Chapter 2:Application Layer

Chapter 2: road map

Some networks apps

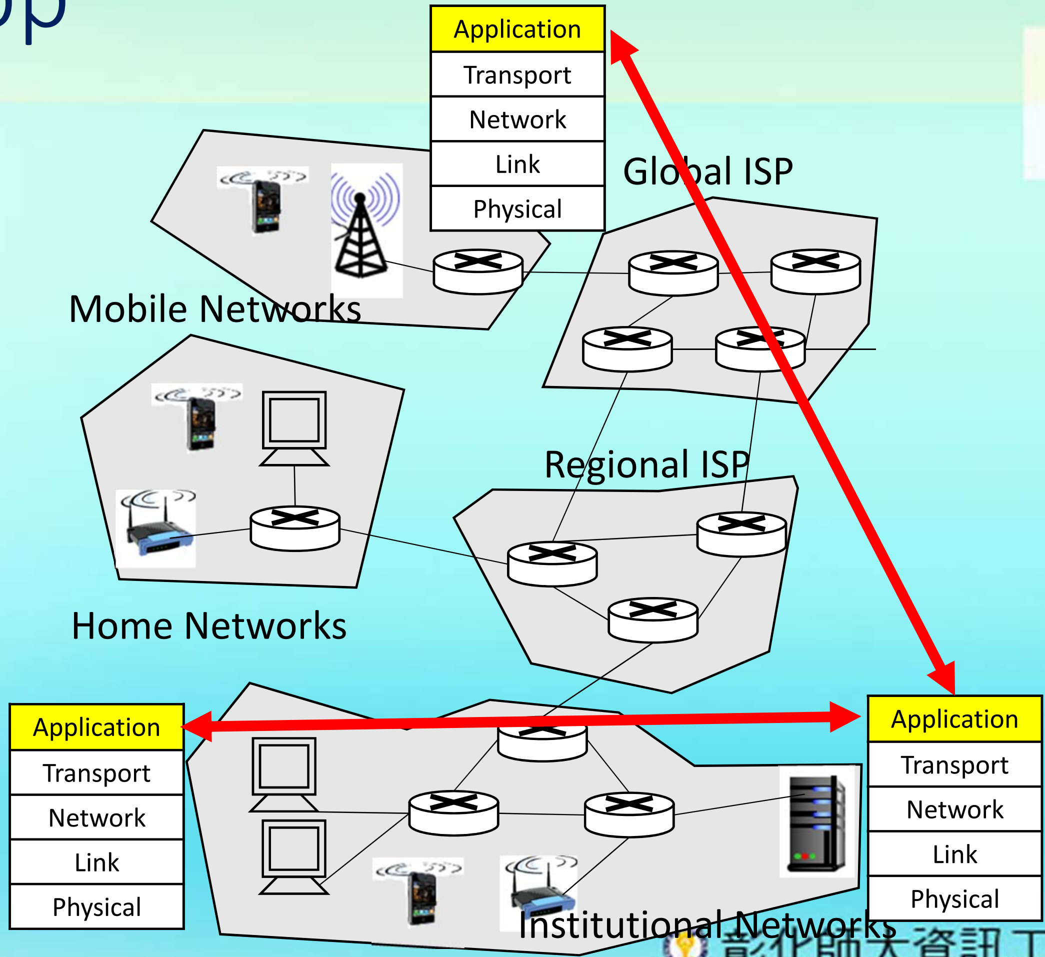
- E-mail
- Web
- Text messaging
- Remote login
- P2P file sharing
- Multi-user network game
- Streaming stored video (Youtube, Hulu, Netflix)
- Voice over IP(e.g. Skype)
- Real-time video conferencing
- Social Networking
- Search
- Etc…

Create a network app

- ***Write program that:***
  - Run on(different) *end systems*
  - Communicate over network
  - E.g. web server software communicates with browser software

No need to write software for network-core devices

- Network-core do not run user applications
- Applications on *end systems* allow for rapid app development, propagation



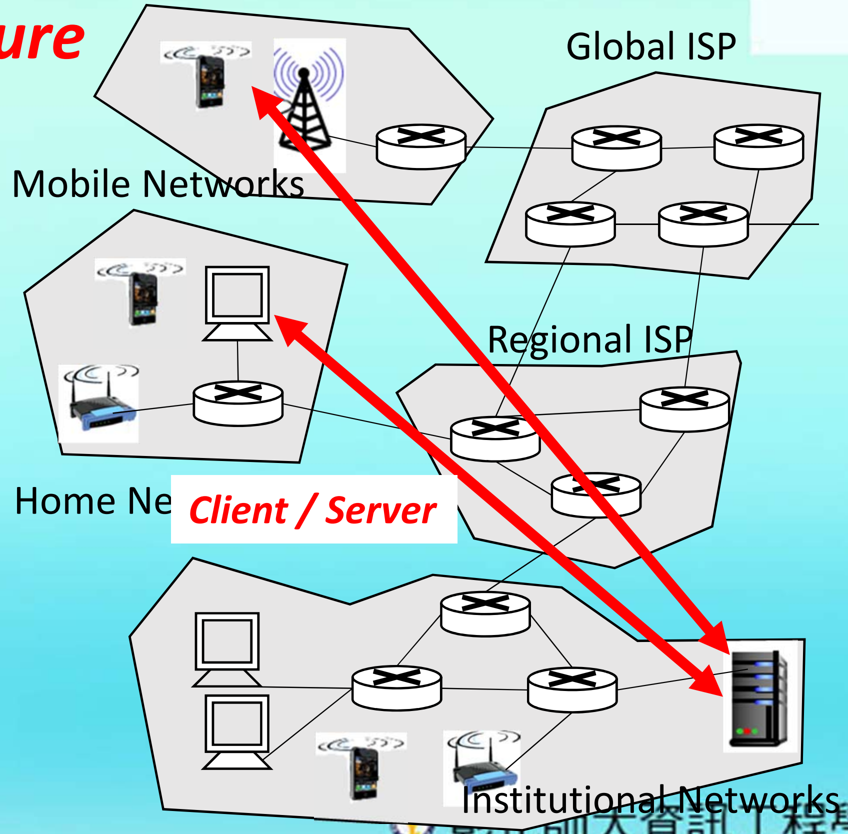
2.1 Principles of Network Applications

2.1.1 Network Application Architecture

Client-Server architecture

- **Server :**
  - Always-on host
  - Permanent IP address
  - *Data centers* for scaling

Clients:
  - Communicate with server
  - May be intermittently connected
  - May have dynamic IP
  - Do not communicate directly with each other



2.1.2 Processes communicating

Process : program running within a host

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

Client, Servers

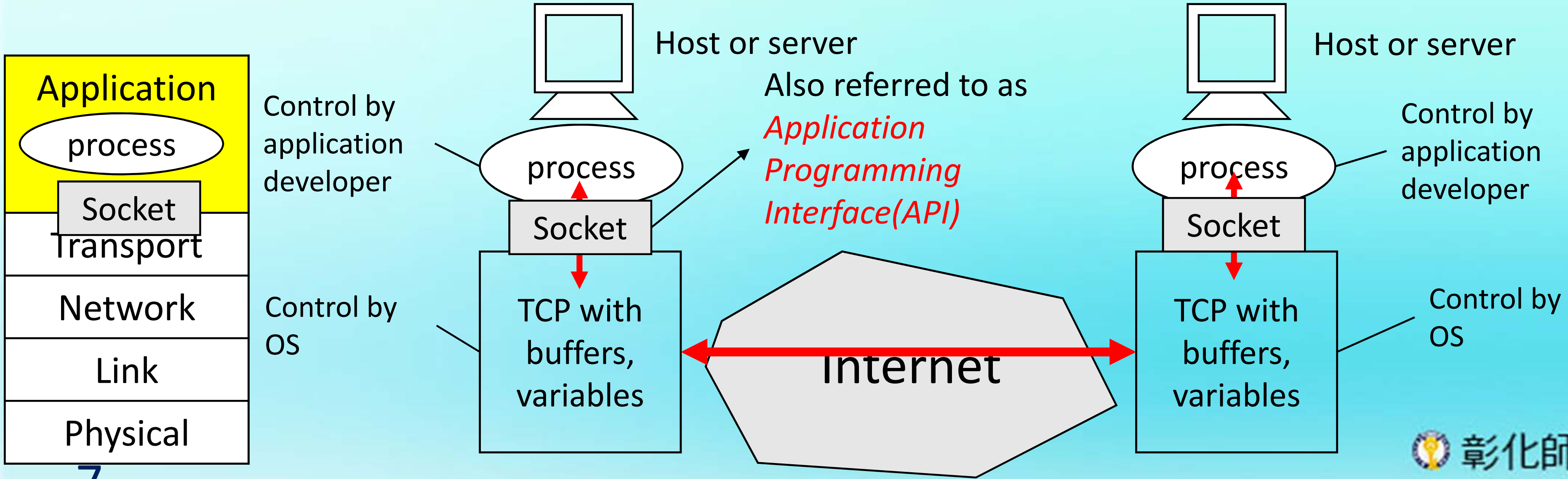
Client process: process that initiates communication

Server process: process that waits to be contacted

Aside: applications with P2P architectures have ***client processes & sever processes***

2.1.2 Processes communicating – Sockets

- **process sends/receives *messages* to/from its *sockets***
- **Sockets analogous to door**
  - Sending process shoves message out door
  - Sending process relies on transport infrastructure on other side of door to deliver message to *sockets* at receiving process



2.1.2 Processes communicating – Addressing Processes

- To receives messages, process must have *identifier*
- Host device has unique *32-bit IP address*
- **Q: Does IP address of host on which process runs suffice for identifying the process?**
  - ***A: no, many processes can be running on same host***

- *Identifier* includes *IP address* and *port numbers* associated with process on host.
- Example port numbers:
  - HTTP server : 80
  - Mail server : 25
  - FTP: 21
- To send HTTP message to gaia.cs.umass.edu web server:
  - IP address : 128.119.245.12
  - Port number: 80

2.1.2 Processes communicating – *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 information in fields
- Rules for when and how process send & responds to messages

Open protocols:

- *Defined in RFCs*
- *Allows for interoperability*
- *E.g. HTTP, SMTP*

Proprietary protocols:

- *E.g. Skype*

2.1.3 Transport Services Available to applications

What transport service does an app 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 minimum amount of throughput to be "effective"
- Other apps ("elastic app") make use of whatever throughput they get

Security

- Encryption, data integrity …

2.1.4 Transport Services Provided by the Internet

Transport service requirements: common apps

application	data loss	throughput	Time sensitive
File transfer	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web document	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	
Interactive game	Loss-tolerant	Few kbps up	Yes few secs
Text messaging	No loss	Elastic	Yes, 100's msec *yes and no*

