CS 331: Computer Networks

Sameer G Kulkarni

Assistant Professor,

Department of Computer Science and Engineering,

Indian Institute of Technology, Gandhinagar

TAKEAWAY QUESTION..

Q: What is the role of L7 in ISO/OSI and L5 in TCP/IP?

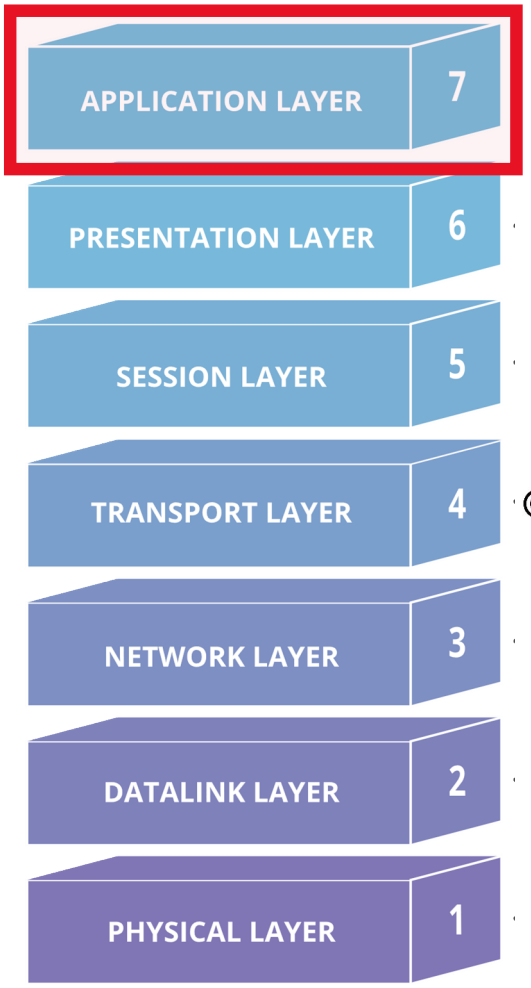
Application Layer Protocols!

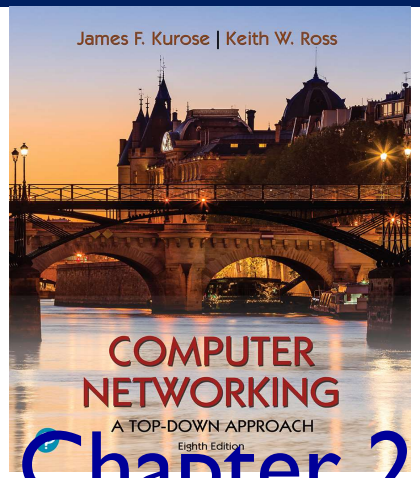
Today's Focus:
- Networking Applications (NApps)
- Common Characteristics of NApps
- Development and Deployment Paradigms.

APPLICATION LAYER

our goals:

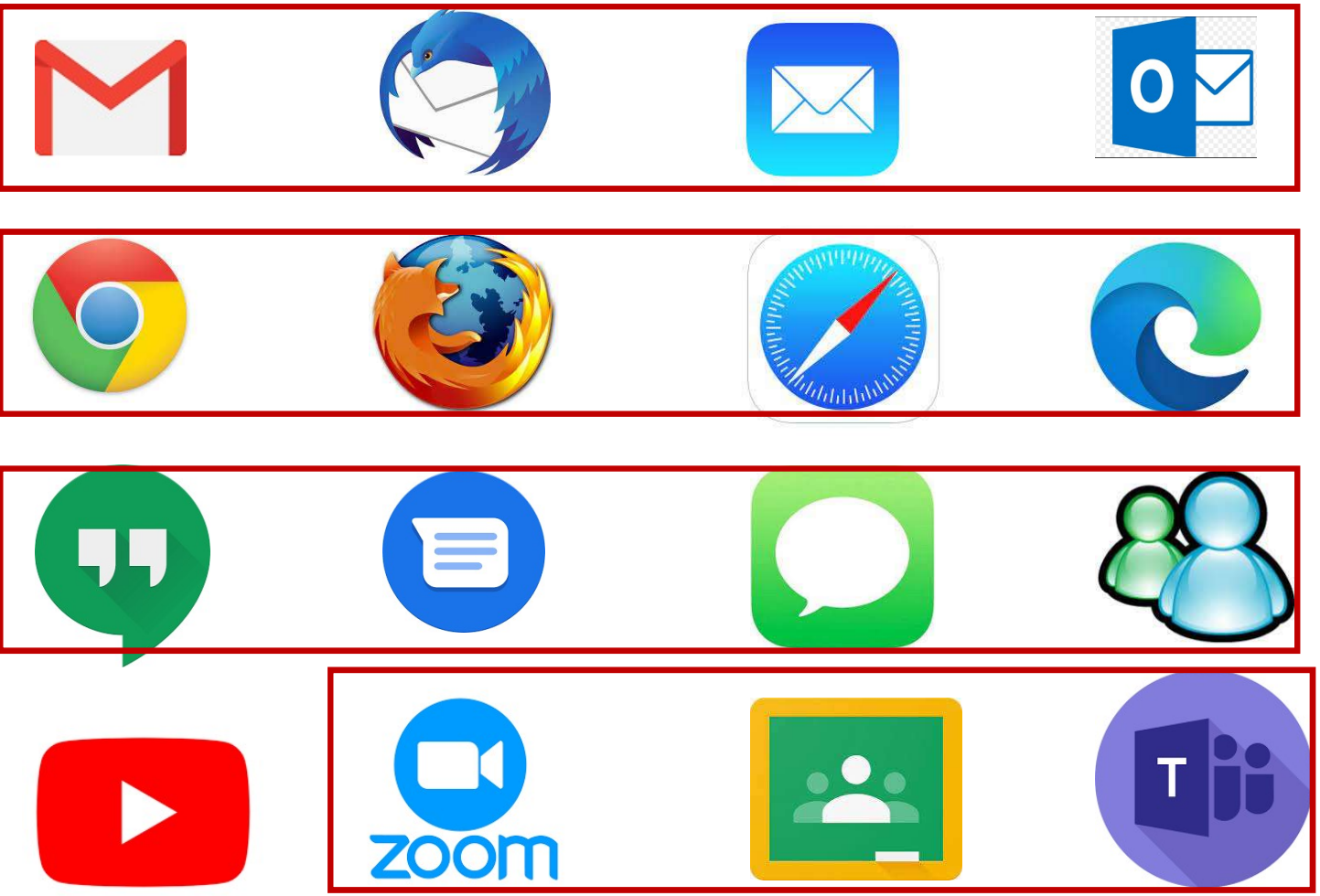
- conceptual, implementation aspects of network application protocols
  - client-server paradigm
  - peer-to-peer paradigm
  - transport-layer service models
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- creating network applications
  - socket API *(possibly in Tutorial ~~2~~ 3)*





Some Slides are also adapted from:
Computer Systems approach (Bruce Davie), Internetworking (D Comer).

SOME COMMONLY USED NETWORK APPLICATIONS..

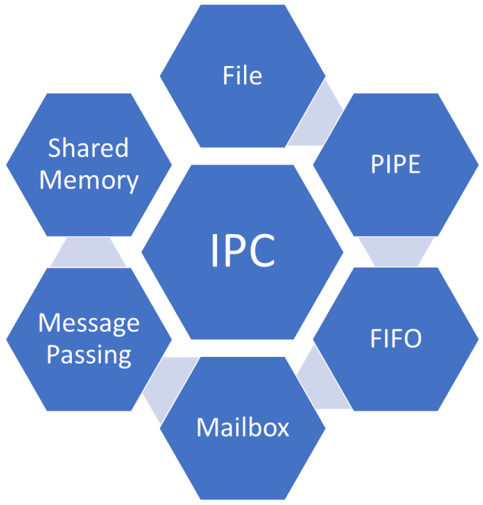


- e-mail
- Web (search engines, websites)
- text messaging
- remote login
- streaming stored video (YouTube, Hulu, Netflix)
- voice over IP (e.g., Google Voice)
- real-time video conferencing
- P2P file sharing
- multi-user network games
- social networking
- E-commerce (shopping, banks, trade)
- …

NETWORKING APPLICATIONS: PROCESSES THAT COMMUNICATE

process: program running within a host/device.

- within same host, two processes communicate using inter-process communication (***OS APIs***)



- processes in different hosts communicate by exchanging well-defined messages (***network protocols + OS APIs***).

clients, servers

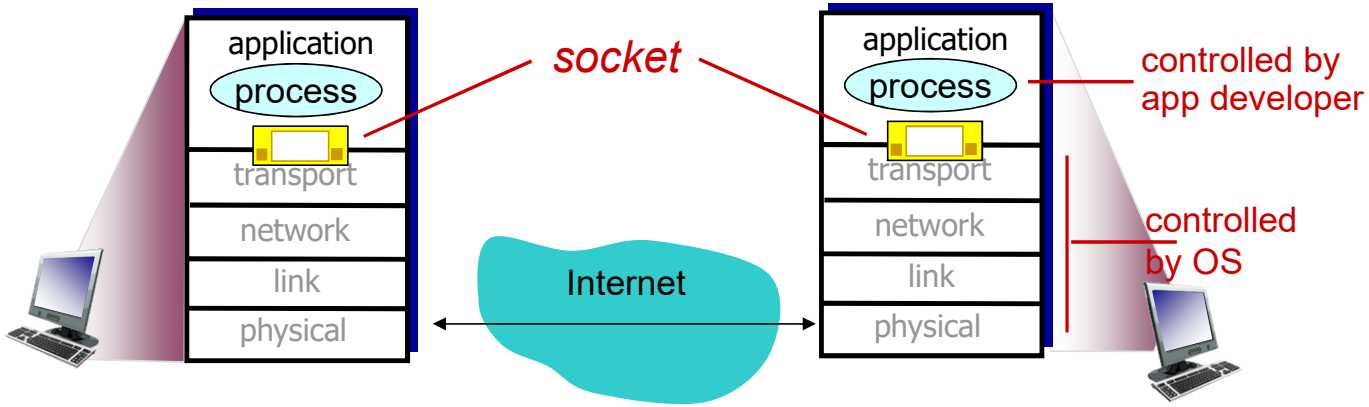
client process: process that initiates communication

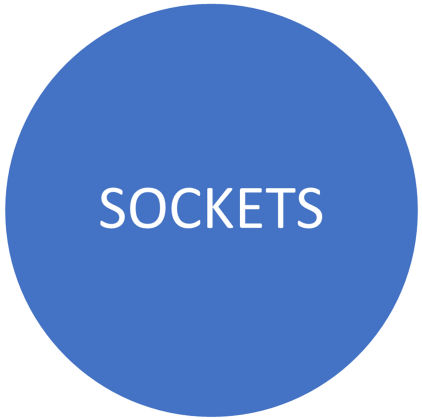
server process: process that waits to be contacted

SOCKETS

SOCKETS

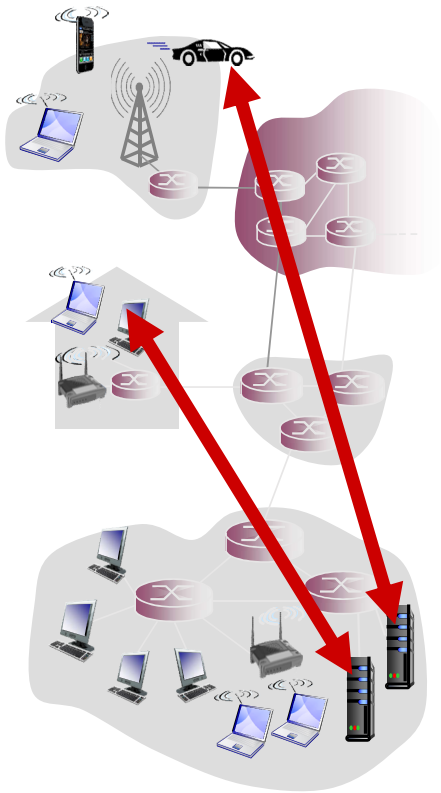
- process sends/receives messages to/from its socket
- **socket** analogous to ***door***
  - sending process pushes the message out of the door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process





CLIENT-SERVER VS. PEER2PEER ARCHITECTURE

client/server



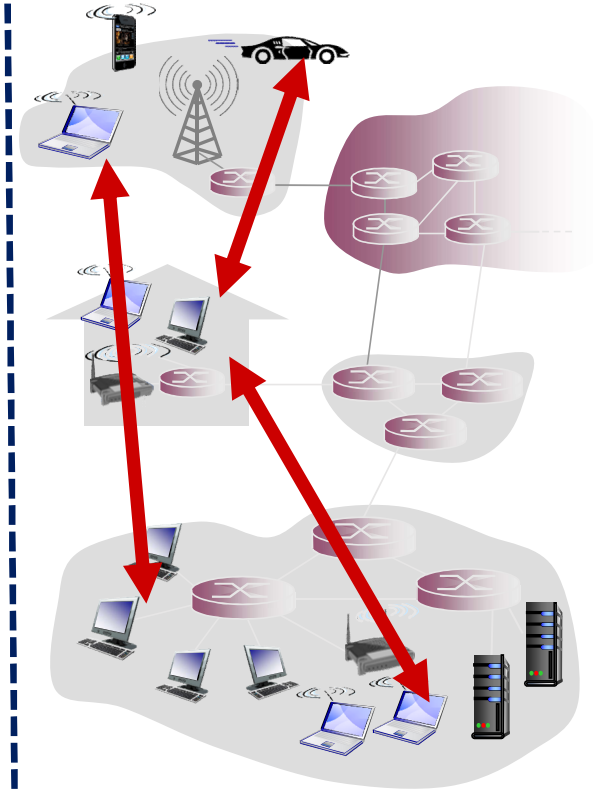
clients:

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

server:

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

peer-2-peer



- arbitrary end systems communicate directly
- *no* always-on server
- peers request service from other peers, provide service in return to other peers
- *self scalability* – peers bring new service capacity, as well as new service demands
- peers change IP ADDR. & are intermittently connected
- complex management

APPLICATION LAYER PROTOCOL DEFINES

- types of messages exchanged:
  - e.g., request, response
- message syntax and semantics:
  - what fields in messages & how fields are delineated
  - meaning of information in fields
- 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., FaceTime, ~~Skype~~, Zoom,..

Application Layer Protocols (L7/L5 protocols) provide common services to different user (networking) applications and build on the underlying transport services.

