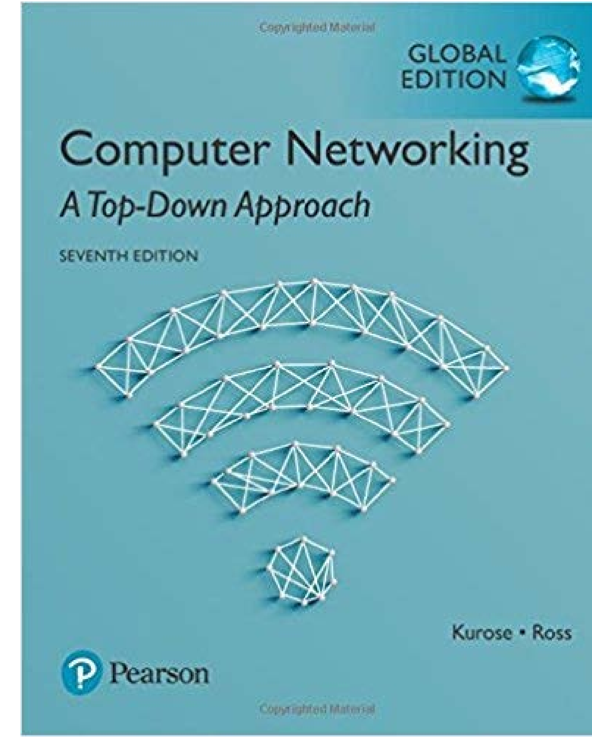
Chapter 2
Application Layer part 2



Computer Networking: A Top Down Approach

7th edition
Jim Kurose, Keith Ross
Pearson, 2017

School of Computing
Gachon Univ.
Joohyung Lee

Most of slides from J.F Kurose and K.W. Ross. And, some slides from Prof. Joon Yoo

Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

Electronic Mail

- Electronic Mail (E-mail) has been around since the beginning of the Internet – and still remains most important and utilized application
- E-mail is an 비동기 *asynchronous* communication
  - People can send out messages when it is convenient for them, no need to coordinate other people's schedules
  - Faster, easer to distribute, and inexpensive compared to postal mail

보내는 사람과 받는 사람이 서로 타이밍을 안 맞춰도 된다

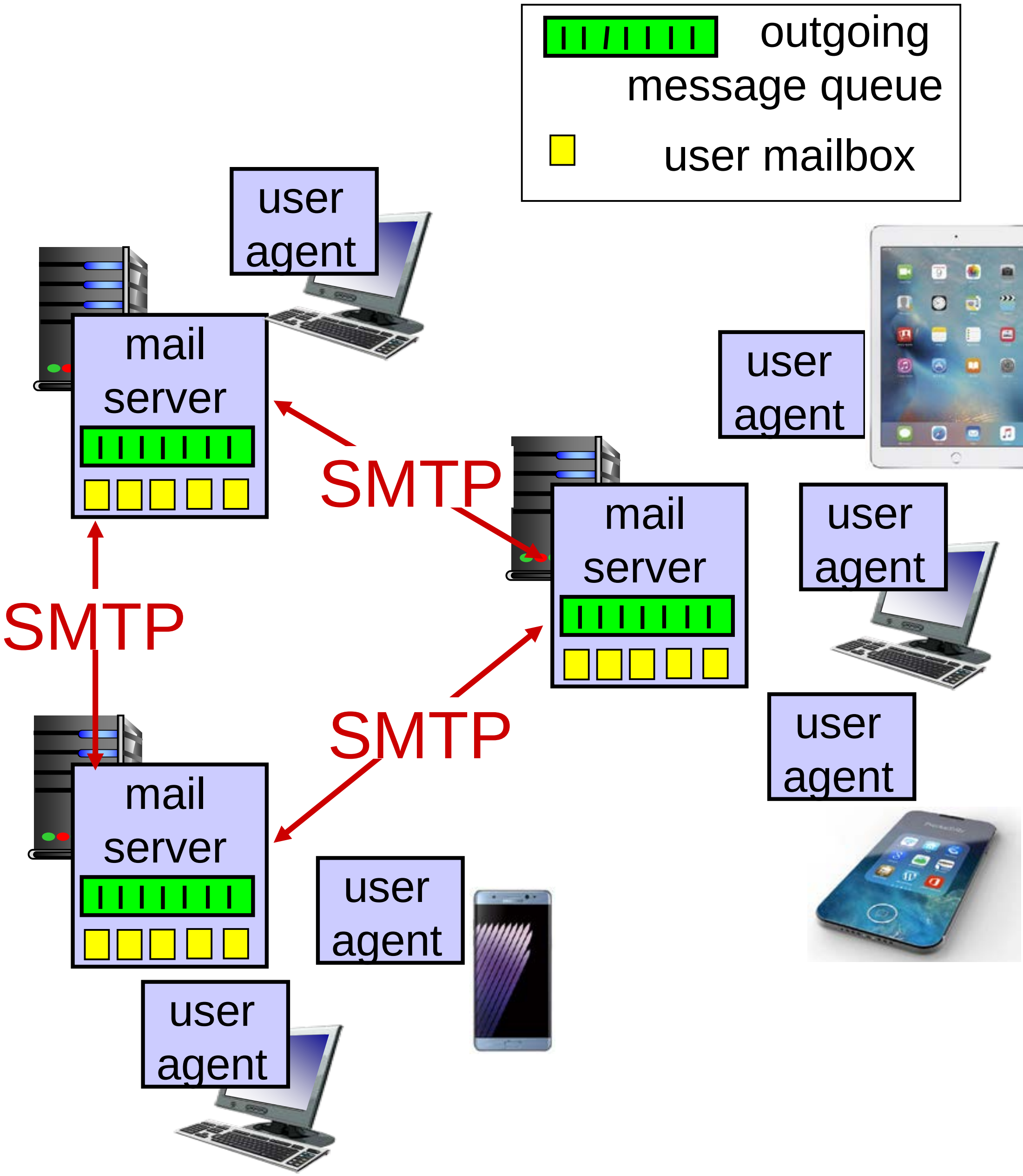
Electronic mail

Three major components:

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

User Agent

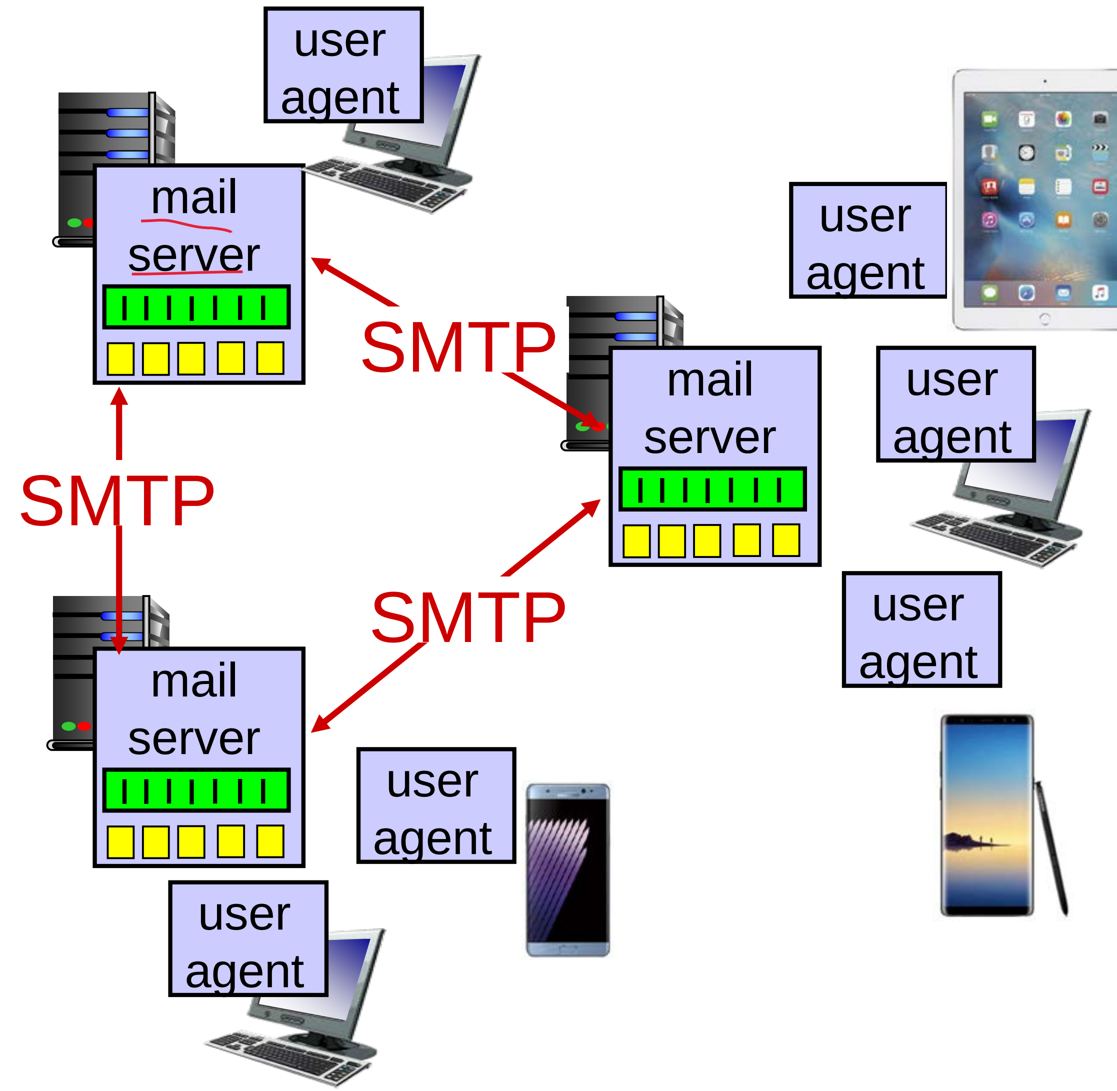
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Naver web client, Android google web client, iPhone mail client



Electronic mail: mail servers

mail servers:

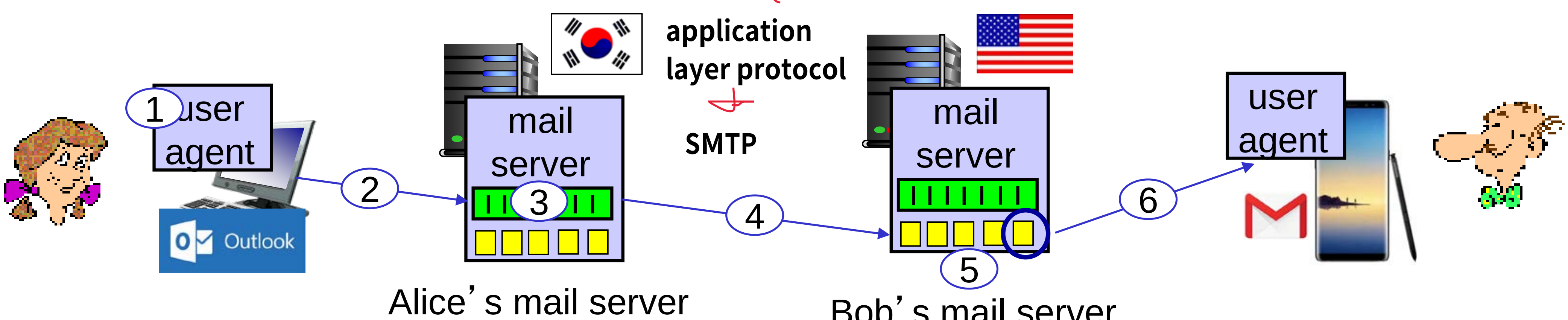
- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send/receive email messages
  - Simple Mail Transfer Protocol (SMTP)