2.1.4 Transport Services Provided by the Internet

Internet transport protocols services

TCP Services:

- *Reliable transport:* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *Congestion control:* throttle sender when network overloaded
- *Does not provide:* timing, minimum throughput guarantee, security
- *Connection-oriented:* setup required between client and server processes

UDP Services:

- *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 bother? Why is there a UDP?*

2.1.4 Transport Services Provided by the Internet

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
Remote terminal access	Telnet[RFC 854]	TCP
E-mail	SMTP[RFC 2821]	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	SIR, RTP, proprietary (e.g. Skype)	TCP or UDP

2.1.5 Application-Layer Protocols

An application-layer protocol defines:

- *The types of messages exchanged*
  for example, request messages and response messages
- *The syntax of the various message types*
  such as the fields in the message and how the fields are delineated
- *The semantics of the fields*
  that is, the meaning of the information in the fields
- Rules for determining when and *how a process sends* messages and *responds to messages*

2.1.5 Application-Layer Protocols

Securing TCP

TCP & UDP

- No encryption
- Cleartext passwords sent into sockets traverse Internet in cleartext

SSL

- Provides encrypted TCP connection
- Data integrity
- End-point authentication

SSL is at app layer

- Apps use SSL libraries, which “talk” to TCP

SSL socket API

- Cleartext passwords sent into socket traverse Internet encrypted
- Ch. 7

2.2 The Web and HTTP

2.2.1 Overview of HTTP

First a review…

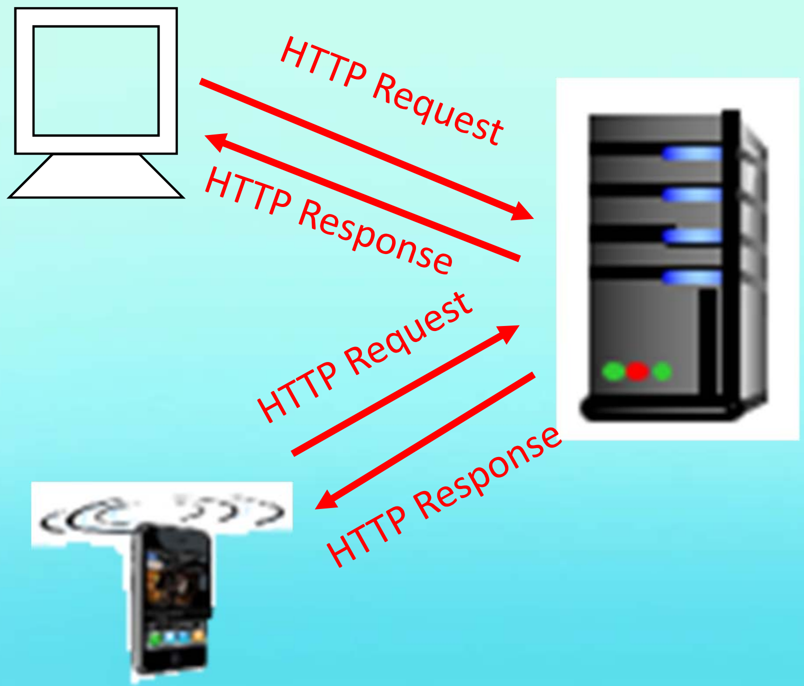
- *Web page consists of objects*
- *object can be HTML file, JPEG image, Java applet, audio file, …*
- *Web page consists of base HTML-file which includes several reference objects*
- *each object is addressed by a URL, e.g.,*

$$\underbrace{\textit{http://www.ncue.edu.tw}}_{\text{Host name}}\underbrace{\textit{/bin/home.php}}_{\text{Path name}}$$

2.2.1 Overview of HTTP

HTTP: Hypertext Transfer protocol

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



2.2.1 Overview of HTTP

Use TCP:

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

HTTP is "stateless"

- Server maintains no information

Protocols that maintain "state" are complex!

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

2.2.2 Non-Persistent and Persistent Connections

Non-persistent HTTP

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

Persistent HTTP

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

2.2.2 Non-Persistent and Persistent Connections

Suppose user enters URL:

www.ncue.edu.tw/bin/home.php

(Contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server(process) at *www.ncue.edu.tw/bin/home.php* On port 80

1b. HTTP server at host www.ncue.edu.tw/bin/home.php waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP request message(containing URL) into TCP connection sockets. Message indicates that client wants object *www.ncue.edu.tw/bin/home.php*

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

time

2.2.2 Non-Persistent and Persistent Connections

4. HTTP server closes TCP connection.

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

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

time

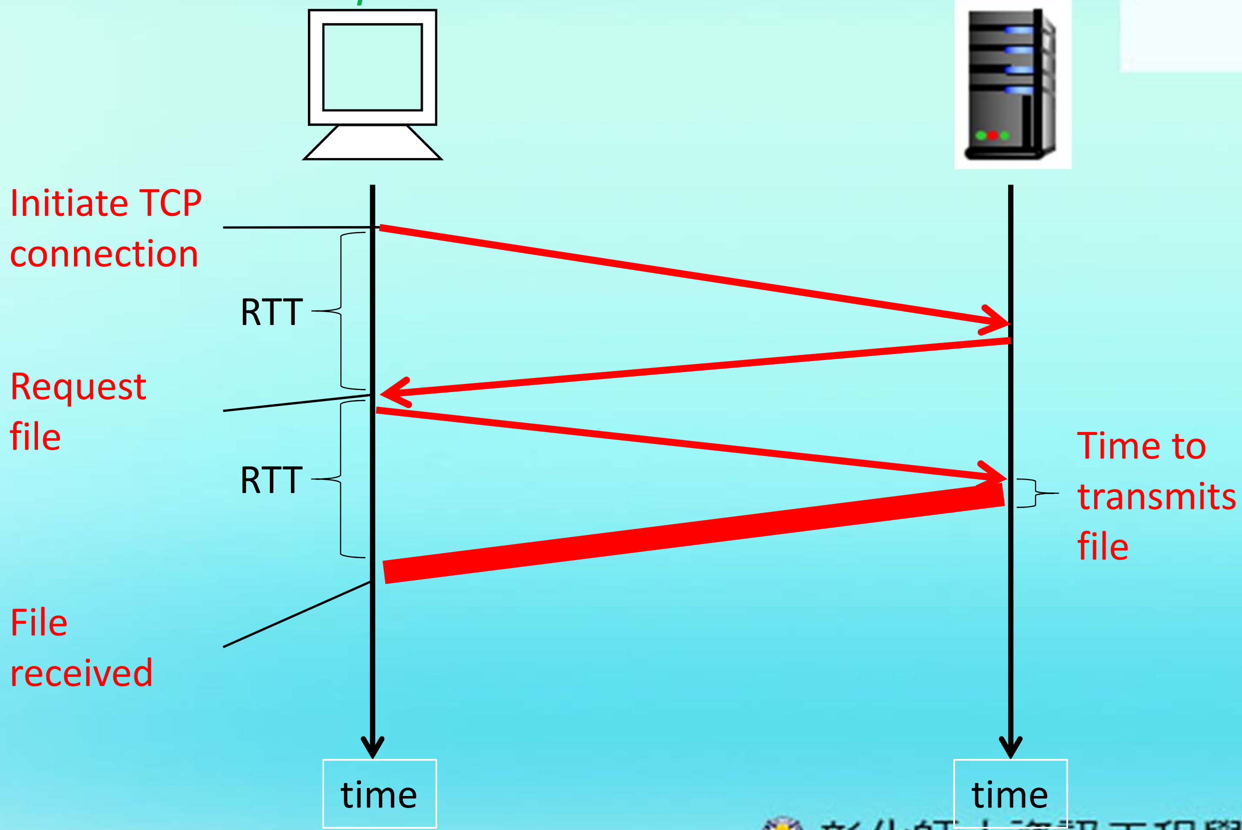
2.2.2 Non-Persistent and Persistent Connections

Non-persistent HTTP: response time

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



2.2.2 Non-Persistent and Persistent Connections

Persistent HTTP

Non-persistent HTTP issues:

- Requires 2 RTTs per object
- OS overheard 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 over open connection
- Client sends requests as soon as it encounters a referenced object
- As little as one RTT for all the referenced objects

2.2.3 HTTP Message Format
HTTP request message

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

Carriage return character

Line-feed character

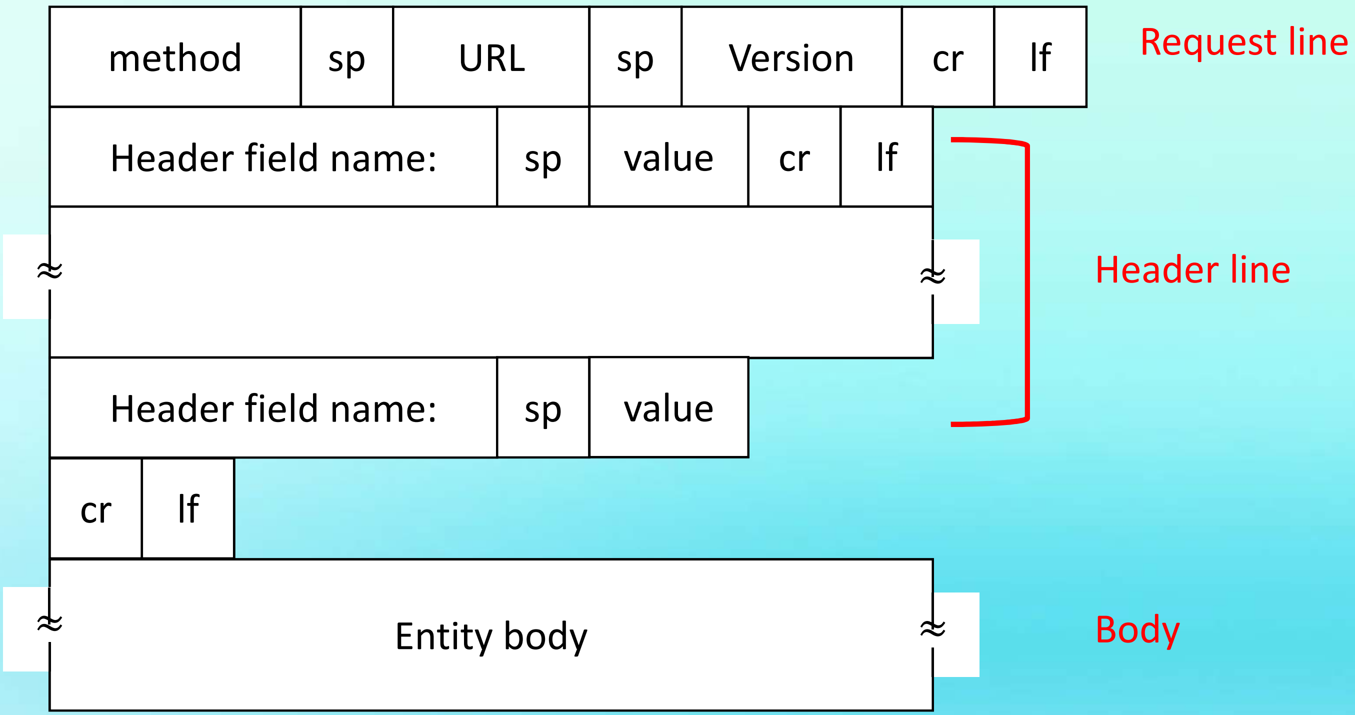
Request line (GET, POST, HEAD commands)

```
GET /somedir/page.html HTTP/1.1 \r\n
Host: www.someschool.edu\r\n
Connection: close\r\n
User-agent: Mozilla/5.0\r\n
Accept-Language: fr\r\n
\r\n
```

