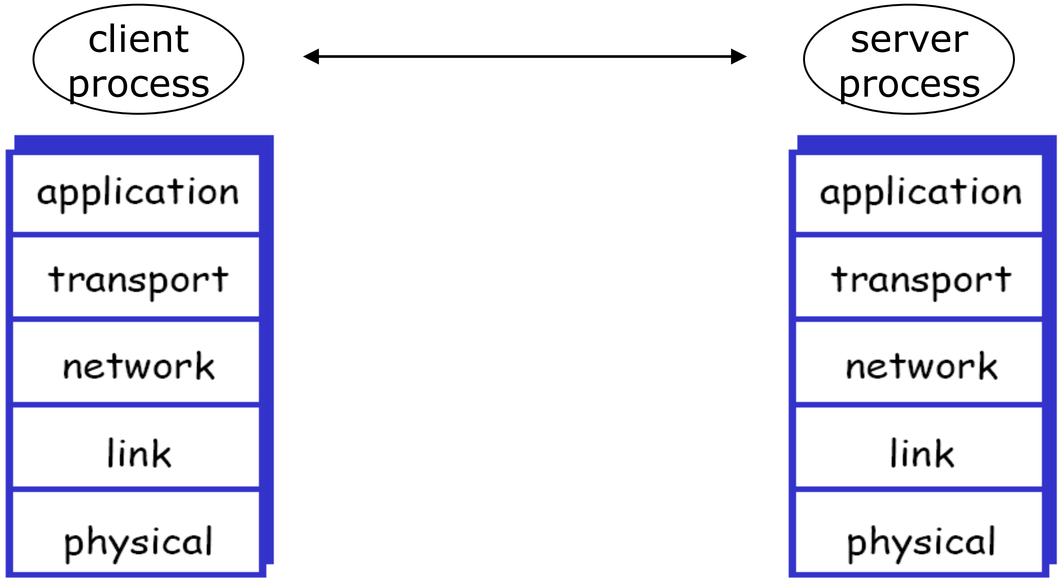
Chapter 2: Application Layer

- Principles of network applications
- Application Layer Protocols
- Network Programming Interface

Contents

Application Layer

- Application layer communication paradigm
  - client-server paradigm
  - peer-to-peer paradigm
  - publisher-subscriber paradigm

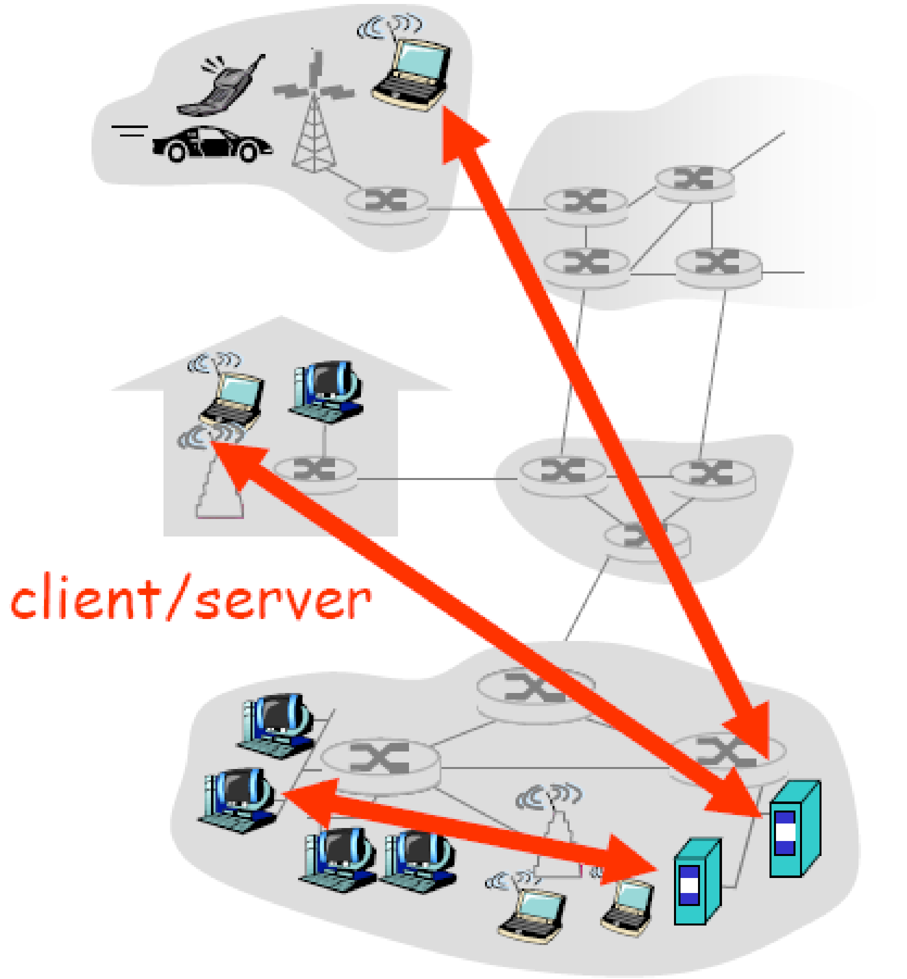


Application Layer

- Popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- Programming network applications
  - socket API

Application paradigm: Client-server

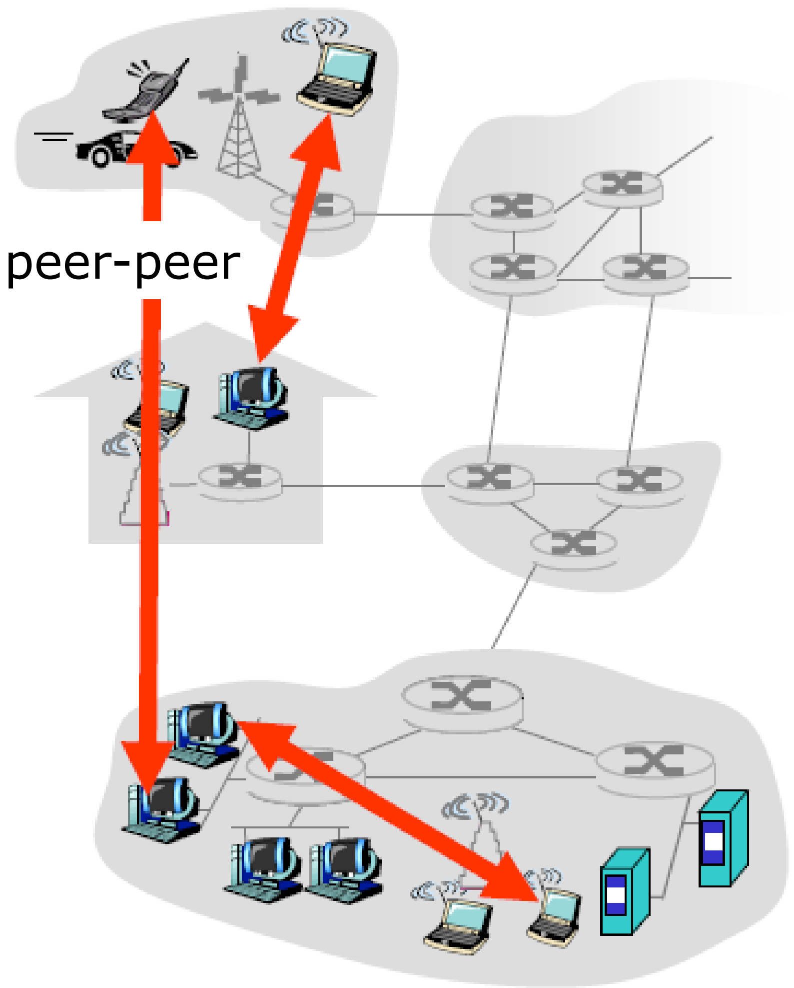
- server:
  - Provide a service to clients; always-on host
  - permanent IP address
  - server farms for scaling
- clients:
  - Receive a service from a server
  - communicate with server when needed
  - do not communicate directly with each other



Application paradigm: peer-to-peer (P2P)

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

Highly scalable but difficult to manage

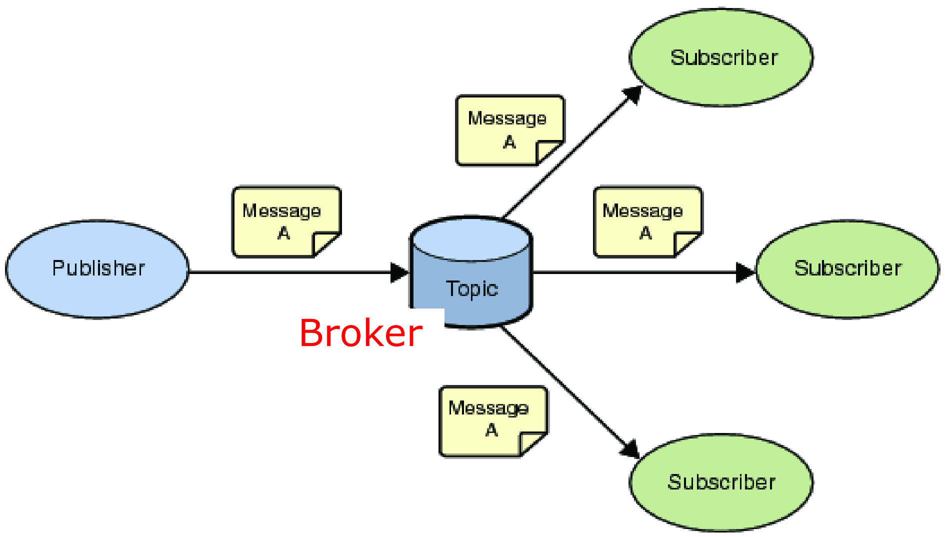


Hybrid of Client-server and P2P

- Skype
  - Voice-over-IP P2P application
  - centralized server: finding address of remote party
  - client-client connection: direct (not through server)
- Instant messaging
  - chatting between two users is P2P
  - centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

Application paradigm: Publisher-subscriber

- Publisher
  - generates and publishes message
- Subscriber
  - subscribes to a topic and consumes messages
- Broker



Processes Communicating

- Process: program running within a host
- within same host, two processes communicate using inter-process communication (in OS)
- communication end-point: processes
  - processes in different hosts communicate by exchanging messages

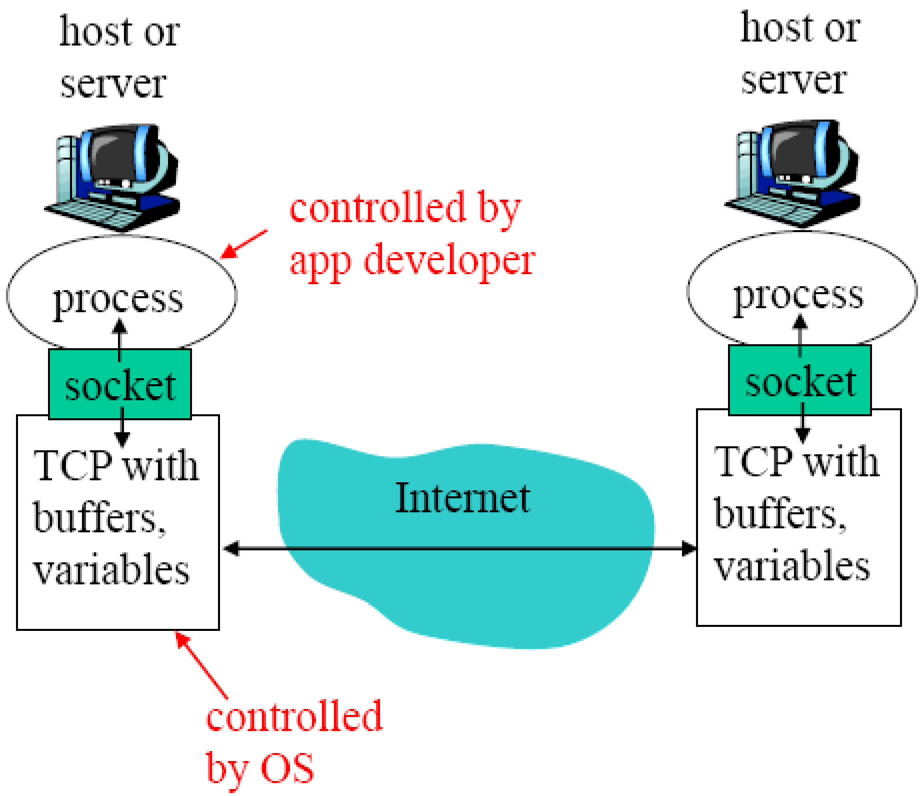
Client process: process that initiates communication

Server process: process that waits to be contacted

p2p applications have client processes & server processes

Sockets

- Process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process



Addressing Processes

- *Identifier* to recognize the comm. end-point (process)
- 32-bit IP address : identifies a host device
- *Q:* does IP address of host suffice for identifying the process?
- identifier (IP addr., port numbers) associated with process on host
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- to send HTTP message to cicweb.ulsan.ac.kr web server:
  - IP address: 203.250.77.119
  - Port number: 80

App-layer Protocol Defines

- Types of messages exchanged
  - e.g., request, response
- Message format
- Message semantics
  - meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:

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

Proprietary protocols:

- e.g., Skype

Transport Service for an Application

- Data loss
  - most apps (e.g., file transfer, telnet) require 100% reliable data transfer
  - some 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”
  - many apps (“elastic apps”) make use of whatever throughput they get
- Security
  - Encryption, data integrity, …

Transport Service Requirements of Common Applications

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, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Internet Transport Protocols Services

Stream-oriented service: TCP

- *connection-oriented:* setup required between client and server processes
- *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 guarantees, security

Internet Transport Protocols Services

Datagram service: UDP

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

Q: Why is there a UDP?

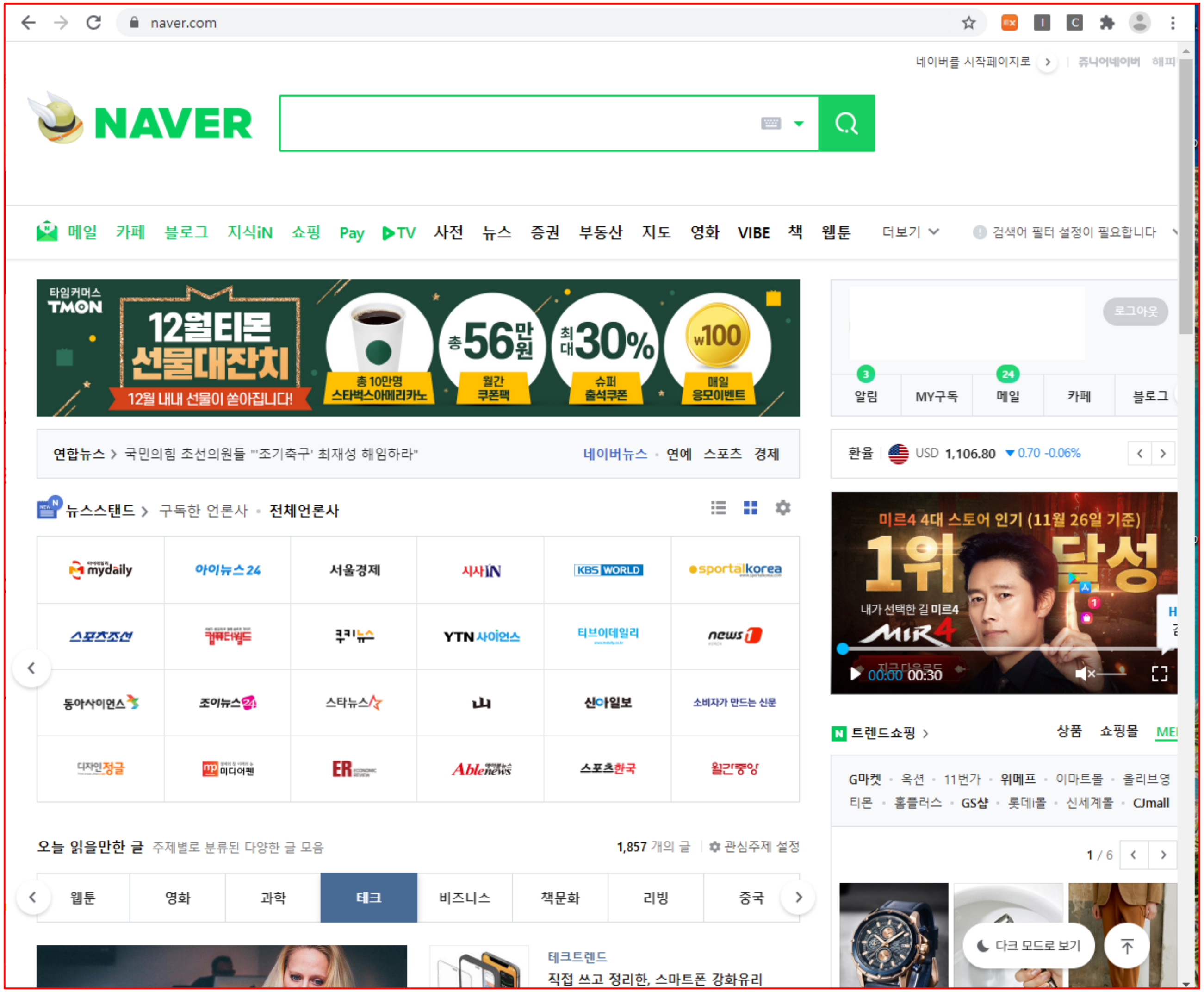
Transport Protocols for Internet Apps

Application	Application layer protocol	Transport protocol
web	HTTP [RFC 2616]	TCP
e-mail	SMTP [RFC 2821]	TCP
telnet	Telnet [RFC 854]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	Typically UDP

Chap 2

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail : smtp, pop3, imap
- DNS
- P2P applications
- Socket programming

Web and HTTP


naver.com
네이버를 시작페이지로
NAVER
메일 카페 블로그 지식iN 쇼핑 Pay TV 사전 뉴스 증권 부동산 지도 영화 VIBE 책 웹툰 더보기
검색어 필터 설정이 필요합니다
12월티몬 선물대잔치
12월 내내 선물이 쏟아집니다!
연합뉴스 국민의힘 초선의원들 "'조기축구' 최재성 해임하라"
네이버뉴스 연예 스포츠 경제
뉴스스탠드 구독한 언론사 전체언론사
오늘 읽을만한 글 주제별로 분류된 다양한 글 모음
1,857 개의 글 관심주제 설정
웹툰 영화 과학 테크 비즈니스 책문화 리빙 중국
테크트렌드
직접 쓰고 정리한, 스마트폰 강화유리
로그아웃
알림 MY구독 메일 카페 블로그
환율 USD 1,106.80 0.70 -0.06%
트렌드쇼핑 상품 쇼핑몰
G마켓 옥션 11번가 위메프 이마트몰 올리브영
티몬 홈플러스 GS샵 롯데몰 신세계몰 CJmall
1 / 6
다크 모드로 보기