Scenario: Alice sends message to Bob

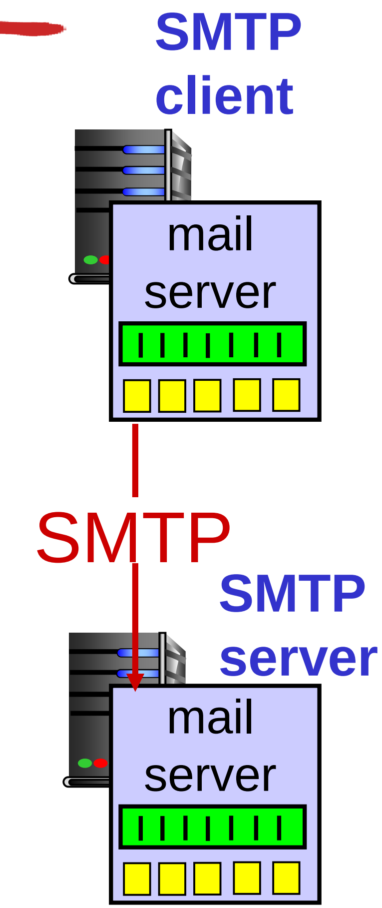
1) Alice uses UA to compose message “to” `bob@someschool.edu`
2) Alice’s UA sends message to her mail server; message placed in message queue
3) client side of SMTP opens TCP connection with Bob’s mail server
4) SMTP client sends Alice’s message over the TCP connection
5) Bob’s mail server places the message in Bob’s mailbox
6) Bob invokes his user agent to read message

TCP



Mail Server-to-server : SMTP

- direct transfer: sending server (SMTP client) to receiving server (SMTP server)
- uses TCP transport protocol to reliably transfer email message from client to server, port 25
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure



SMTP: Comparison with HTTP

clien <---http request--->server
http response

- HTTP: transfer files from Web server to Web client (browser)
  - **Pull protocol**: HTTP client pulls the information from server
  - TCP connection is initiated by machine that wants to receive
- SMTP: transfer files from one mail server to another mail server
  - **Push protocol**: sending mail server pushes the file to the receiving mail server
  - TCP connection is initiated by machine that wants to send
- SMTP requires message (header & body) to be in **7-bit ASCIIs**
  - Need to encode all binary multimedia data into ASCII before sending over SMTP (No such restriction in HTTP)

    decoding

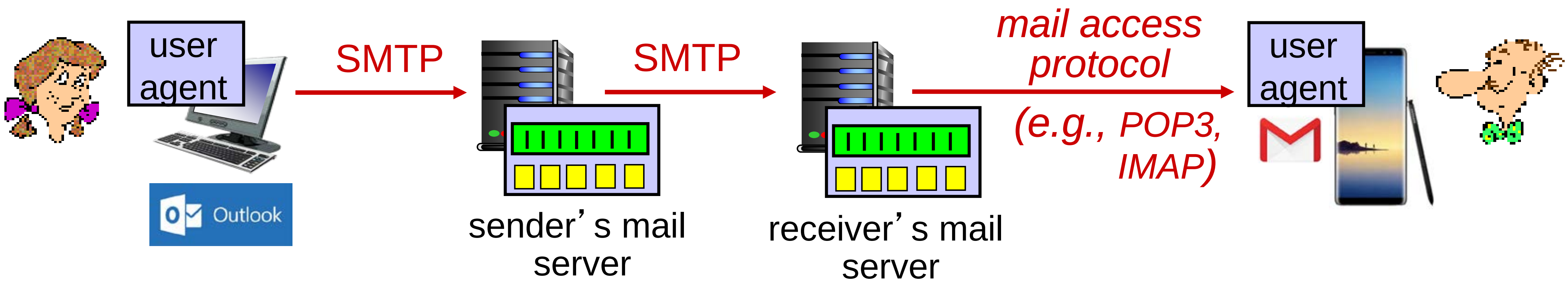
    - Image (sender) → 7-bit ASCII text (in SMTP msg) → Image (receiver)
  - This made sense in early 80s when transmission capacity was scarce, so all messages were text – but now it is archaic

    부족한, 드문

    구식의

A mail server ---mail---> B mail server

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. 보편적으로 사용

IMAP & Web mail

POP3

받는이

- Transfer mail from recipient's mail server to user agent (client)
- POP3 uses "download and delete" mode

IMAP

다운로드 하는 기능만

- keeps all messages in one place: at server – doesn't delete

Web-based E-mail

- User agent is Web browser and communicate with mailbox via HTTP (rather than SMTP, POP3, or IMAP)
- The mail server still uses SMTP to send/receive messages to/from other mail servers

push하는 구간에서는 SMTP 유저에게는 HTTP

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

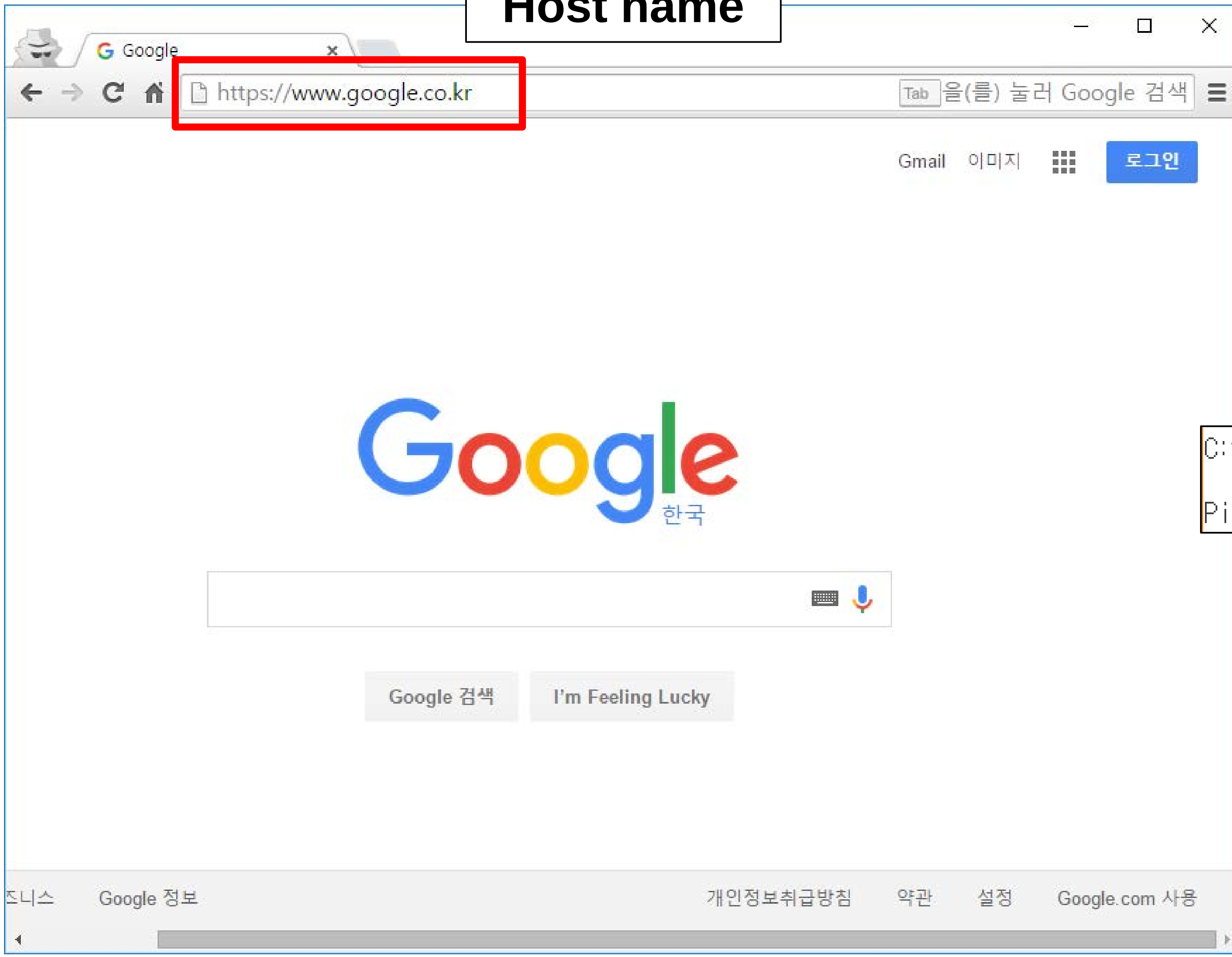
2.2 Web and HTTP

2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

Host name



But, Google server's actual address is **IP address**!

```
C:₩Users₩jyoo>ping google.co.kr

Ping google.co.kr [59.18.46.113] 32바이트 데이터 사용:
```

host를 구별하는 식별자는 ip 주소

IP address

The network ID identifies the network segment to which the host belongs.
The host ID identifies an individual host on some specific network segment.

- **IP address** is used to identify Internet hosts (Ch. 4)
  - 4-bytes (=32 bits), *hierarchical* structure **계층적 구조**
    - *e.g.*, 121.7.106.83
      - each period separates one byte (0~255)
      - try finding the current IP address of your machine - use commands such as ipconfig, ifconfig, …
  - It is an **ID** of the host – unique IP address for each Internet Host
  - It shows the **network** of the host - IP address is composed of **network id** & **host id**
    - *e.g.*, 121.7.106.83

network id is used to locate the network

host가 어느 네트워크에 속했는지

네트워크 내에서 어떤 앤지

가천대학교 Gachon University

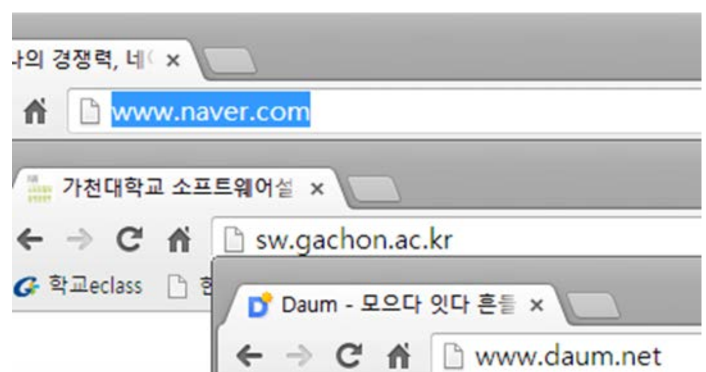
DNS: domain name system

people: many identifiers:
- SSN (≈주민번호), driver's license#, passport #, …

Internet hosts, routers:
- **IP address** or **host name** (e.g., www.yahoo.com)
- People prefer ____ and routers prefer _____