Header lines

Carriage return, line feed at start of line indicates end of header lines

2.2.3 HTTP Message Format-
HTTP request message: general format



2.2.3 HTTP Message Format-
HTTP request message: uploading from 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

2.2.3 HTTP Message Format-
HTTP request message: method type

HTTP/1.0:

- GET
- POST
- HEAD
  - Ask server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - Uploads file in entity body to path specified in URL field
- DELETE
  - Delete file specified in the URL field

2.2.3 HTTP Message Format- HTTP response message

Status line (protocol status code phrase)

HTTP/1.1 200 OK \r\n

Header line

Connection: close \r\n
Date: Tue, 18 Aug 2015 15:44:04 GMT \r\n
Server: Apache/2.2.3 (CentOS) \r\n
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT \r\n
Content-Length: 6821 \r\n
Content-Type: text/html

Data, e.g., requested HTML file

(data data data data ...)

2.2.3 HTTP Message Format- HTTP response status codes

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

200 OK
- Request succeeded, requested object later in this msg

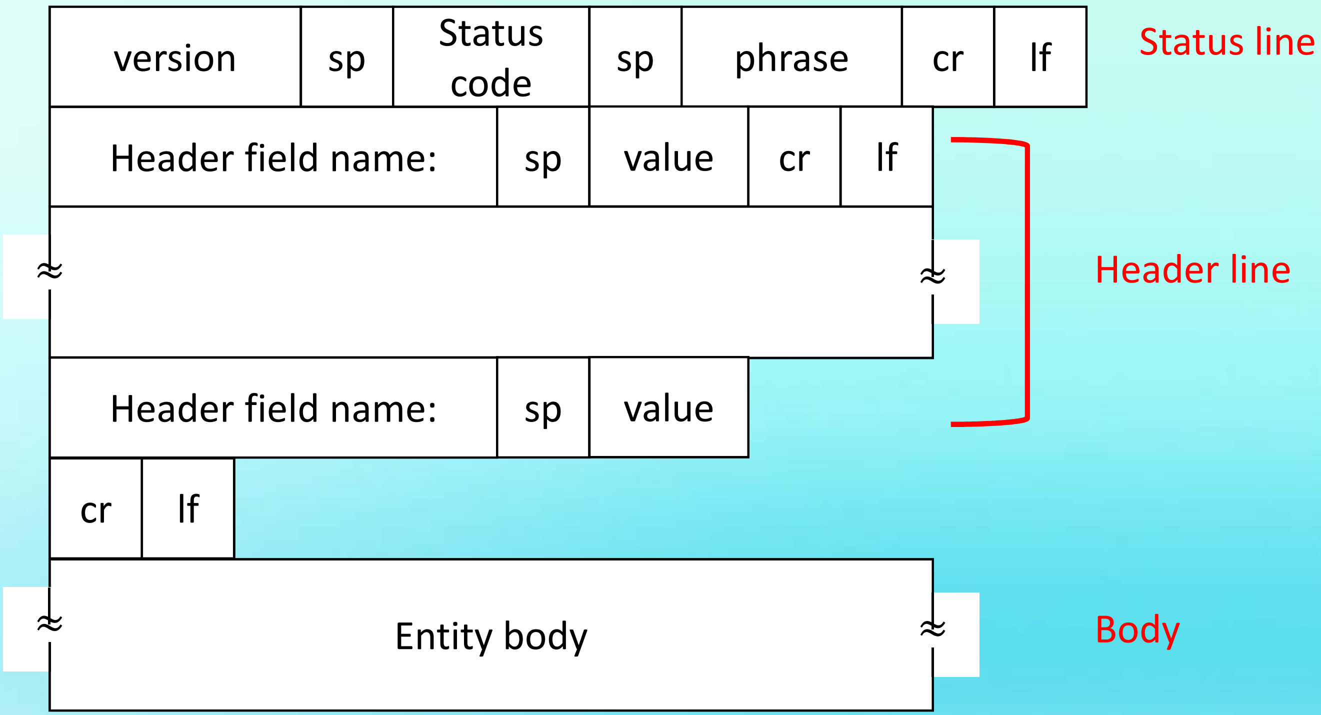
301 Moved Permanently
- Requested object moved, new location specified later in this msg (Location:)

400 Bad Request
- Request msg not understood by server

404 Not Found
- Requested document not found on this server

505 HTTP Version Not Supported

2.2.3 HTTP Message Format-
HTTP response message: general format



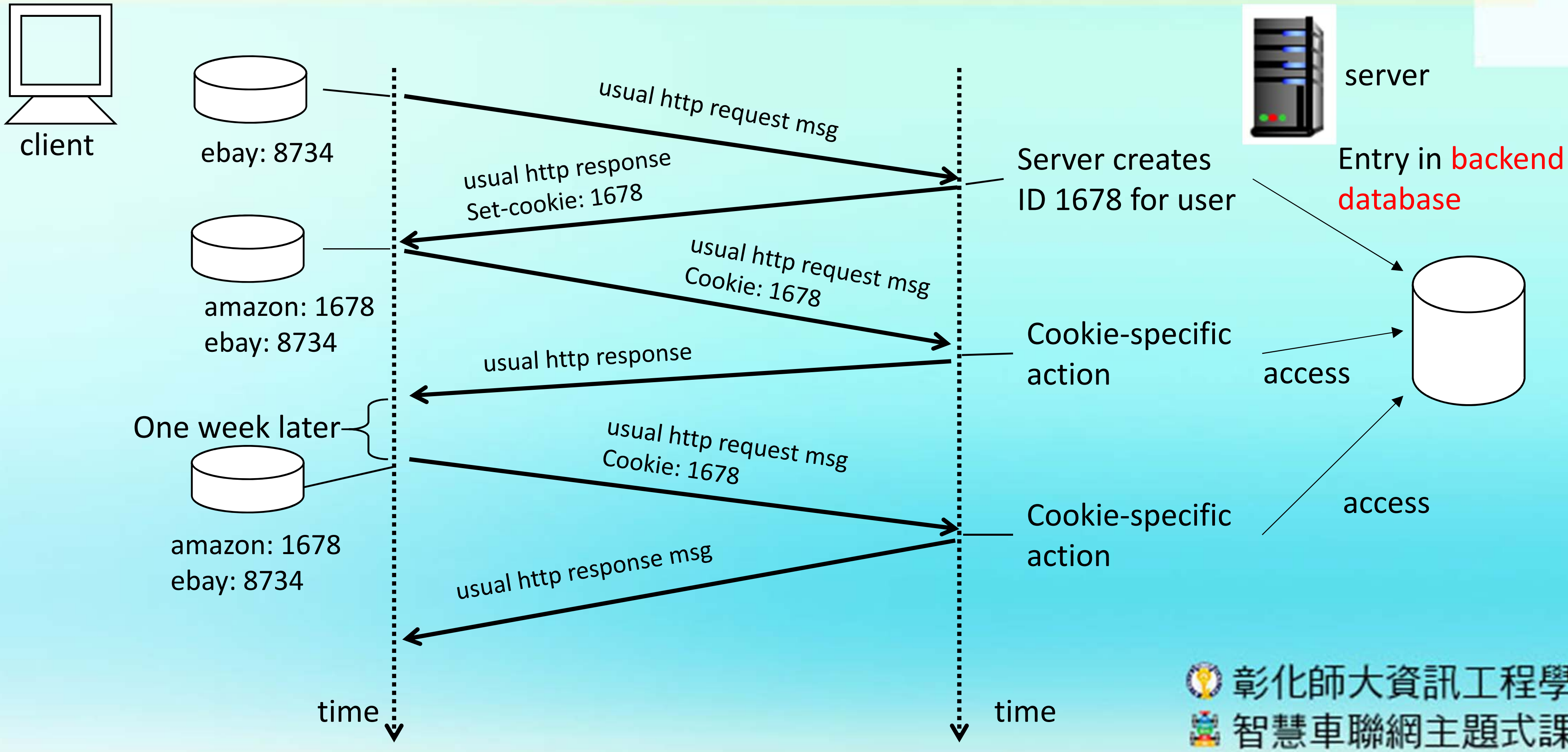
2.2.4 User-Server Interaction: Cookies

- Many Web sites use cookies *four components:*
  1. Cookie header line of HTTP *response* message
  2. Cookie header line in next HTTP *request* message
  3. Cookie file kept on user's host, managed by user's browser
  4. Back-end database at Web site

Example :

- Susan always access Internet from PC
- Visit specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates:
  - Unique ID
  - Entry in backend database for ID