WHAT SERVICE REQUIREMENTS DO THE APPLICATIONS NEED?

data integrity

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

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

throughput

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

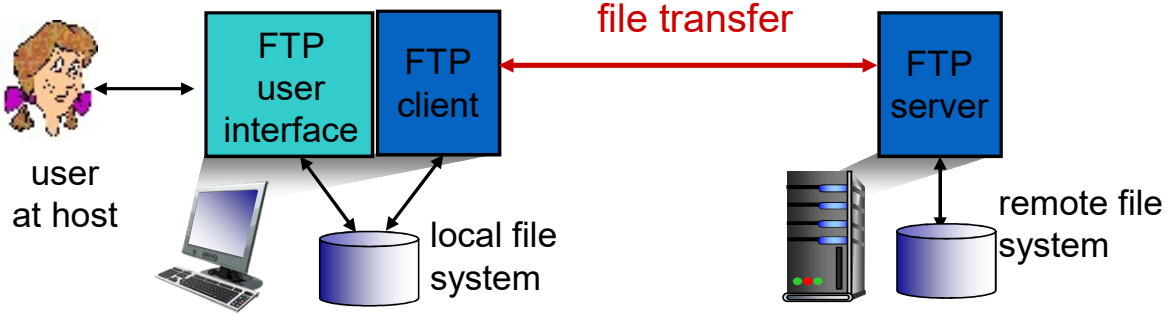
application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes and no
interactive games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

security

- Confidentiality, Integrity, Availability, Privacy, including Authentication, Authorization, Non-repudiation, Accountability, …

Let us look into some specific Networking Applications and associated application Layer Protocols

FTP: THE FILE TRANSFER PROTOCOL



- transfer file(s) to/from remote host
- client/server model
  - *client:* side that initiates transfer
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

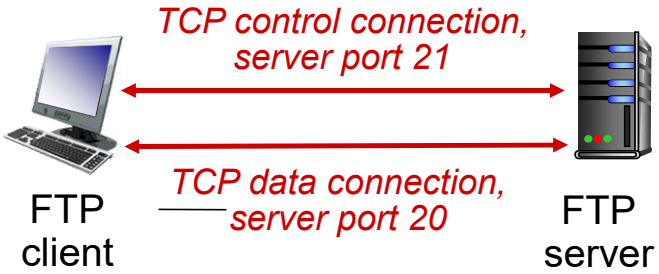
Check out: Abhay K. Bhushan

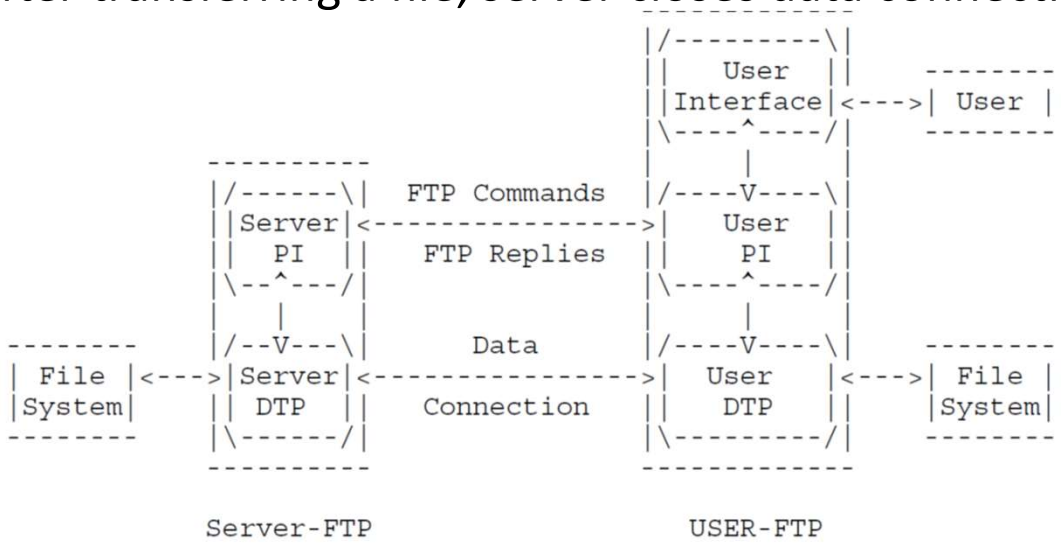
https://en.wikipedia.org/wiki/Abhay_Bhushan

RFC 114

FTP: SEPARATE CONTROL, DATA CONNECTIONS

- FTP client contacts FTP server at **port 21**
- client is *authorized* over control connection
- client *browses* remote directory, ***sends commands*** over control connection
- when server receives file transfer command, *server* opens *2nd* TCP data connection (for file) *to* client
- after transferring a file, server closes data connection





- server opens another TCP data connection to transfer another file
- control connection: *"out of band"*
- FTP server maintains "state": current directory, earlier authentication

FTP COMMANDS, RESPONSES

sample commands (verbs):

- sent as ASCII text over control channel
- **`USER`** ***`username`***
- **`PASS`** ***`password`***
- **`LIST`** return list of file in current directory
- **`RETR filename`** retrieves (gets) file
- **`STOR filename`** stores (puts) file onto remote host

sample return codes

- status code and phrase (as in HTTP)
- **`331 Username OK, password required`**
- **`125 data connection already open; transfer starting`**
- **`425 Can't open data connection`**
- **`452 Error writing file`**

Some Food for Thought

Can you transfer executables via FTP?
Is there a need to distinguish regular text vs binary data?

Try to understand the variants of FTP protocols:
What are they and Why we need them?
FTP and TFTP (Trivial FTP)
SFTP and FTPS (Secure FTP and FTP secure)

aside

PTP and MTP:

- Photo and Media transfer Protocols

Check whether your browser supports FTP or not?
Firefox: FTP support has been removed since July 13 2021
Firefox Release 90 [Link]

In this Session

HTTP Protocol:

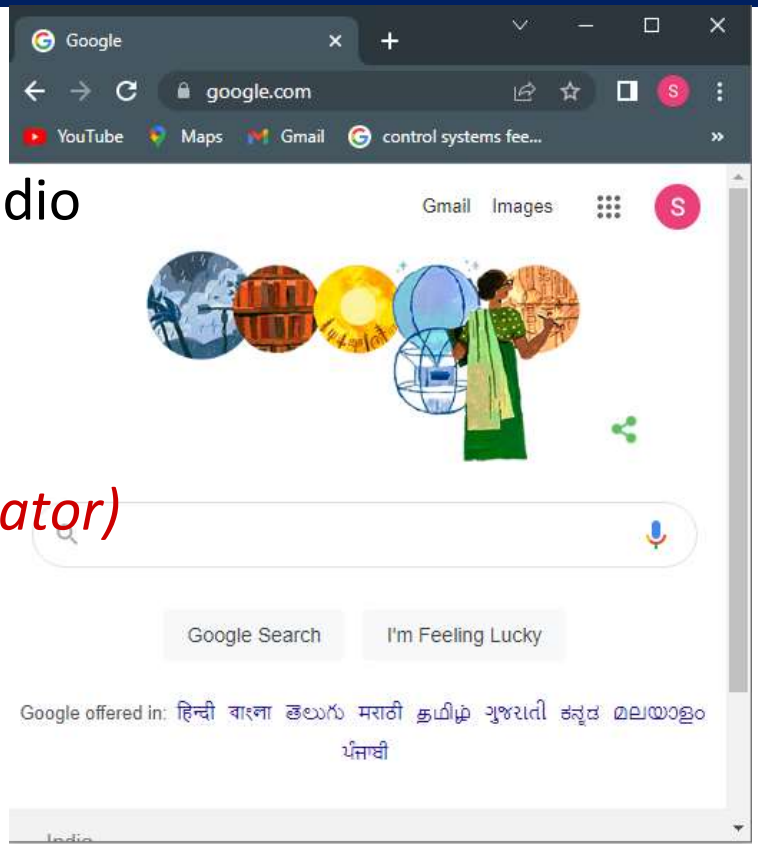
- Stateless, TCP port 80
- Request Message
- Response Message
- HTTP Methods
- Safe and Idempotent

HTTP Protocol:

- Cookies
- Persistent vs Non-Persistent connections
- Head-of-Line Blocking
- Web Caches

WEB AND HTTP

- *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*
- object is addressable by a *URL (uniform resource locator)* e.g.,



`https://www.google.com/home/abc?q=CN#Ans`

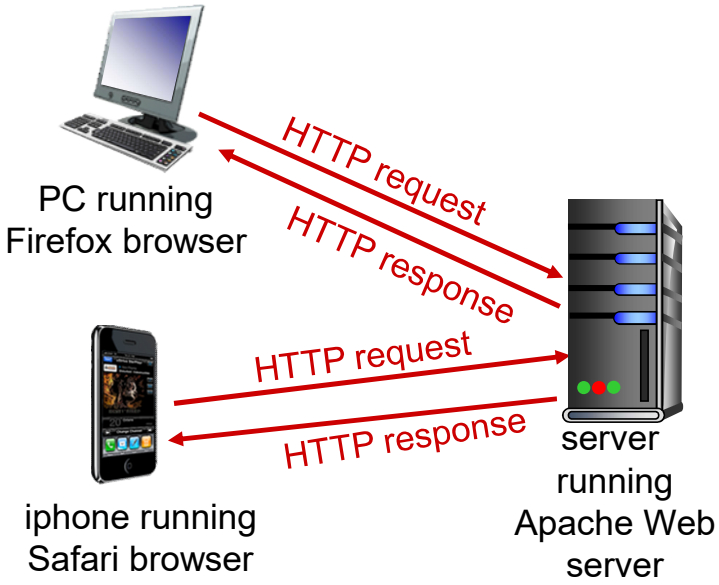
Protocol / Scheme — Authority/hostname Domain:Port — Path — Parameters — Anchor

URL specifies both **location** (host name, path internally to a file) and **protocol** (http, ftp, https) to retrieve object.

APPLICATION LAYER PROTOCOL: HTTP – A QUICK OVERVIEW

HTTP: Hypertext transfer protocol

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

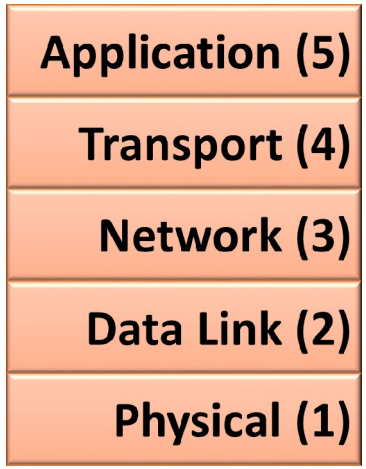


HTTP is "stateless"

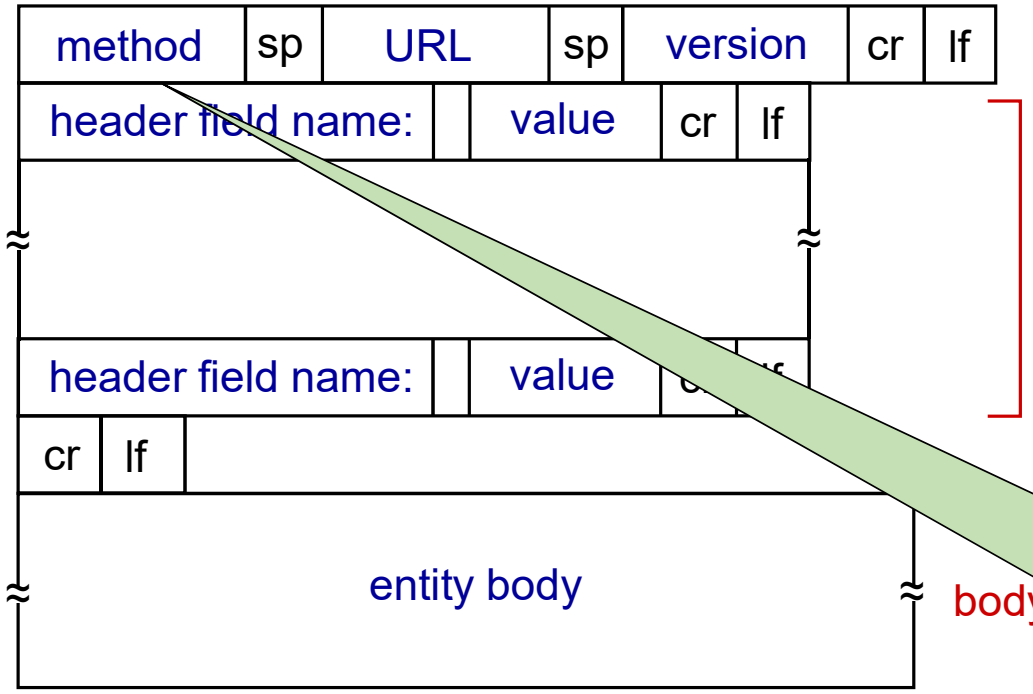
- server maintains no information about past client requests

*Transport layer Service: uses TCP…, **well not necessarily true!***

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



HTTP REQUEST MESSAGE FORMAT (SYNTAX AND SEMANTICS)

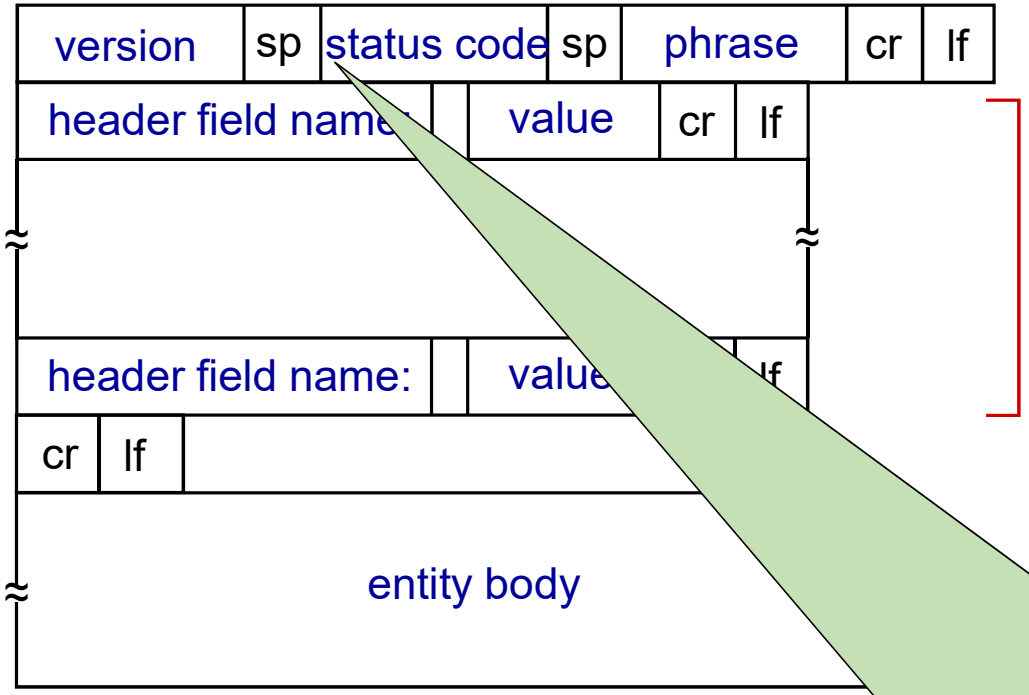



```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html\r\n
Accept-Language:en-us,en;q=0.5\r\n
Accept-Encoding:gzip,deflate\r\n
Accept-Charset:ISO-8859-1\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