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

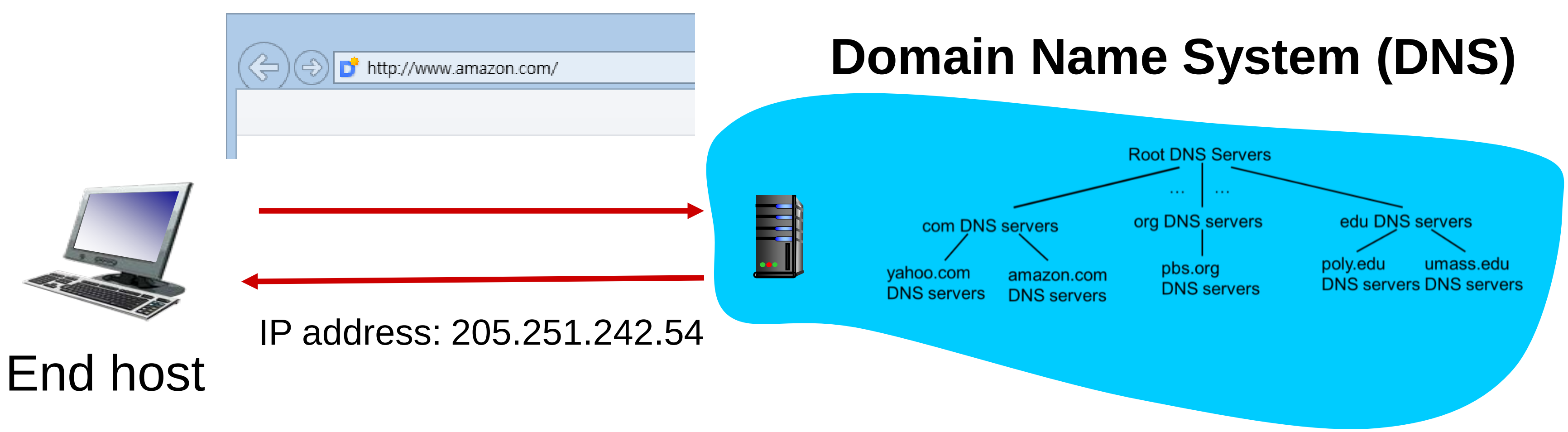
A: DNS



DNS: domain name system

Domain Name System (DNS):

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers employ DNS to *translate* host names into IP addresses



DNS: distributed vs. central

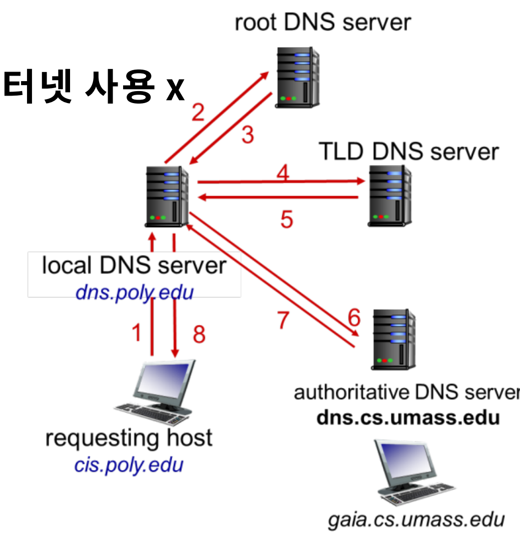
Single centralized server

- Ask one DNS server to translate name to IP address
- Very simple!

Then, why not centralize DNS?

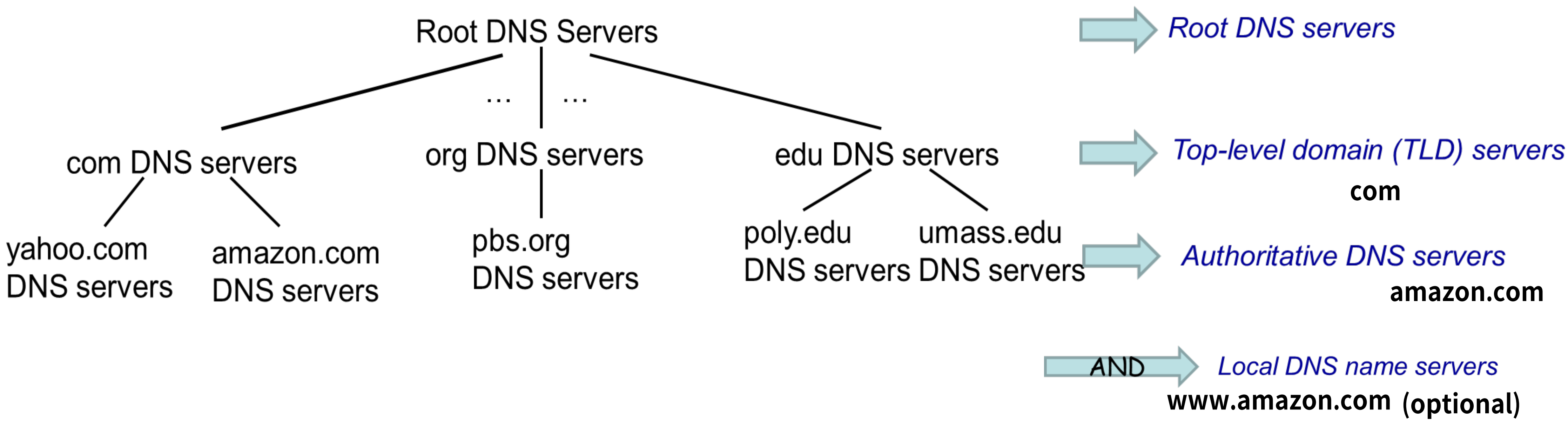
- single point of failure 서버가 1대 있는데 죽으면 전세계 사람들이 인터넷 사용 x
- traffic volume
- distant centralized database
- maintenance

vs.



A: doesn't scale!

DNS: a **distributed**, **hierarchical** database

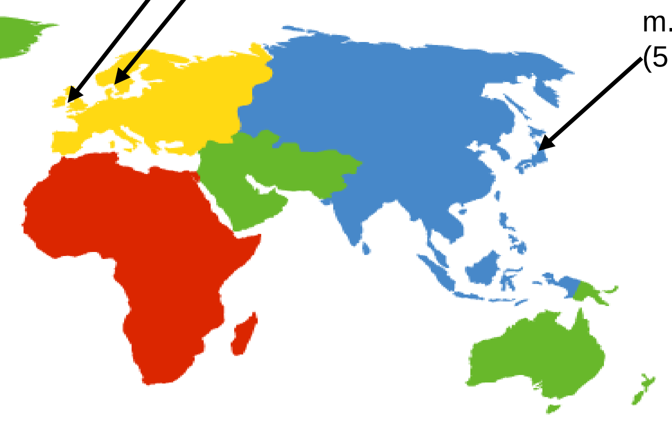


client wants IP address for `www.amazon.com`*; 1st approx:*

- client queries **root server** to find TLD (.com) DNS server TLD : .com들만 모아져 있는 서버
- client queries `.com` DNS server to get authoritative (`amazon.com`) DNS server 이제 amazon.com이라는 authoritative DNS server가 어디있는지 물어봄
- 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
  - returns mapping to local name server



13 root name "servers" worldwide

TLD, 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`

authoritative DNS servers:

ex) amazon.com

- organization's own DNS server(s), providing aut hostname to IP mappings for organization's nan
- can be maintained by organization or service pr

Local DNS name server

- each ISP (residential ISP, company, university) has one
  - also called “default name server”

ipconfig/all

```
IPv4 주소 . . . . . . . . . . . . : 121.135.107.150(기본 설정)
서브넷 마스크 . . . . . . . . . . : 255.255.255.0
임대 시작 날짜. . . . . . . . . . : 2014년 9월 21일 일요일 오후 1:59:47
임대 만료 날짜. . . . . . . . . . : 2014년 9월 21일 일요일 오후 11:11:20
기본 게이트웨이 . . . . . . . . . : 121.135.107.1
DHCP 서버 . . . . . . . . . . . . : 121.137.7.58
DHCPv6 IAID . . . . . . . . . . . : 199754650
DHCPv6 클라이언트 DUID. . . . . . : 00-01-00-01-18-39-90-22-E8-03-9A-66-87-44
DNS 서버. . . . . . . . . . . . . : 168.126.63.1
                                    168.126.63.2
Tcpip를 통한 NetBIOS. . . . . . . : 사용
```

- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS query에 보내기 전에 local DNS server에 query가 보내진다. ip과 host 주소의 관계는 바뀔 수도 있다. ip주소를 재할당하거나 한다면 local DNS 서버에 저장된 정보가 out of date가 되기도 한다. cacheing을 하면 싱크가 안맞을수도.

가장 최근에 검색했던 host name과 ip 주소간의 mapping을 caching 해서 가지고 있다.
web cache와 같은 proxy 역할을 한다. 바로 root DNS 서버에 물어볼 필요 없이.
가지고 있지 않으면 그 query를 hierarchy에 따라 전달하게 된다.

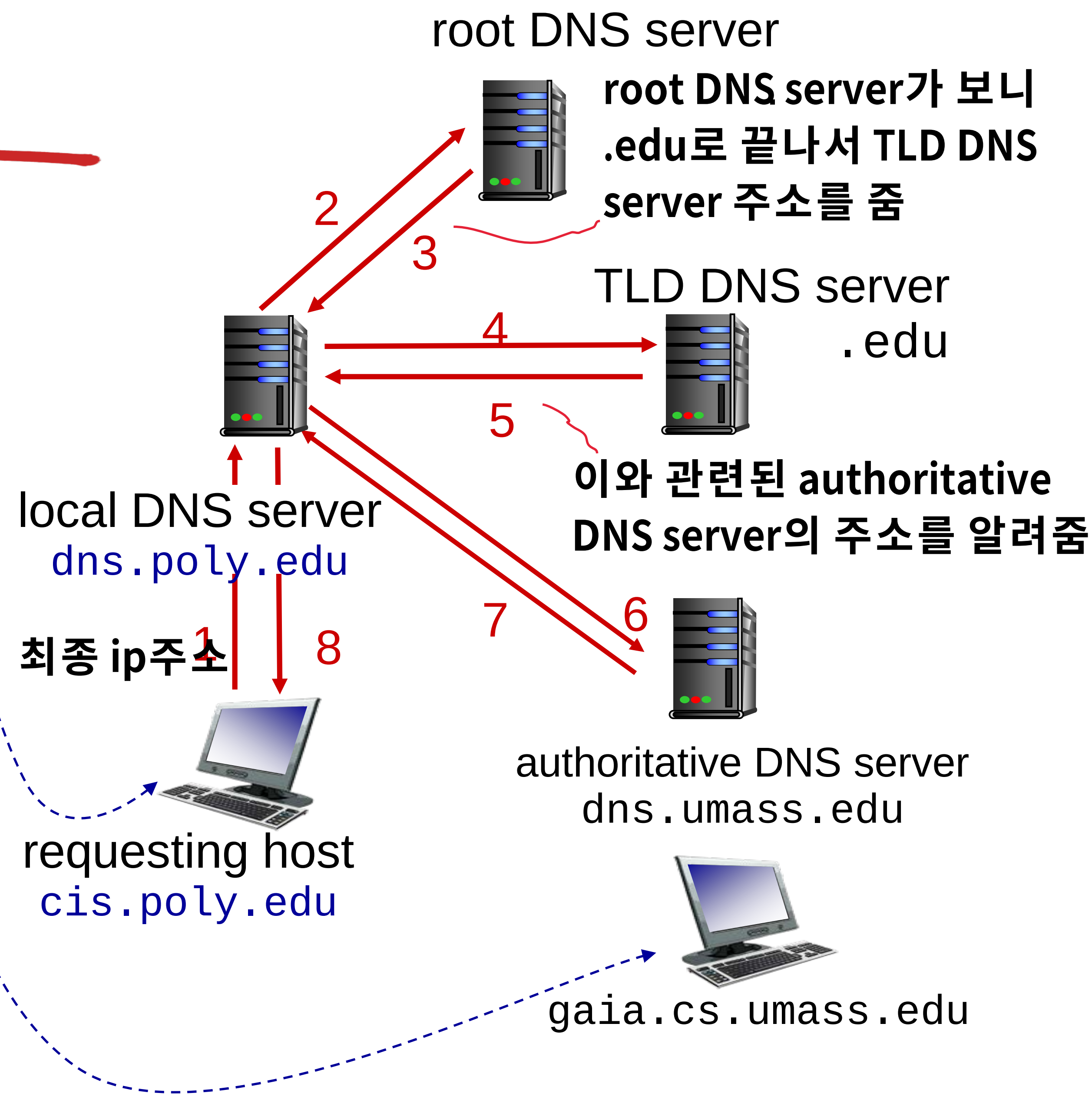
DNS name resolution example

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

반복하는

iterated query:

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

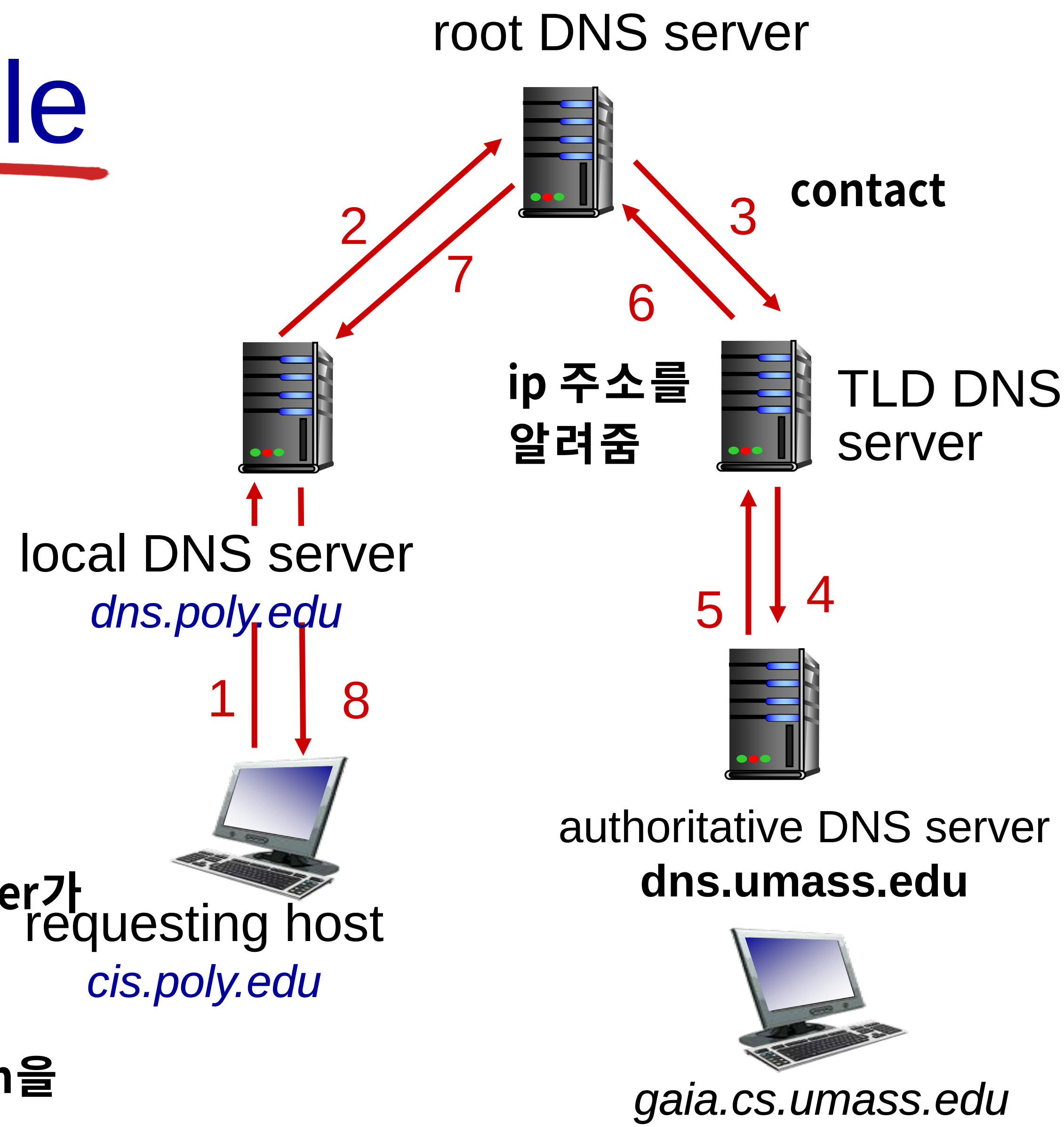


DNS name resolution example

recursive query:

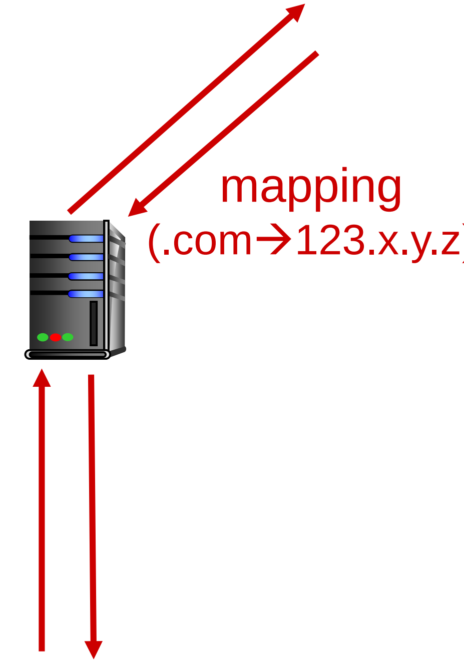
- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?

각각의 contact하려고 하는 name server가 얻어와서 전달까지 하는 역할을 하니 contact하는 name server의 burden을 나눠주면서 local DNS server의 burden을 완화시켜준다.



전세계에 13개밖에 없는 root DNS server에 상위 레벨이니 heavy load가 발생할 수 있다.

Questions



Is it appropriate that (any) name server *caches* mapping after it learns the mapping ?

If so, what is a possible problem occurred from DNS caching?

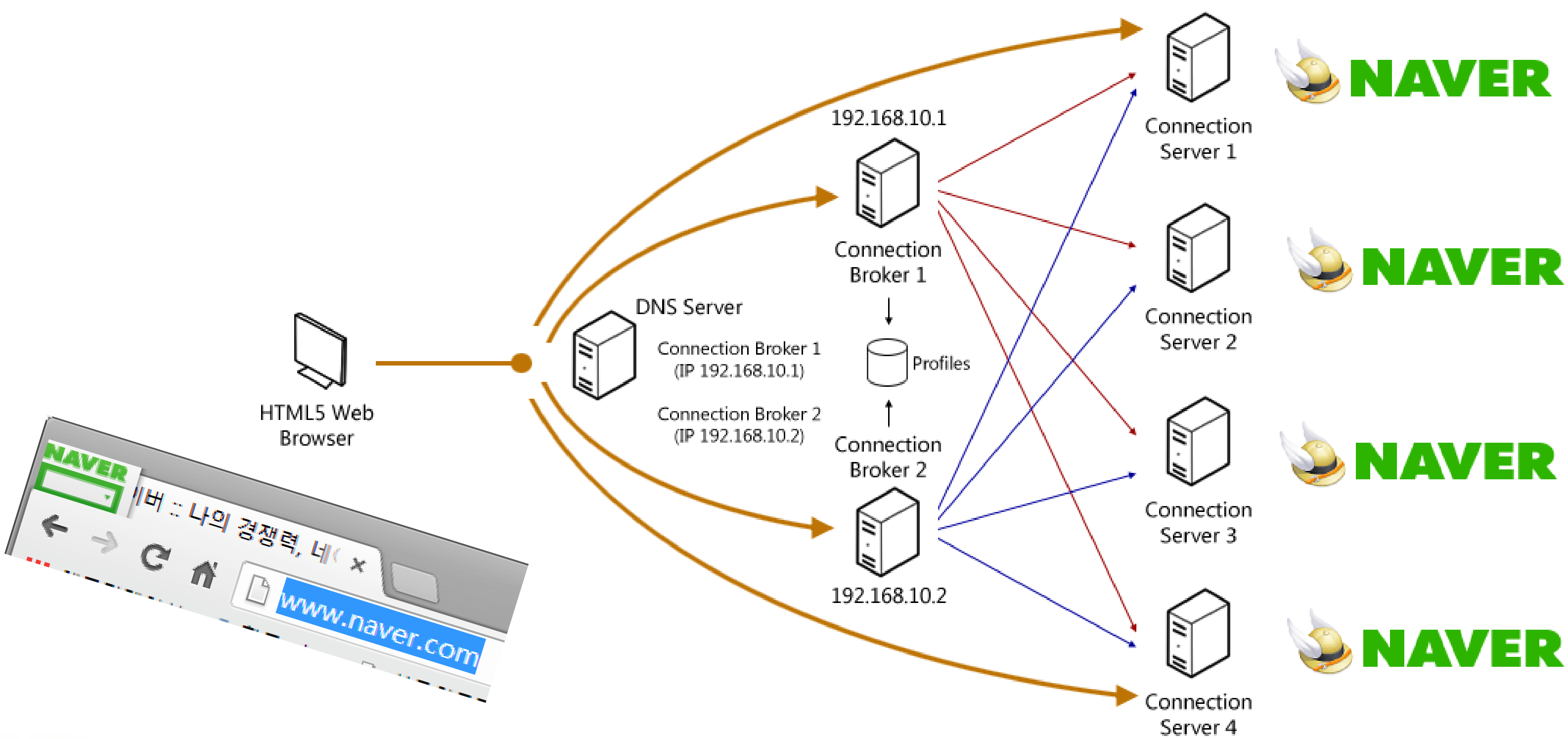
DNS: caching, updating records

- once (any) name server learns mapping, it *caches* mapping

  일정 시간 지난 것들은 날려버린다.
  - 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!) (계속 보관하게 되면)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire

DNS load distribution

- Help web servers do **load distribution (or load balancing)**
  - replicated Web servers: set of IP addresses for one canonical name (정식 이름)



Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS

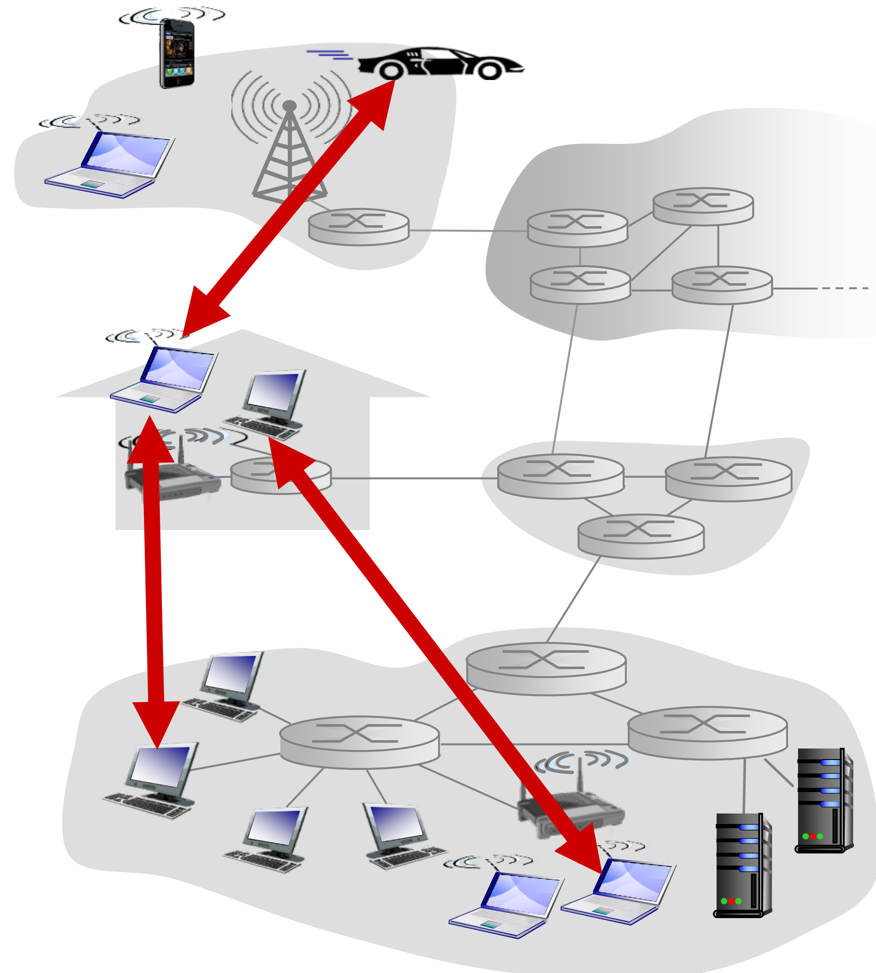
2.5 P2P applications

Pure P2P architecture

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

examples:

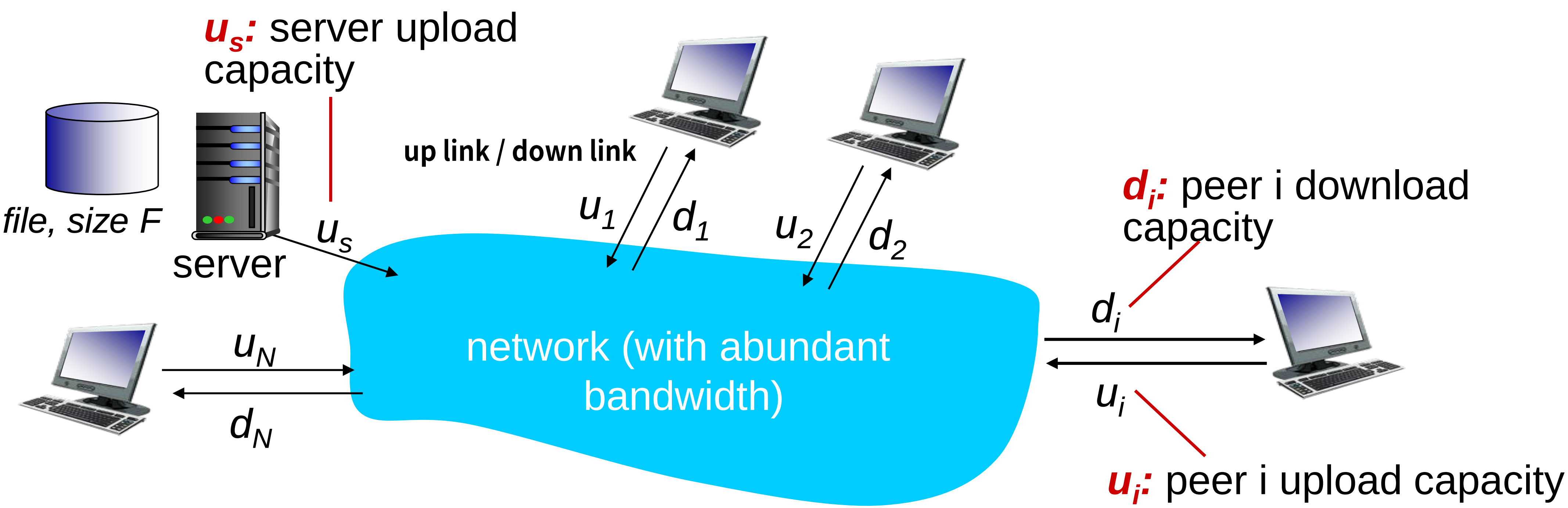
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)
  voice ip



File distribution: client-server vs P2P

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

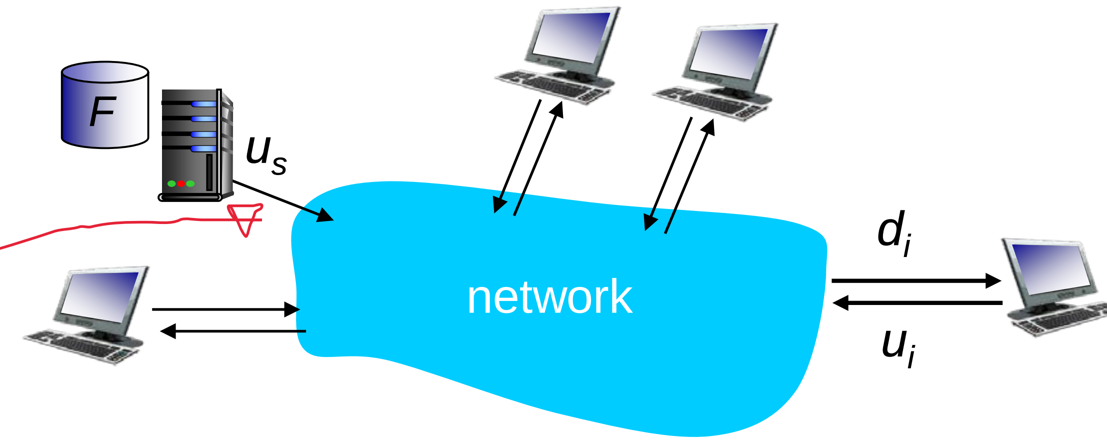
- Server upload capacity & peer upload/download capacity are limited resources



가천대학교 Gachon University

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}



꼴찌가 받았으면 다른 클라이언트도 다 받았으니.

배포 받는 유저 수가 10->10000이 되면 서버는 클라이언트가 받으려는 파일을 네트워크로 흘려보내야 하니까 시간이 커짐

서버가 파일을 배포시키는 시간

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

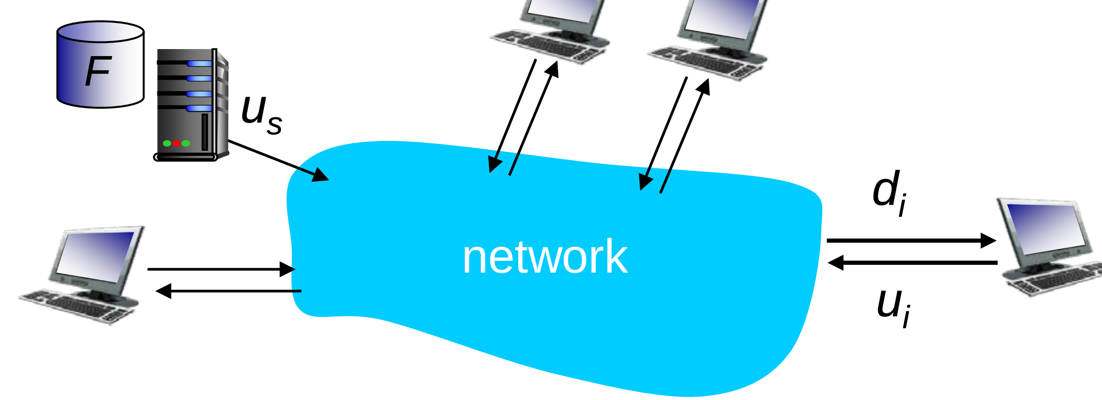
$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

클라이언트가 전달과 동시에 다운 받는 시간.

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 (limting max download rate) is $u_s + \Sigma u_i$



파일을 배포하는 주체가 서버만 있는게 아니라 파일을 받기만 하면 업로드 하는 주체가 된다.

time to distribute F to N clients using P2P approach

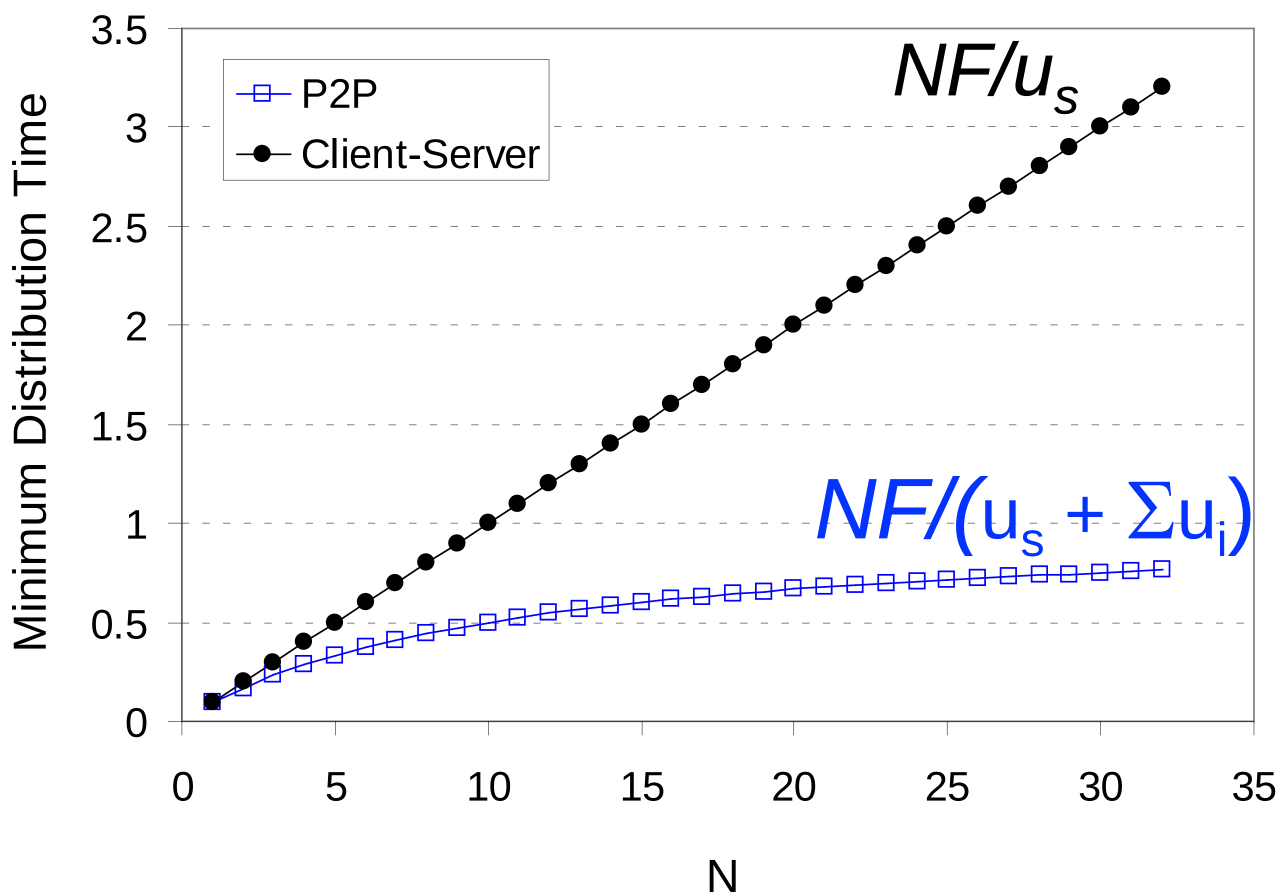
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