2.2.4 User-Server Interaction: Cookies



2.2.4 User-Server Interaction: 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 : http messages carry state

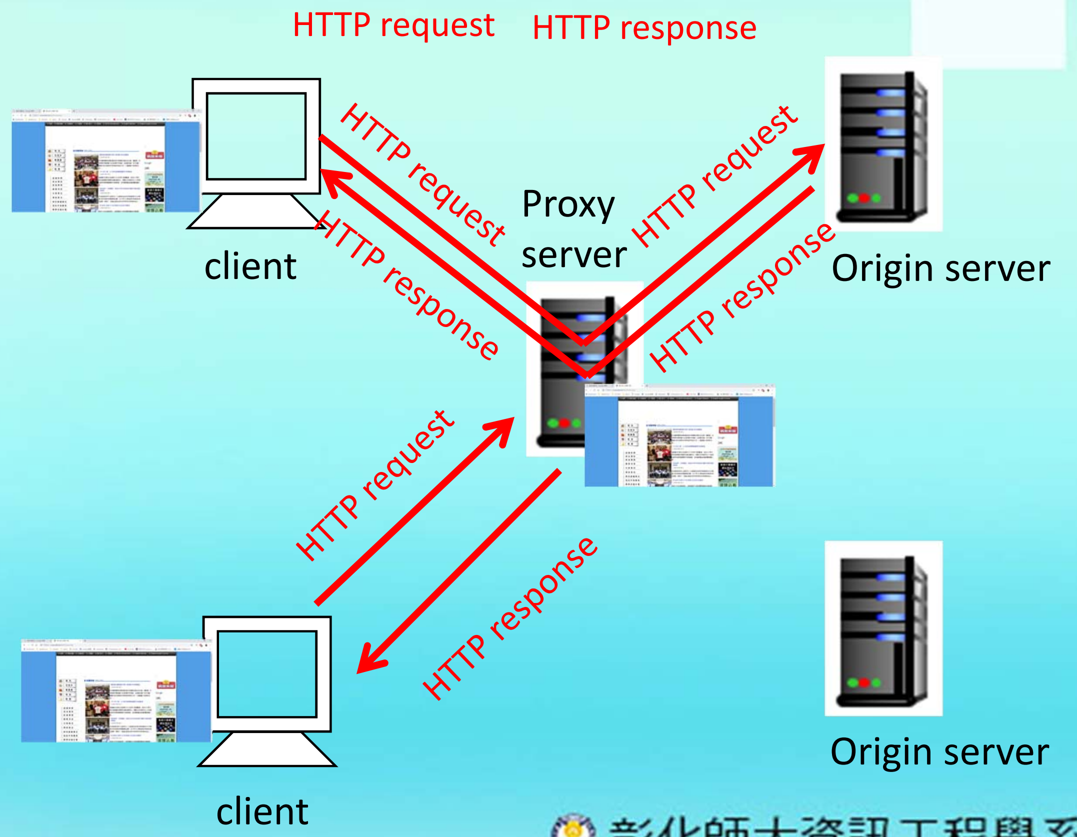
Cookie and privacy:

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

2.2.5 Web Caching (proxy server)

Goal : *satisfy client request without involving origin server*

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



2.2.5 Web Caching (proxy server)

- Cache acts as both client and server
  - Server for 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 an institution’s access link
- Internet dense with caches: enables “poor” content providers to effectively deliver content (so too does P2P files)

2.2.5 Web Caching (proxy server)- Caching example

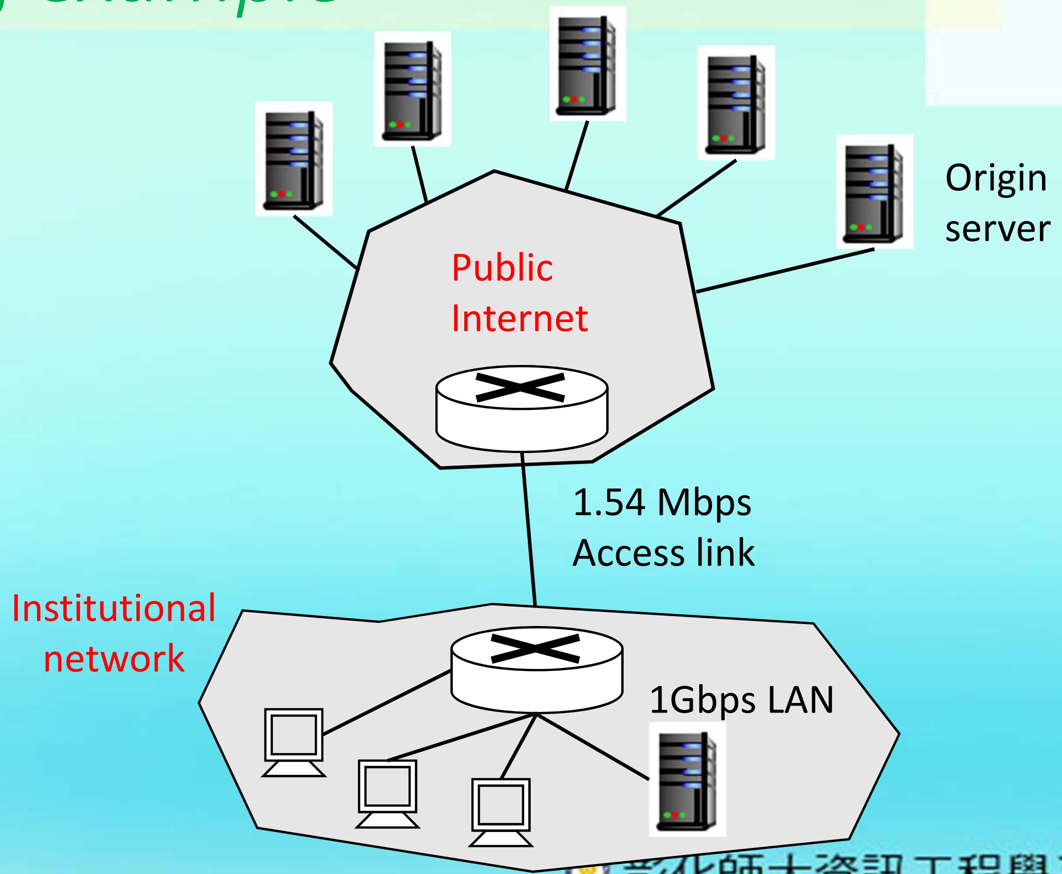
assumptions:

- 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: 15% *Problem!*
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + minutes + usecs



2.2.5 Web Caching (proxy server)-
fatter access link

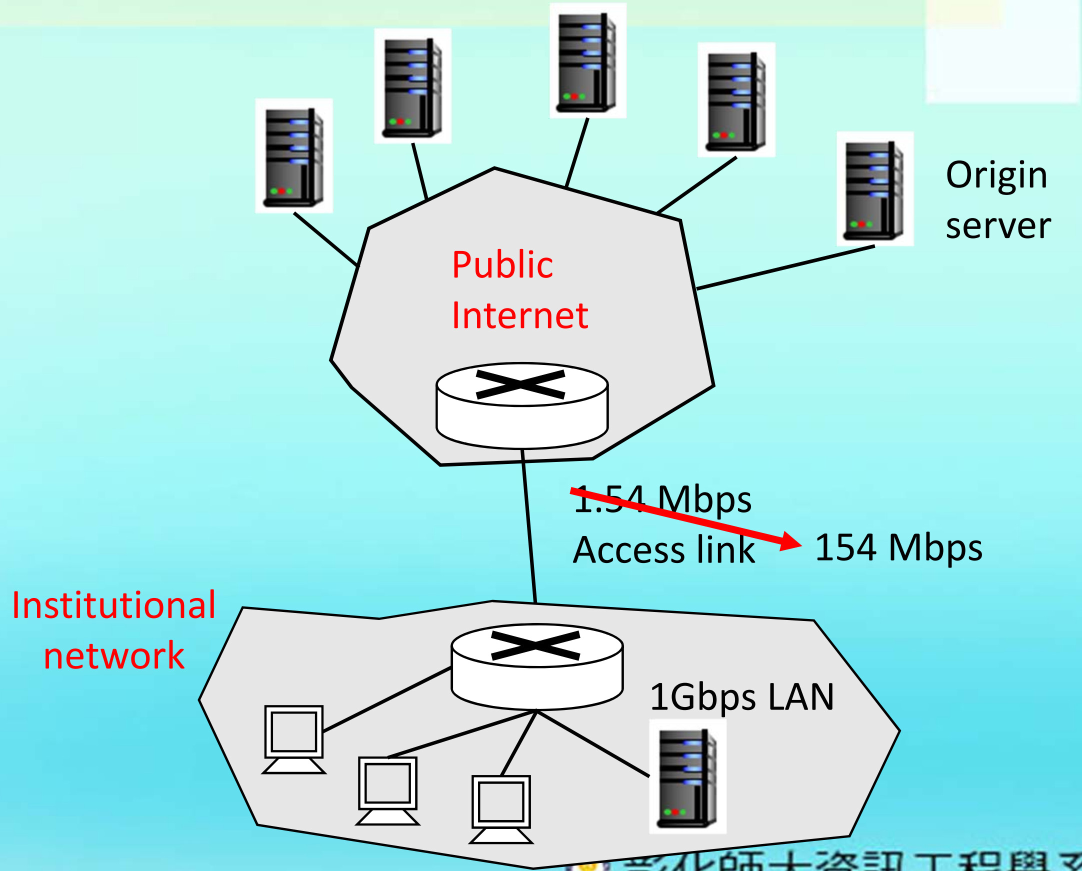
assumptions:

- 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~~ 154 Mbps

consequences:

- LAN utilization: 15%   *Problem!*
- access link utilization = ~~99%~~ 9.9%
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + ~~minutes~~ msec + usecs



Cost: increased access link speed (not cheap)

2.2.5 Web Caching (proxy server)- install local cache

assumptions:

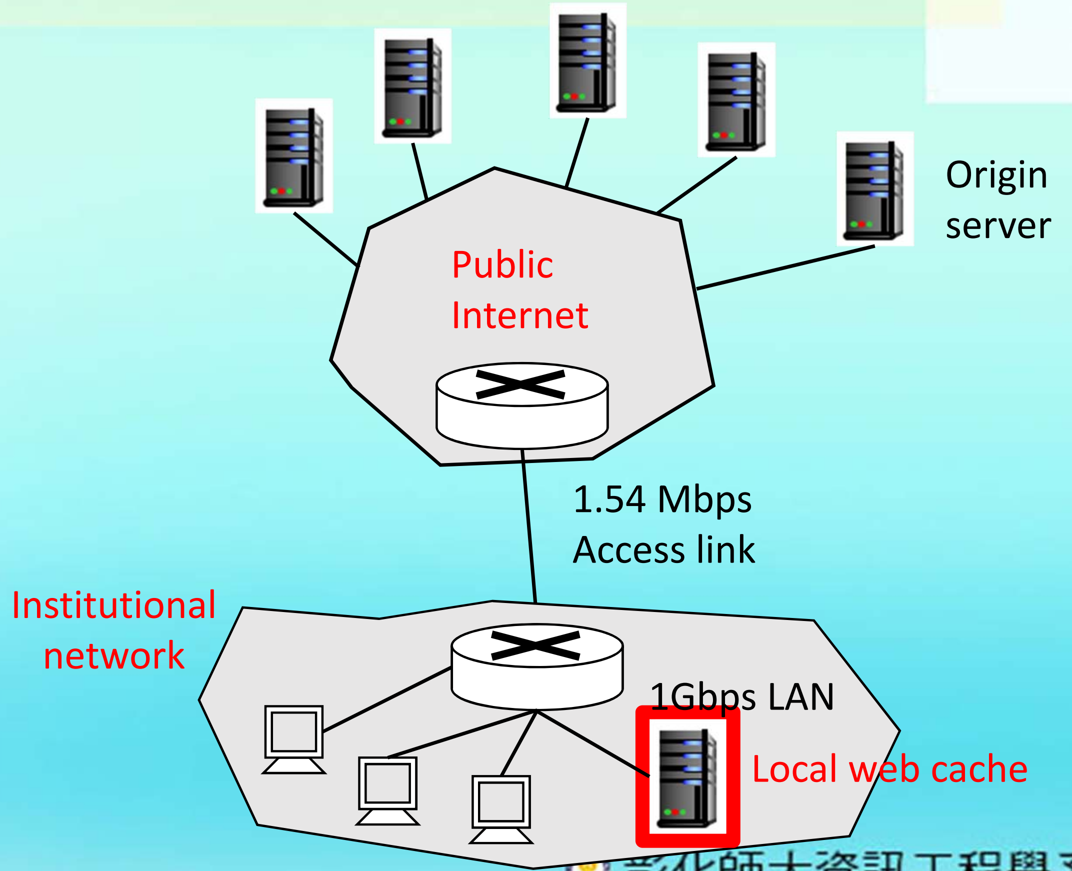
- Avg object size: 100K bits
- Avg request rate from browsers to origin servers:15/sec
- Avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ?
- total delay = ?

How to compute link utilization, delay?

Cost: web cache (cheap!)



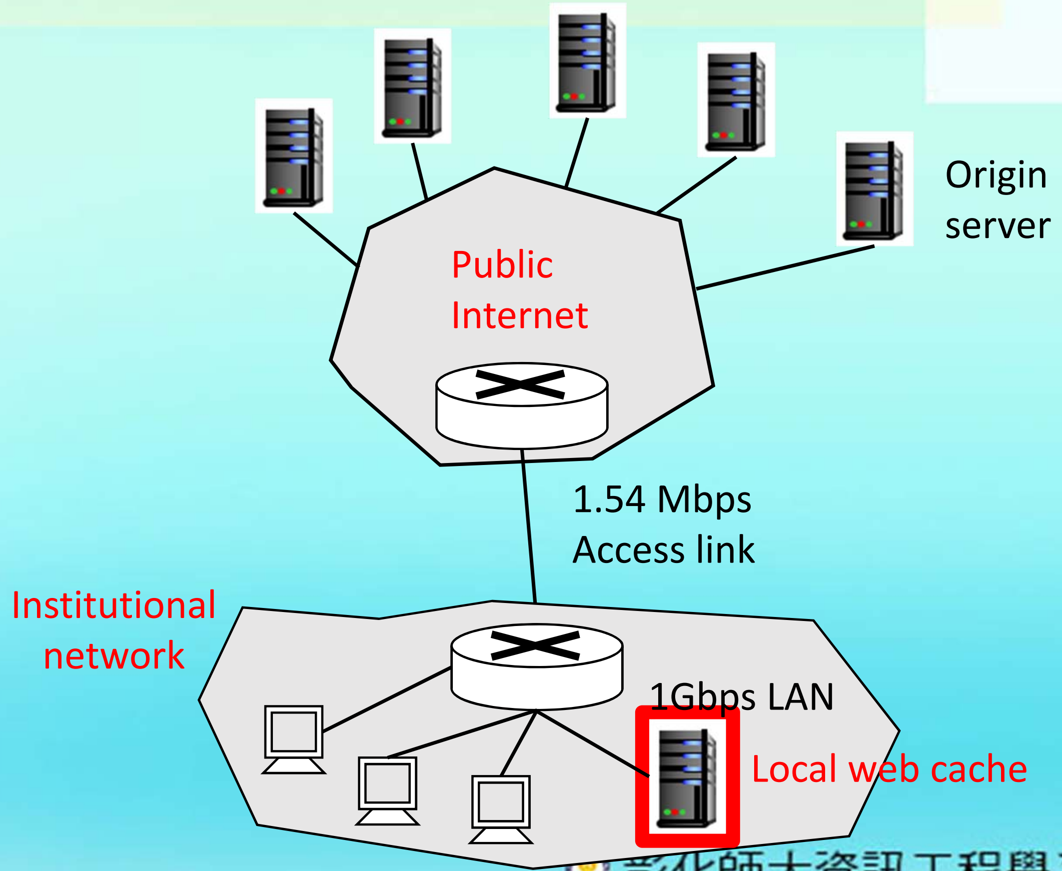
2.2.5 Web Caching (proxy server)- *install local cache*

Calculating access link utilization, delay with cache:

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

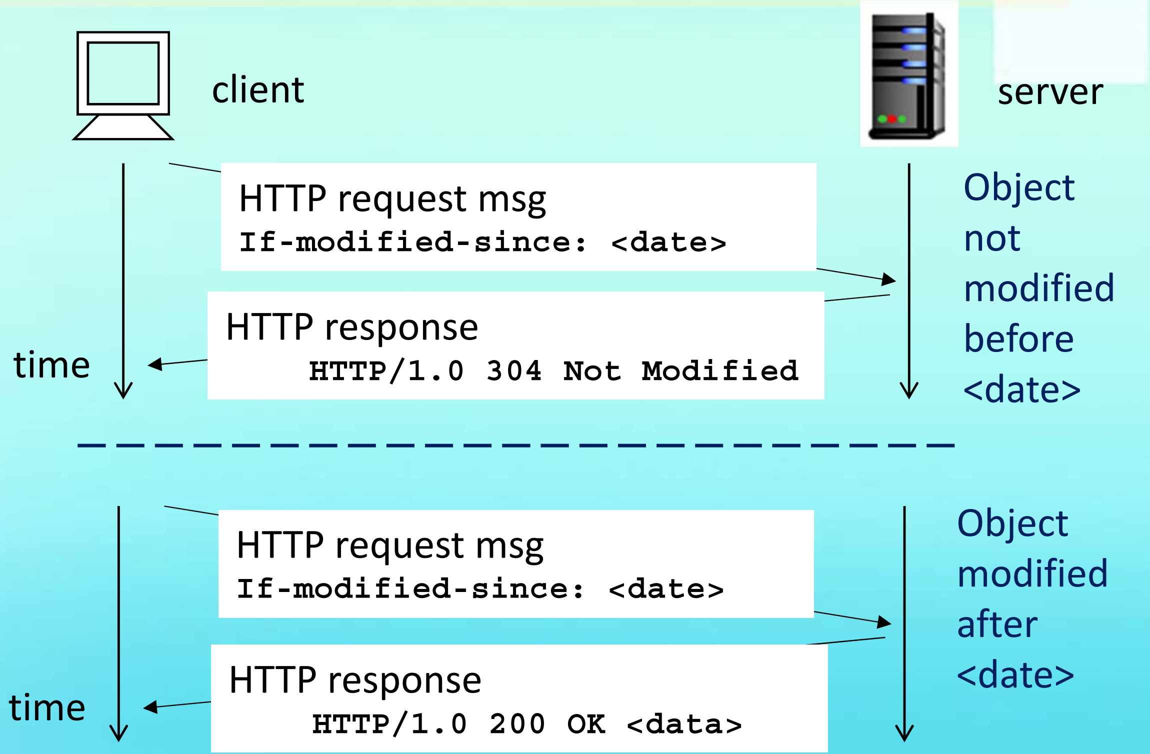
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link = 0.6*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .58

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

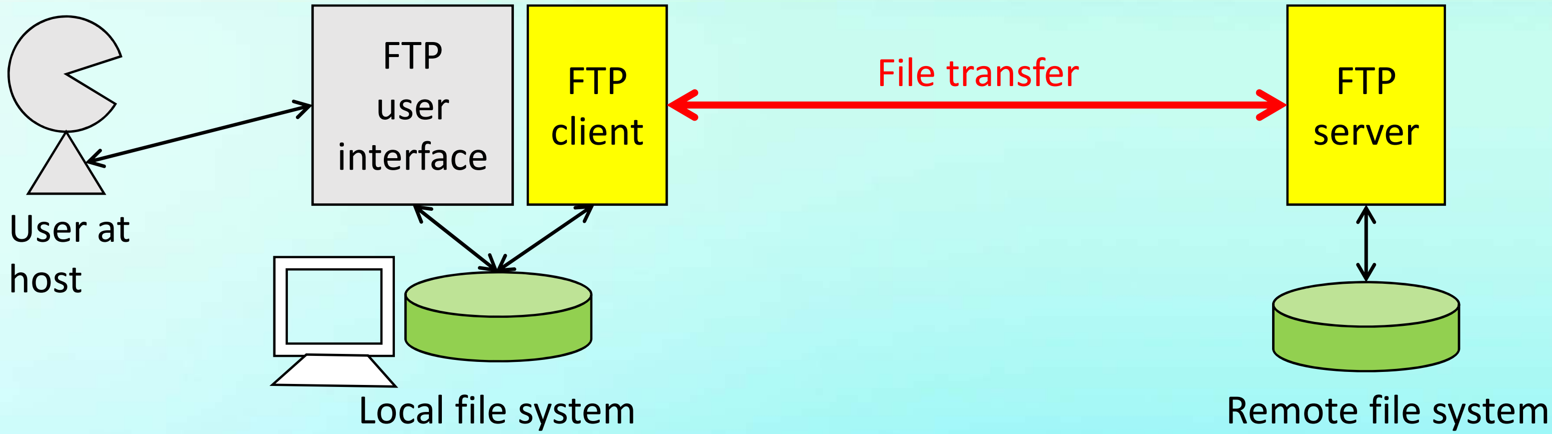


2.2.5 Web Caching (proxy server)- Conditional GET

- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *cache:* specify date of cached copy in HTTP request
  `If-modified-since: <date>`
- *server:* response contains no object if cached copy is up-to-date:
  `HTTP/1.0 304 Not Modified`