Web and HTTP

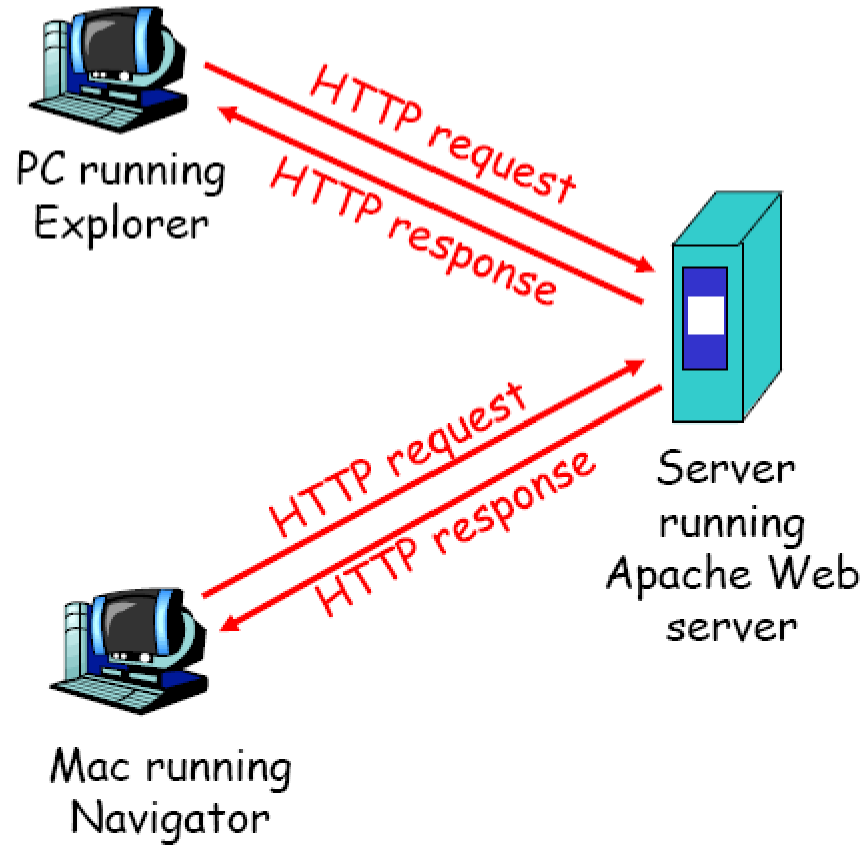
- Web page consists of objects
  - Base page: HTML file (base page)
  - Referenced objects: JPEG image, Java applet, audio file, …
  - Each object is addressable by a URL

    (e.g.) http://cicweb.ulsan.ac.kr/Undergraduate/course.html
- Web browser
  - gets a base HTML file
  - interprets HTML file
  - gets more objects referenced in the base HTML file
  - displays the information after rendering

WWW Service: HTTP protocol

HTTP: hypertext transfer protocol

- Request-Response message exchange
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests



HTTP Overview

Uses TCP:

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

HTTP is "stateless"

- server maintains no information about past client requests

aside

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

HTTP Connections

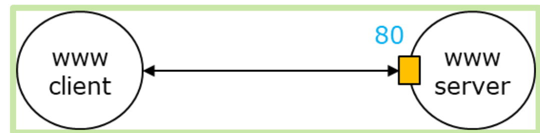
- Non-persistent HTTP
  - At most one object is sent over a TCP connection
  - for each object:

    connection setup – data exchange – connection close

- Persistent HTTP
  - Multiple objects can be sent over a single TCP connection between client and server

Non-persistent Connection in HTTP

- URL: www.someSchool.edu/someDept/home.html
  - It has 10 referenced objects (jpeg images)



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

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

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

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

time

Non-persistent Connection in HTTP

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

time

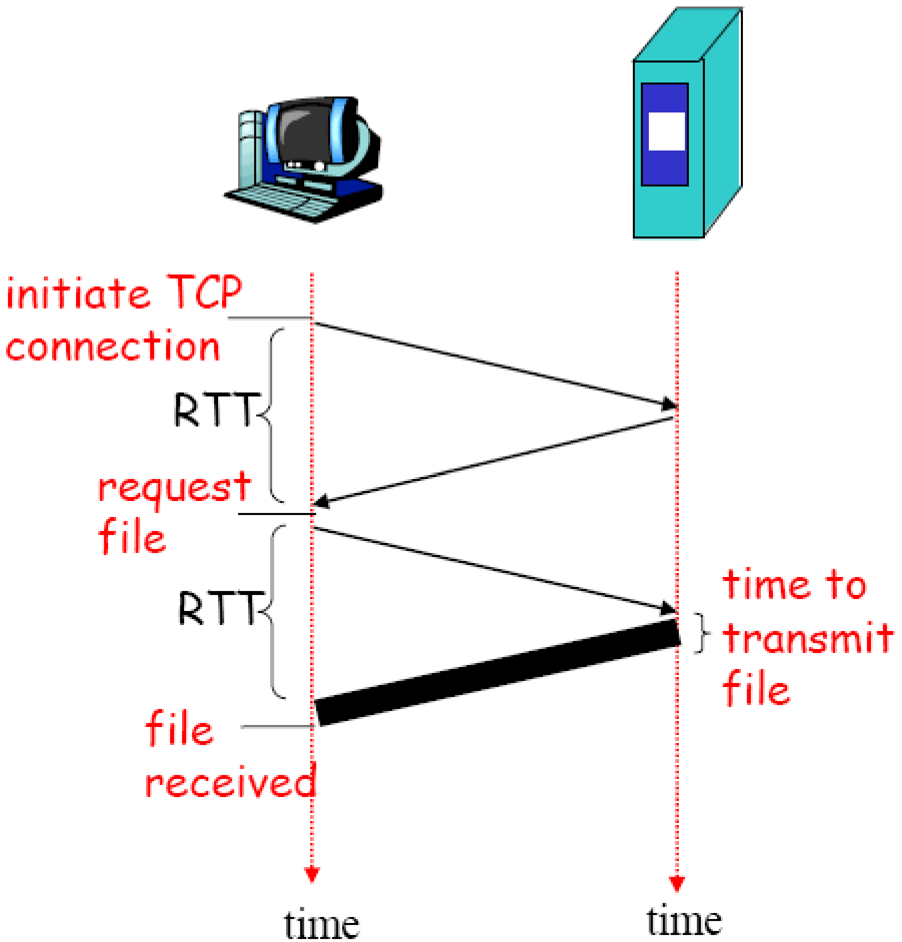
6. Steps 1-5 repeated for each of 10 jpeg objects

Non-Persistent HTTP

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

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
- total = 2*RTT + file transmit time



Persistent HTTP

Non-persistent HTTP issues:

- requires 2 RTTs per object
- OS 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
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

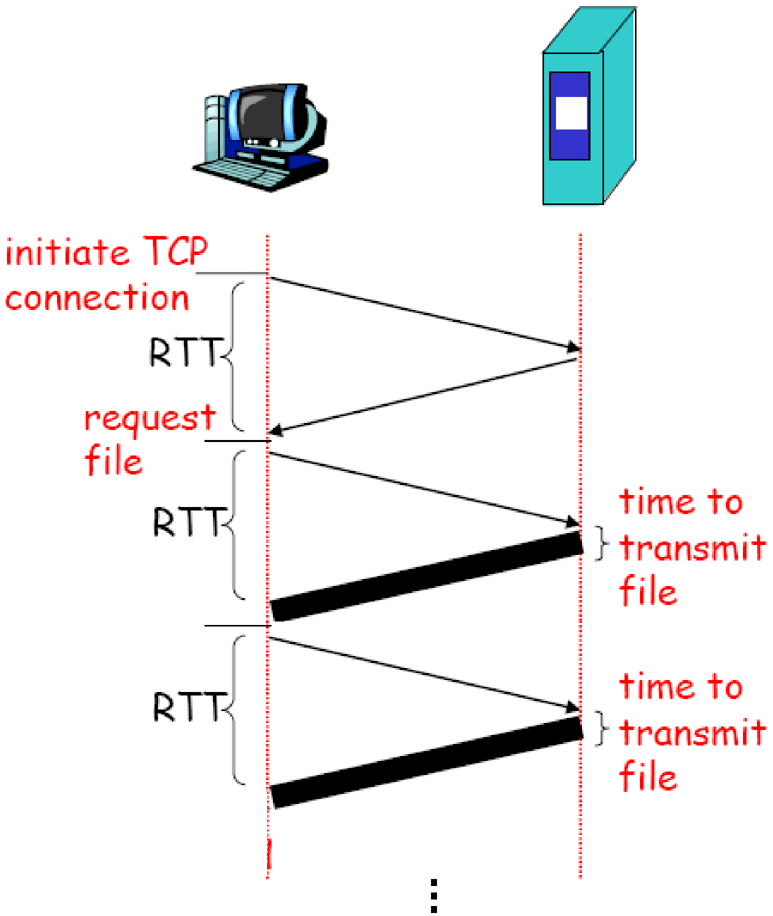
Response Time

Non-persistent HTTP:

- N: number of objects to download
- response time = N * (2*RTT + file transmit time)

Persistent HTTP:

- N: number of objects to download
- response time = RTT + N * (RTT + file transmit time)



HTTP Message

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

IH	TH	HTTP message

http://www.someschool.edu/somedir/page.html

request line
(GET, POST,
HEAD commands)

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header
lines

Carriage return,
line feed
indicates end
of message

(extra carriage return, line feed)

Method Types

HTTP/1.0

- GET
- POST
- HEAD: asks server to send only HTTP header information

HTTP/1.1

- GET, POST, HEAD
- PUT: uploads file in entity body to path specified in URL field
- DELETE: deletes file specified in the URL field

HTTP Response Message

- HTTP response message format

status line (protocol status code status phrase)

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

header lines

data, e.g., requested HTML file

HTTP Response Status Codes

`200 OK`

- request succeeded, requested object later in this message

`301 Moved Permanently`

- requested object moved, new location specified later in this message (Location:)

`400 Bad Request`

- request message not understood by server

`404 Not Found`

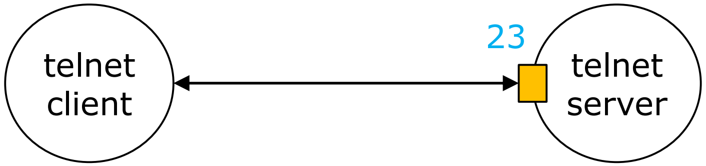
- requested document not found on this server

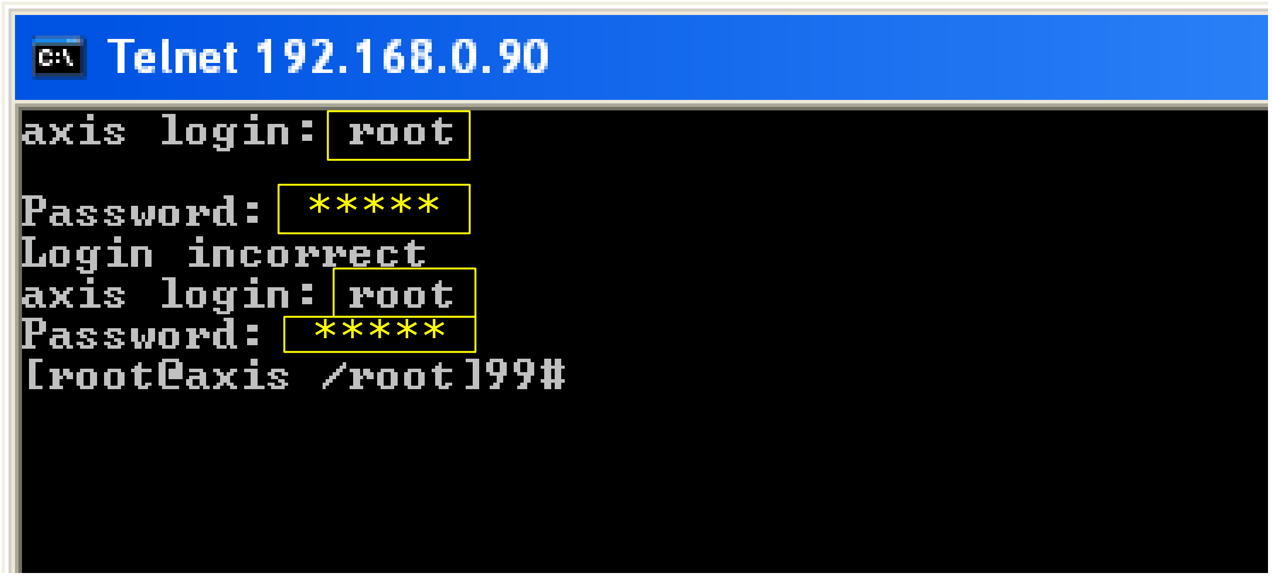
`505 HTTP Version Not Supported`

Trying-out HTTP for Yourself

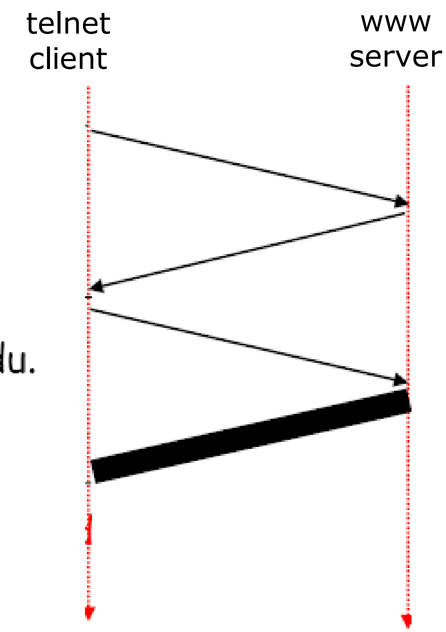
telnet: remote login service

- telnet <host>
- telnet uses TCP
- telnet uses port 23 by default





Trying-out HTTP for Yourself



1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. Anything typed in sent to port 80 at cis.poly.edu

2. Type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Look at response message sent by HTTP server!

User-server State: Cookies

Many Web sites use cookies for customized service

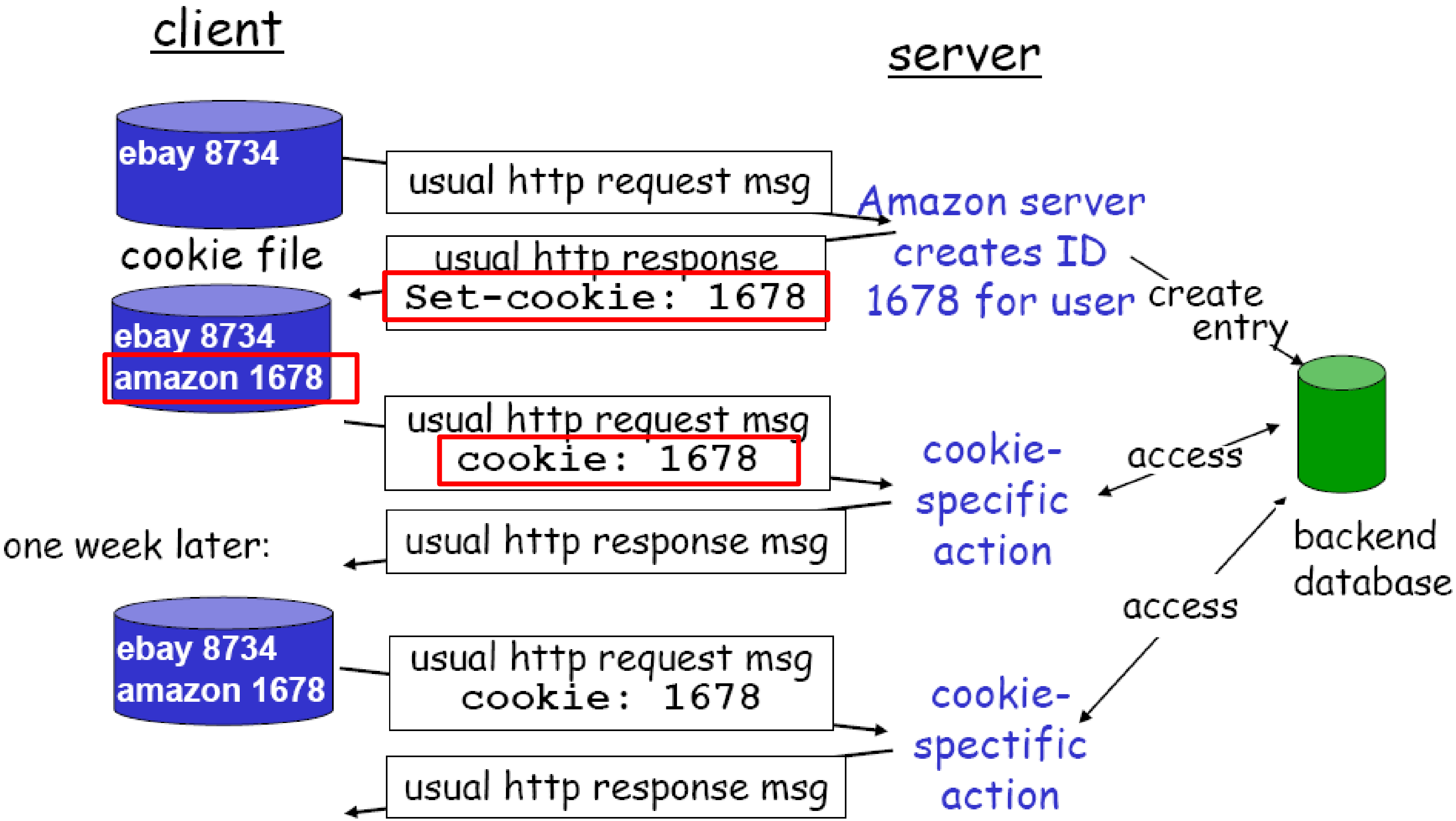
Four components:

1) cookie header line of HTTP *response* message
2) cookie header line in HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end DB at Web site

Example:

- when visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - creates an entry in backend DB for ID and stores some info. for the ID

User-server State: Cookies



User-server State: Cookies

What cookies can bring:

- 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

aside

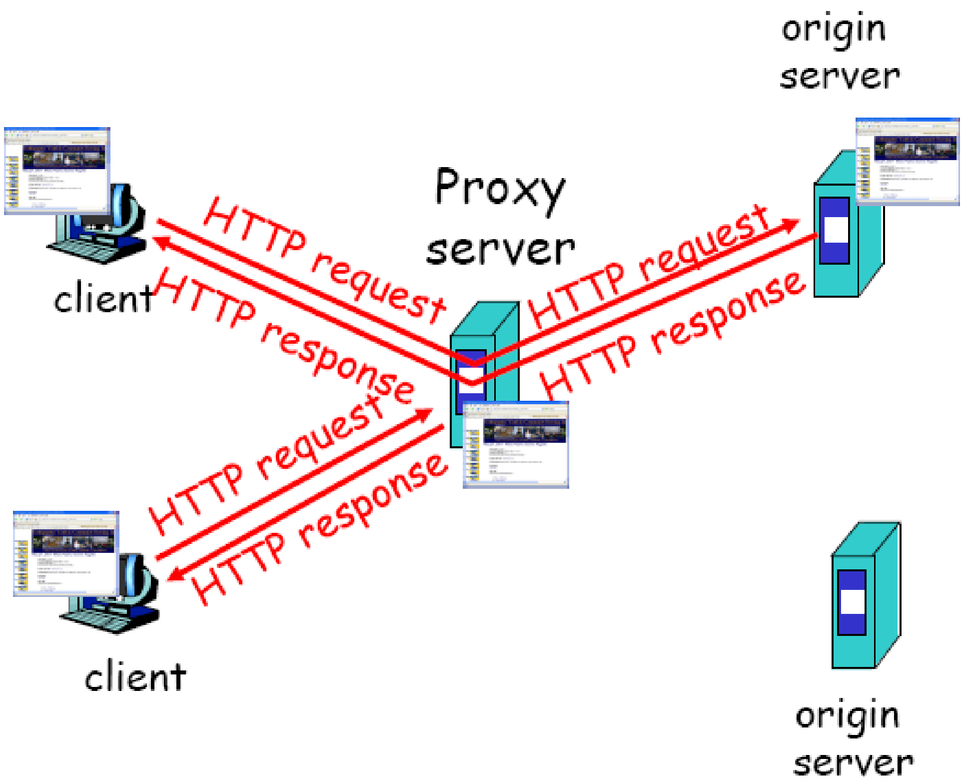
Cookies and privacy:

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

Web caches (proxy server)

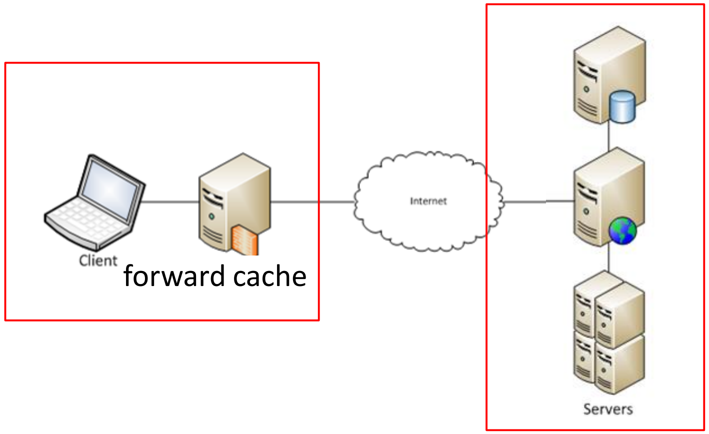
Goal: satisfy client request without involving origin server

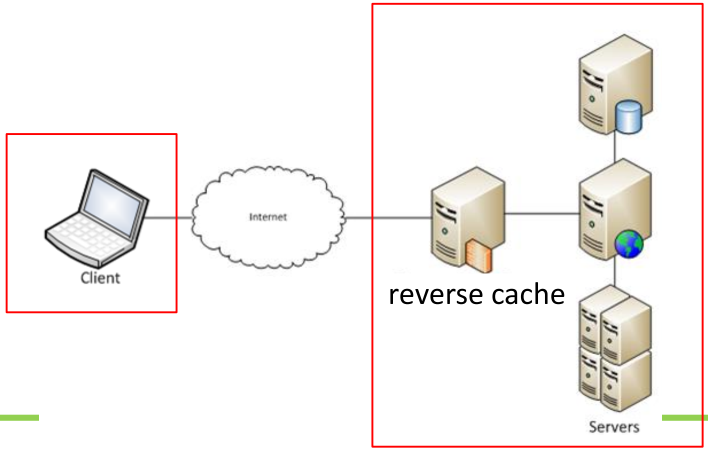
- user sets browser: proxy server; www accesses via proxy
- browser sends all HTTP requests to proxy
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



Web caches (proxy server)

- cache acts as both client and server





<u>Why Web caching?</u>

- Forward cache:
  - reduce response time for client request
  - reduce traffic on an institution's access link
- Reverse cache:
  - reduces and balances the load of the server

Caching Example

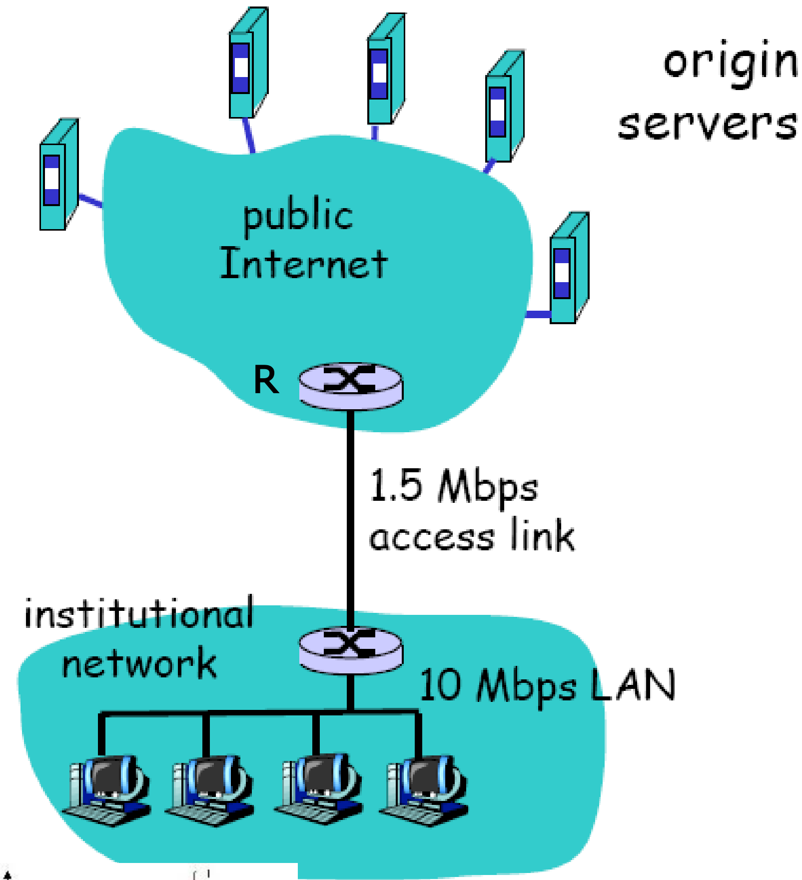
Assumptions

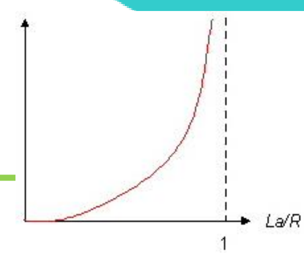
- average object size = 100,000 bits
- avg. request rate from institution's browsers to origin servers = 15/sec
- delay from institutional router (R) to any origin server and back to router = 2 sec

Consequences

- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access link delay + LAN delay

= 2 sec + minutes + milliseconds





Caching Example

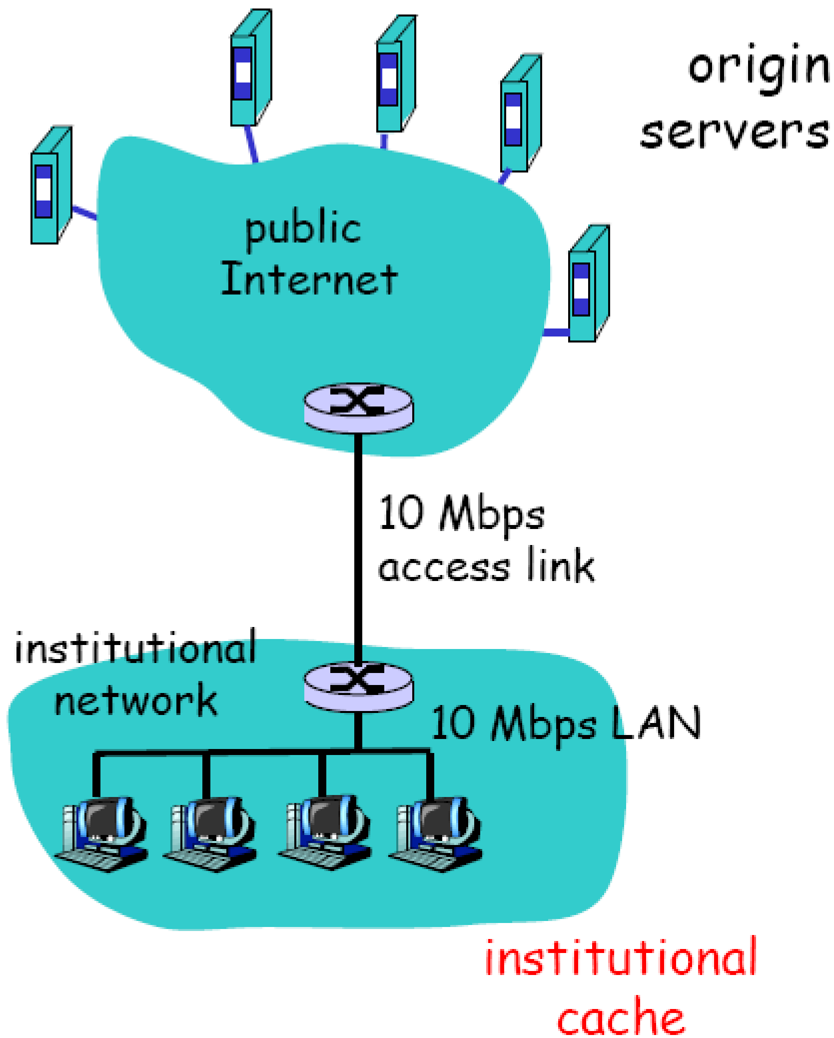
possible solution

- increase bandwidth of access link to, say, 10 Mbps

consequence

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access link delay + LAN delay

  = 2 sec + msecs + msecs
- expensive !!!



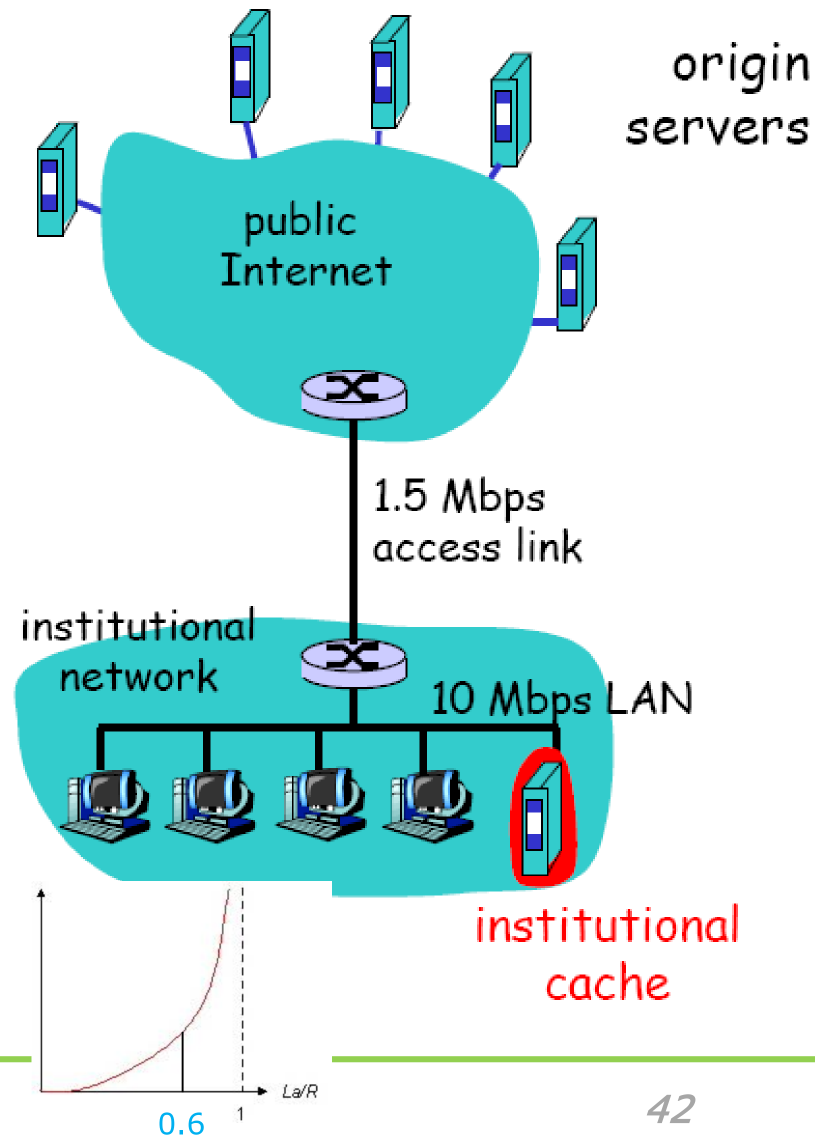
Caching Example

possible solution: install cache

- suppose cache hit rate is 0.4

Result

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access link delay + LAN delay
  = .6*(2.01) secs + .4*milliseconds < 1.4 secs



Conditional GET

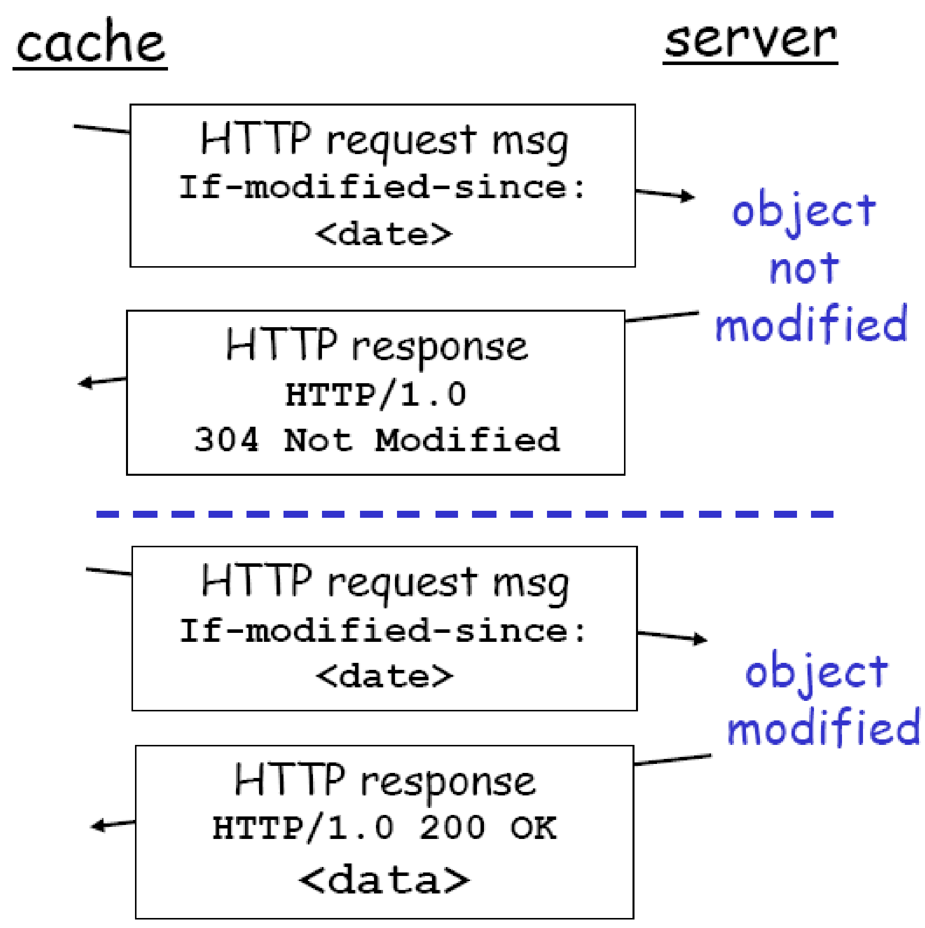
Goal: don't send object if cache has up-to-date cached version

- 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`



Homework #1

Due date: 4월 2일

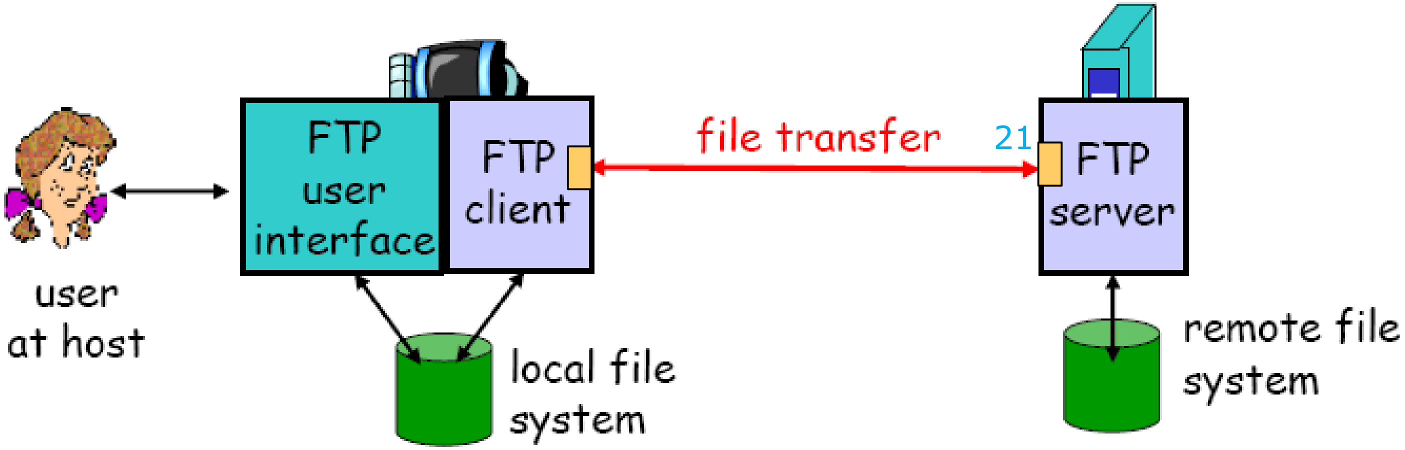
- HTTP protocol
  - Request headers: 종류, 의미
  - Response headers: 종류, 의미