```
RFC 1945 - HTTP 1.0
RFC 7321 - HTTP-1.1
RFC 9110 - HTTP Semantics
RFC 9111 - HTTP Caching
RFC 9112 - HTTP-1.1
RFC 9113 - HTTP-2.0
RFC 9114 - HTTP-3.0
```

- GET (0.9)
- HEAD (1.0)
- POST (1.0)
- PUT (1.1)
- OPTIONS (1.1)
- DELETE (1.1)
- TRACE (1.1)
- CONNECT (1.1)
- PATCH (1.1)

HTTP Response message format (Syntax and Semantics)



status line

header lines

body

```
HTTP/1.1 200 OK\r\n
Date: Sun,22 Aug 2021 00:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Mon,23 Aug 2021
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 ...
```

- 200 ➔ OK = *request succeeded, data has the requested object*
- 301 ➔ Moved Permanently = *new location specified in 'location:'*
- 400 ➔ Bad Request = *request message not understood by 'server'*
- 401 ➔ Unauthorized = *wrong or missing authentication 'id' /access*
- 404 ➔ Not Found = *requested data not found on the 'server'*

Idempotent and Safe Methods

Safe methods:

- Do not modify the resources.
- Can be cached and prefetched.

Idempotent methods:

- Can be called multiple times (same outcome)
- Necessary for fault-tolerance.

Method	Idempotent	Safe	Cacheable	Pipelining
OPTIONS	YES	YES	NO	YES
GET	YES	YES	YES	YES
HEAD	YES	YES	YES	YES
PUT	YES	NO	NO	YES*/NO
POST	NO	NO	NO*	NO
CONNECT	NO	NO	NO	NO
DELETE	YES	NO	NO	YES*/NO
PATCH	NO	NO	NO*	NO
TRACE	YES	YES	NO	YES*/NO

All Safe methods are idempotent.

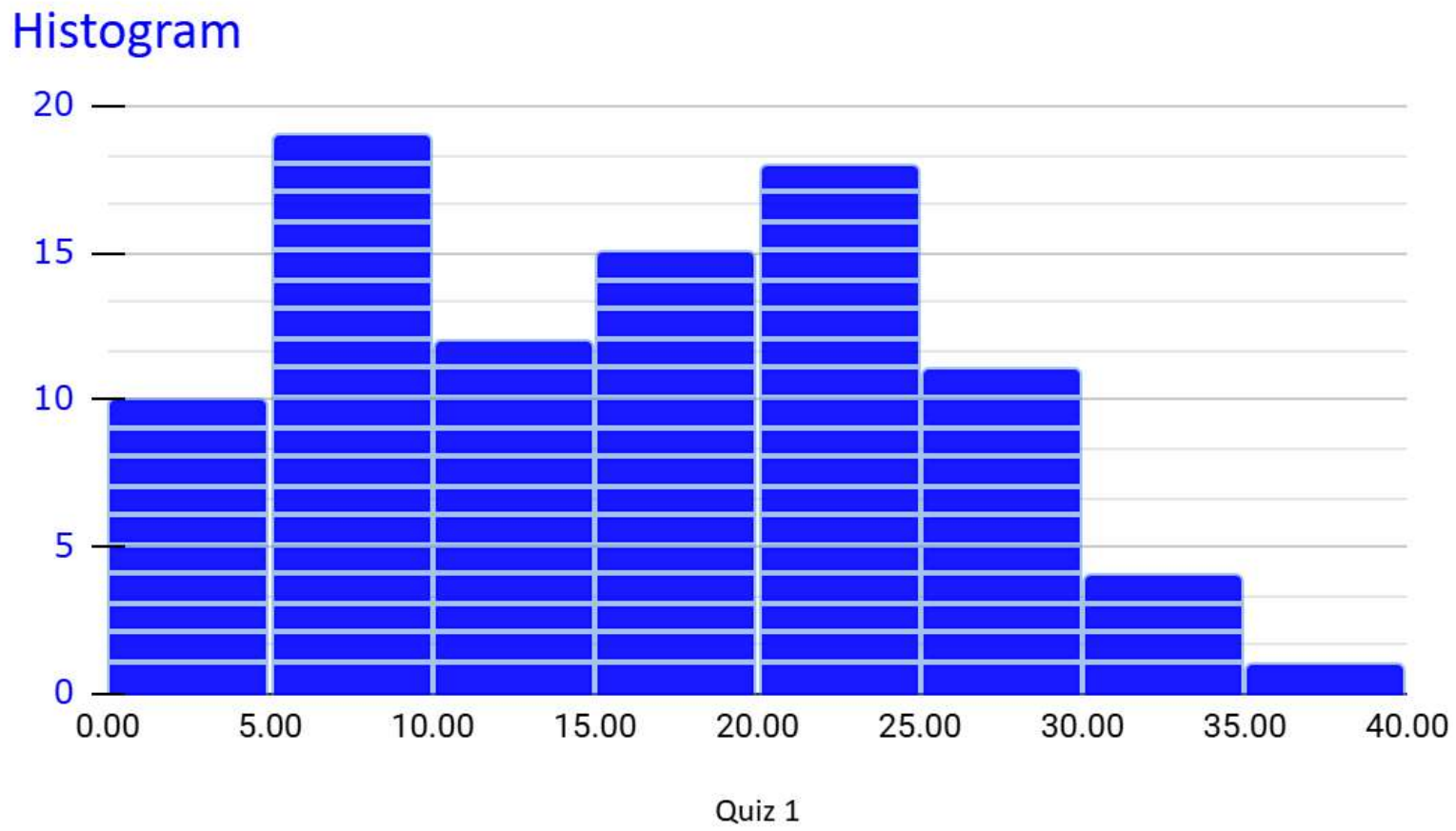
But,

not all idempotent methods are safe.

GET vs POST

Properties	GET (URL)	POST
Data Submission Format	Encoded into the URL as query string parameter.	Sent as a message body of HTTP request.
Data type	Only ASCII characters	Any data type including binary data is allowed.
Data Length	Restricted to URL Length (2048 characters)	No restrictions.
Visibility	Displayed in the browser's address bar. (visible to everyone)	Not displayed/not visible.
Security	Less Secure as data is sent directly as part of URL	Safer than GET as data can be encoded separately
Usability	Not to be used for login/password or other sensitive details.	Recommended for login/password/sensitive data.
Cacheability	Can be cached	Cannot be cached.
Browser History	Parameters are cached in browser history	Parameters are not saved in the browser history
Bookmarkable	Can be bookmarked	Cannot be bookmarked.

Quiz-01 Score Distribution



USER-SERVER STATE: COOKIES

Many Web sites use cookies!!

- a way for Web applications to maintain application state.

*Source: Cartoonstock.com

HTTP is "stateless"

- server maintains no information about past client requests

Is this correct?

What is a Cookie? (HTTP Cookie):

-- usually small text files placed on users device (web browser).

-- help collect user data and enable personalization (user-specific) features and tracking.

Session management

Personalization & Tracking

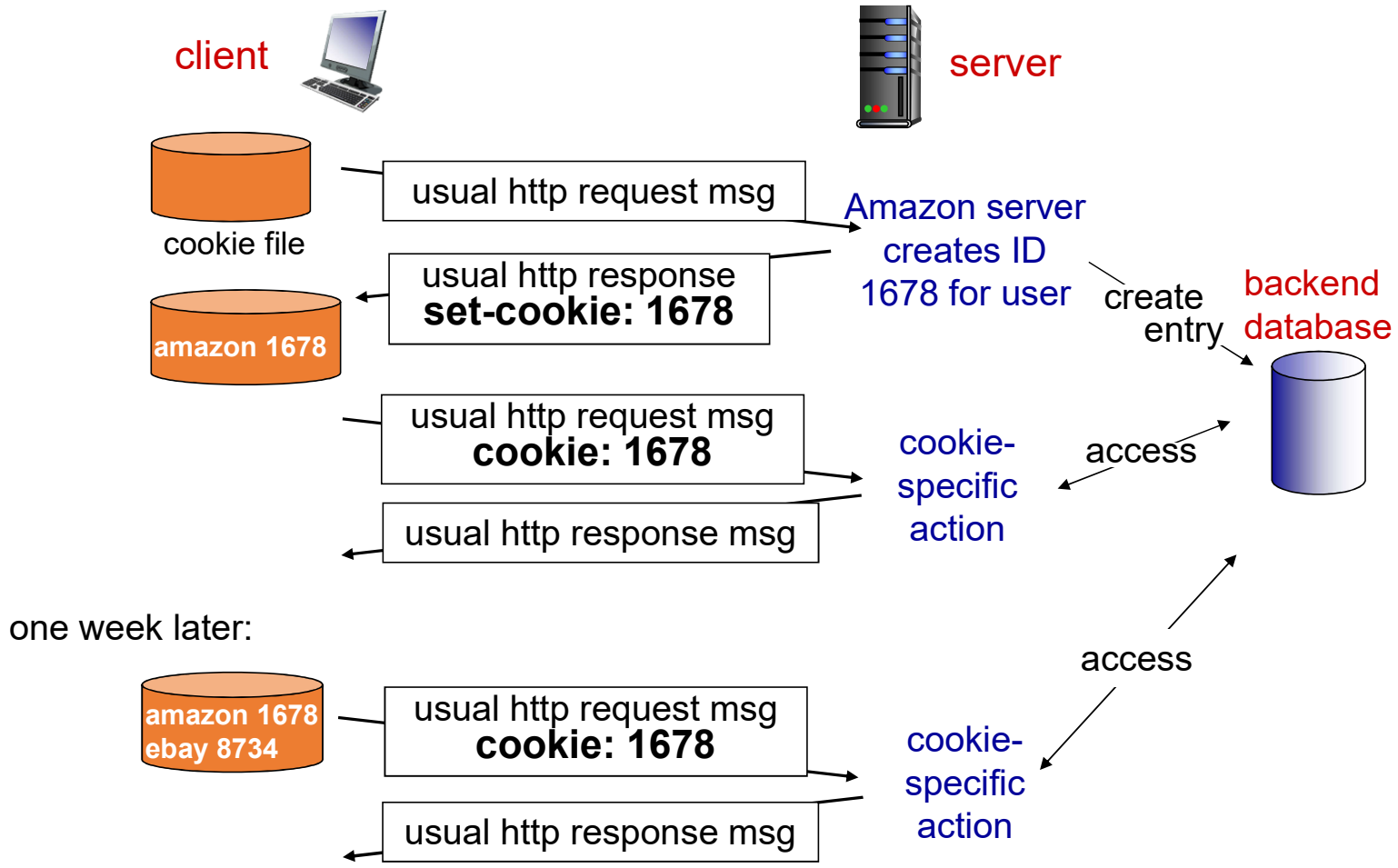
example:

- Suman always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

HTTP is "stateless" but not "Sessionless"

- server maintains no information about past client requests but maintain stateful sessions.

How do the Cookies work?



COOKIES

what cookies can be used for:

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

how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: subsequent http messages carry state

aside

Kinds of Cookies:

- First vs Third party
- Session vs Persistent
- Super Cookies; Zombie Cookies…

cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

Check out

Find out IMS Cookies:

- Find out what cookies does the IMS student portal store when you login?

Private browsing and Cookies:

- What happens to cookies when using private mode of browsing?

TAKEAWAY QUESTION..

Q: How to quantifythe performance of HTTP?

Page Load Time!

Today's Focus:

- Connection Types
- Response Time
- Web Caching.
- Summary of HTTP Evolution!

HTTP CONNECTIONS