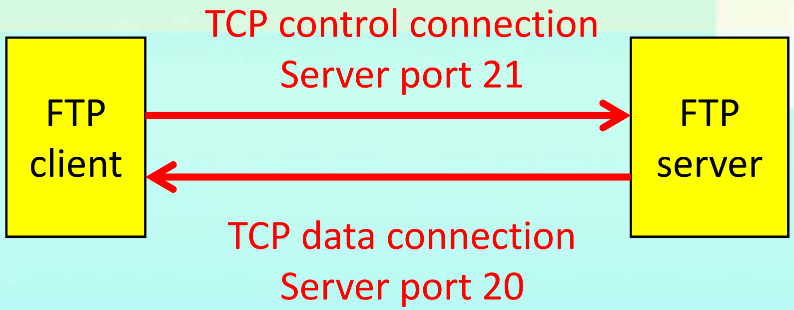
FTP: the file transfer protocol



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

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, using TCP
- client 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 one 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:

- 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**

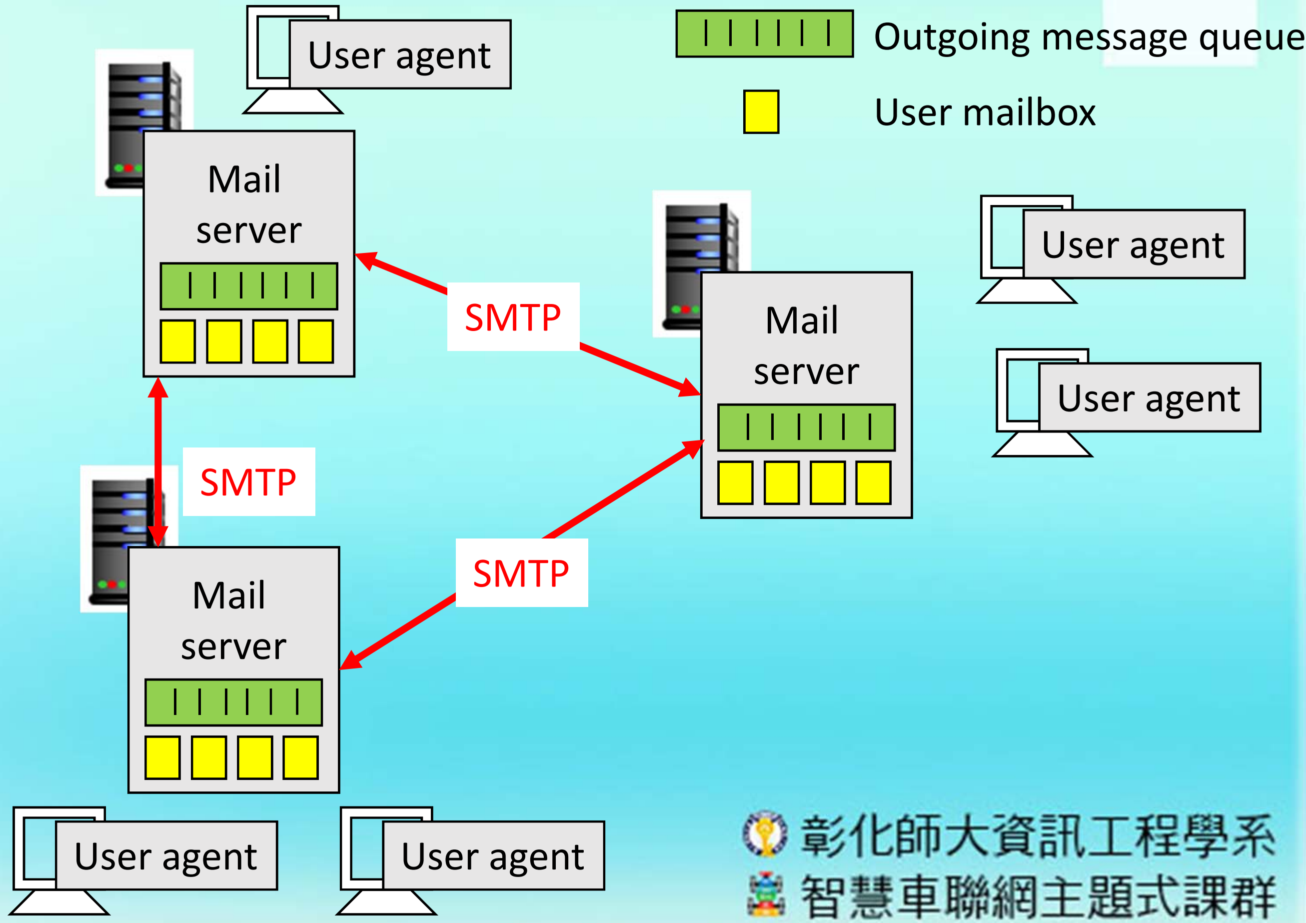
2.3 Electronic Mail in the Internet

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

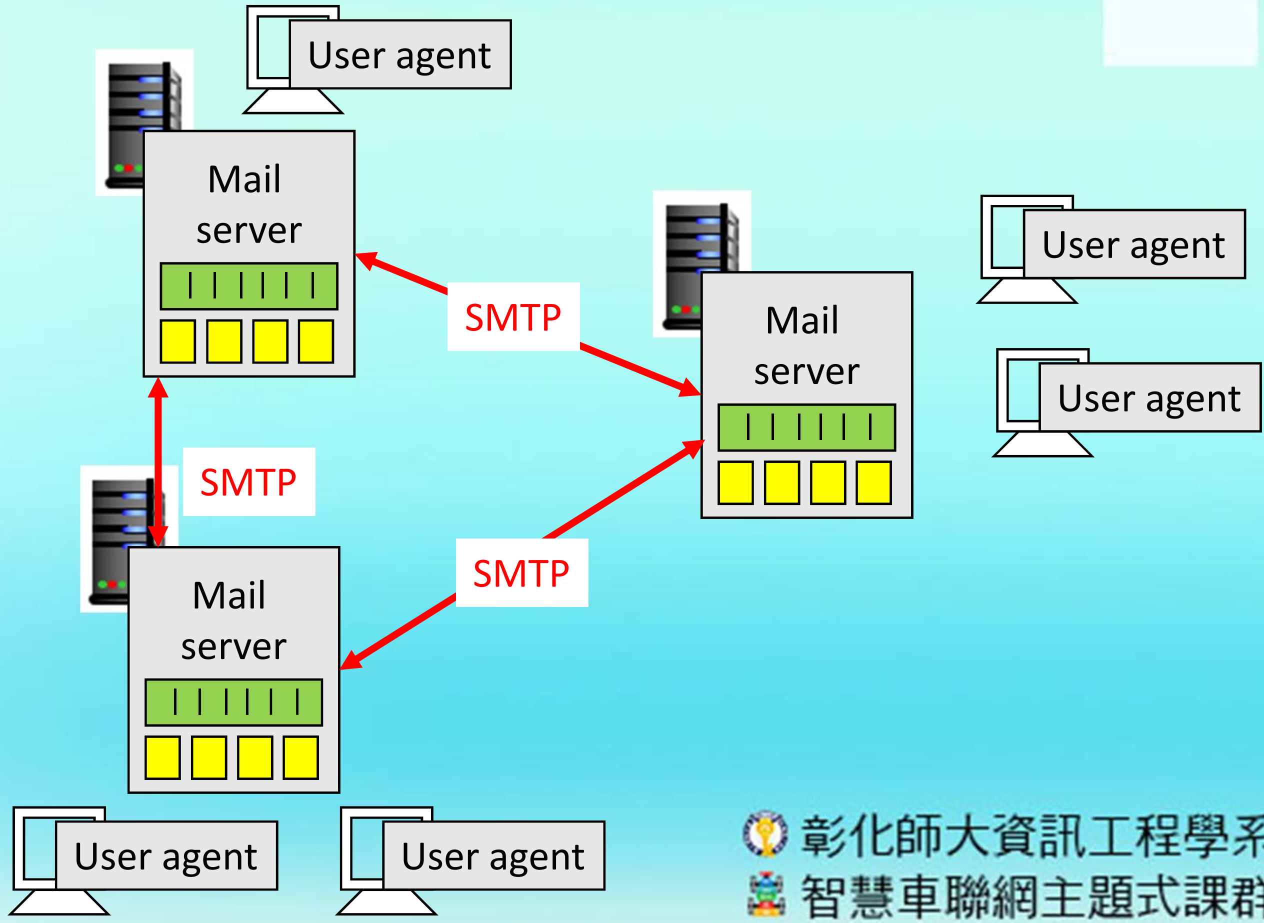
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



2.3 Electronic Mail in the Internet

mail servers:

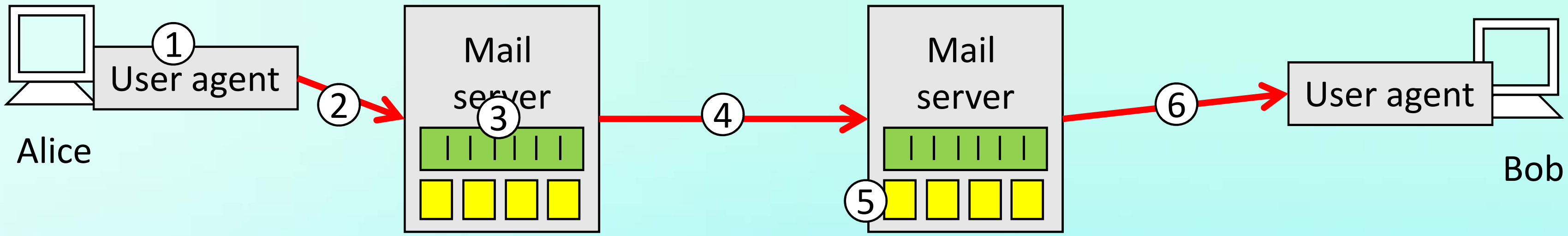
- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server



2.3 Electronic Mail in the Internet
2.3.1 SMTP (RFC 2821)

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP, FTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCI

2.3.1 SMTP (RFC 2821)-
Scenario: Alice sends message to Bob



1) Alice uses UA to compose message "to" `bob@someschool.edu`
2) Alice's UA sends message to her mail server; message placed in message queue
3) 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

2.3.1 SMTP (RFC 2821)-
Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

2.3.2 Comparison with HTTP

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses `CRLF.CRLF` to determine end of message

