 13 root name servers but more than 600 copy servers.
  - Using anycast addressing in multiple geographical locations.



13 root name "servers" worldwide

TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for .com, .org, .net, .edu, and all top-level country domains, e.g. kr, uk, fr, ca, jp

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

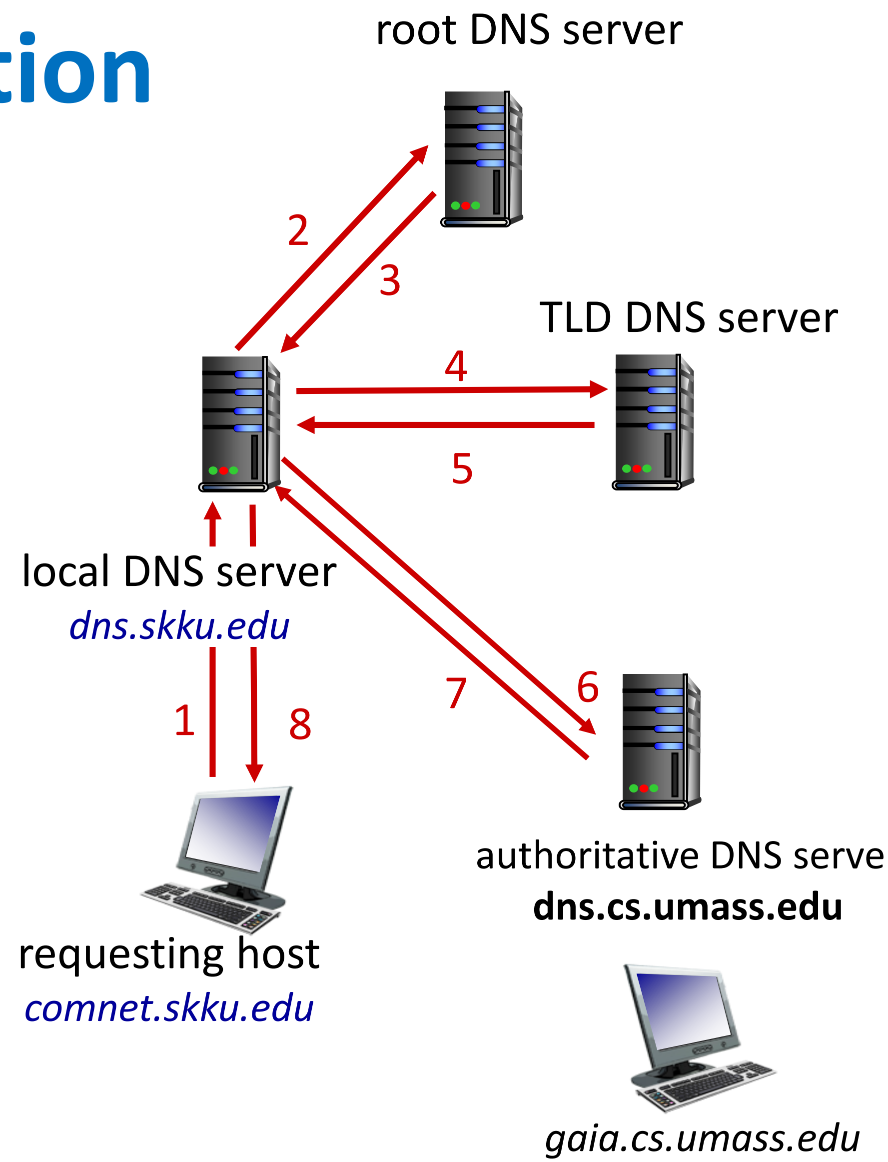
- Each ISP (residential ISP or university) has one
  - also called "default name server"
- When host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS name resolution

- host at **comnet.skku.edu** wants IP address for **“gaia.cs.umass.edu”**

iterated query:

- contacted server replies with the next server to contact.