Two Types: Non-persistent and Persistent

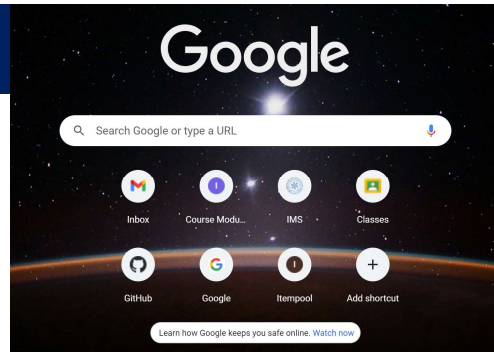
non-persistent HTTP

- at most one object sent over a TCP connection
  - TCP connection then gets closed
- downloading multiple objects require multiple TCP connections

persistent HTTP

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

NON-PERSISTENT HTTP



suppose user enters URL:

`www.Google.com/index.html`

(contains text, references to say 10 jpeg images)

time

1b. HTTP client initiates TCP connection to HTTP server (process) at www.Google.com on port 80

1a. HTTP server at host www.Google.com waiting for TCP connection at port 80.

1c. "accepts" connection, and notifies the client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object gmailicon.jpg

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

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

5. HTTP server closes TCP connection.

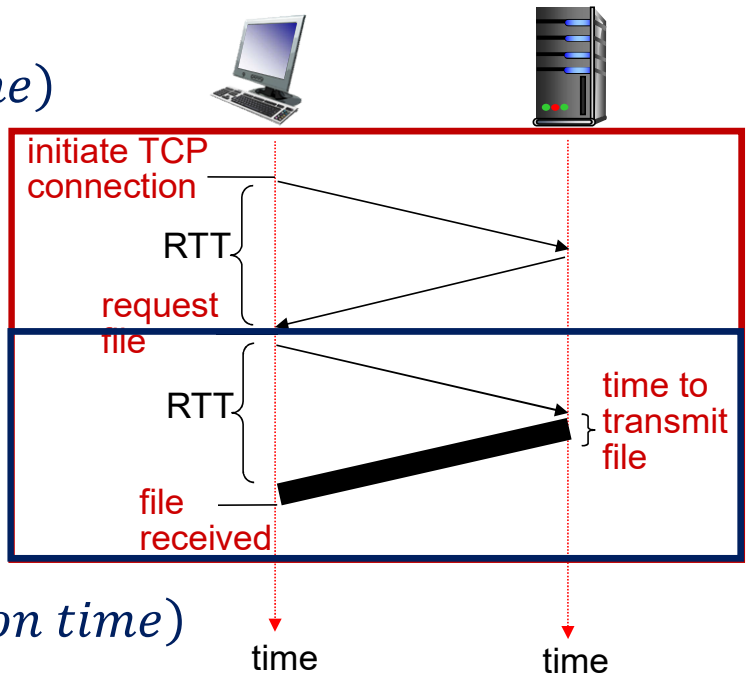
6. Steps 1-5 repeated for each of 10 jpeg objects

WEB PERFORMANCE METRIC: RESPONSE TIME

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

HTTP response time (non-persistent connections):

- one RTT to initiate TCP connection
- one RTT for HTTP request and HTTP response to return:
  - Response includes File transmission time.
- HTTP response time = $\sum_{i=1}^{n}(2RTT + file\ tranmission\ time)$



persistent HTTP connections:

- server leaves connection open after sending response
- subsequent HTTP messages use existing TCP connection
- client sends HTTP requests for all referenced objects
- as little as one RTT for all the referenced objects
- HTTP response time =$RTT + \sum_{i=1}^{n}(RTT + file\ tranmission\ time)$

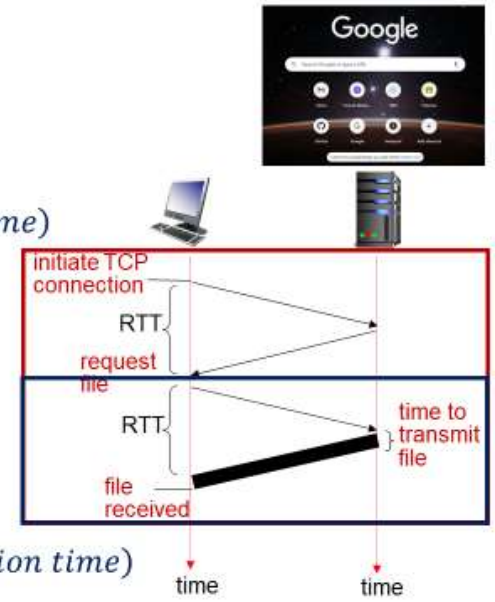
NON-PERSISTENT VS PERSISTENT – *ANY CONCERNS?*

WEB PERFORMANCE METRIC: RESPONSE TIME

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

HTTP response time (non-persistent connections):

- one RTT to initiate TCP connection
- one RTT for HTTP request and HTTP response to return:
  - Response includes File transmission time.
- HTTP response time = $\sum_{i=1}^{n}(2RTT + file\ tranmission\ time)$



persistent HTTP connections:

- server leaves connection open after sending response
- subsequent HTTP messages use existing TCP connection
- client sends HTTP requests for all referenced objects
- as little as one RTT for all the referenced objects
- HTTP response time =$RTT + \sum_{i=1}^{n}(RTT + file\ tranmission\ time)$

IIT Gandhinagar CS433: Computer Networks 30

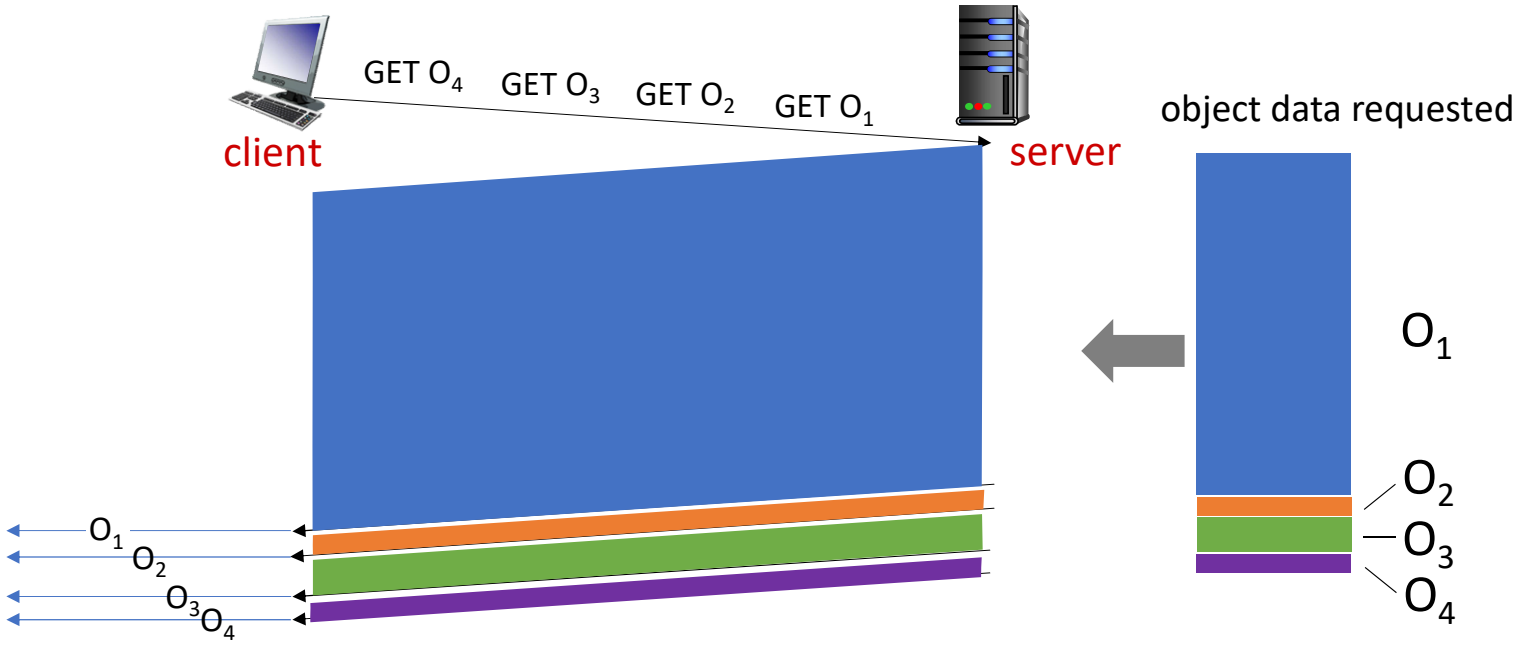
Now, suppose we have a web page with 4 contents to download:

1.) Large video file (4GB), 2) a style file (10KB), and 2 Image files (1MB, 10KB resp.)

HOL BLOCKING

Persistent mode or Persistent + Pipelined mode

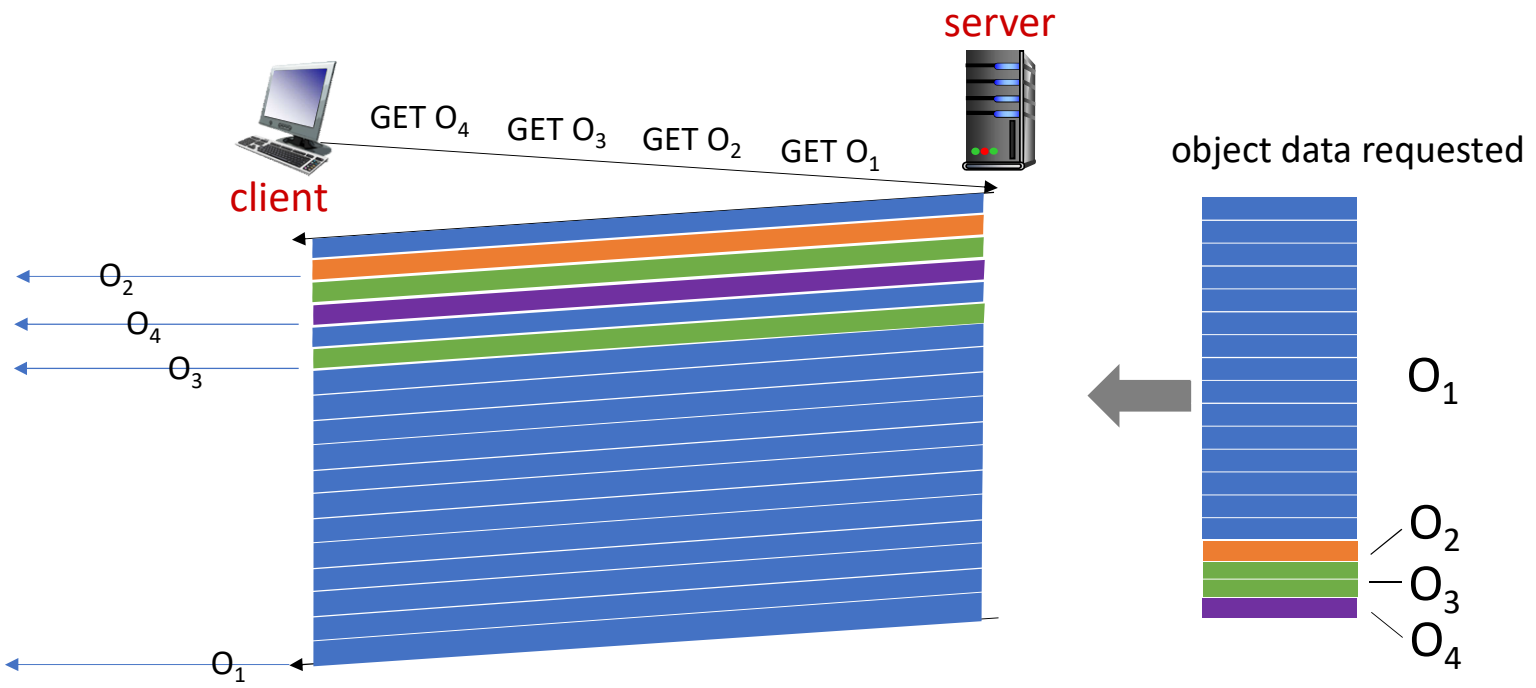
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



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

HTTP/2: MITIGATING HOL BLOCKING

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



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

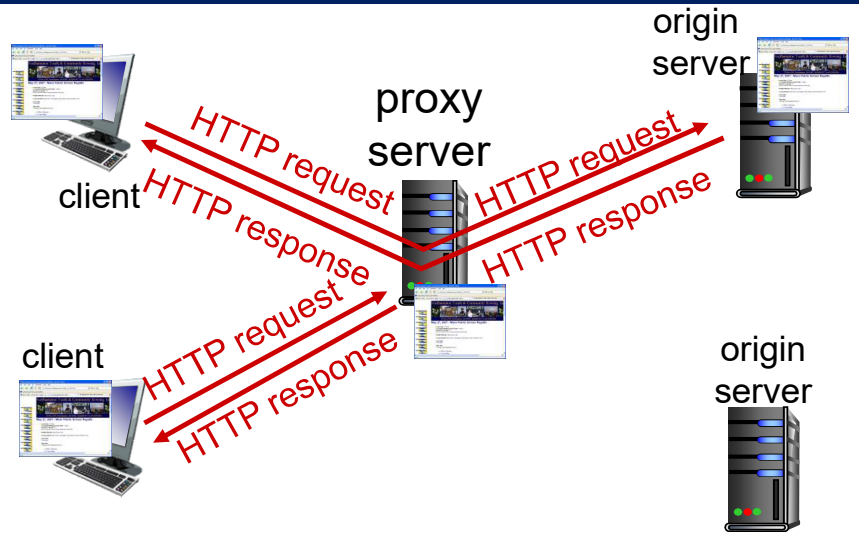
COMPUTER NETWORKS – HEY, DO WE ALSO NEED A CACHE?

- user sets browser: Web accesses via cache
- Browser sends all HTTP requests to cache

CACHE (Proxy Server)

IF(requested object in cache): NOP
ELSE: Request object from origin server;
return object;

- cache acts as both client and server
  - server for the 'original' requesting client
  - client to origin server
- typically, cache is installed by ISP (university, company, residential ISP)



Why Web caching?

- reduce response time for client request
- reduce traffic on access links
- Internet dense with caches: enables "poor" content providers to effectively deliver content

Cache: Helps scale, minimize network traffic and load on the origin server(s).

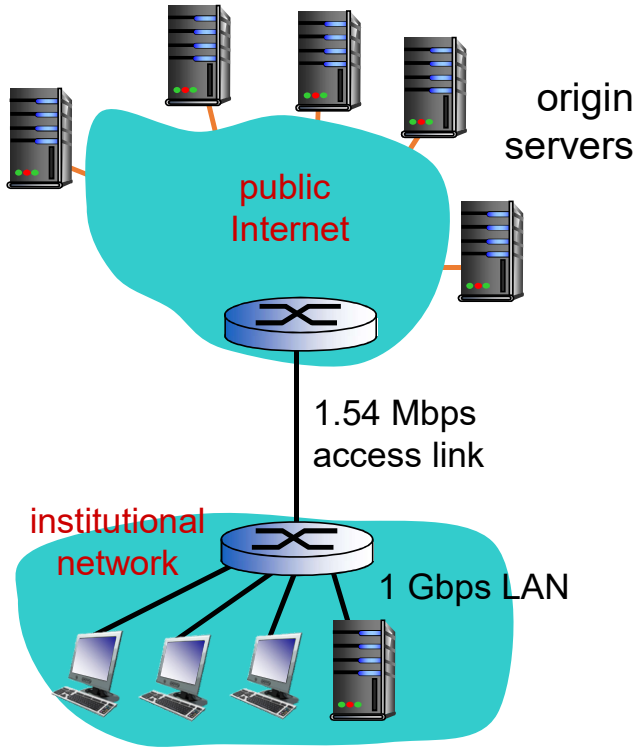
CACHING EXAMPLE:

assumptions:

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

consequences:

- LAN utilization: 0.15%
- access link utilization = 99% *problem!*
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + seconds/msec + usecs



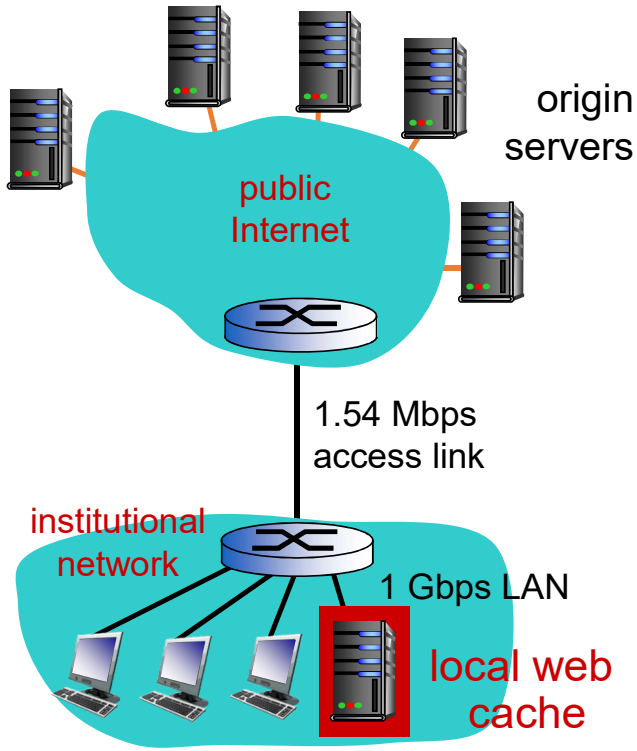
CACHING EXAMPLE: INSTALL LOCAL CACHE

Calculating access link utilization, delay with cache:

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

- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link = 0.6*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .58

- total delay
  - = 0.6 * (delay for origin servers) +0.4 * (delay for cache hits)
  - = 0.6 (2.01) + 0.4 (~msecs)
  - = ~ 1.2 secs
  - less than with 154 Mbps link (and cheaper too!)



CHALLENGES WITH CACHING

Are there any consequences/problems with Caching?

CONDITIONAL HTTP REQUESTS:

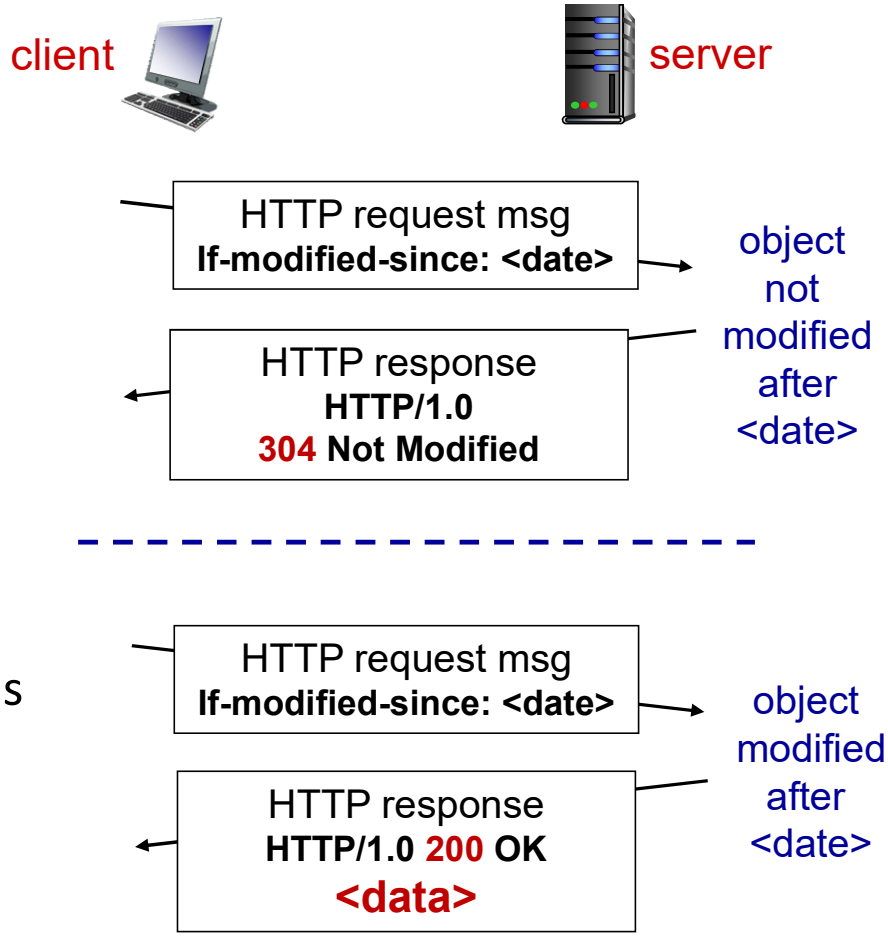
- *Goal:* don't send object if cache (in the client's browser) has up-to-date cached version
  - no object transmission delay
  - lower link utilization

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

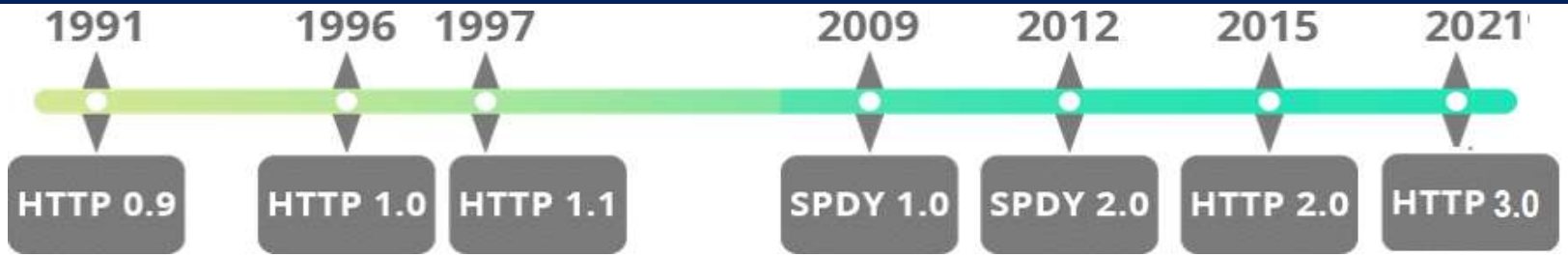
  `If-modified-since: <date>`

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

  `HTTP/1.0 304 Not Modified`



HTTP Evolution



- Hypertext only request/response model.
- One-line GET only.
- No HTTP headers
- NO status/error code

HTTP 0.9

- HTTP headers
- HEAD and POST
- Version, status/error code
- More content types.

HTTP 1.0

- More methods (remaining 6)
- Long-lived connections
- Persistent/pipelined conn.
- Chuncked transfers …

HTTP 1.1

SPDY 1.0

- Multiplexed Stream with prioritization support.
- Binary Framing layer.
- Server Push.
- Better Compression.

SPDY 2.0

HTTP 2.0

HTTP 3.0

- Change Transport protocol
- QUIC secure connection
- Client/Server Migration
- Improved performance
- Enhanced privacy

Reference Materials

More information and Tutorials on HTTP and Web development tools

Mozilla Developer Network – Documentation and Tutorials
https://developer.mozilla.org/en-US/docs/Web/HTTP

W3 Schools: Documentation and Tutorials:
https://www.w3schools.com/tags/ref_httpmethods.asp

IN THIS SESSION

E-mail Protocol(s):

- send email
- receive email
- mail-servers (mail exchanges)

DNS:

- Infrastructure
- Database
- Protocol
- Mode of Operation

YOU 'VE GOT A MAIL!




ELECTRONIC MAIL (E-MAIL)

Three major components:

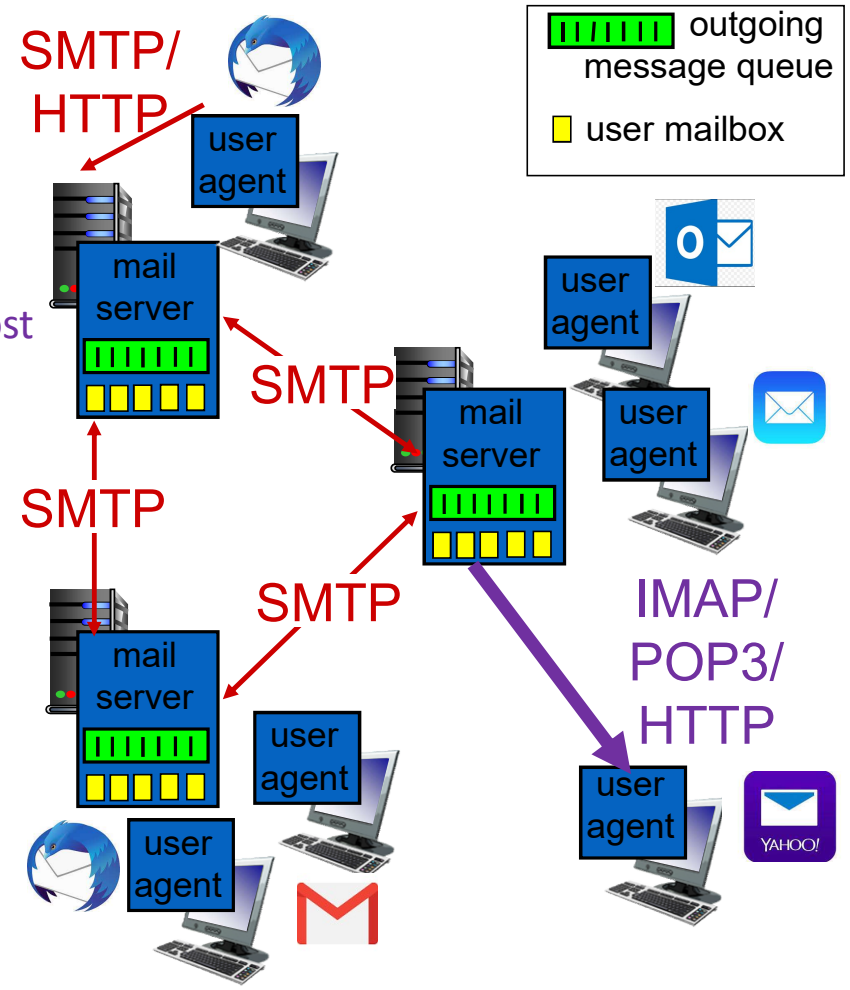
- user agents
- mail servers
- Protocols: To send and Receive e-mails.
  - Send→ ***SMTP***: Simple mail transfer protocol (RFC 5321)
  - Receive → ***IMAP/POP3***: Internet Message Access Protocol / Post Office Protocol-v3 (RFC 9051, RFC 5034)

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

composing, editing, reading mail messages
e.g., Outlook, Thunderbird, iPhone mail client.

mail servers:

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



SCENARIO: ALICE SENDS MESSAGE TO BOB

1) Alice uses UA to compose message "to" `bob@iitgn.ac.in`

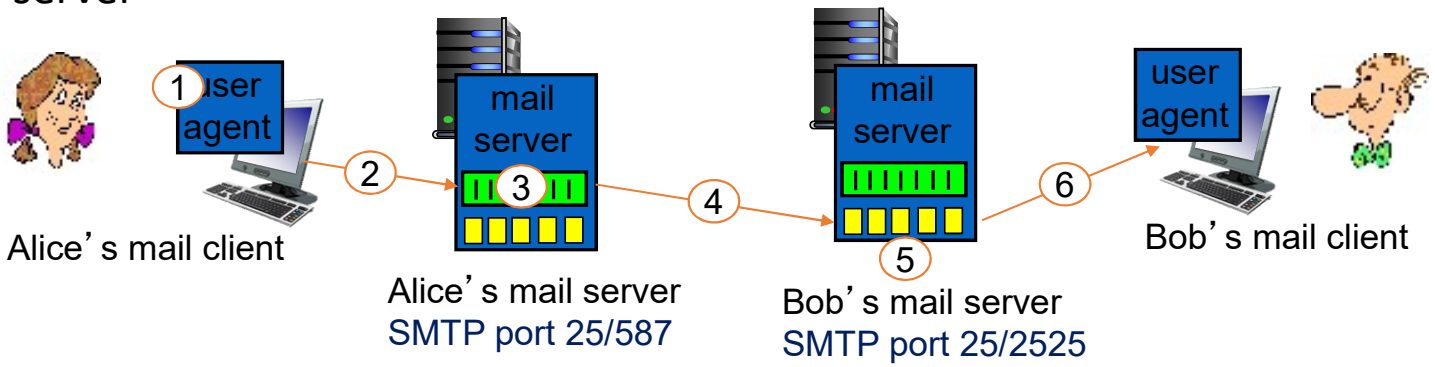
2) Alice's UA sends message to her mail server; message placed in message queue

3) client side of SMTP opens TCP connection with Bob's mail server

4) SMTP client sends Alice's message over the TCP connection

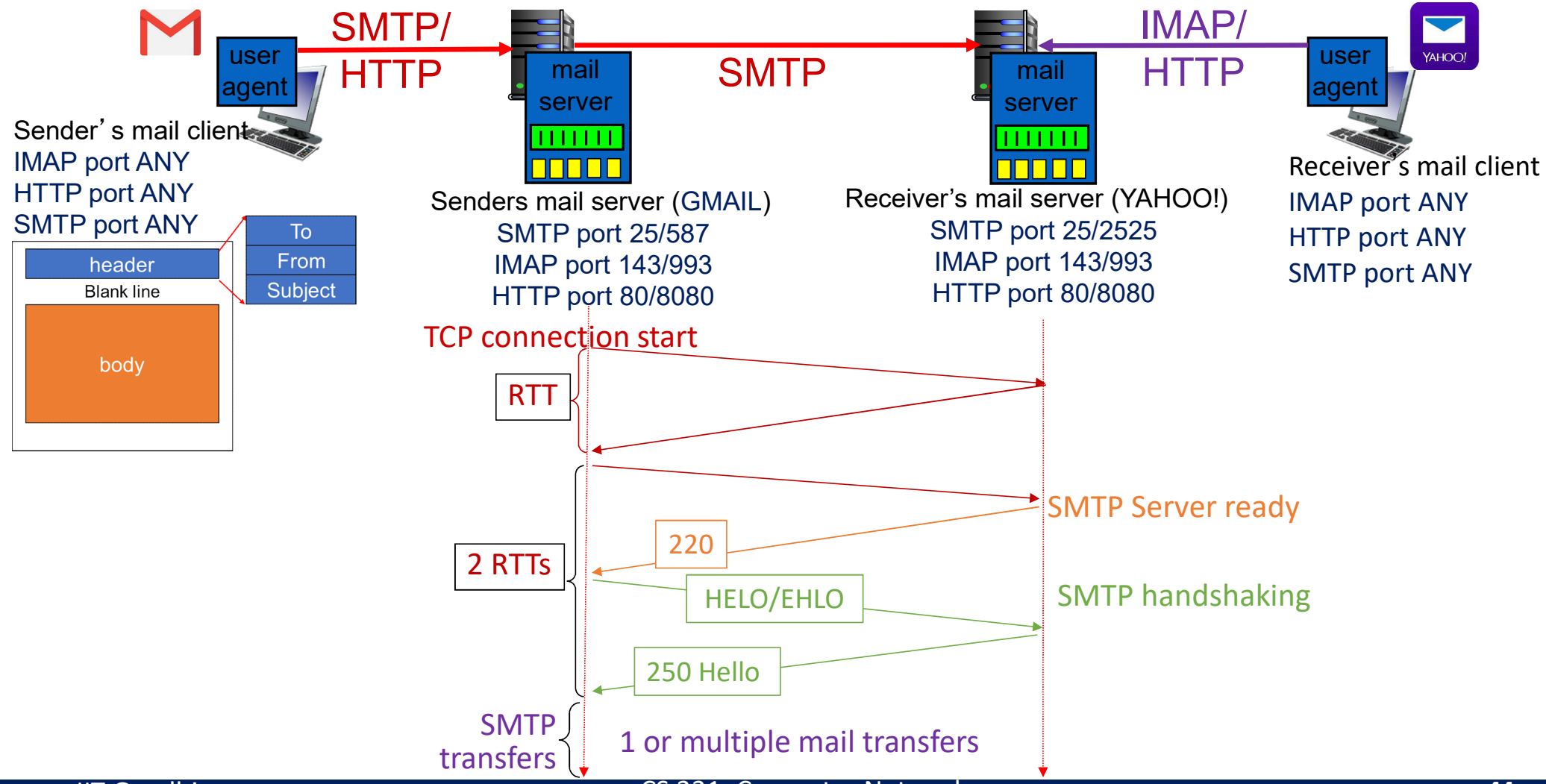
5) Bob's mail server places the message in Bob's mailbox

6) Bob invokes his user agent to read message



- SMTP/IMAP use persistent TCP connections
- SMTP/IMAP require message (hdr, body) to be in 7-bit ASCII
- SMTP server uses `CRLF.CRLF` to determine end of message.

EMAIL DELAYS AND FAILURE TO DELIVER THE MESSAGES!



In this Session

DNS:

- Infrastructure
- Database
- Protocol
- Mode of Operation

~~P2P and CDN:~~

- ~~Infrastructure~~
- ~~Management~~
- ~~Hashing~~
- ~~Replication~~

DNS: DOMAIN NAME SYSTEM

people: many identifiers:

- name, PAN#, Aadhar#, passport #, DL#