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

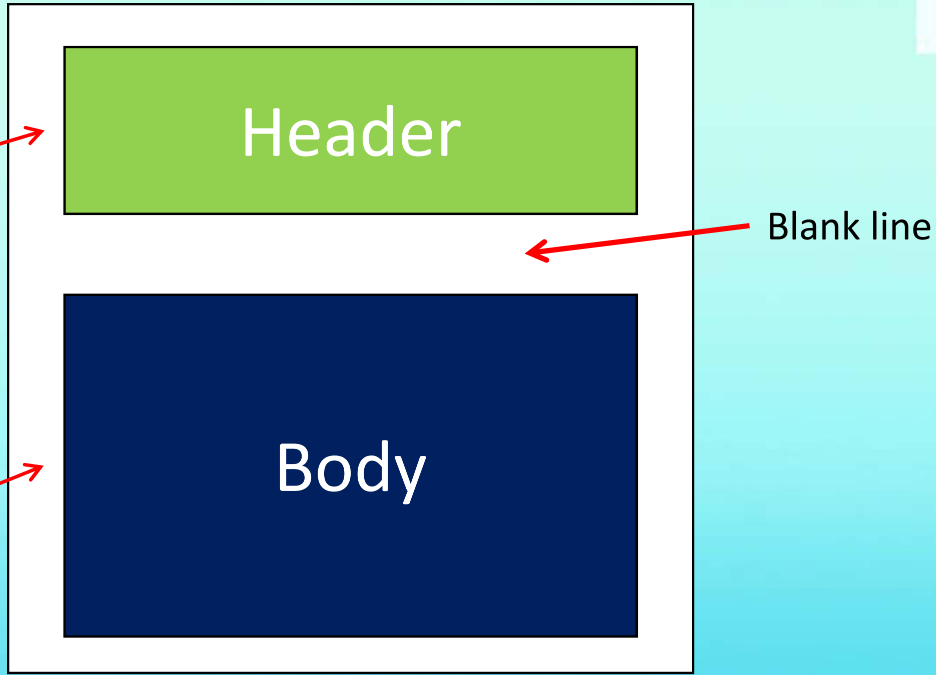
2.3.3 Mail Message Formats

SMTP: protocol for exchanging email msgs

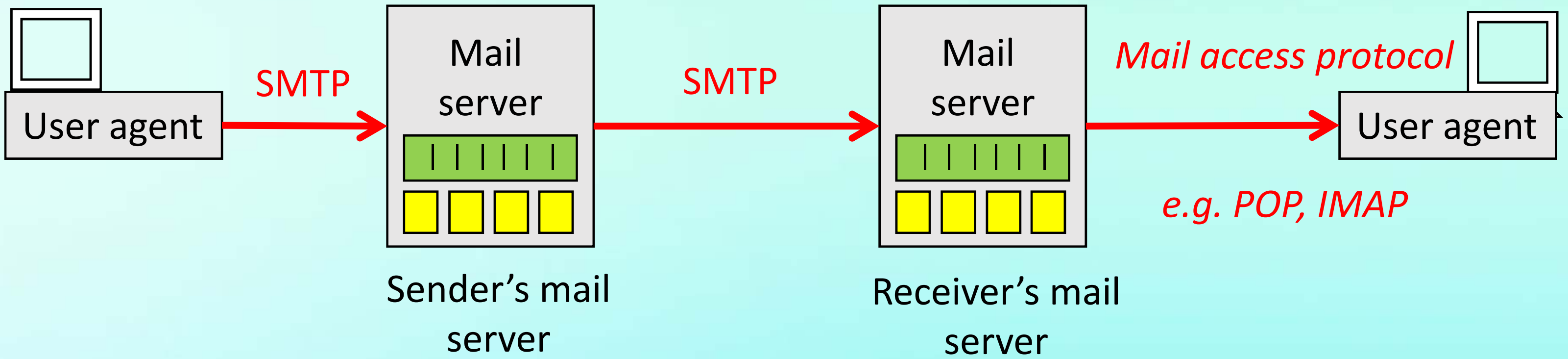
RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only



2.3.4 Mail Access Protocols



- ***SMTP***: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - **POP**: Post Office Protocol [RFC 1939]: authorization, download
  - **IMAP**: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
  - **HTTP**: gmail, Hotmail, Yahoo! Mail, etc.

2.3.4 Mail Access Protocols- POP3 protocol

authorization phase

- client commands:
  - **user:** declare username
  - **pass:** password
- server responses
  - **+OK**
  - **-ERR**

transaction phase, client:

- **list:** list message numbers
- **retr:** retrieve message by number
- **dele:** delete
- **quit**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

2.3.4 Mail Access Protocols-
POP3 protocol(more) and IMAP

more about POP3

- previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

2.1 DNS – The Internet's Directory Service

DNS: Domain Name System

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., 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"

2.1 DNS – The Internet's Directory Service

2.4.1 Services Provided by DNS

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

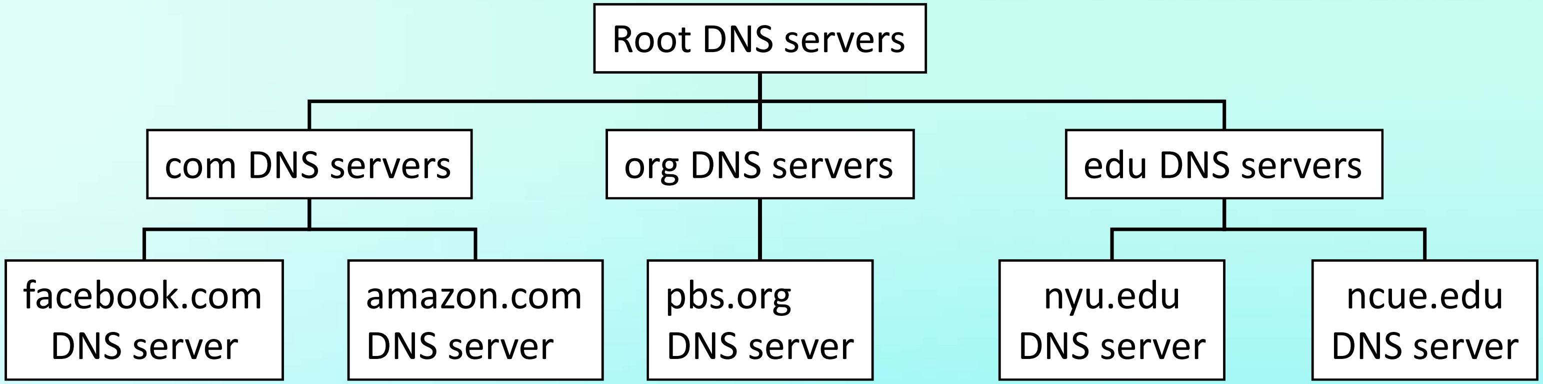
why not centralize DNS?

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

A: doesn't scale!

2.4.2 Overview of How DNS works

DNS: a distributed, hierarchical database



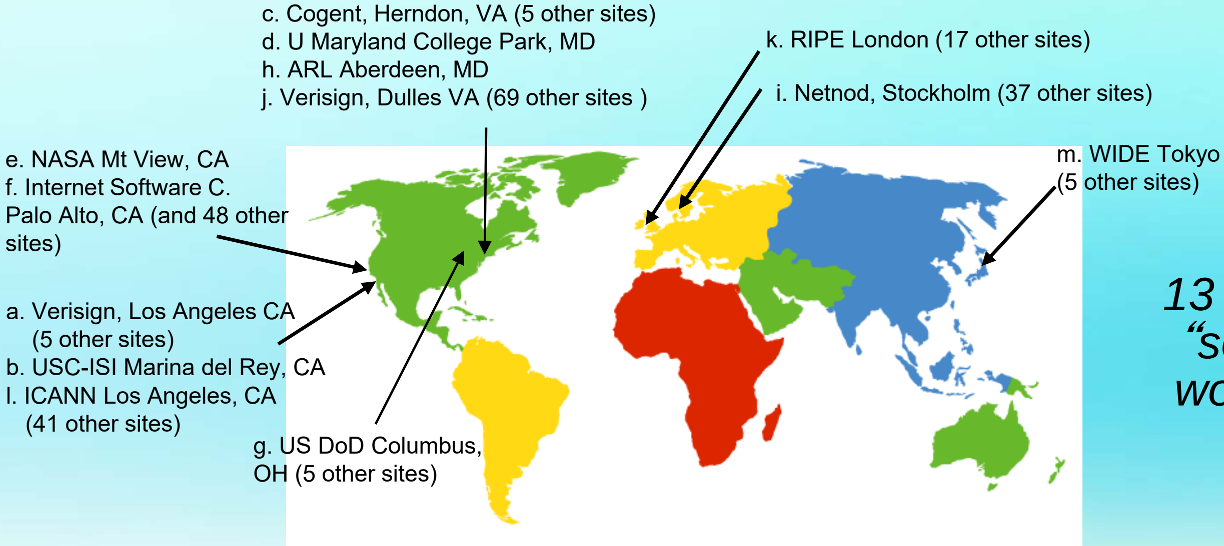
client wants IP for www.amazon.com; 1st approx:

- 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

2.4.2 Overview of How DNS works

DNS : root name servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server



13 root name "servers" worldwide

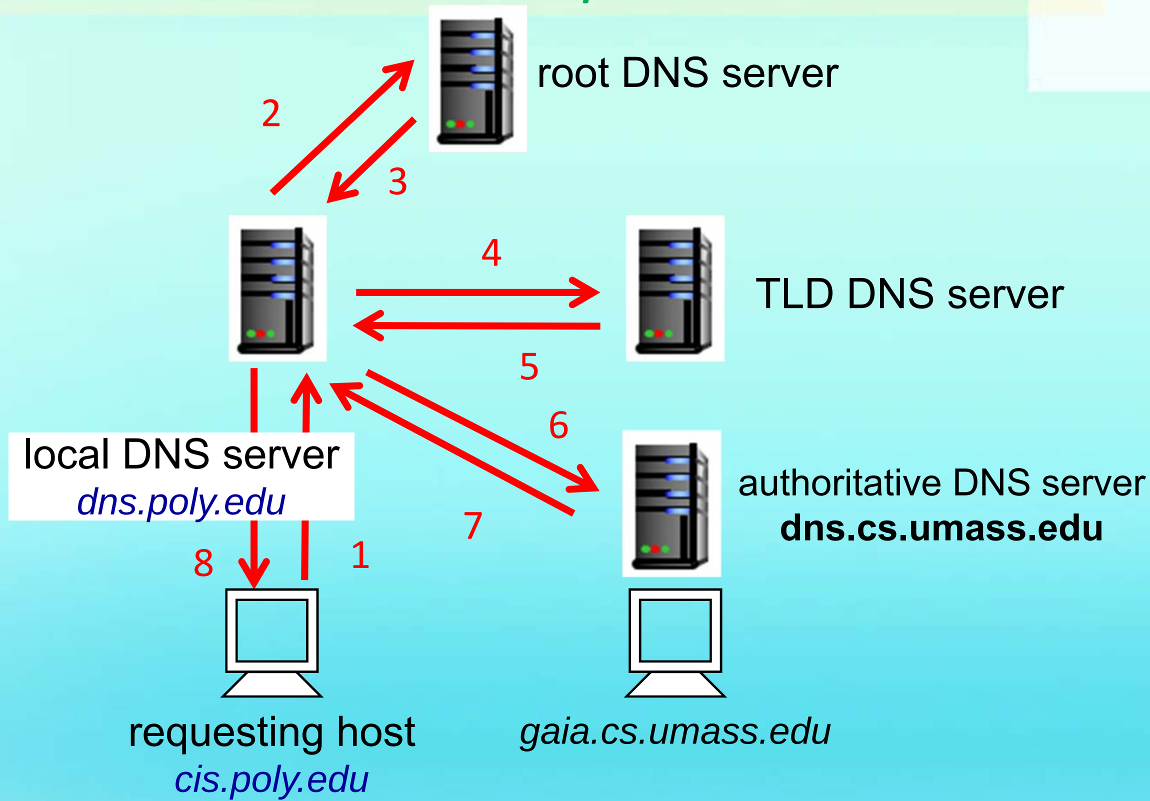
2.4.2 Overview of How DNS works

DNS name resolution example

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

iterated query:

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

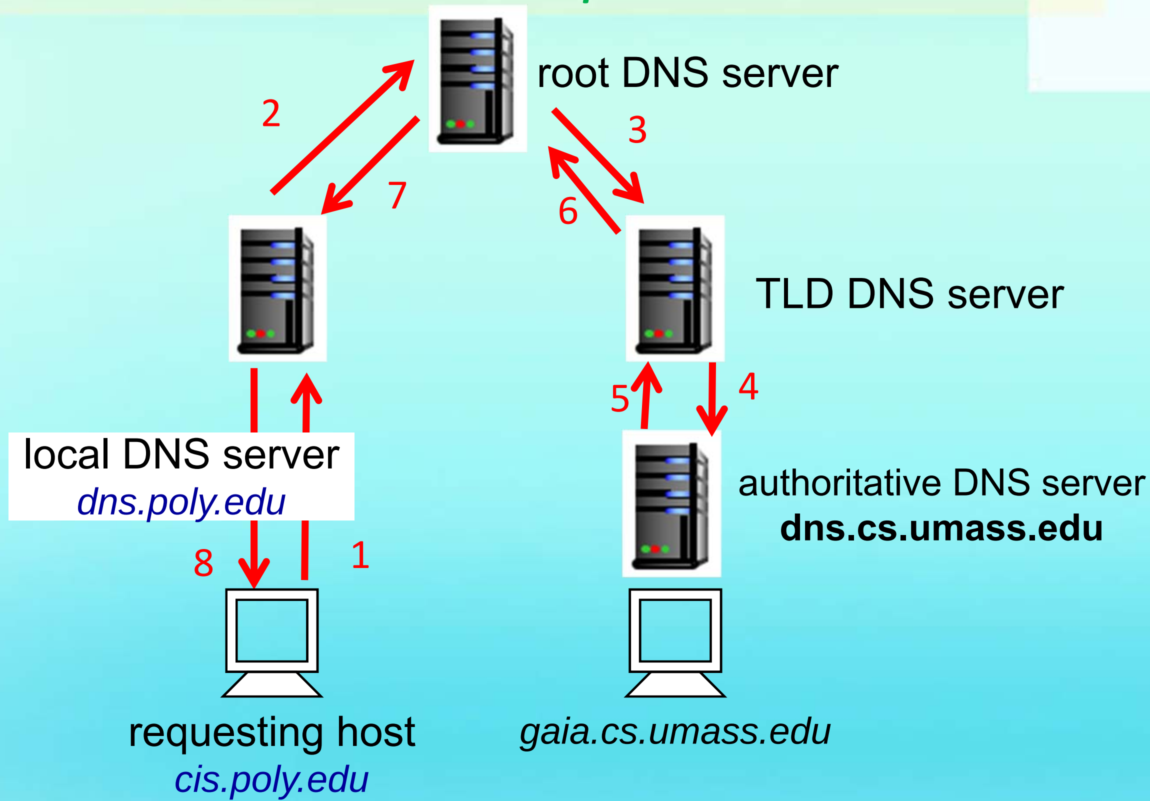


2.4.2 Overview of How DNS works

DNS name resolution example

recursive query:

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



2.4.2 Overview of How DNS works
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 name host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard
  - RFC 2136

2.4.4 DNS Records and Messages

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

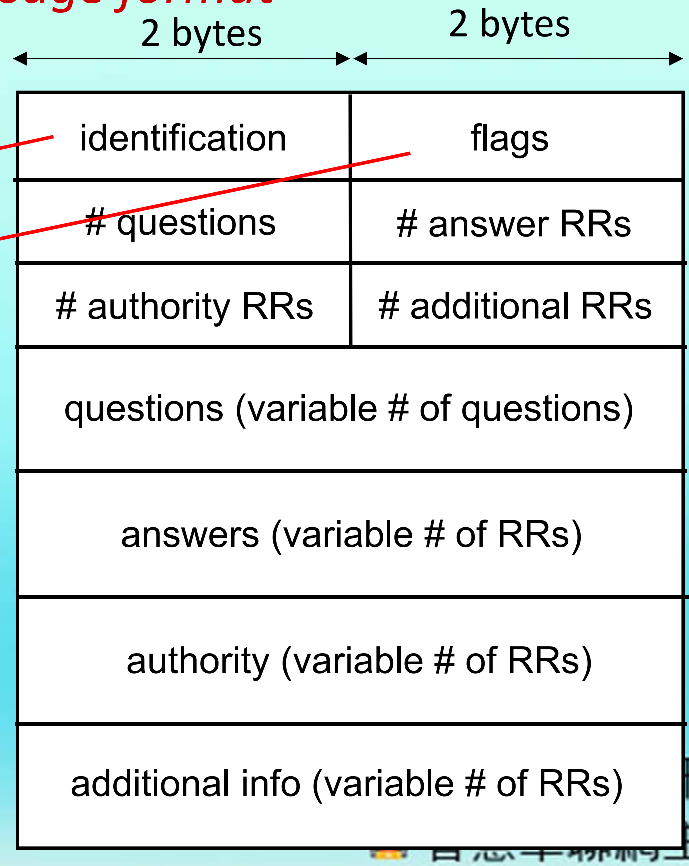
- **name** is alias name for some "canonical" (the real) name
- **www.ibm.com** is really **servereast.backup2.ibm.com**
- **value** is canonical name

type=MX

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

2.4.4 DNS Records and Messages
DNS protocol, messages

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

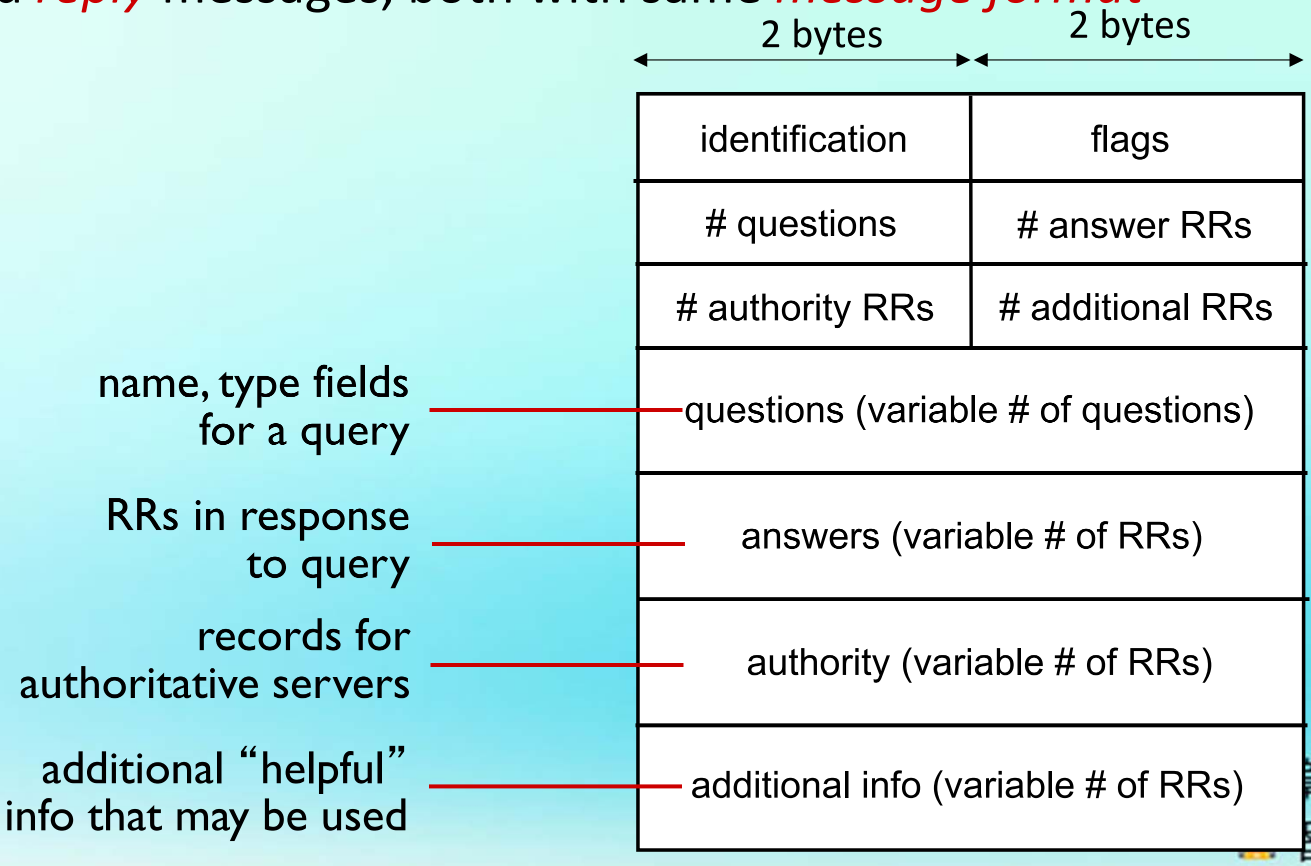


msg header

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

2.4.4 DNS Records and Messages
DNS protocol, messages

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



2.4.4 DNS Records and Messages

Inserting Records into the DNS Database

- example: new startup “Network Utopia”
- register name networkuptopia.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.networkuptopia.com; type MX record for networkutopia.com

2.4.4 DNS Records and Messages
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 bypass
- Bombard TLD servers
  - Potentially more dangerous

Redirect attacks

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

Exploit DNS for DDoS

- Send queries with spoofed source address: target IP
- Requires amplification