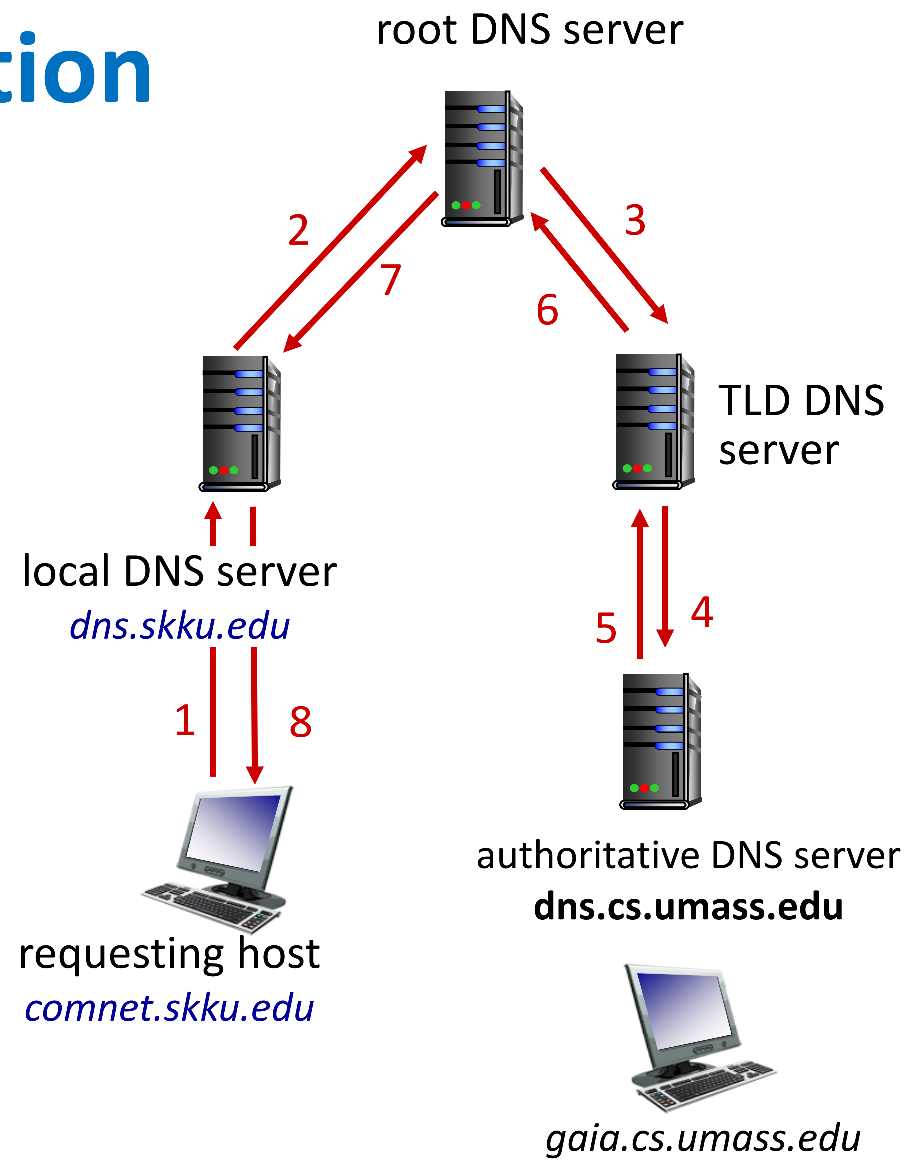


DNS name resolution

recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?

* In practice, what is the typically used as a query method?



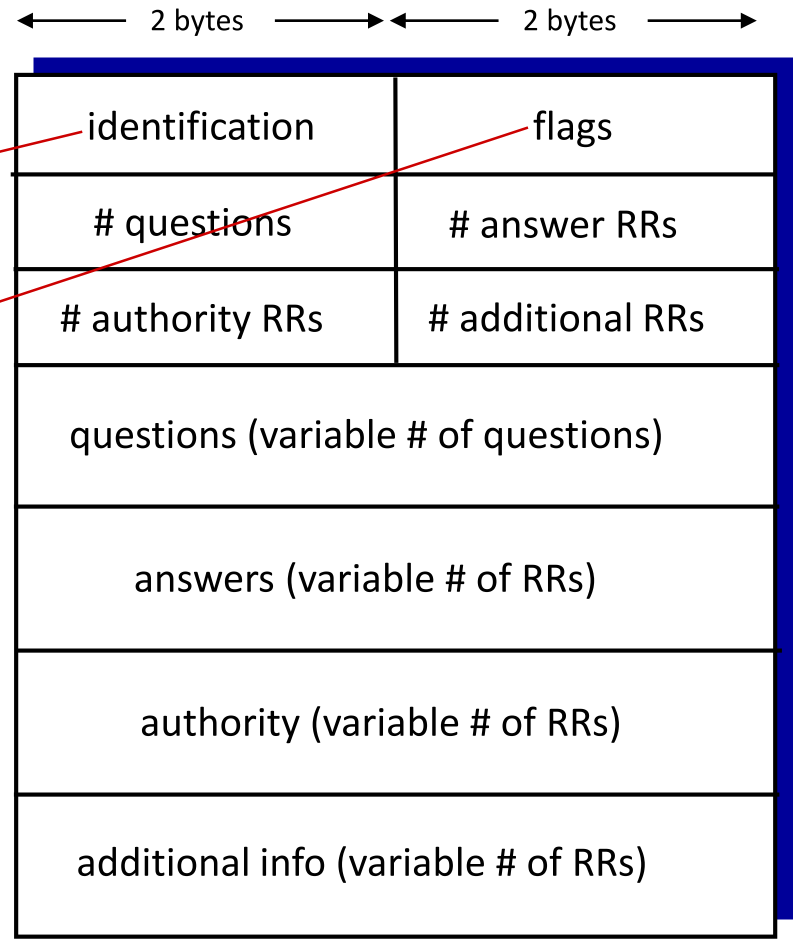
DNS messages

- *query* and *reply* messages have same *message format*

msg header

- **identification:**
  16 bit for query, reply to query uses the same value

- **flags:**
  - query or reply
  - recursion desired



DNS: caching, updating records

- Once (any) name server learns mapping, it ***caches*** the mapping
  - cache entries timeout after some time **(TTL)**
  - TLD servers typically cached in local name servers (thus root name servers not often visited)
  - if a host name changes an IP address, may not be known Internet-wide until all **TTLs** expire
- Update/notify mechanisms proposed IETF standard
  - RFC 2136

Attacking DNS

DDoS attacks

- Attack root servers with massive traffic
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Attack TLD servers
  - Potentially more dangerous

Redirect attacks

- DNS poisoning
  - Intercept queries
  - Send bogus relies to DNS server, which caches

Exploit DNS for DDoS

- Send queries with spoofed source address: target IP

Summary

Typical request/reply message exchange:

- client requests info or service
- server responds with data and status code

Message formats:

- headers giving info about data
- data being communicated

Important themes:

- control vs. data msgs (in-band, out-of-band)
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable
- complexity at network edge