Internet hosts, routers:

- IP address - used for addressing datagrams
- "URL", 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 many *name servers* organized as a hierarchy
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation) to addresses
  - note: this is a core Internet function, implemented as an *application-layer protocol*
- DNS is a *critical network function* that a *user may not necessarily interact with*

What do we have on Internet and Where?

DNS = Phonebook of Internet.

WORKING OF DNS: EXAMPLE – WEB APP

- **Domain Name**: *A human-readable address (mapped to a IP address), that defines a realm of administrative autonomy/authority within the Internet.*

- To send HTTP request message to Web server: *http://www.google.com.*
  - user machine runs client side of the DNS application
  - browser extracts hostname from URL, passes it to client side of DNS application.
  - **DNS client** sends query containing hostname *www.google.com.* to a **DNS server**
  - DNS client receives reply - including IP address for hostname. → *172.217.26.132.*
  - Once browser receives IP address from DNS, it can initiate a TCP connection to HTTP server at port 80 at that IP address.

- Typically DNS query/reply messages are sent using **UDP datagrams.**
- **DNS service: TCP/UDP port 53**.

DATABASE VIEW: *DNS RECORDS*

DNS: distributed db storing resource records (RR)

https://www.nslookup.io/

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

- A
- AAAA
- NS
- CNAME
- MX
- PTR
- SRV
- SOA
- CAA,
- DNSKEY
- RRSIG
- *& many more*

type=A → *IP address of a given domain*

- **name** is hostname
- **value** is IP address

`(wikipedia.org, 103.102.166.224, A, 499)`

type=NS → *server name containing the DNS records*

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

`(example.com, ns1.example.com, NS, 14400)`

type=CNAME → *name points to a different domain name*

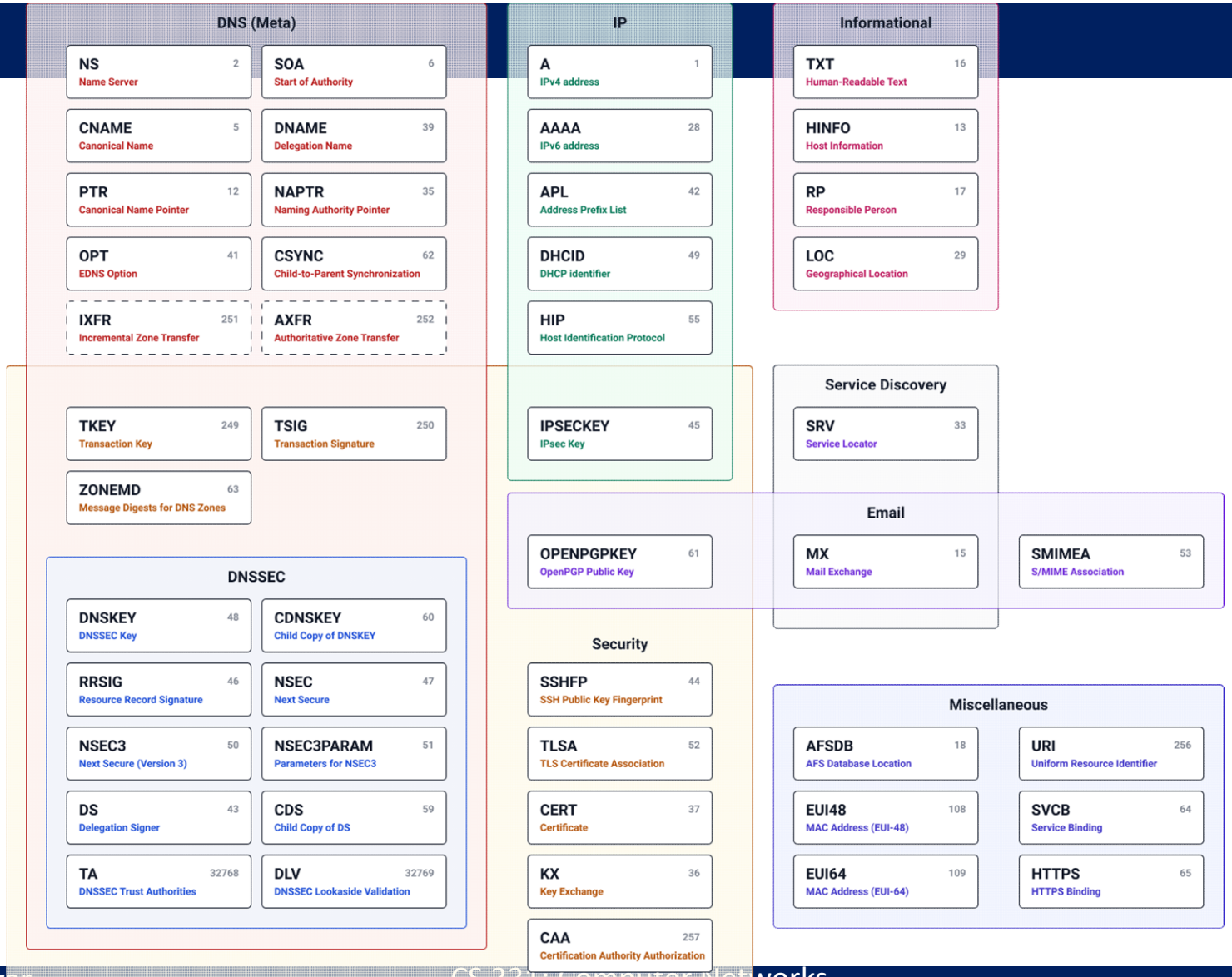
- **name** is an ***alias*** name for some "***canonical***" (the real) name
- `mail.yahoo.com` is really `edge.qycpi.b.yahoodns.net`
- **value** is canonical name

`(mail.yahoo.com, edge.gycpi.b.yahoodns.net, CNAME, 14400)`

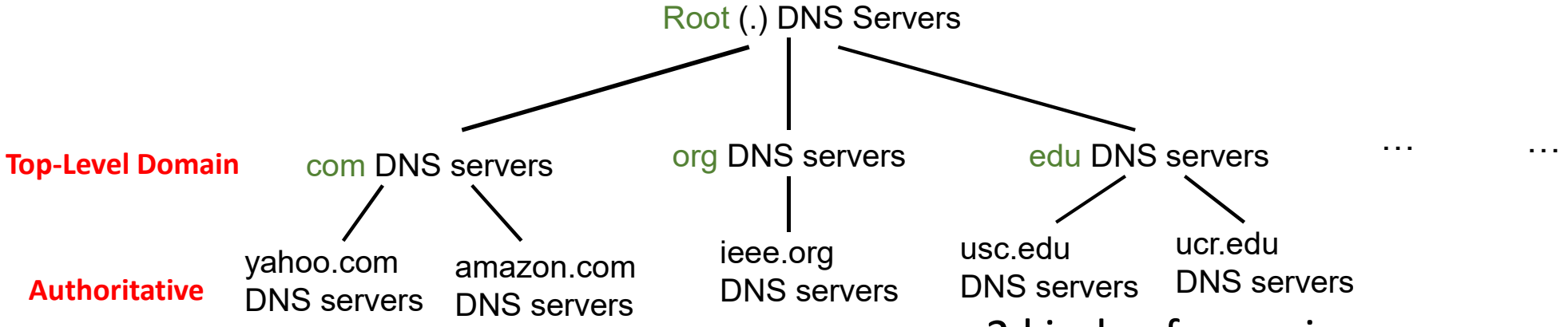
type=MX

- **value** is name of mailserver associated with **name**
- **Priority*** if multiple records

`(gmail.com, gmail-smtp-in.l.google.com., MX, 1386, 5)`


DNS (Meta)
NS 2 Name Server
SOA 6 Start of Authority
CNAME 5 Canonical Name
DNAME 39 Delegation Name
PTR 12 Canonical Name Pointer
NAPTR 35 Naming Authority Pointer
OPT 41 EDNS Option
CSYNC 62 Child-to-Parent Synchronization
IXFR 251 Incremental Zone Transfer
AXFR 252 Authoritative Zone Transfer
TKEY 249 Transaction Key
TSIG 250 Transaction Signature
ZONEMD 63 Message Digests for DNS Zones
DNSSEC
DNSKEY 48 DNSSEC Key
CDNSKEY 60 Child Copy of DNSKEY
RRSIG 46 Resource Record Signature
NSEC 47 Next Secure
NSEC3 50 Next Secure (Version 3)
NSEC3PARAM 51 Parameters for NSEC3
DS 43 Delegation Signer
CDS 59 Child Copy of DS
TA 32768 DNSSEC Trust Authorities
DLV 32769 DNSSEC Lookaside Validation
IP
A 1 IPv4 address
AAAA 28 IPv6 address
APL 42 Address Prefix List
DHCID 49 DHCP identifier
HIP 55 Host Identification Protocol
IPSECKEY 45 IPsec Key
Informational
TXT 16 Human-Readable Text
HINFO 13 Host Information
RP 17 Responsible Person
LOC 29 Geographical Location
Service Discovery
SRV 33 Service Locator
Email
OPENPGPKEY 61 OpenPGP Public Key
MX 15 Mail Exchange
SMIMEA 53 S/MIME Association
Security
SSHFP 44 SSH Public Key Fingerprint
TLSA 52 TLS Certificate Association
CERT 37 Certificate
KX 36 Key Exchange
CAA 257 Certification Authority Authorization
Miscellaneous
AFSDB 18 AFS Database Location
URI 256 Uniform Resource Identifier
EUI48 108 MAC Address (EUI-48)
SVCB 64 Service Binding
EUI64 109 MAC Address (EUI-64)
HTTPS 65 HTTPS Binding

INFRASTRUCTURE VIEW OF DNS: *A DISTRIBUTED, HIERARCHICAL DATABASE*