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



Self Scalability

P2P file distribution: BitTorrent

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

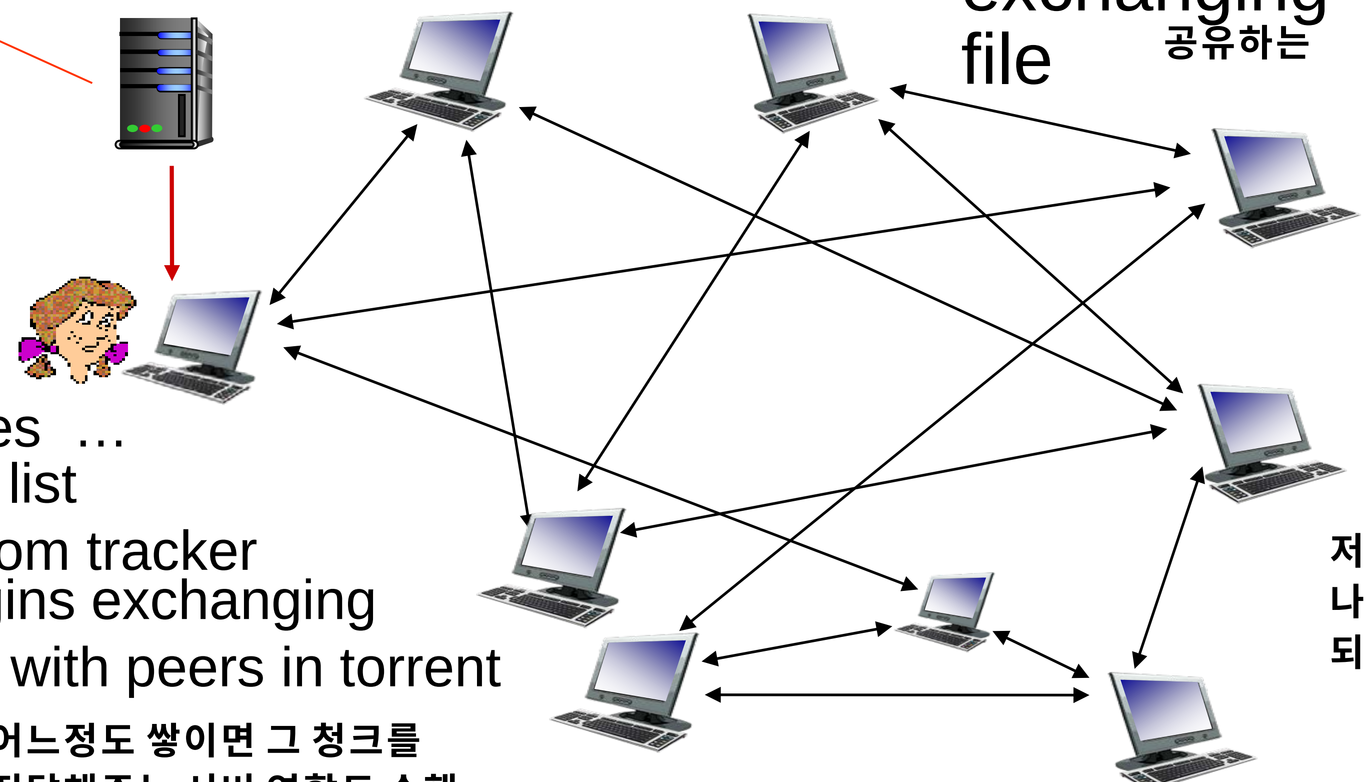
추적/관리

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file

공유하는

청크를 다 받은
peer 들은 토렌트로부터
빠져나감



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

저작권은 배포자가 잘못인데
나도 모르고 배포한 서버가
되어서 범죄자가 됨.

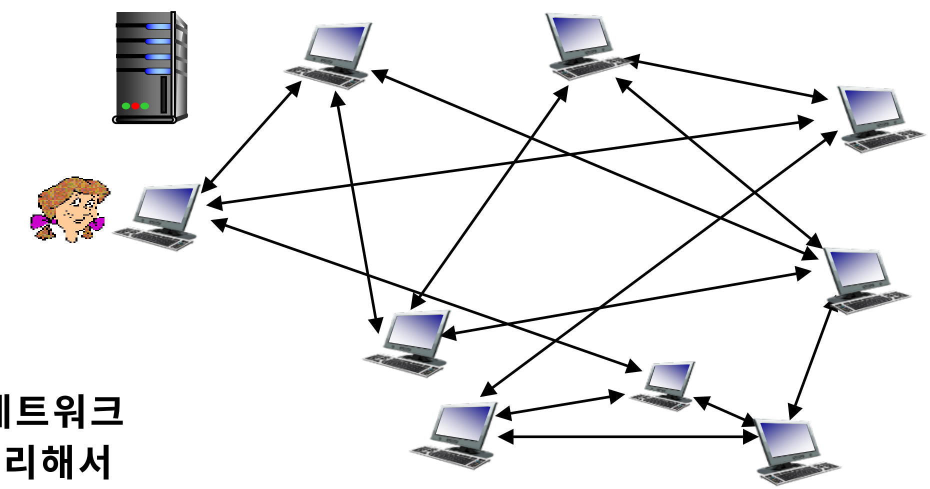
어느정도 쌓이면 그 청크를
전달해주는 서버 역할도 수행.

P2P file distribution: BitTorrent

BitTorrent

Read Chapter 2.5 (page 172-175) in textbook!

- 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”)



peer 들이 나갈 수 있고 네트워크
가 빠른 곳에서 받는게 유리해서
peer들이 바뀜.

- 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

이타적으로

BitTorrent: requesting, sending file chunks

p2p구조

빨리 받으려면 전략적으로 받는게 유리.

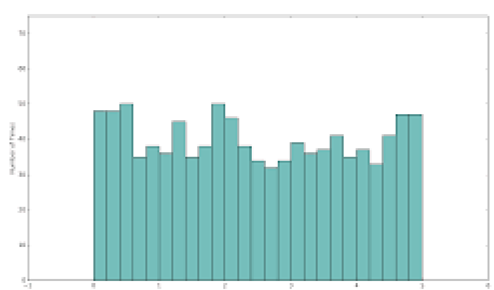
requesting chunks:

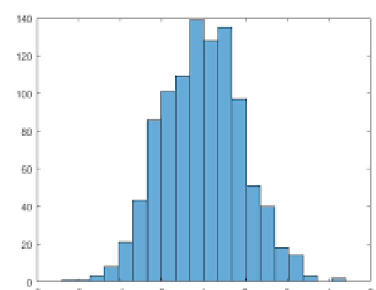
- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, *rarest first*

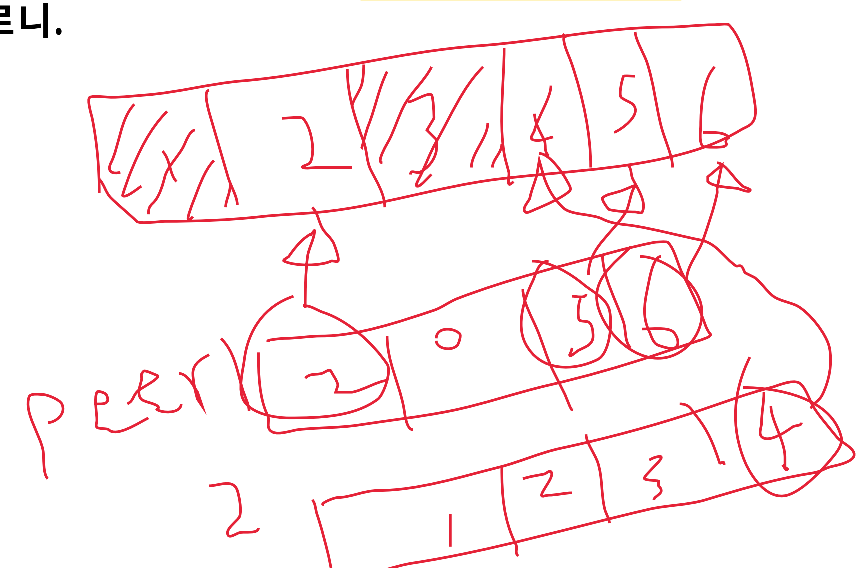
언제 나갈지 모르니.

1번 chunk : 2명
2번 chunk : 5명
3번 chunk : 10명

peer가 10명



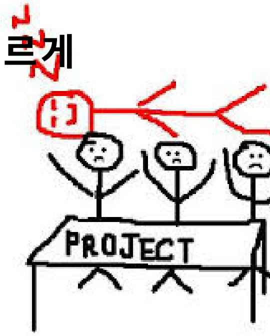




BitTorrent: requesting, sending file chunks

sending chunks: tit-for-tat

나한테 가장 빠른 속도로
보낸 애들한테 = 나한테 빠르게
줬으니 통신이 빠름.



- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are *choked* by Alice (do not receive chunks from her) = chunk를 받지 못한다.
  - re-evaluate top 4 every10 secs
    계속 평가해서

나로부터 멀리 떨어지는 노드들 중에서 통신 속도가 잘 나오는
애들을 놓칠 수 있어서.

- every 30 secs: randomly select another peer, starts sending chunks
  - "*optimistically unchoke*" this peer
  - newly chosen peer may join top 4

운영하다보면 가장 큰 이슈는 chunk를 받자 마자 바로 떠나는 문제.
다운하려는 링크는 많지만 업링크 하는 곳은 적음.
이를 해결하기 위해? 연구중.