- wireshark
  - wireshark 기능, 설치, 사용법 정리
  - cicweb.ulsan.ac.kr 홈페이지 접속
    - HTTP protocol request 패킷 분석
    - HTTP protocol response 패킷에서 header 정보를 가진 패킷 분석

Chap 2

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail : smtp, pop3, imap
- DNS
- P2P applications
- Socket programming

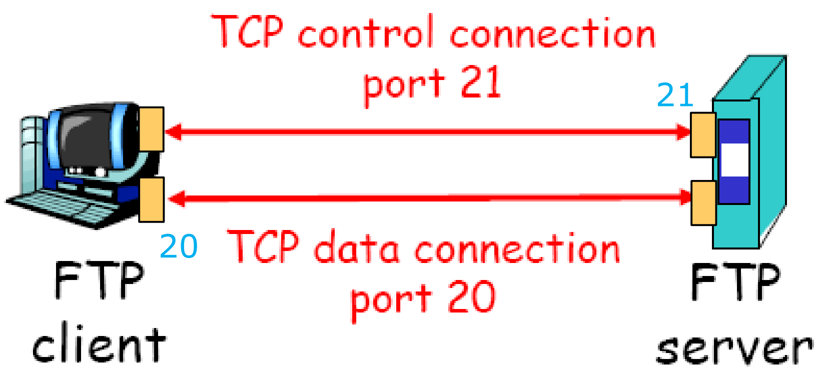
FTP (File Transfer Protocol)



- 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

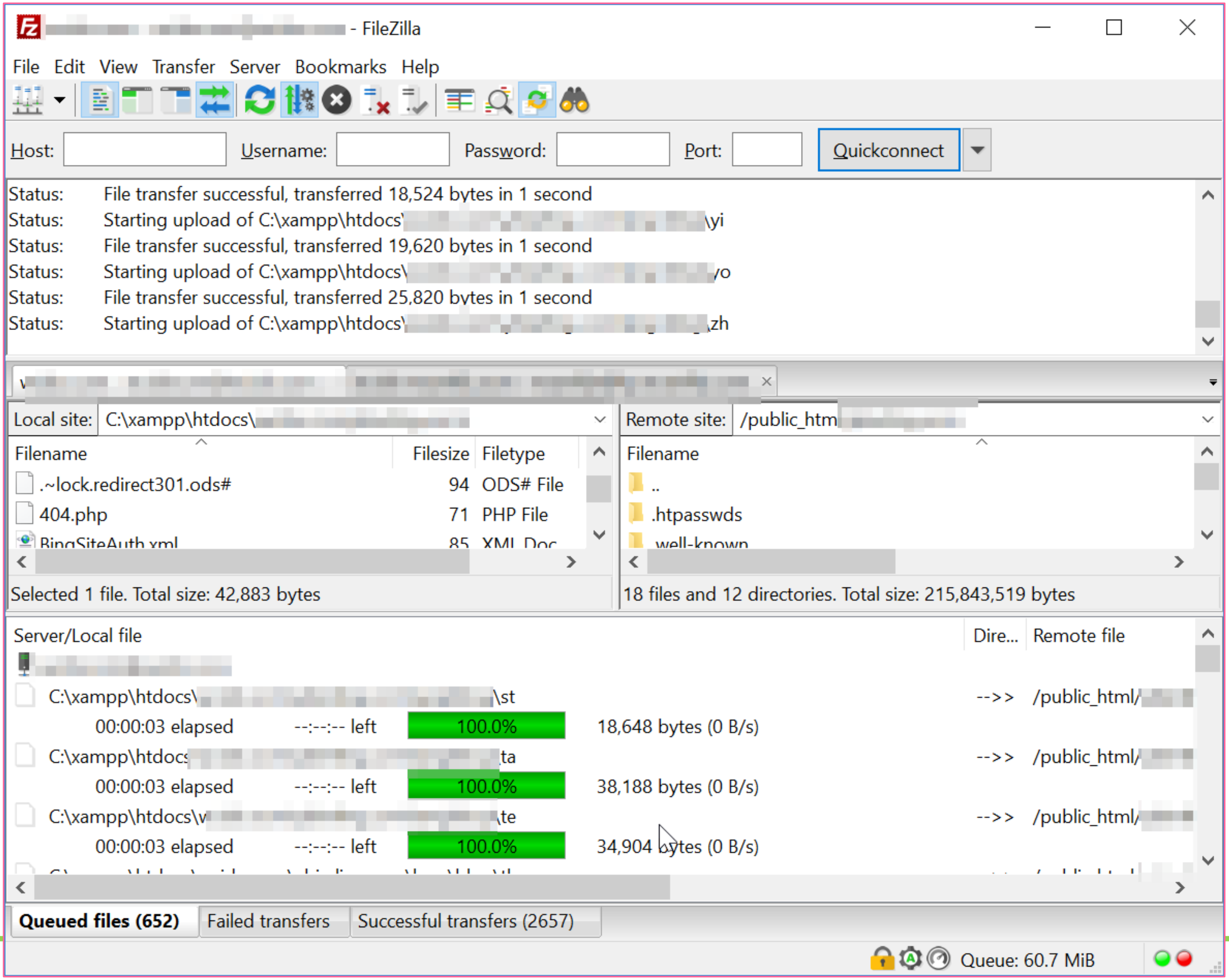
- FTP client contacts FTP server at port 21, TCP
- client authorized over control connection
- client browses remote directory by sending commands over control connection
- when server receives file transfer command, server opens *2-nd* TCP connection to send data to client
- after transferring one file, server closes data connection



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

FTP: Separate Control, Data Connections

- ftp agent: FileZilla



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

Chap 2

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail : smtp, pop3, imap
- DNS
- P2P applications
- Socket programming

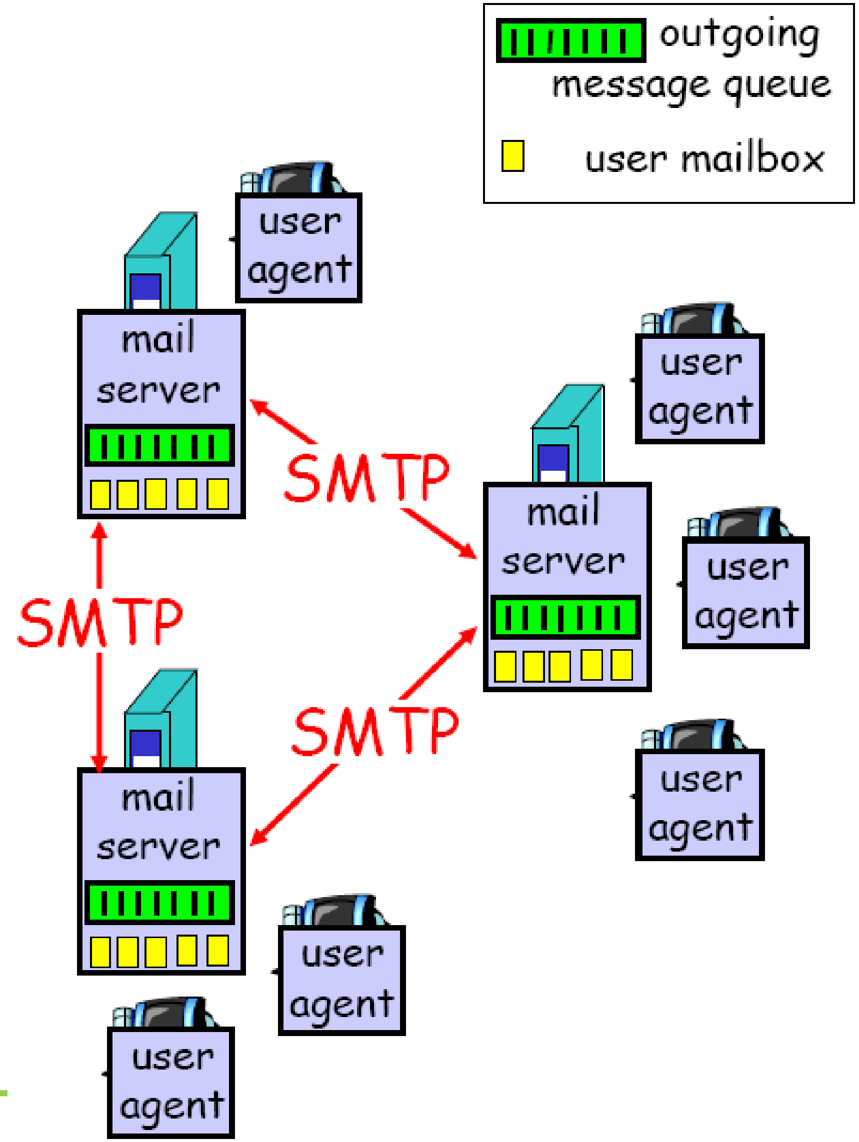
Electronic Mail

<u>Three major components:</u>

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

<u>User Agent</u>

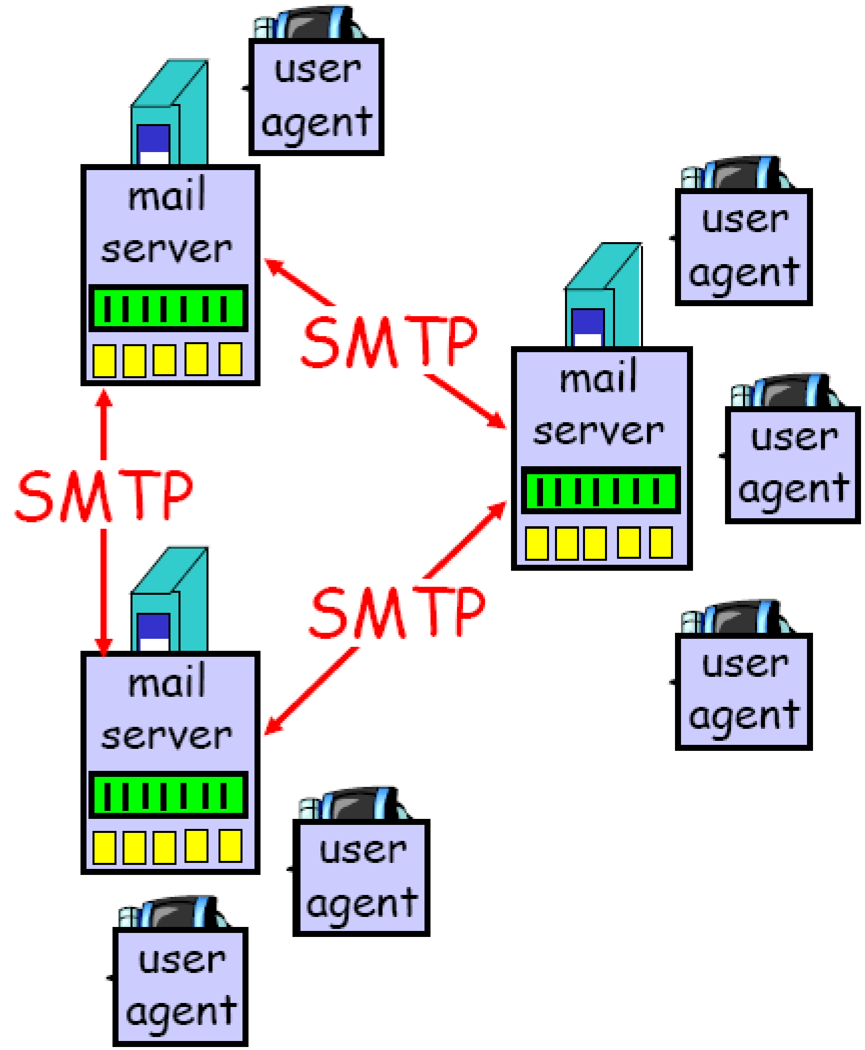
- reading email messages; composing, editing, and sending email messages
- e.g., Eudora, Outlook, elm, Mozilla Thunderbird
- outgoing, incoming messages stored on server



Electronic Mail

Mail Servers

- mailbox contains incoming messages for user
- email ID: mkkim@ulsan.ac.kr
- 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



Electronic Mail: SMTP [RFC 2821]

- Uses TCP : port 25
- Direct transfer: sending server to receiving server
- Three phases of transfer
  - handshaking (greeting) – transfer of messages – closure
- Command/response interaction
  - commands: ASCII text
  - response: status code and phrase
- Messages must be in 7-bit ASCII

Scenario: Alice sends message to Bob

1) Alice uses UA to send email to "`bob@mail.someschool.edu`"

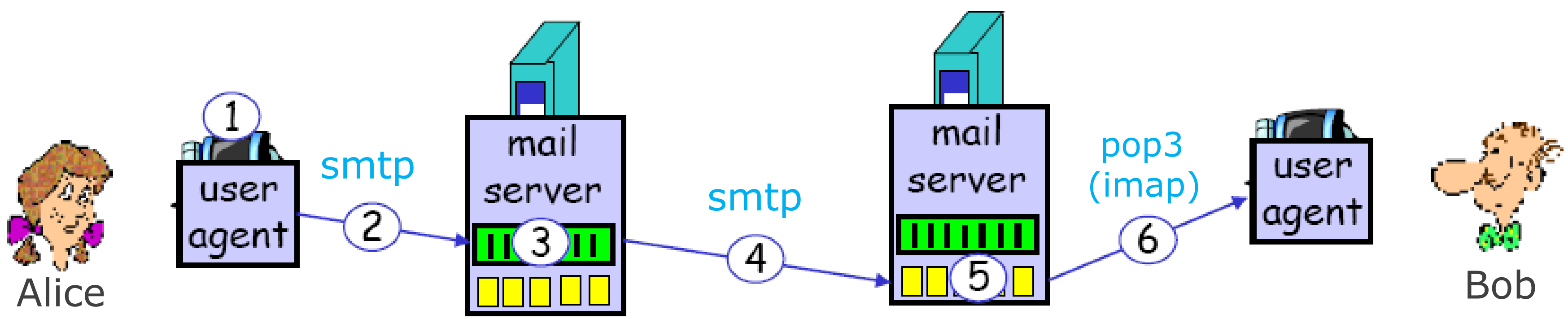
2) Alice's UA sends message to her mail server; message placed in message queue

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

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

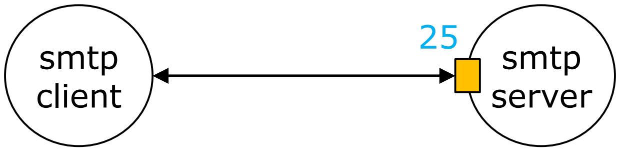
5) Bob's mail server places the message in Bob's mailbox

6) Bob invokes his UA to read message



Sample SMTP Interaction

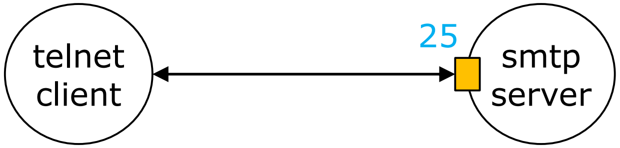
tcp connection setup




```
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
```

Try SMTP Interaction for Yourself

- **telnet <servername> 25**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands




```
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
```

SMTP

- 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

Comparison with HTTP:

- HTTP: pull
- SMTP: push

- both have ASCII command/response interaction, status codes

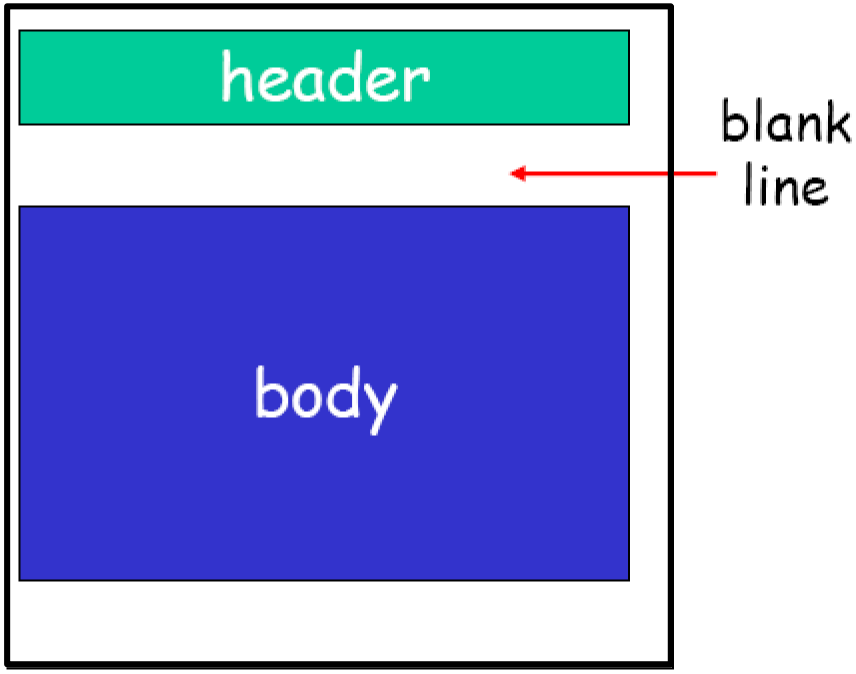
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart message

Mail Message Format

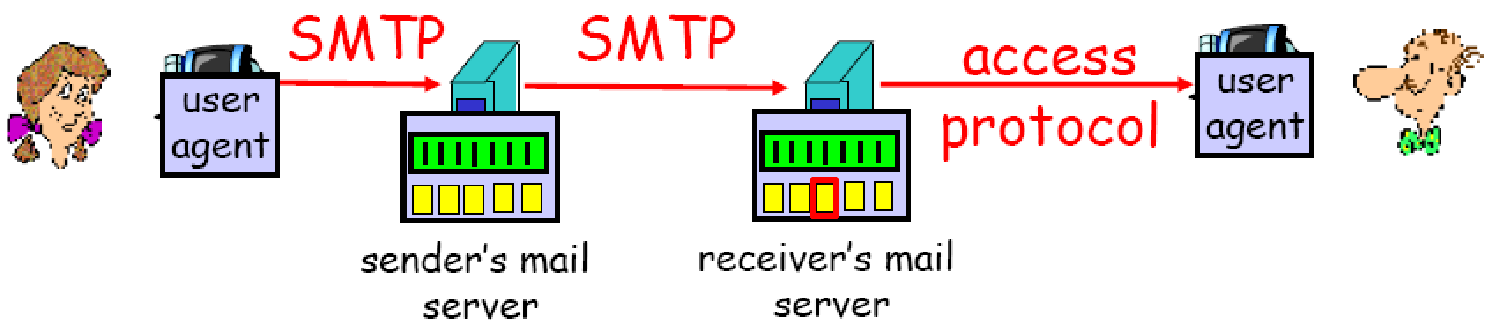
RFC 822: standard for text message format; obsoleted by RFC 2822

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

  different from SMTP commands!
- body
  - the "message", ASCII characters only; extended to MIME types (content-type header)



Mail Access Protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]; port 110
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]; port 143
  - HTTP: port 80
    - gmail, naver mail, hotmail, yahoo! mail, etc.