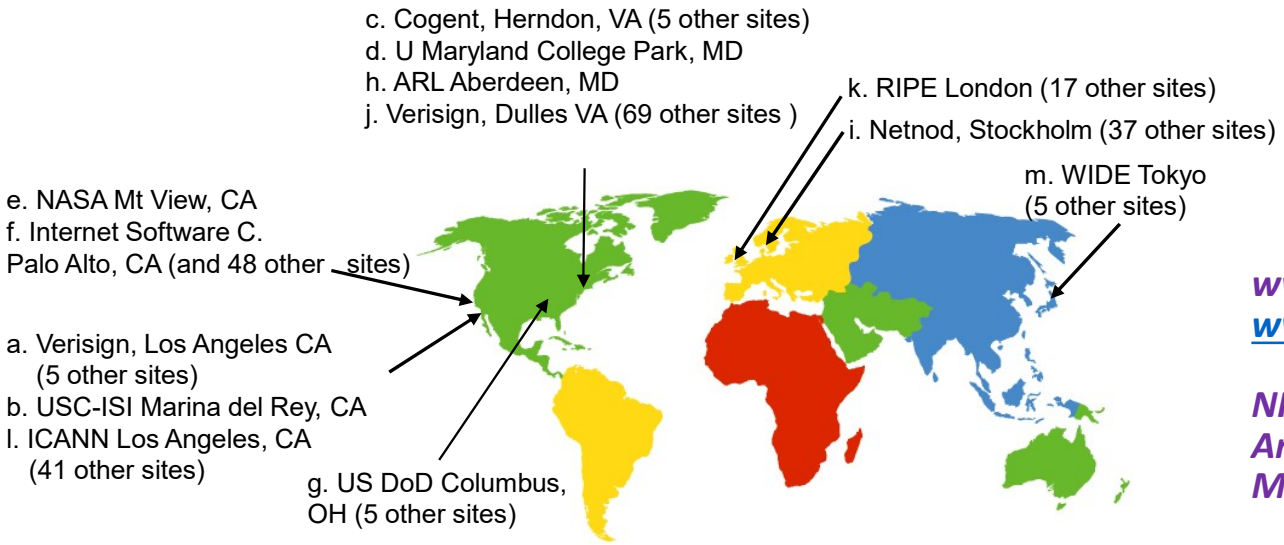
- A tree structure starts with the root (.)
- Each node represents a domain name
- Valid domain names:
  - Each component: [a-zA-Z0-9\-]{1,63}
  - Note: symbols such as (*), (_) (!) are not allowed!
  - Full domain name < 256 chars
- Case insensitive
  - www.cs.iitgn.ac.in is same as WWW.CS.IITGN.AC.IN

Refer to RFC1035 for more details on: DOMAIN NAMES – Specification & Implementation

- 2 kinds of mapping:
  - Forward mapping
    - hostnames → IP addresses
  - Reverse mapping
    - IP addresses → hostnames
- Fully Qualified Domain Name
  - **turing**.iitgn.ac.in.
    - turing– logical hostname
    - iitgn.ac.in – domain name
    - The last dot "." – root

DNS: ROOT NAME SERVERS

- contacted by local name server
- root name server:
  - contacts authoritative name server (AS) if mapping not known
  - gets mapping from AS
  - returns mapping to local name server



13 root name "servers" worldwide**

10 in USA
01 each in Netherlands, Sweden and Japan

But, -- over 1000 root-server instances across world.

www.iana.org/domains/root/servers
www.root-servers.org

NIXI – (Almost around 54 root server instances) Amritsar, Bangalore, Chennai, Hyderabad, Mumbai, Kolkatta, Delhi

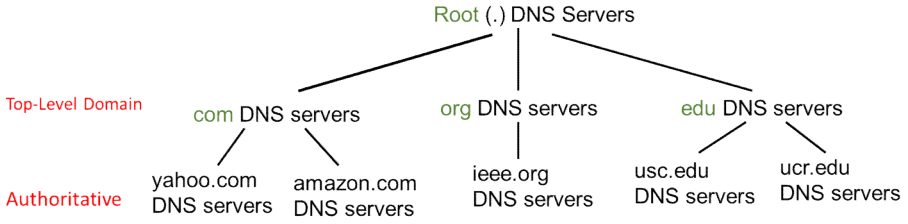
TOP LEVEL DOMAIN AND AUTHORITATIVE SERVERS

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp, de

- gTLDs (generic TLDs)
  - com, edu, net, org, gov, mil, int, arpa
  - aero, biz, coop, info, jobs, museum, name, pro

- ccTLDs (country code TLDs)
  - au, ca, br, de, fi, fr, jp, se, hk, cn, tw, my, …
  - Network Solutions maintains servers for .com TLD
  - Educause for .edu TLD

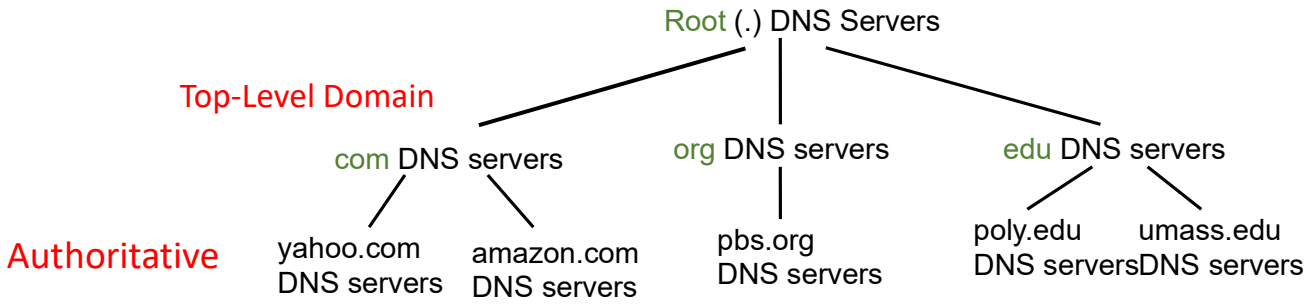


Authoritative DNS servers:

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

As of June 2020, the IANA root database includes 1,514 TLDs, and 316 ccTLDs
As of March 2021, the IANA root database includes 1,589 TLDs, and 316 ccTLDs
As of April 2022, the IANA root database includes 1,591 TLDs, and 316 ccTLDs
As of Aug 2023, the IANA root database includes 1,49x TLDs, and 308 ccTLDs
As of Jan 2025, the IANA root database includes 1,601 TLDs, and 341 ccTLDs

DNS NAME RESOLUTION: RECURSIVE VS. NON-RECURSIVE



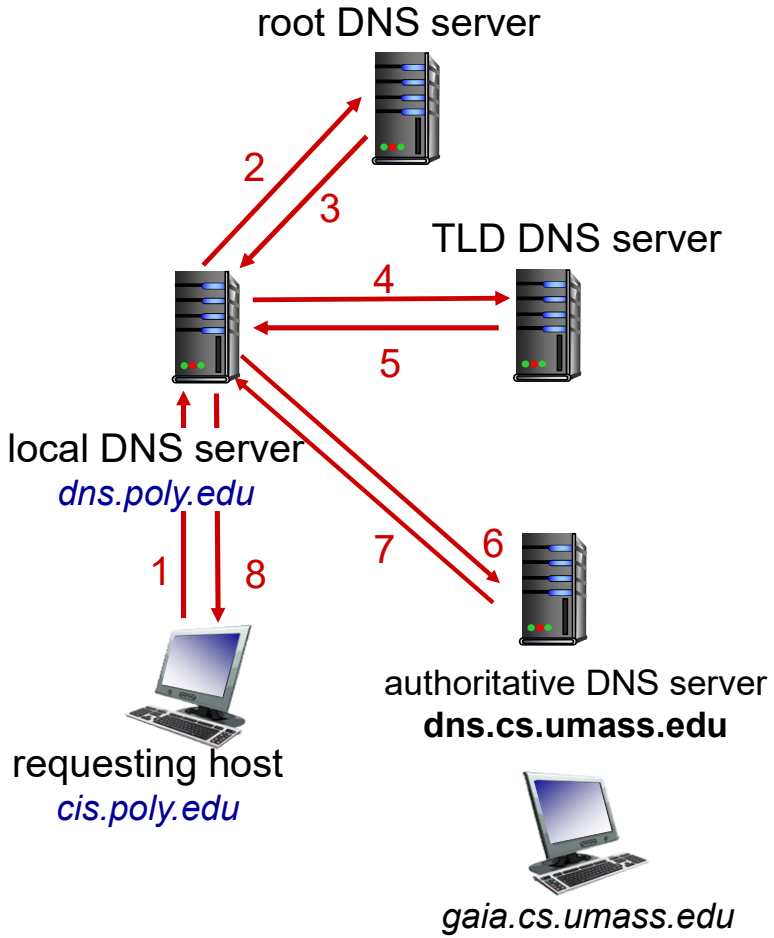
- How it works? *: Delegation*
  - All name servers read all the **root servers** from a local configuration file.
  - Those root servers in turn knows all the TLDs.
    - ***.com*** knows *.google.com*
    - ***.in*** knows *.ac.in*
- Recursive
  - As a DNS resolver:
  - Queries on a client behalf until it returns either an answer or an error
- Non-recursive (Iterative)
  - Refers the client to another server if it can't answer a query.

DNS NAME RESOLUTION (EXAMPLE)

- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

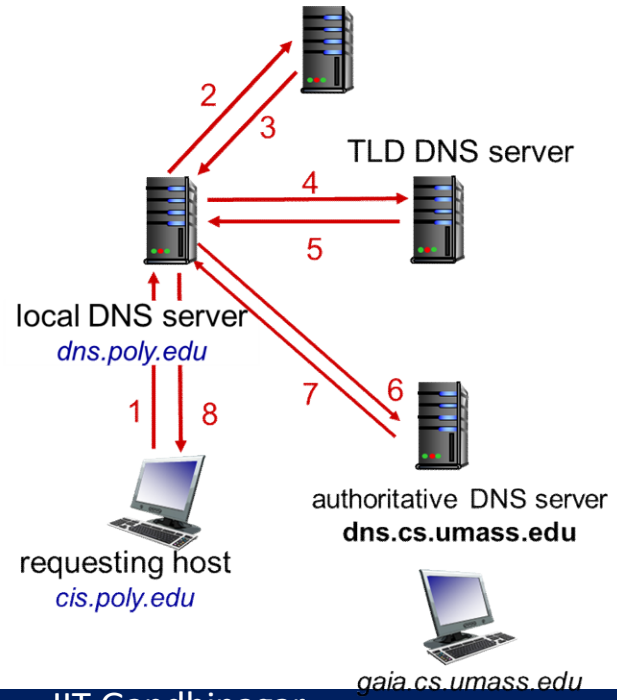
- contacted server replies with the name of server to contact
- "I don't know this name, but ask this server"

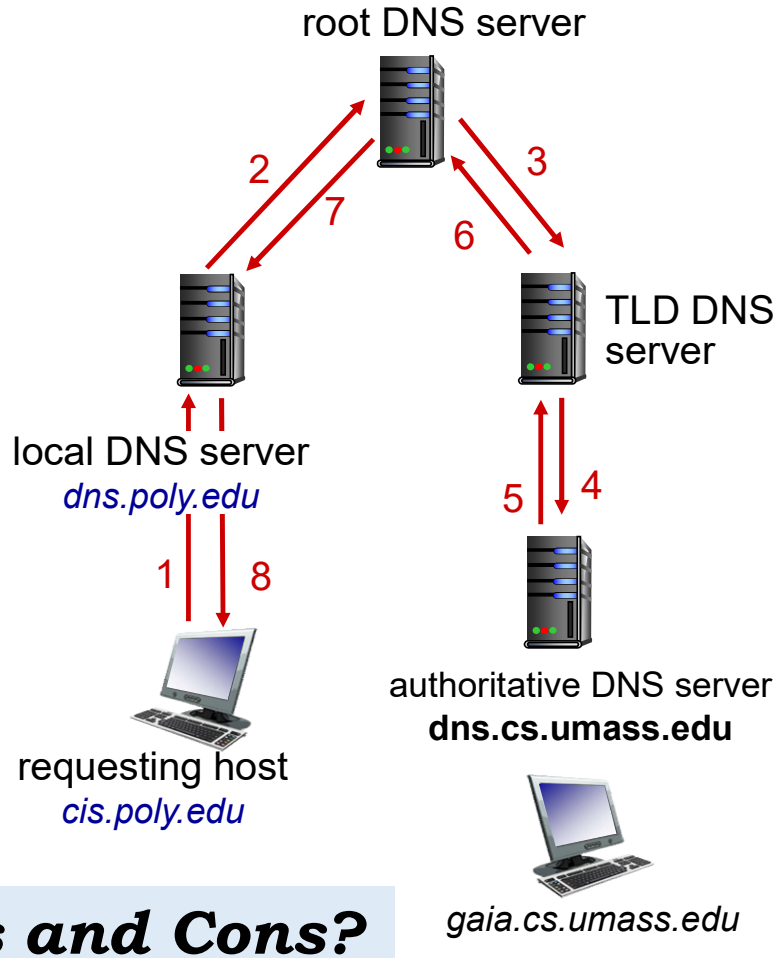


DNS Name Resolution (Example)

recursive query:

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

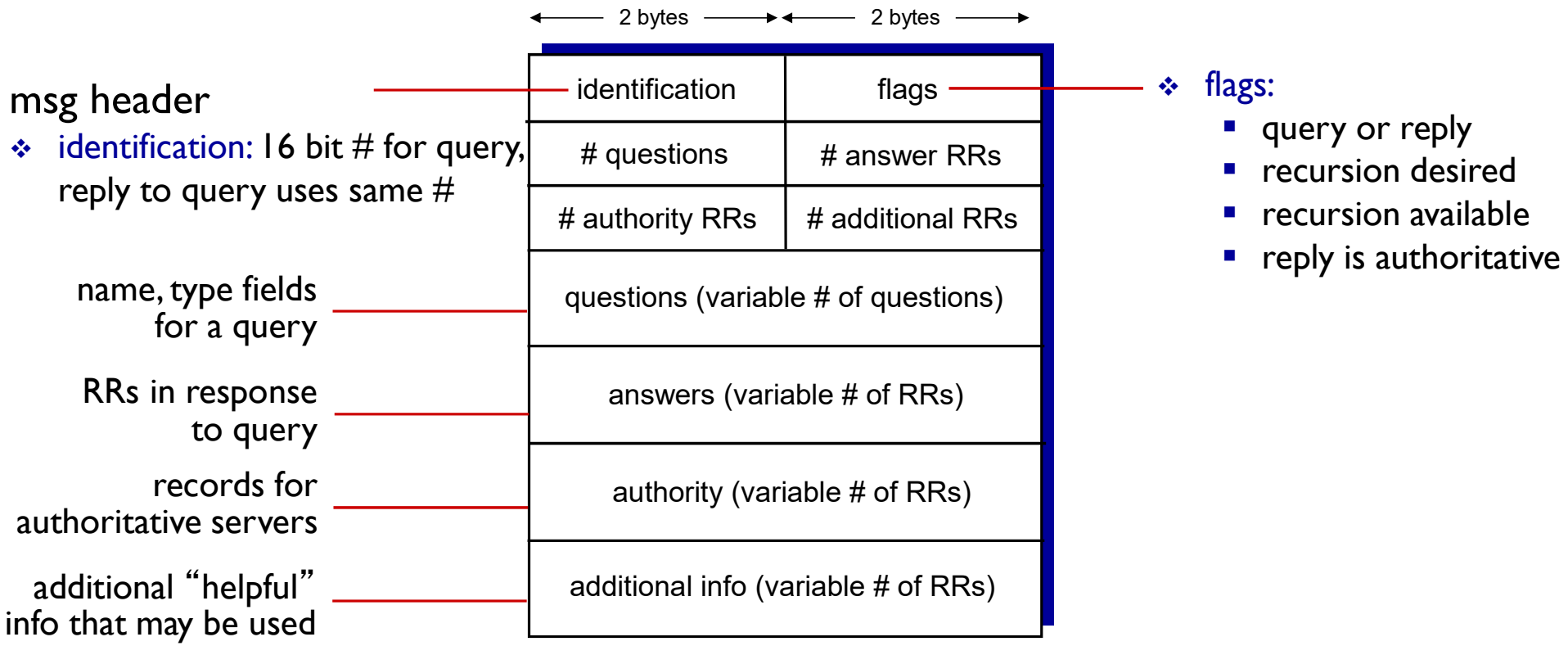




What are the pros and Cons? Any Better Alternatives?

DNS PROTOCOL, MESSAGES

- *query* and *reply* messages, both with same *message format*
- *May use TCP or UDP as underlying transport protocol.*



WHAT IT MEANS TO START A WEBSITE – INSERTING NEW RECORDS INTO DNS

- example: A new IIT starts, say "IIT Ahmedabad"
- register the name iitad.ac.in at *DNS registrar* (e.g., registry.in)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts two RRs into .in TLD server:

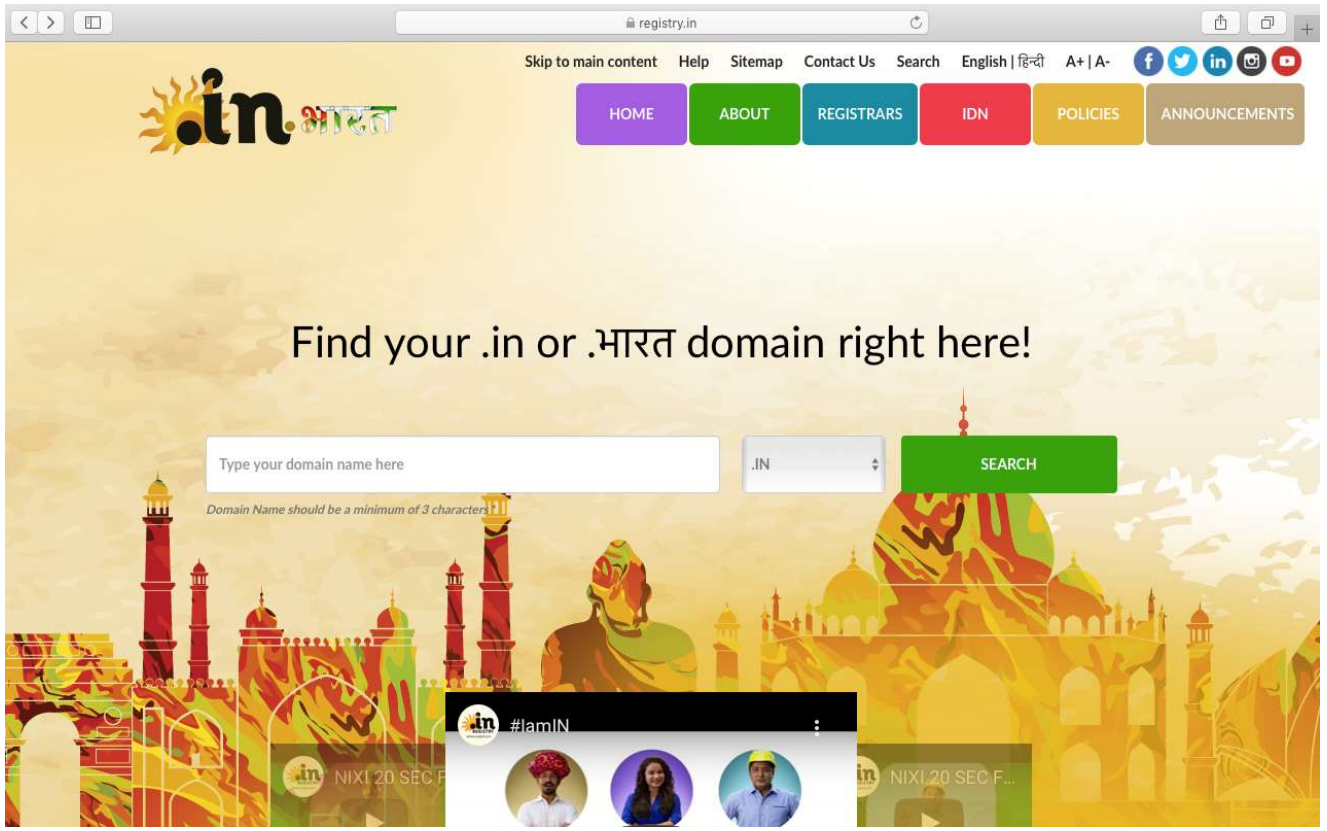
```
(iitad.ac.in, dns1.iitad.ac.in, NS)
(dns1.iitad.ac.in, 102.103.105.105, A)
```

- Create an authoritative server locally with IP address 102.103.105.105
  - type A record for www.iitad.ac.in