BitTorrent: tit-for-tat

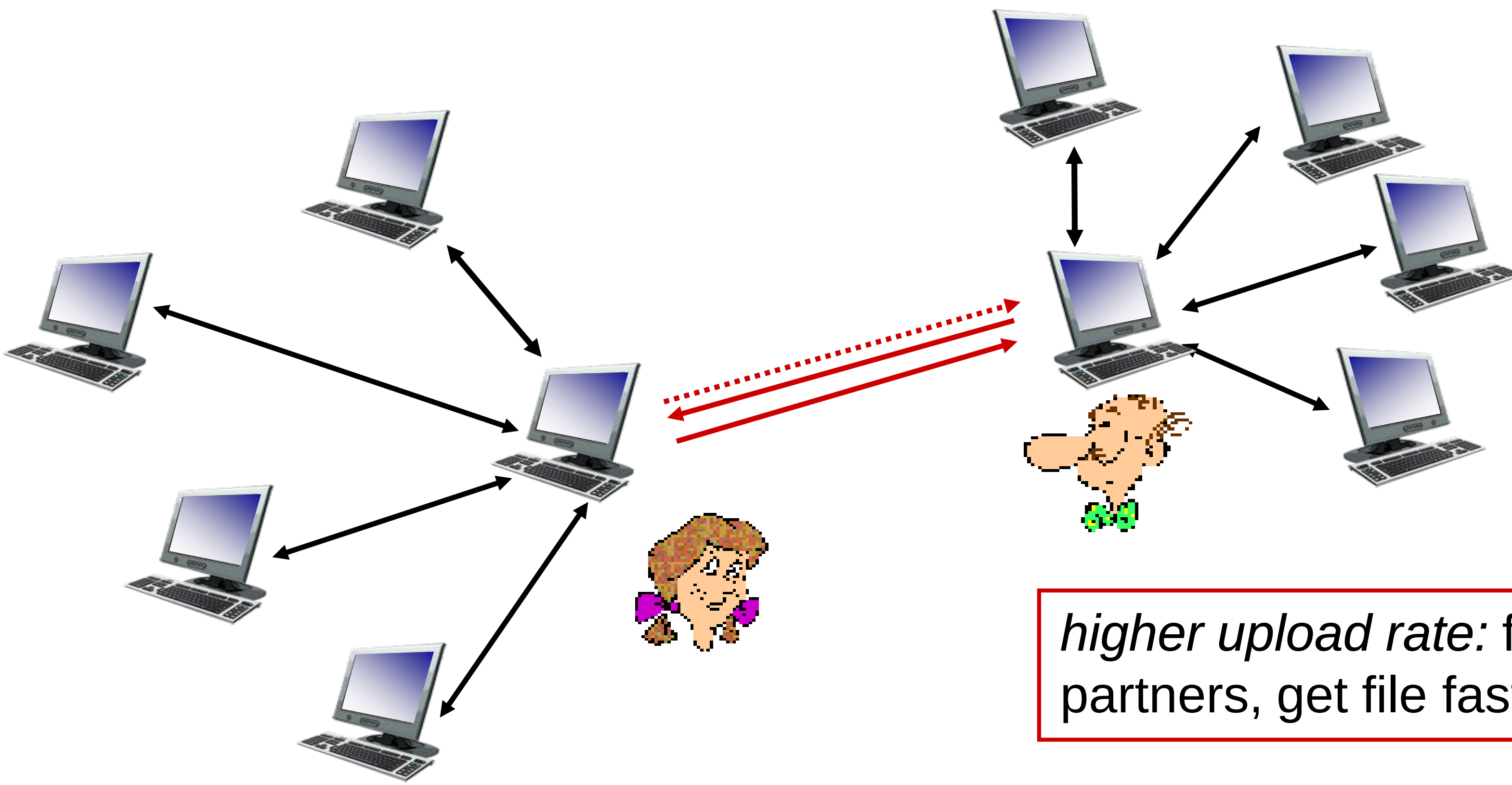
만약에 속도가 잘 안나오면 교역 안하면 됨.
만약에 나오면 top four provider.

(1) Alice "optimistically unchokes" Bob

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

응답하다

(3) Bob becomes one of Alice's top-four providers



The rest?
Choke!

higher upload rate: find better trading partners, get file faster !

Chapter 2: outline

traffic의 상당 부분을 video streaming이 차지

Video Streaming and CDNs: context

- video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1B YouTube users, ~75M Netflix users
- challenge: scale - how to reach ~1B (10억) users? scalability 이슈
  - single mega-video server won't work (why?)
- challenge: heterogeneity 이기종성
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor) 전부 다른 환경에 처함(이더넷, 폰)
- *solution:* distributed, application-level infrastructure

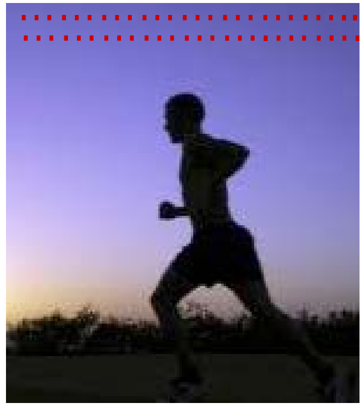
Multimedia: video

- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

픽셀값을 다 전달하는게 아니라
비슷한 색깔을 모아서
색깔 값이랑 얼마나반복되는지

짧은 시간의 이미지는 비슷함. 미세한 변화만 전달.

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)



frame *i*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

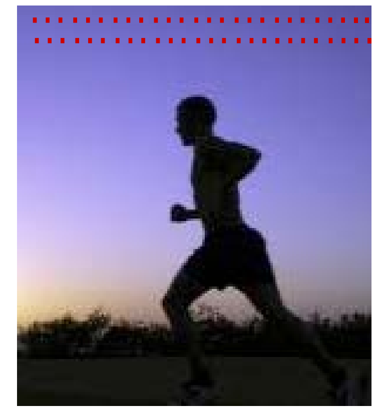
frame *i+1*

Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)

점점 좋아지고 있다. 따라서 네트워크(bandwitdth)를 적게 쓴다.

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)



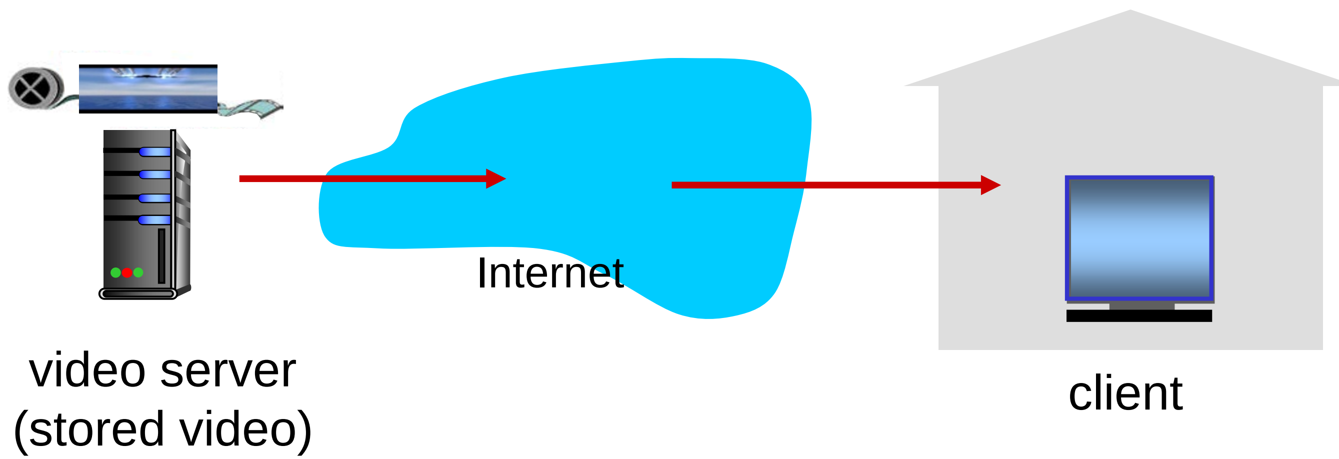
frame *i*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

frame *i+1*

Streaming stored video:

simple scenario:



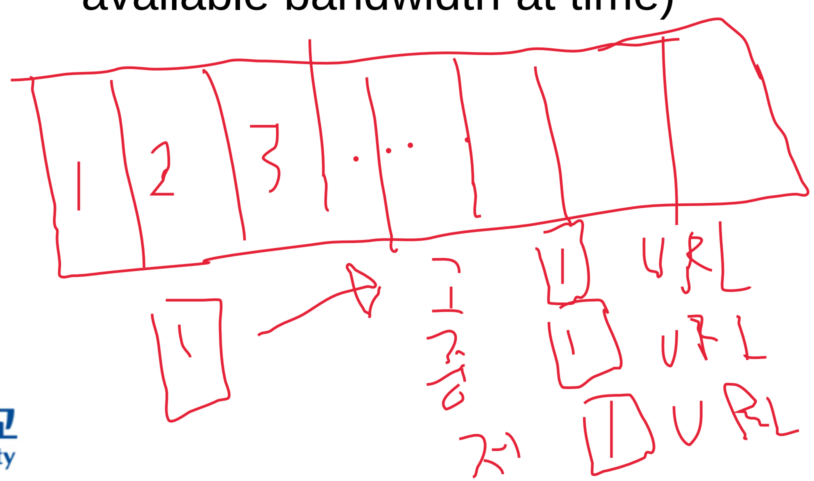
Streaming multimedia: DASH

간단한 아이디어임에도 성능이 좋아서 자주 씀.

- *DASH: Dynamic, Adaptive Streaming over HTTP*
- *server:*
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file:* provides URLs for different chunks

    나타내다
- *client:*
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time

    찾아보다
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at time)



manifest file을 만들어 url을 가지고 있는다.
bandwidth 측정.
매 시간마다 chunk 요청함.(3번 청크 받으면 4번 청크 요청)
측정한 bandwidth가 끊임 없이 재생할 만한 사이즈의 encoding version 선택.

Streaming multimedia: DASH

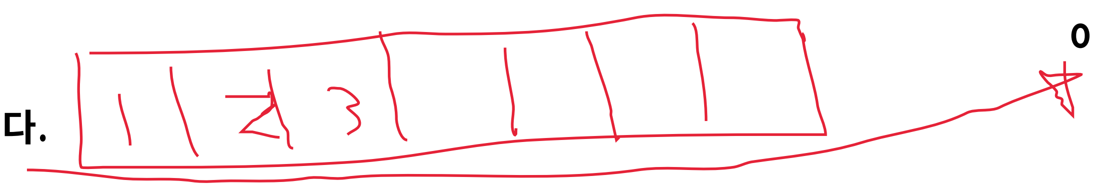
- *DASH: Dynamic, Adaptive Streaming over HTTP*
- *"intelligence"* at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)

client

3초만 보고 그만뒀는데 buffer는 이미 다 채워있으면 안되니 어느정도 채워야하나를 연구중.