POP3 Protocol

authorization phase

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

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

transaction phase, client:

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

```
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
```

Homework #2

Due date: 4월 9일

- SMTP protocol message format
  - email message headers: 종류, 의미
- wireshark
  - 전송 이메일 패킷 캡쳐
    - smtp protocol 패킷 분석

Chap 2

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail : smtp, pop3, imap
- DNS
- P2P applications
- Socket programming

DNS: Domain Name System

People: many IDs:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- name: www.naver.com - used by humans

Q: map between IP addr and name ?

(domain-name, IP-addr)

Domain Name System:

- distributed DB implemented in hierarchy of many name servers
- application-layer protocol to resolve names
  - core Internet function, implemented as application-layer protocol

DNS

DNS services

- hostname to IP address translation
- host aliasing
  - Canonical, alias names
- load distribution
  - replicated web servers: set of IP addresses for one canonical name

Why not centralize DNS?

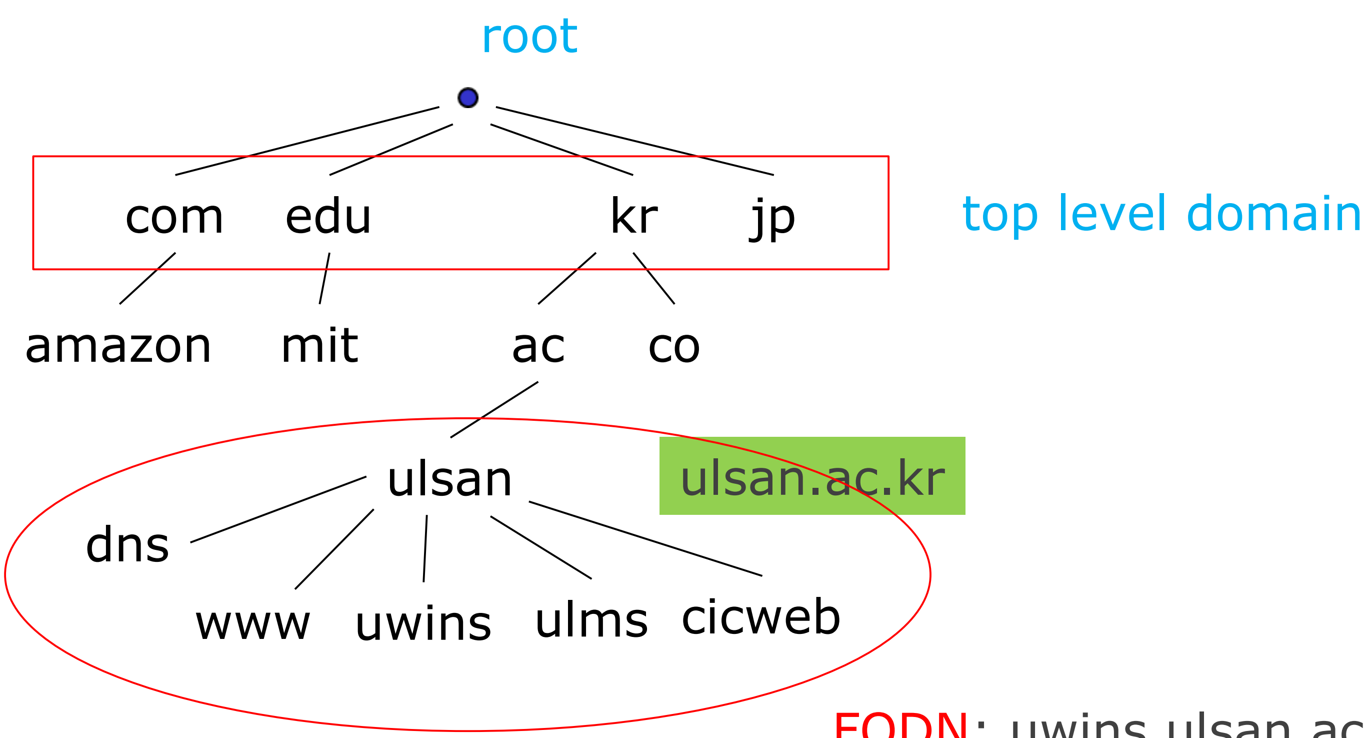
- single point of failure
- traffic volume
- large delay to a distant centralized server

=> doesn't scale!

Distributed, Hierarchical Database

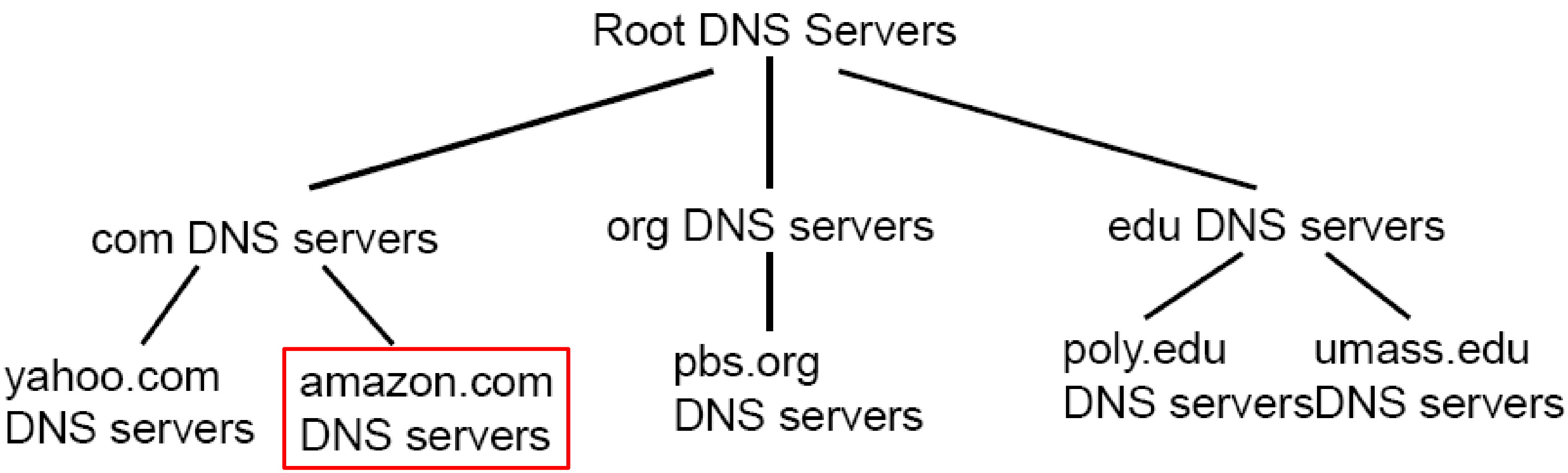
Domain name

- hierarchical name space



FQDN: uwins.ulsan.ac.kr.

Distributed, Hierarchical Database

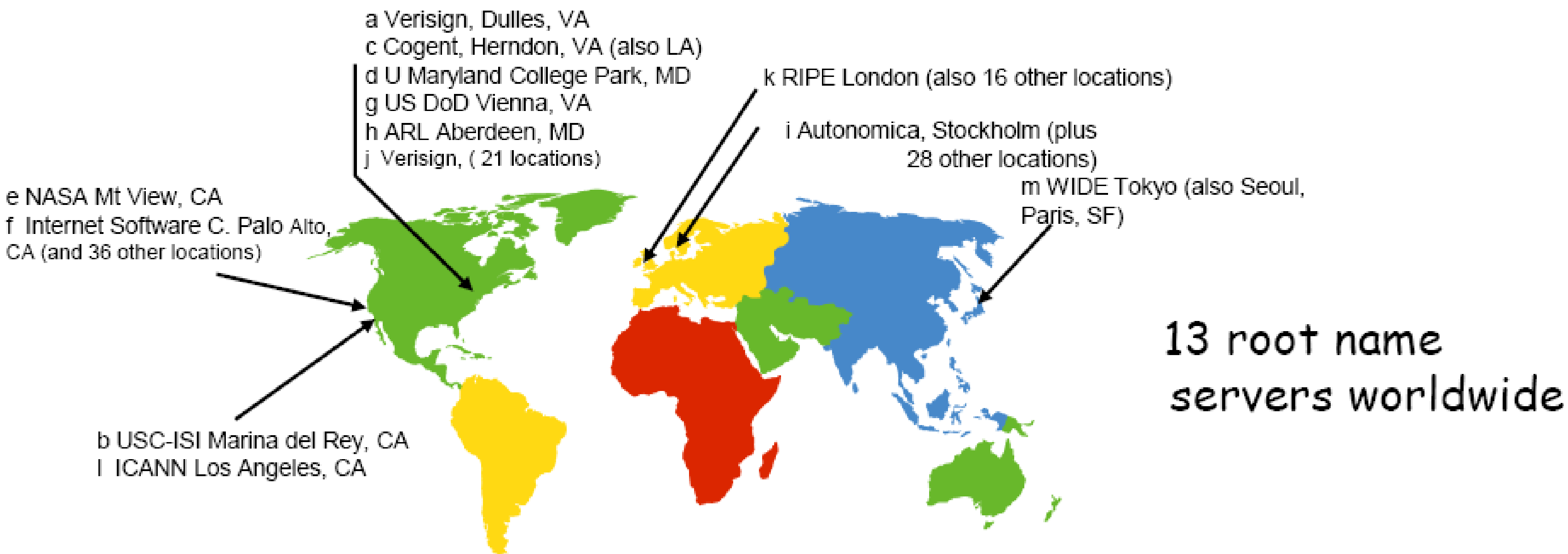


Client wants IP for www.amazon.com

- client queries a 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

- 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



TLD and Authoritative Servers

- Top-level domain (TLD) servers:
  - responsible for gTLD (generic TLD - com, org, net, edu, etc), and ccTLD (country code TLD - uk, fr, ca, jp, kr, etc)
  - KISA maintains servers for kr TLD
  - Educause for edu TLD
- Authoritative DNS servers:
  - organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web, mail)
  - can be maintained by organization or service provider

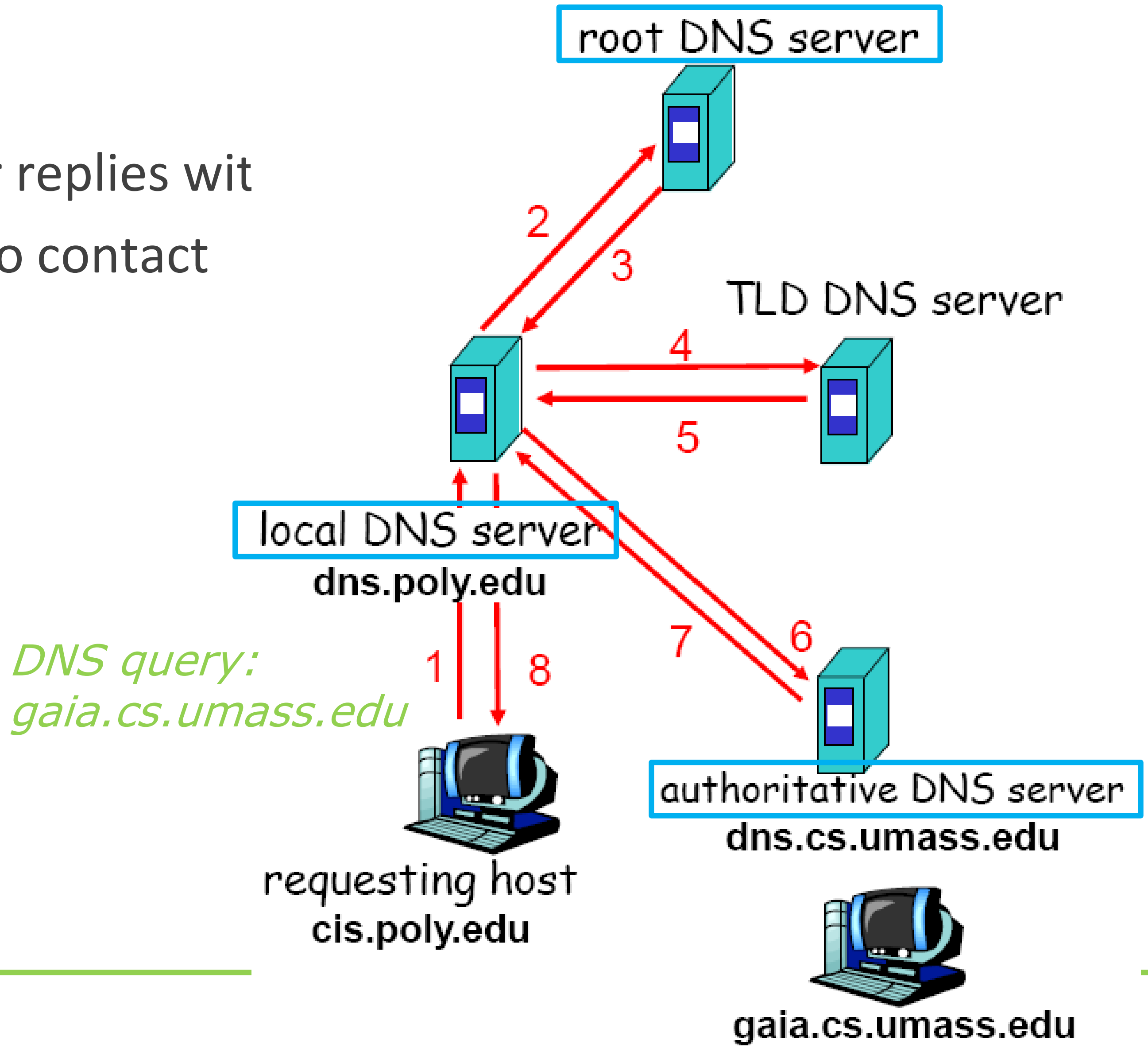
Local Name Server

- does not strictly belong to hierarchy
- each organization (residential ISP, company, university) has one
  - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
  - acts as proxy, forwards query into hierarchy

DNS Name Resolution

Iterated query:

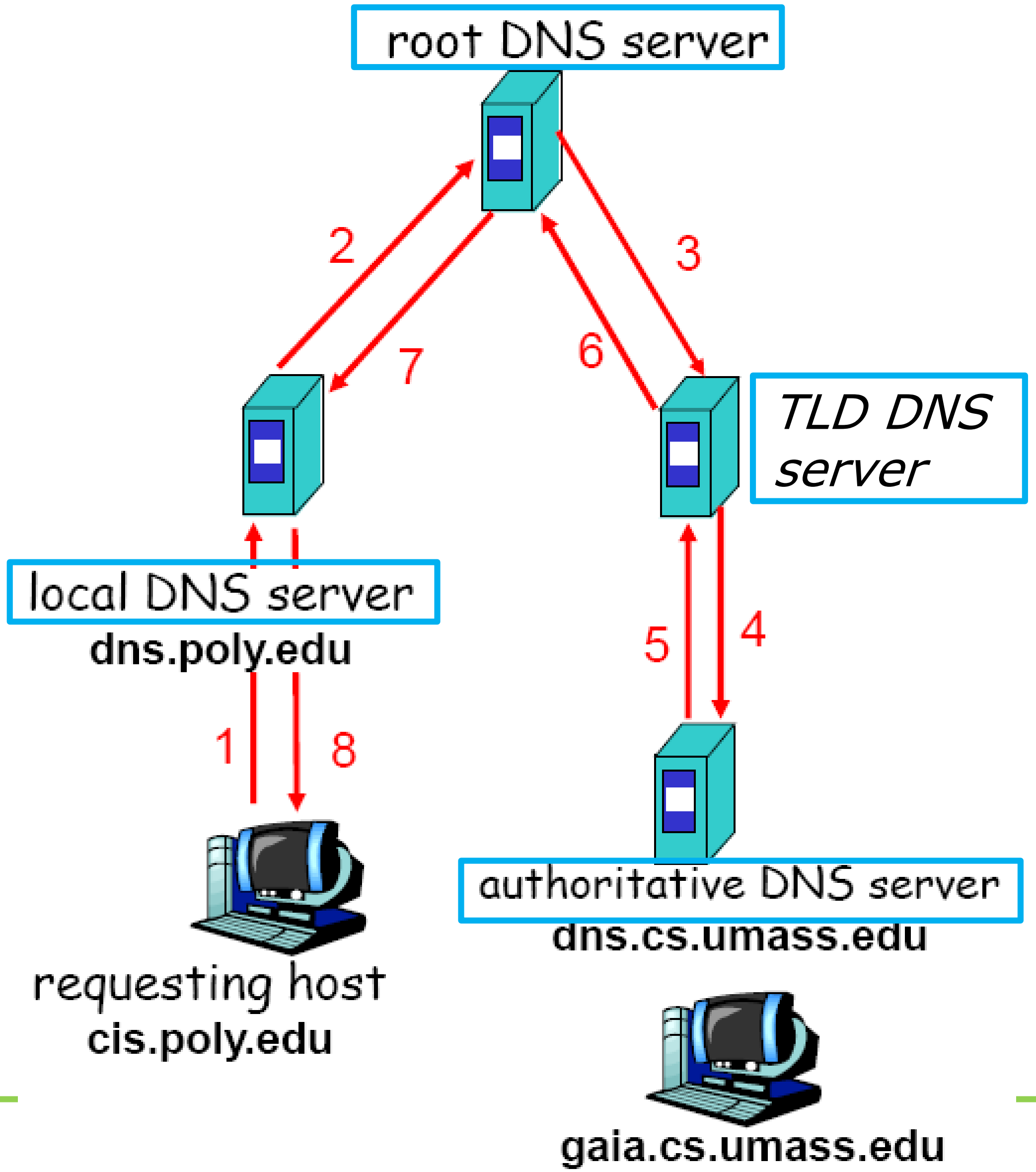
- contacted server replies wit name of server to contact
- load on client