```
(iitad.ac.in, 102.103.105.106, A)
(iitad.ac.in, 102.103.105.106, MX)
```

Registration of Second-level Domain Name

- Examples
  - iitgn.ac.in,
  - gwdg.de
  - ucr.edu
  - sony.co.jp
- Must apply to a ***Registrar*** for the appropriate TLD.
- More than a 1000 registrars worldwide.



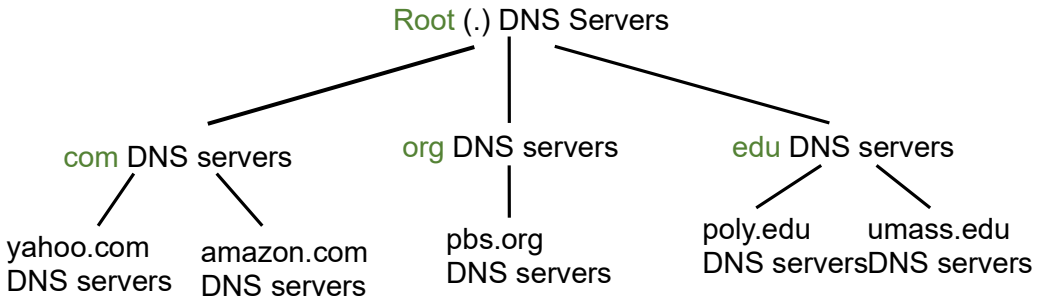
ADDITIONAL FUNCTIONS OF DNS

- Key DNS services in addition to translating host names to IP addresses
- **Load distribution.** DNS also used to perform load distribution among replicated servers, such as replicated Web servers.
  - E.g., ***yahoo.com***, replicated over multiple servers, with each server running on a different end system and each having a different IP address
  - DNS database contains set of IP addresses. When clients make a DNS query for a name that maps to a set of addresses, DNS server responds with entire set of IP addresses. But, rotates ordering of the addresses within each reply.
  - Client policy: result in load balancing across servers
- **Host aliasing** and **mail-server aliasing.** Hostname could have, e.g., two aliases such as ***example.com*** and ***example.net; -- CANNONICAL HOSTNAMES, EMAIL-SERVERS***
  - *Try ping **bbc.com** and **bbc.co.uk***

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
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if named host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard RFC 2136

DNS – Attacks (From and To)



[2016 Dyn Cyberattacks]

[2020 The Aws Attack]

DDoS attacks

- Bombard root servers with traffic
  - Happened long ago in October 2002.
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - Potentially more dangerous – is happening more frequently with Botnets

Redirect attacks

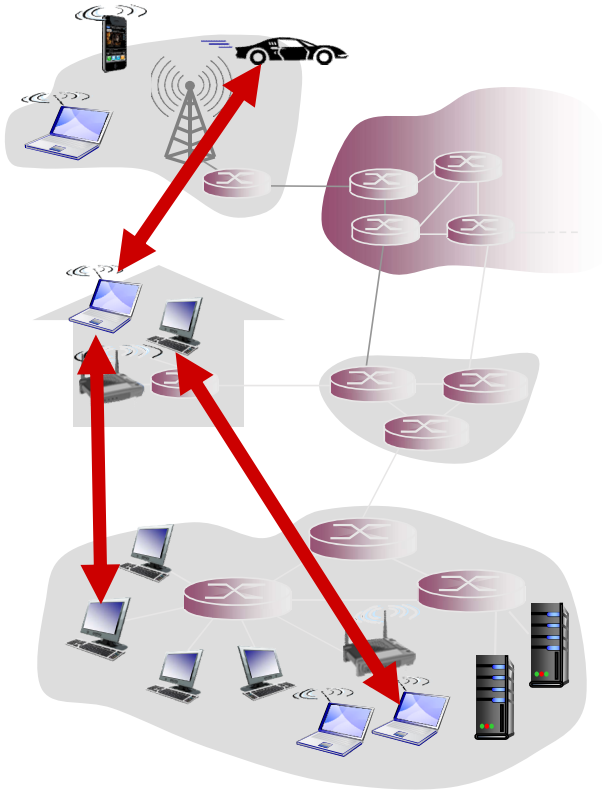
- Man-in-middle attack
  - Intercept queries/responses
- DNS poisoning
  - Send bogus replies to DNS server, which is then cached
- DNS flood attacks:
  - Flood with bogus DNS requests. Apr/2023 @1.3 million DNS queries per second.

PURE P2P ARCHITECTURE

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

examples:

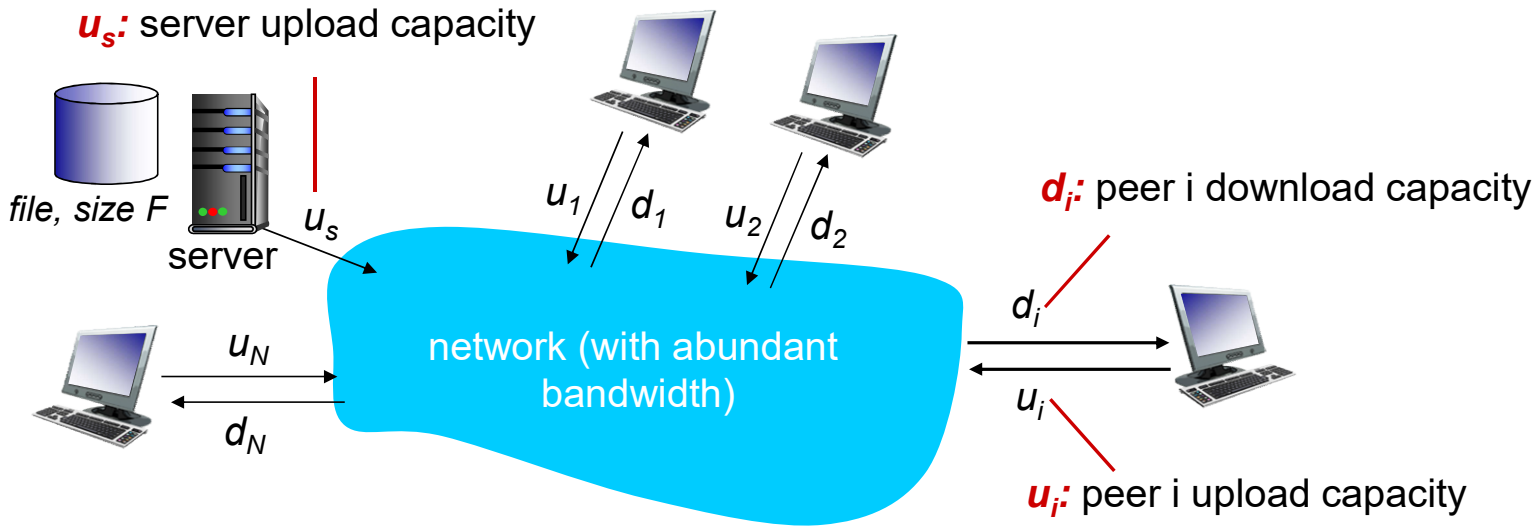
- file distribution (BitComet, BitTorrent, Deluge)
- Streaming (Dacast, Kaltura, Vimeo)
- VoIP (Skype)



FILE DISTRIBUTION: CLIENT-SERVER VS P2P

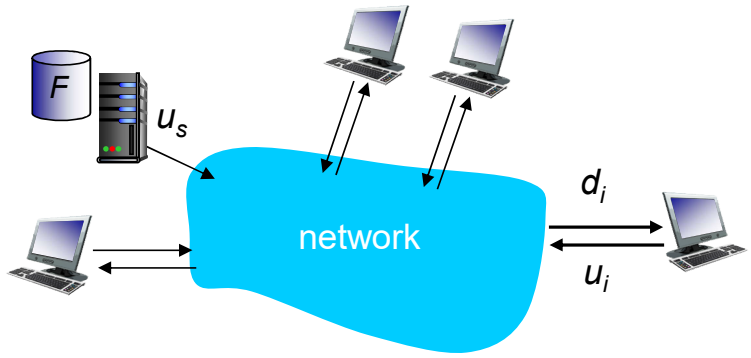
Question: how much time to distribute file (size *F*) from one server to *N* *peers*?

- *peer upload/download capacity is limited resource*



FILE DISTRIBUTION TIME: CLIENT-SERVER

- *server transmission:* must 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}



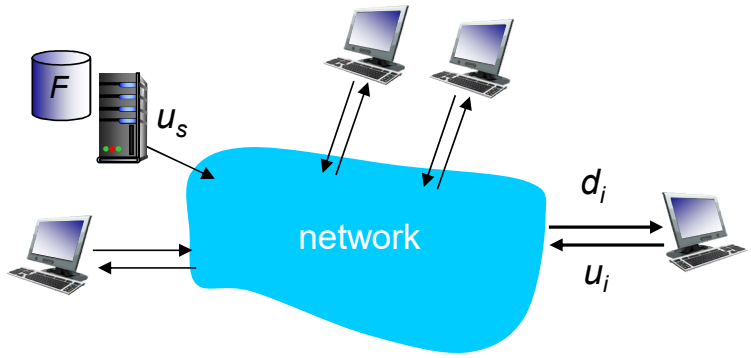
time to distribute F to N clients using client-server approach

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

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

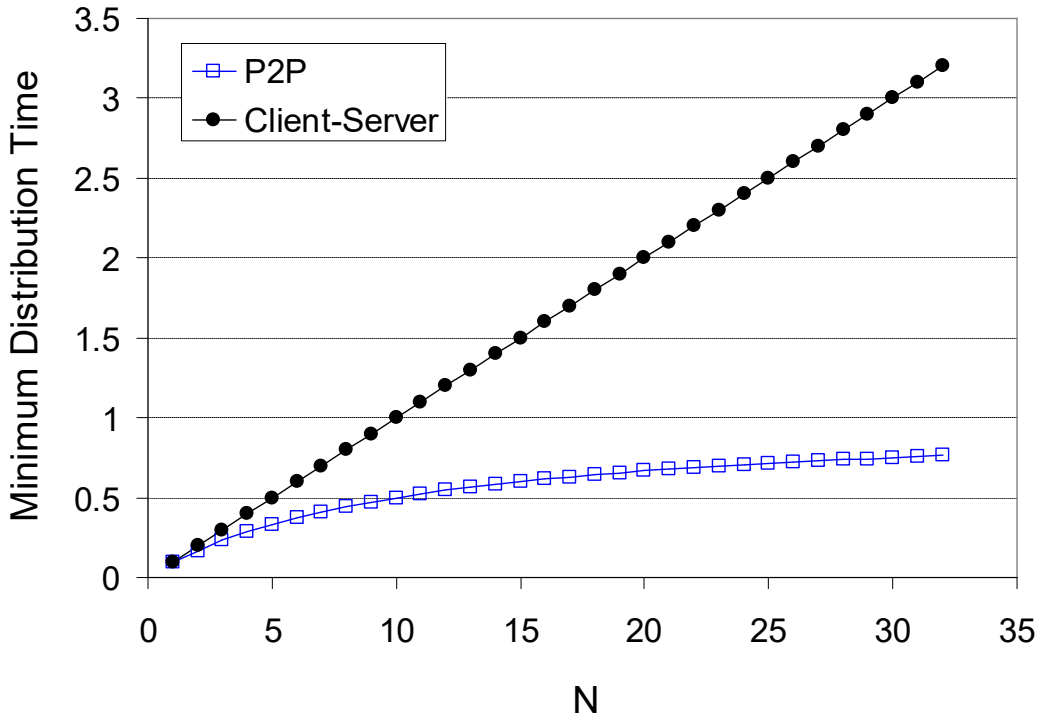
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

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

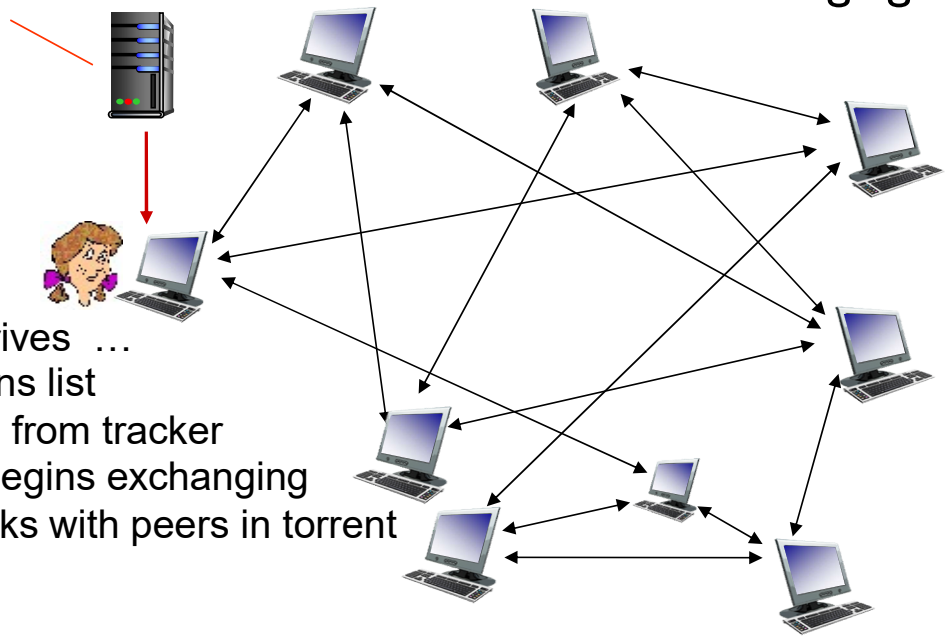


P2P FILE DISTRIBUTION: BITTORRENT

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

tracker: tracks peers participating in torrent

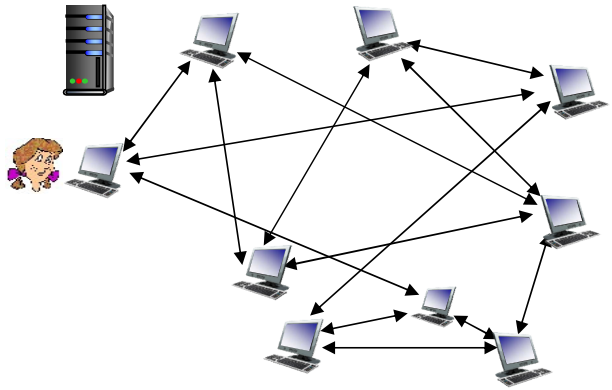
torrent: group of peers exchanging chunks of a file



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

P2P File Distribution: BitTorrent

- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")



- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

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

sending chunks: tit-for-tat

- Alice sends chunks to those (e.g., 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 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4