서버는 청크를 여러 인코딩 버전별로 저장하고 클라이언트에게 제공하는거밖에 없음.
클라이언트는 현재 네트워크에 적당한 url을 고름.
어떤 서버에 있는 청크를 요청할지, 언제 요청할지를 결정하는 주체는 client이다.

chunk를 미리 받아둔다.
버퍼가 비어버리면 비디오가 끊긴다.



buffer

이를 이용한 것은 client가 함

Content distribution networks

- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

- *option 1:* single, large "mega-server"
  - single point of failure
  - point of network congestion 과부하
  - long path to distant clients 서버가 하나만 있기 때문에 여러 지역은 멀어질수도.
  - multiple copies of video sent over outgoing link

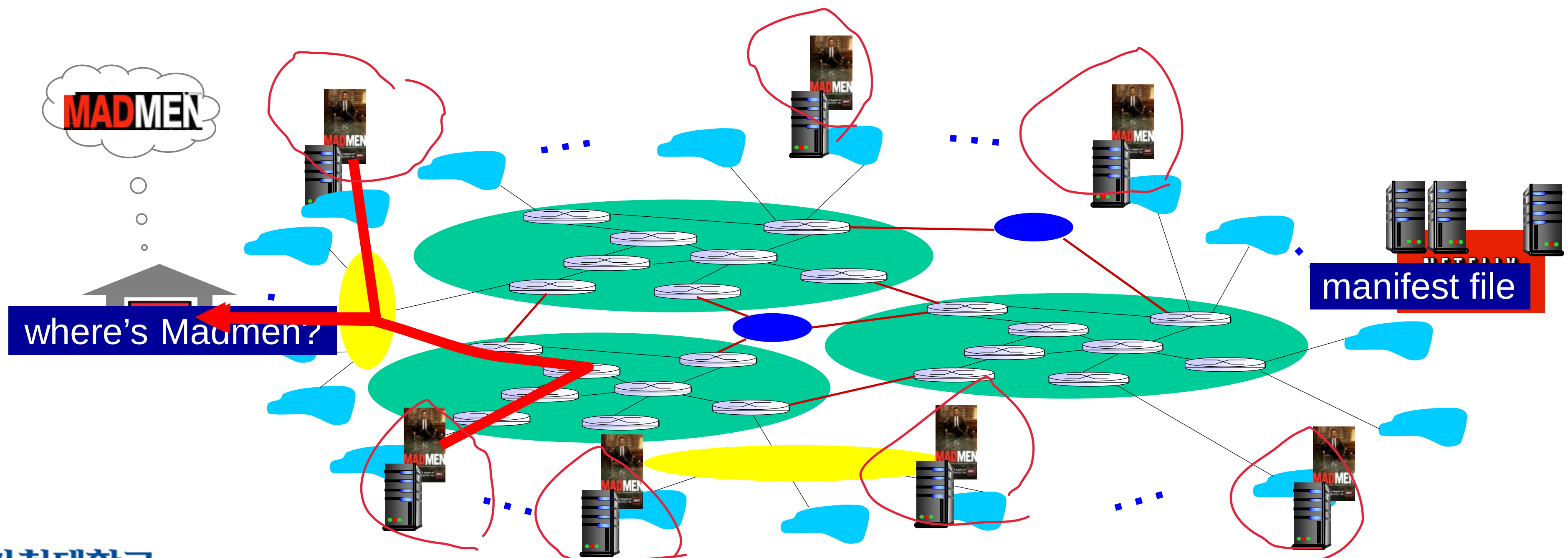
여러 유저들에게 비디오 카피가 전달 되기 때문에 outgoing link가 bottle neck이 될수있다.

….quite simply: this solution *doesn't scale*

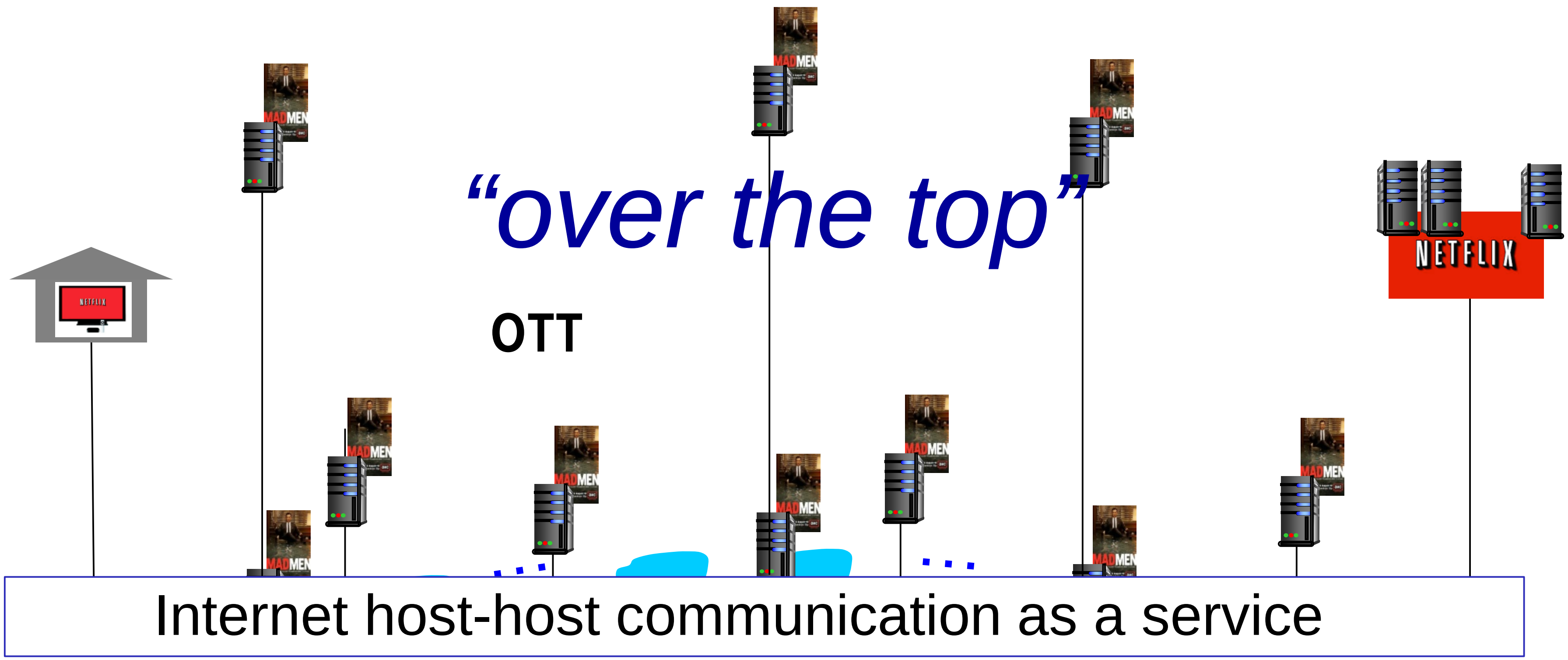
Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- 1 subscriber requests content from CDN
  - 2 directed to nearby copy, retrieves content
  - 3 may choose different copy if network path congested

장점 : 클라이언트 입장에서는 빠르게 안정적으로 stream 받음.
오리지널 서버로부터라면 네트워크 자원이 많이 사용 될 수 있음. -> 덜씀
서버의 부하를 줄임



Content Distribution Networks (CDNs)



OTT challenges: coping with a congested Internet

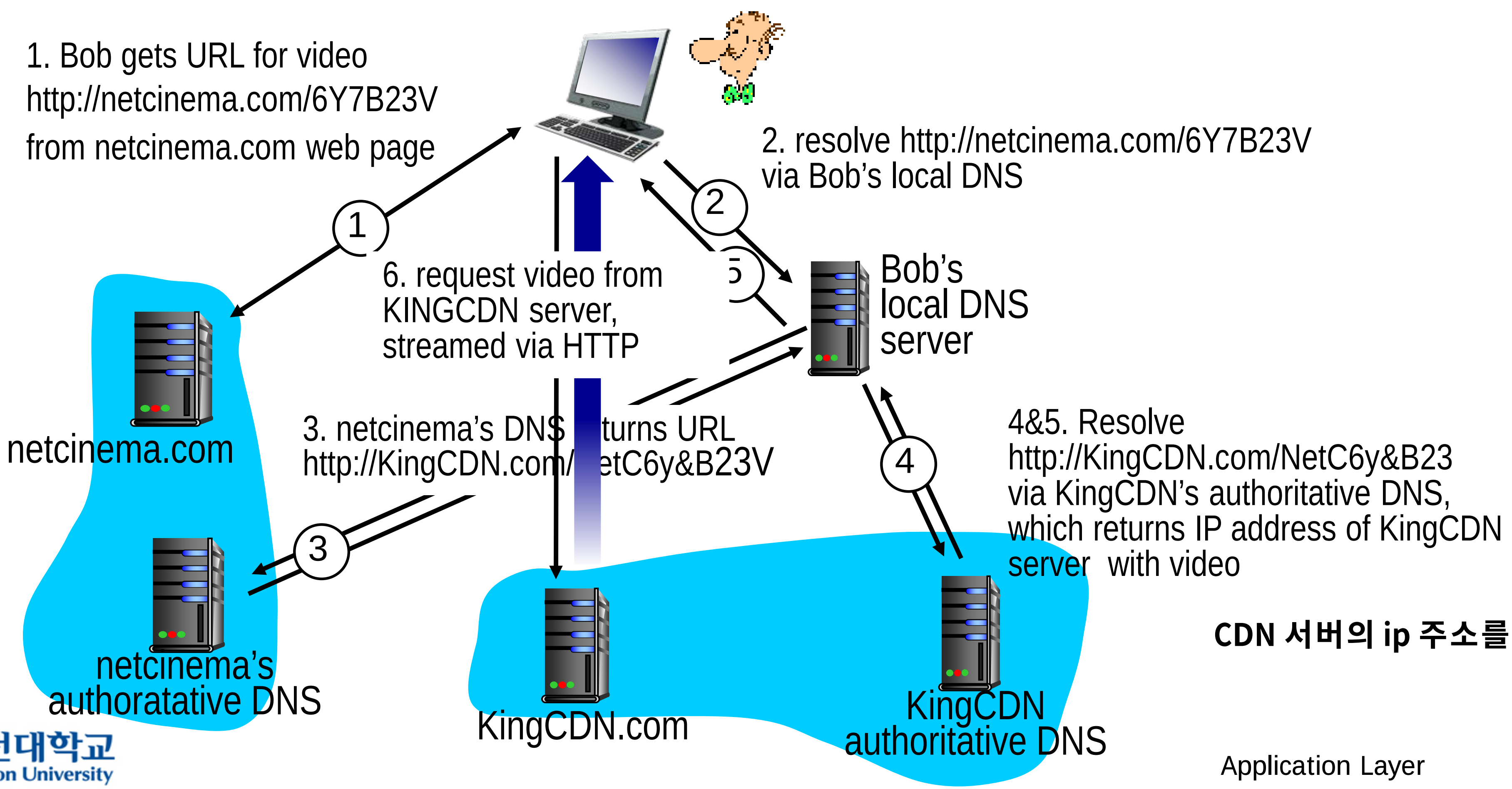
- from which CDN node to retrieve content?
- viewer behavior in presence of congestion?
- what content to place in which CDN node?

more .. in chapter 7

CDN content access: a closer look

Bob (client) requests video http://netcinema.com/6Y7B23V

- video stored in CDN at http://KingCDN.com/NetC6y&B23V



CDN 서버의 ip 주소를 알게됨.

Case study: Netflix

amazon cloud에 있는 netflix 서버