DNS Name Resolution

recursive query:

- puts burden of name resolution on contacted name server
- load on root DNS server



DNS Records

DNS: distributed DB storing resource records (RR)

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

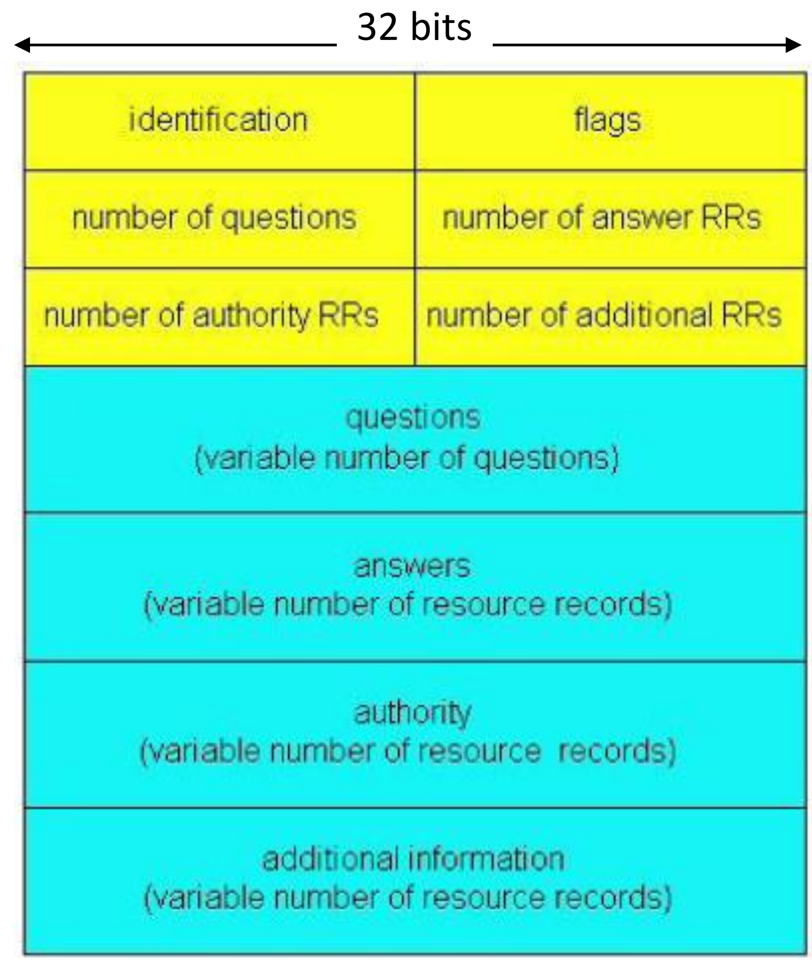
- Type=A
  - name is hostname
  - value is IP address
- Type=NS
  - name is domain
  - value is hostname of authoritative name server for this domain
- Type=CNAME
  - name is alias name for some "canonical" (the real) name
  - value is canonical name
- Type=MX
  - value is name of mail server associated with name

DNS protocol messages

DNS protocol : *query* and *reply* messages, with same format

msg header

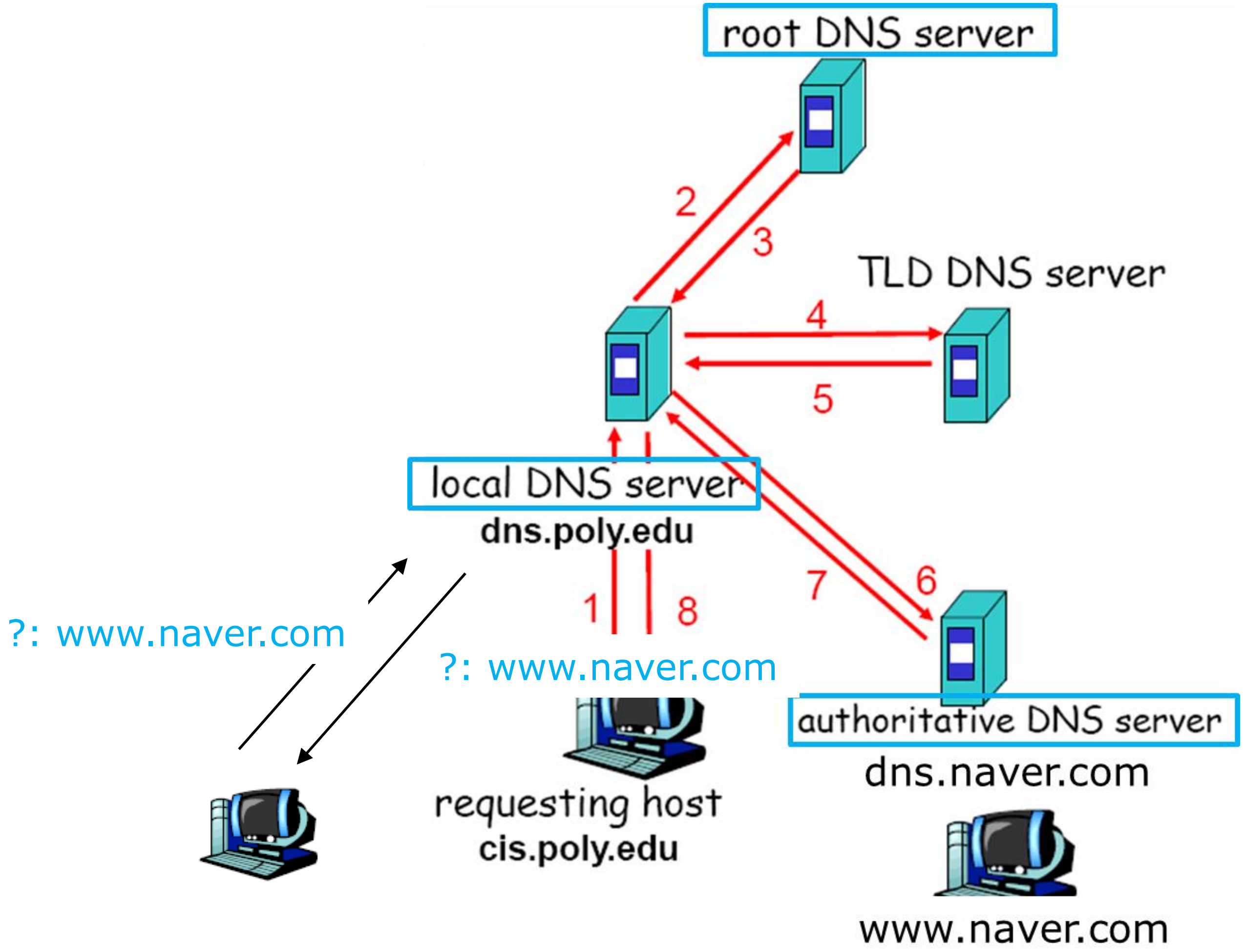
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



DNS: caching and updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited
- DDNS (Dynamic DNS)
  - management mechanisms of (DNS: dynamic IP) mapping
  - (DNS: dynamic IP) maintained in main memory DB
  - RFC 2136

DNS caching


root DNS server
2
3
TLD DNS server
4
5
local DNS server
dns.poly.edu
7
6
1
8
?: www.naver.com
?: www.naver.com
authoritative DNS server
dns.naver.com
requesting host
cis.poly.edu
www.naver.com

Inserting Records into DNS

- Example: new startup "Network Utopia"

- Register name networkuptopia.com at com TLD *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:

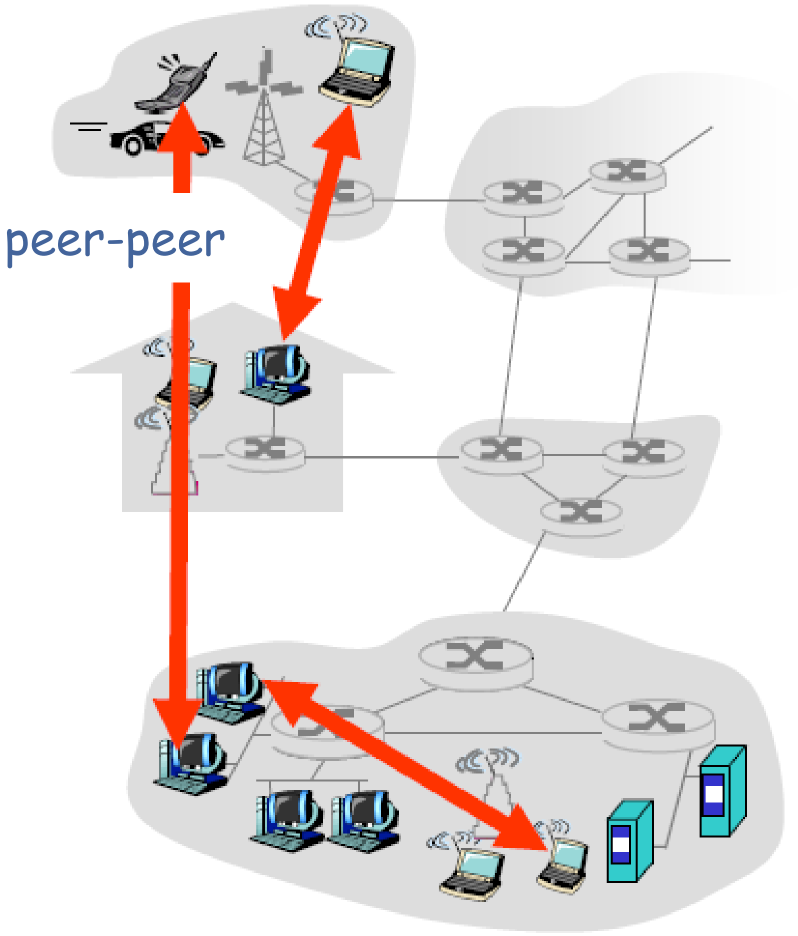
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)

Chap 2

- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail : smtp, pop3, imap
- DNS
- P2P applications
- Socket programming

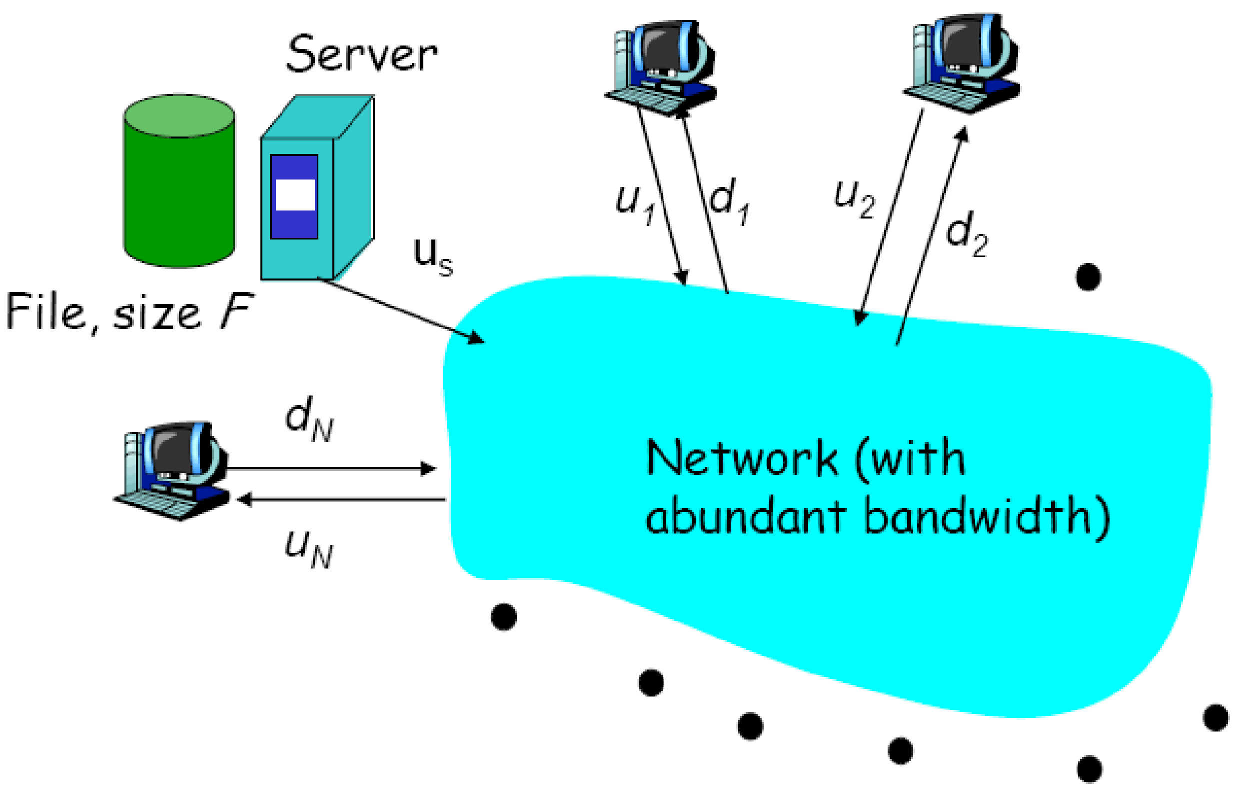
Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses
- Main applications:
  - File distribution
  - Case Study: Skype



File Distribution: Server-Client vs P2P

- *Question* : How much time to distribute file from one server to *N peers*?



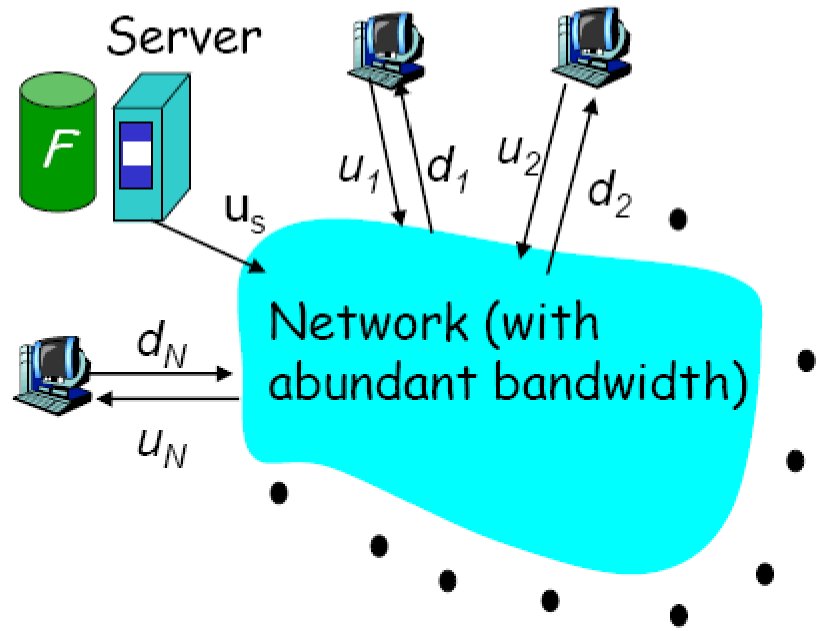
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

File distribution time: server-client

- server sequentially sends N copies:
  - $N*F/u_s$ time
- client *i* takes F/d_i time to download

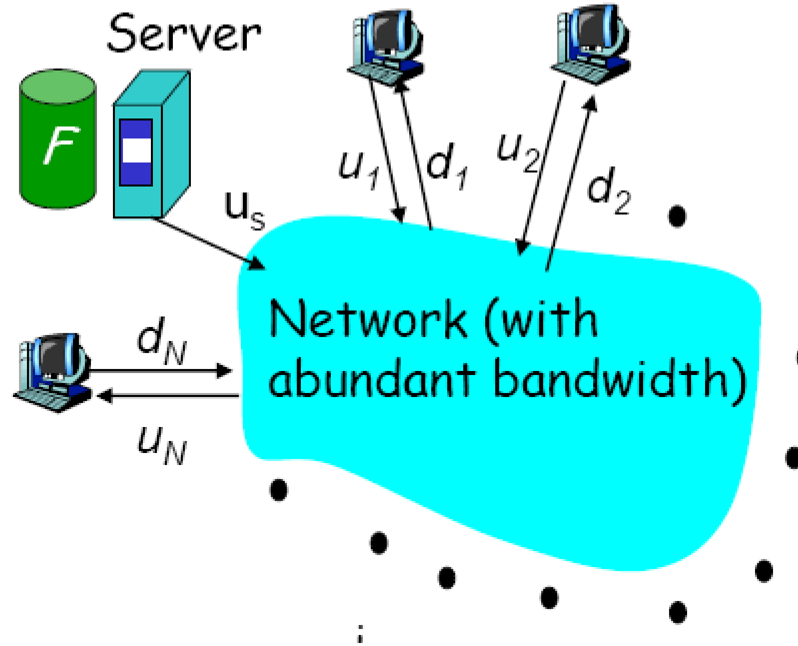


Time to distribute F to N clients using client/server approach $= d_{cs} = \max_i \{ NF/u_s, F/\min(d_i) \}$

Increase linearly in proportion to N (for large N)

File distribution time: P2P

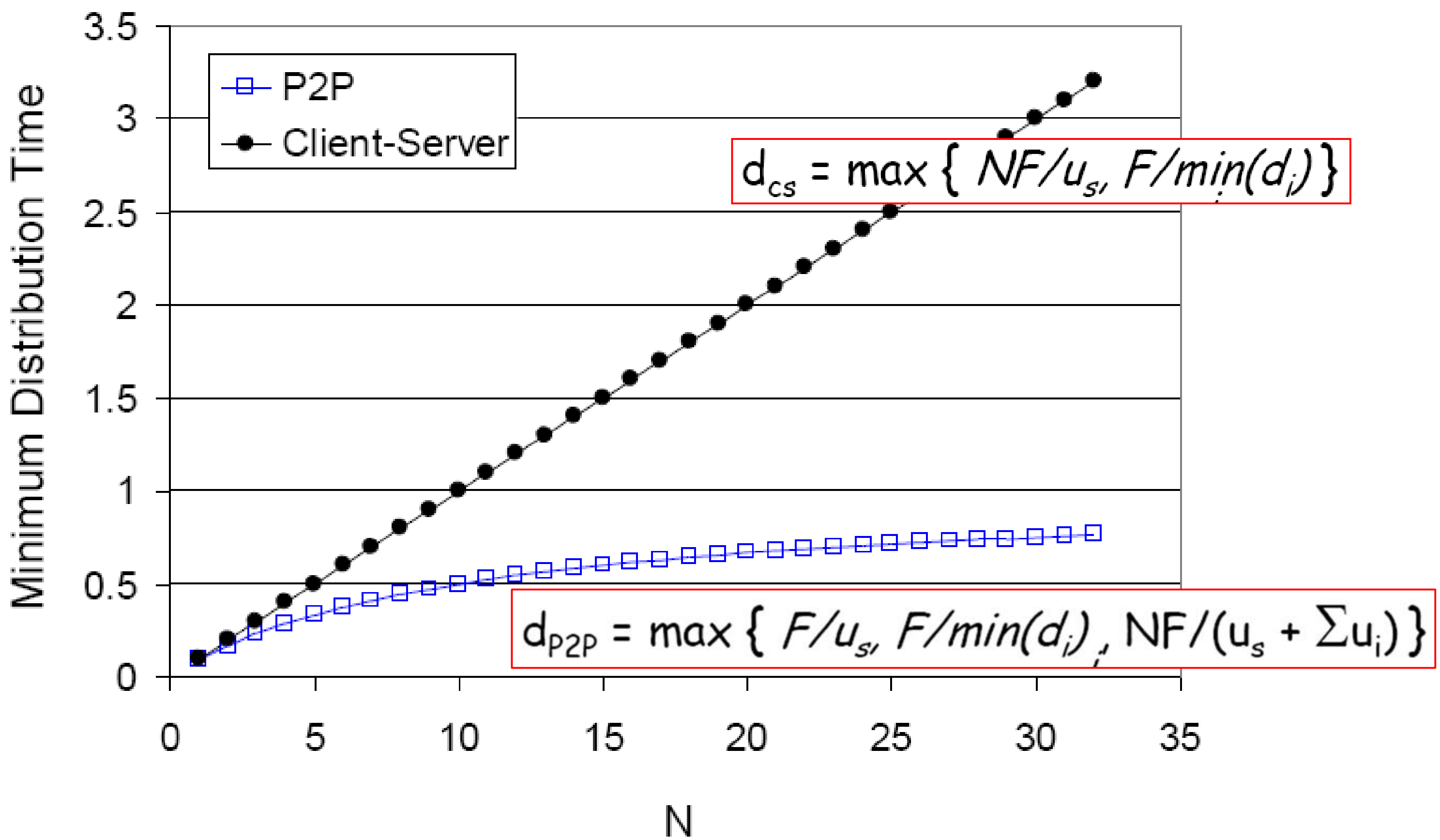
- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- $(N*F)$ bits must be uploaded
- fastest possible upload rate: $(u_s + \Sigma u_i)$



$$d_{P2P} = \max \left\{ F/u_s,\ F/\min_i(d_i),\ NF/(u_s + \Sigma u_i) \right\}$$

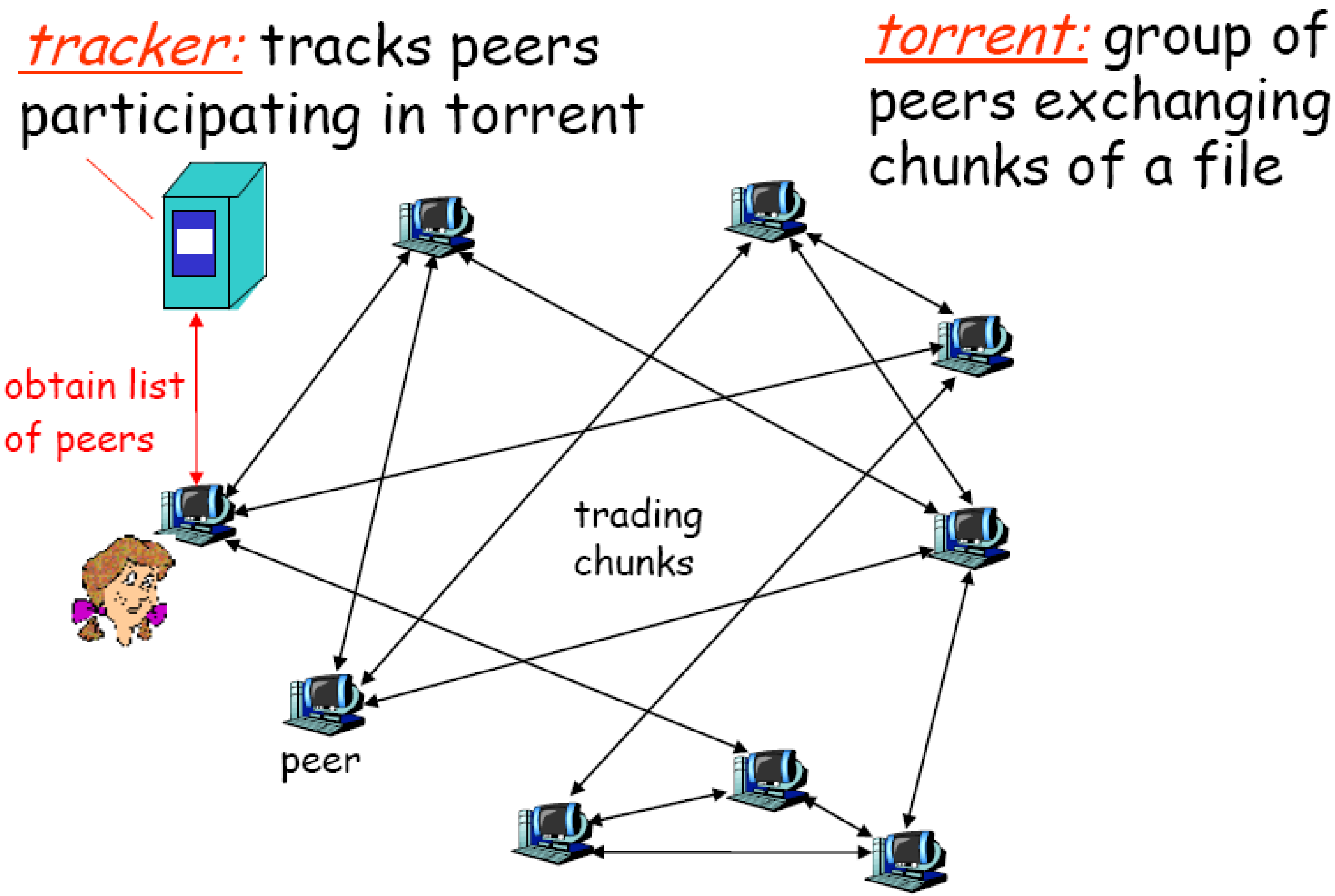
Server-client vs. P2P: example

- Client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$



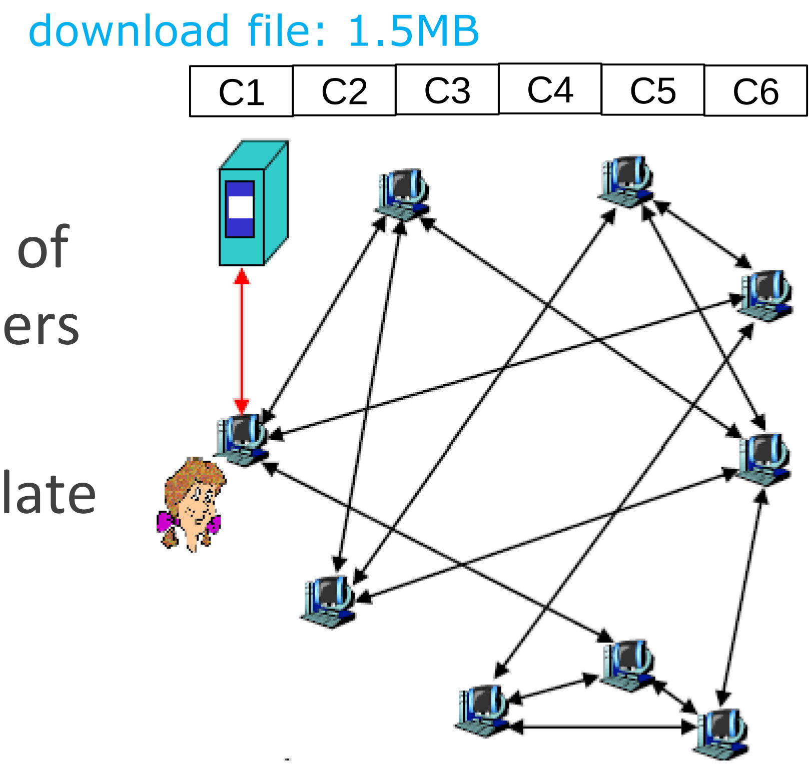
File distribution: BitTorrent

- P2P file distribution



File distribution: BitTorrent

- file divided into 256KB chunks
- peer joining torrent:
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
  - has no chunks, but will accumulate them over time
- while downloading, peer uploads chunks to other peers
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain



File distribution: BitTorrent

Pulling Chunks

- at any given time, different peers have different subsets of file chunks
- periodically, a peer asks each neighbor for list of chunks that they have
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

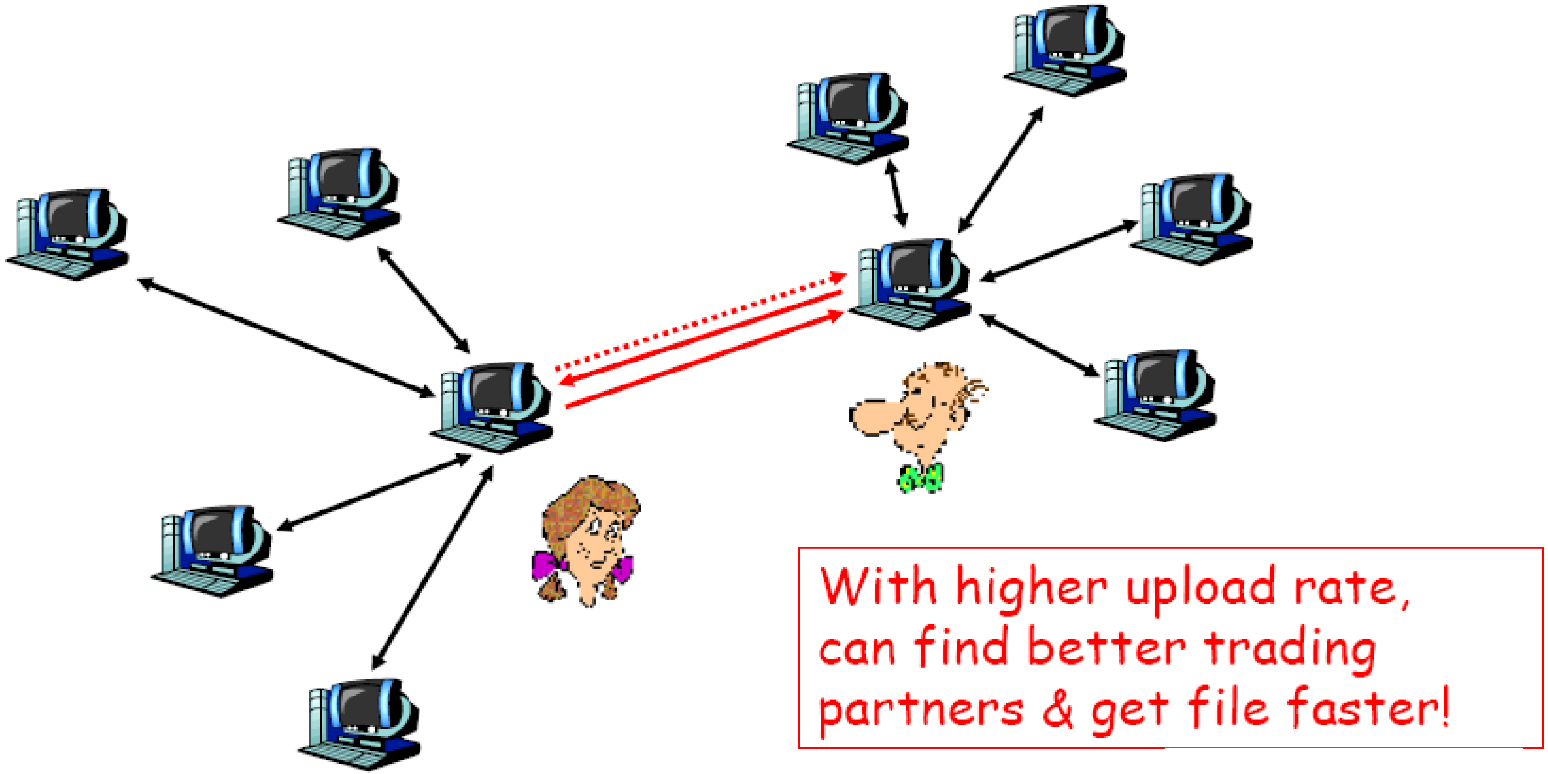
- Alice sends chunks to four neighbors currently sending her chunks at the highest rate
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks ("optimistically unchoke"): newly chosen peer may join top 4

BitTorrent: Tit-for-tat

(1) Alice "optimistically unchokes" Bob

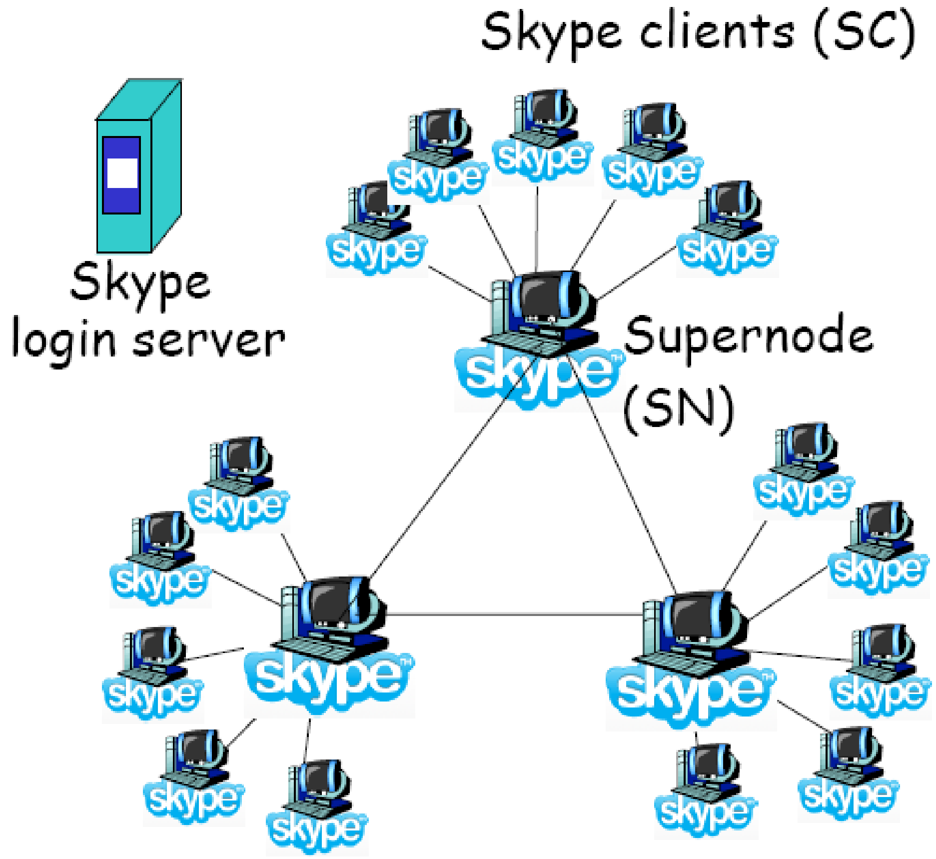
(2) Alice becomes one of Bob's top 4 providers; Bob reciprocates

(3) Bob "optimistically unchokes" Allice



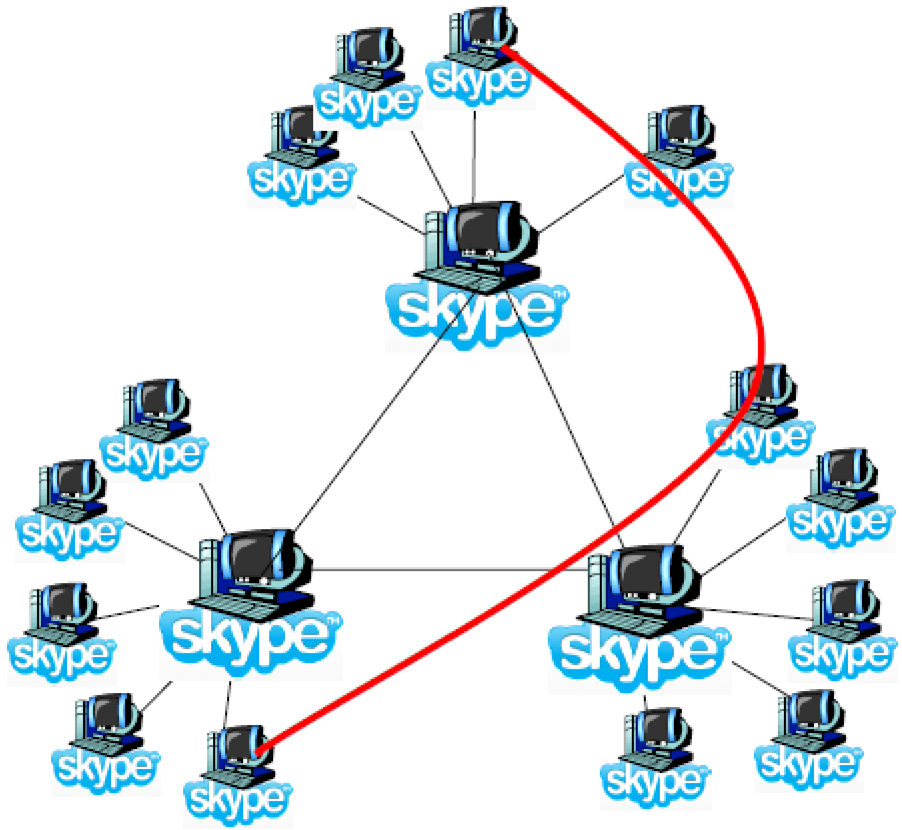
P2P Case study: Skype

- inherently P2P: VoIP service among pairs of users
- proprietary application-layer protocol
- Skype networks: hierarchical overlay with Super Nodes (SNs)
- mapping usernames to IP addresses; distributed over SNs



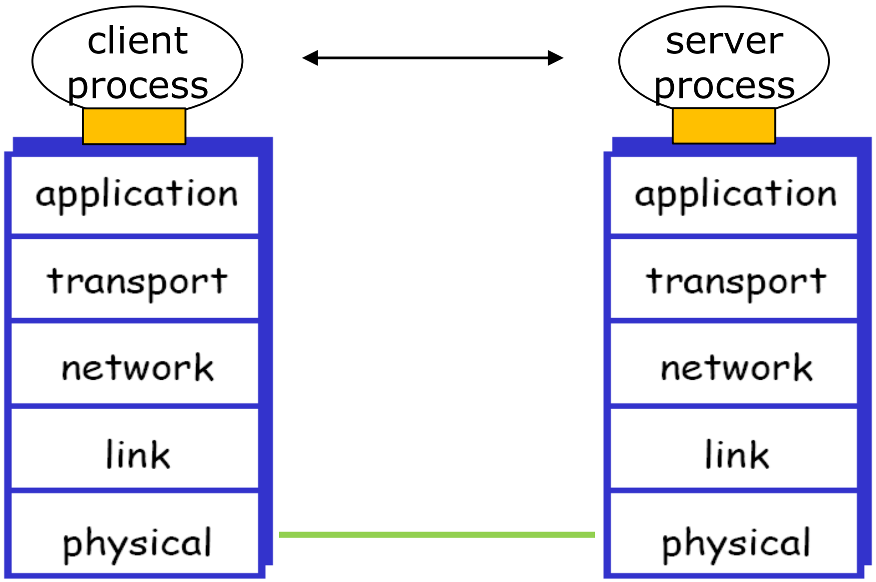
Peers as Relays

- Problem when both Alice and Bob are behind "NATs"
  - NAT prevents an outside peer from initiating a call to insider peer
- Solution:
  - Using Alice's and Bob's SNs, Relay with public IP address is chosen
  - Each peer initiates session with relay
  - Peers can communicate through NATs via relay



Chap 2

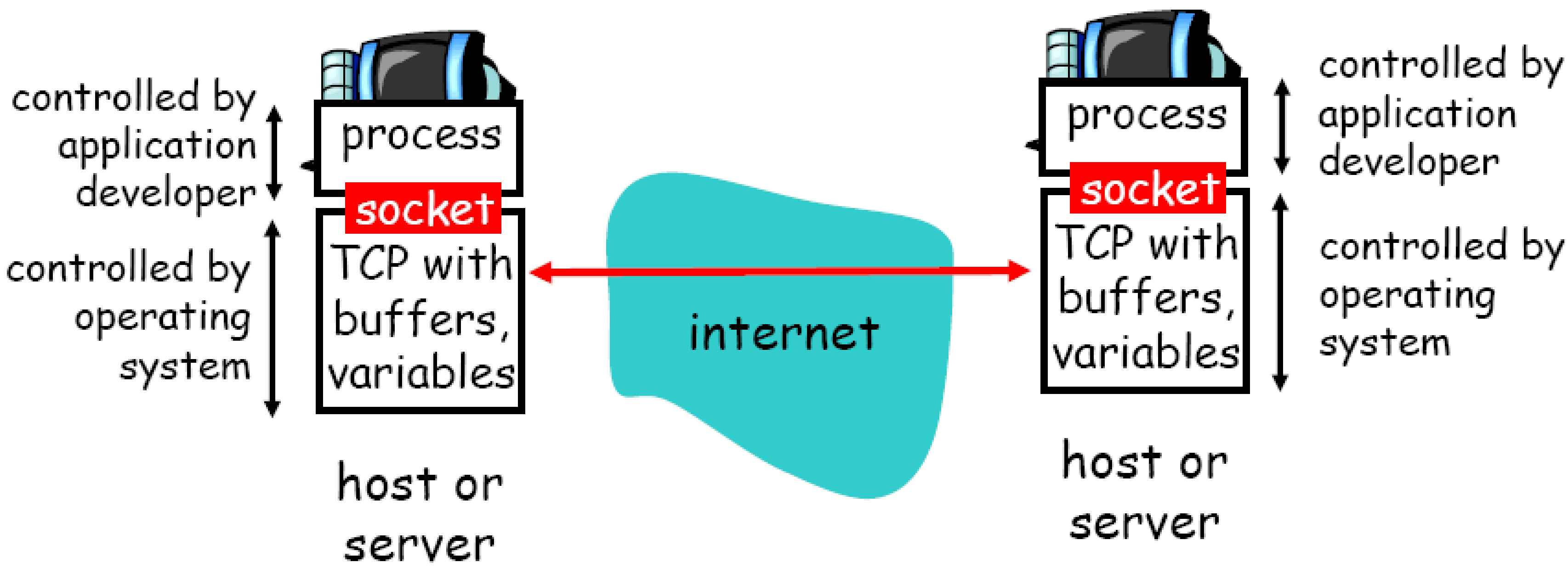
- Principles of network applications
- Web and HTTP
- FTP
- Electronic Mail : smtp, pop3, imap
- DNS
- P2P applications
- Socket programming



Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

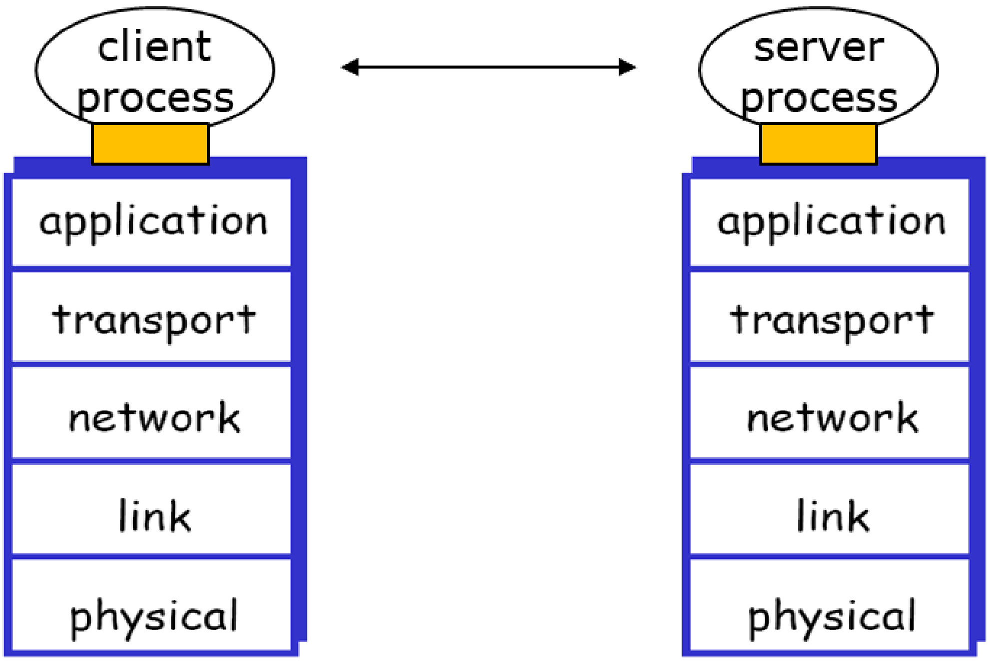
Socket contains information about two communication end-points: (local_IP, local_Port, remote_IP, remote_Port)



Socket-programming using TCP

Client contacts server first:

- server process must first be running
- server must have created a socket (listening socket) that receives client's connection request message



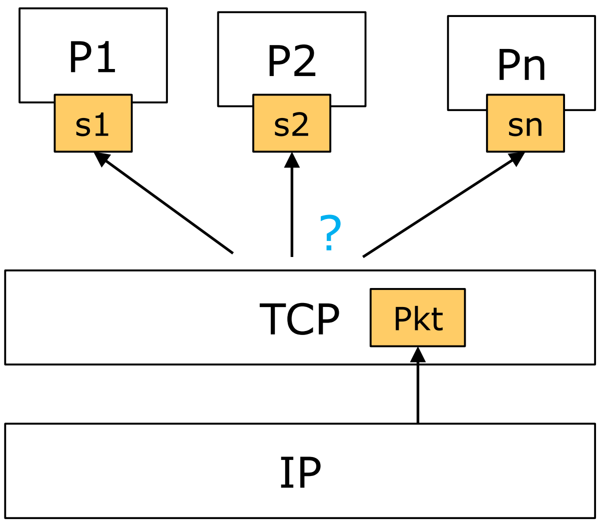
Socket-programming using TCP

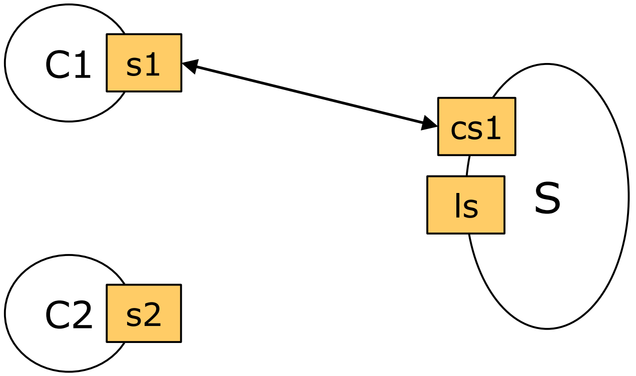
Client contacts server by:

- creating client-local TCP socket
- specifying IP address, port number of server process
- client TCP establishes connection to server TCP
- When contacted by client, server TCP creates a new socket (connected socket) for server process to communicate with client
  - allows server to talk with multiple clients while receiving request from a new client (concurrent server)

Socket-programming using TCP

socket: (loIP, loPort, reIP, rePort)
listening socket: (loIP, loPort, *, *)
connected socket:
(loIP, loPort, reIP, rePort)



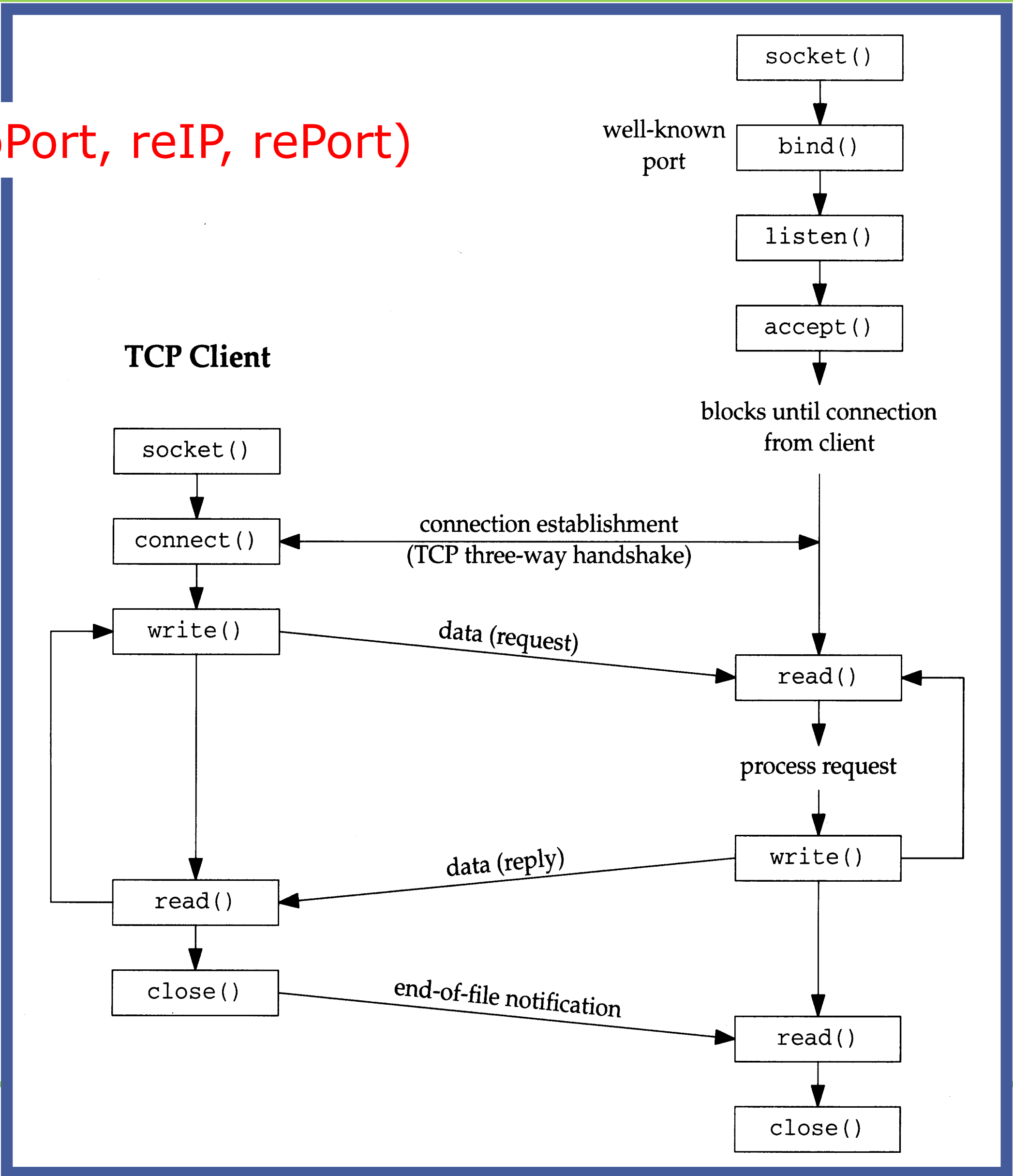


Pkt: (srcIP, dstIP, srcPort, dstPort)

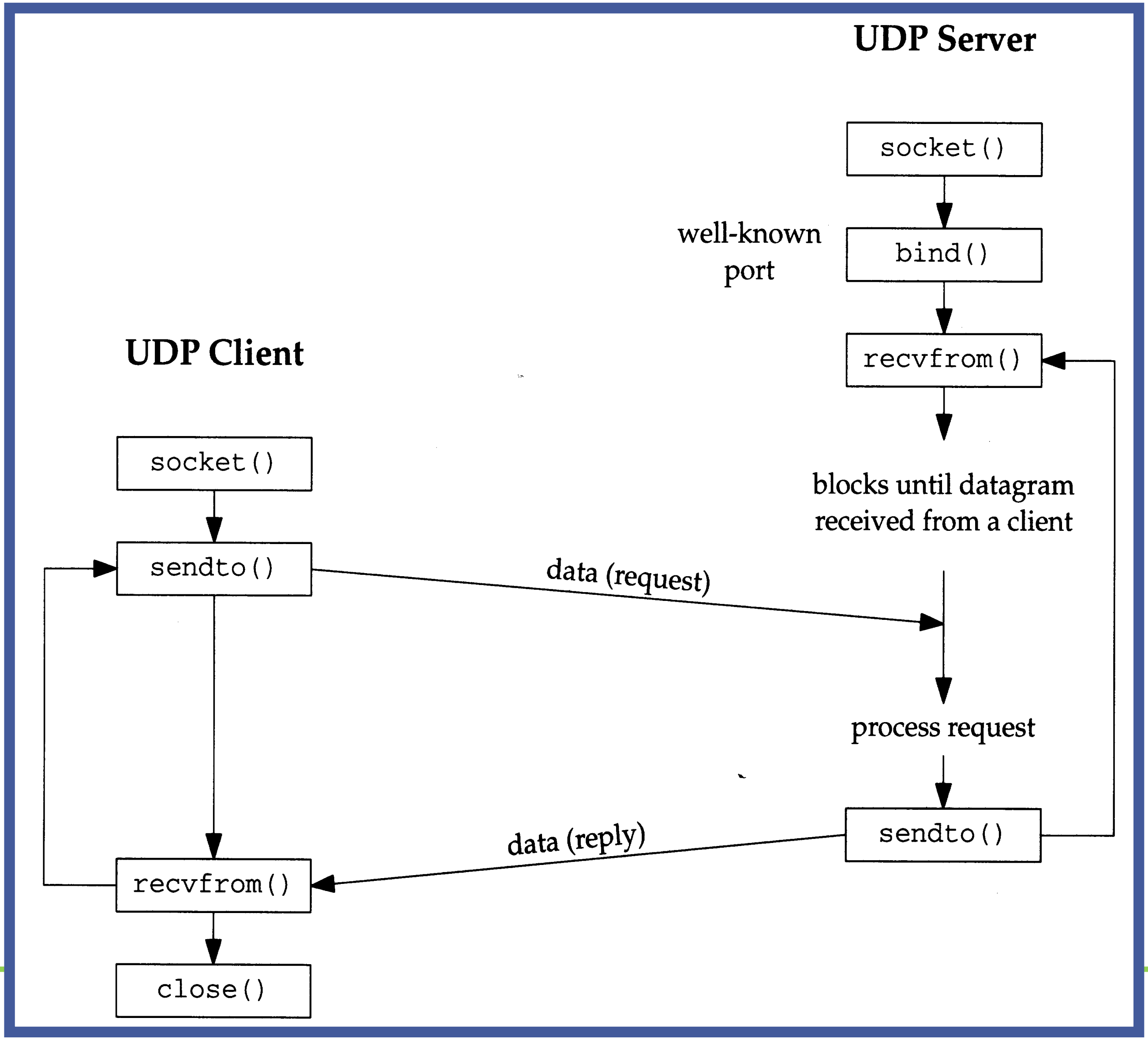
TCP sends to a process with a socket that satisfies
(dstIP=loIP && dstPort=loPort) && (srcIP=reIP && srcPort=rePort)

Client/server socket interaction: TCP

socket: (loIP, loPort, reIP, rePort)



Client/server socket interaction: UDP



Homework #3

- Socket API 조사
  - Windows socket (winsock) API
  - winsock socket 함수, 기능
  - 각 winsock socket 함수 사용 예제
  - 제출:
    - 한글 파일로 제출: 파일 이름 “hw3-학번-이름.hwp”