

# 컴퓨터 네트워크

## - 애플리케이션 계층 2 -

순천향대학교 사물인터넷학과

# 애플리케이션 계층

2.1 네트워크 애플리케이션의 원리

2.2 웹과 HTTP

## 2.3 전자메일

- SMTP, POP3, IMAP

2.4 DNS - 인터넷 디렉토리 서비스

2.5 P2P 파일 분배

2.6 비디오 스트리밍과 컨텐츠 분배 네트워크

2.7 소켓 프로그래밍: 네트워크 애플리케이션 작성

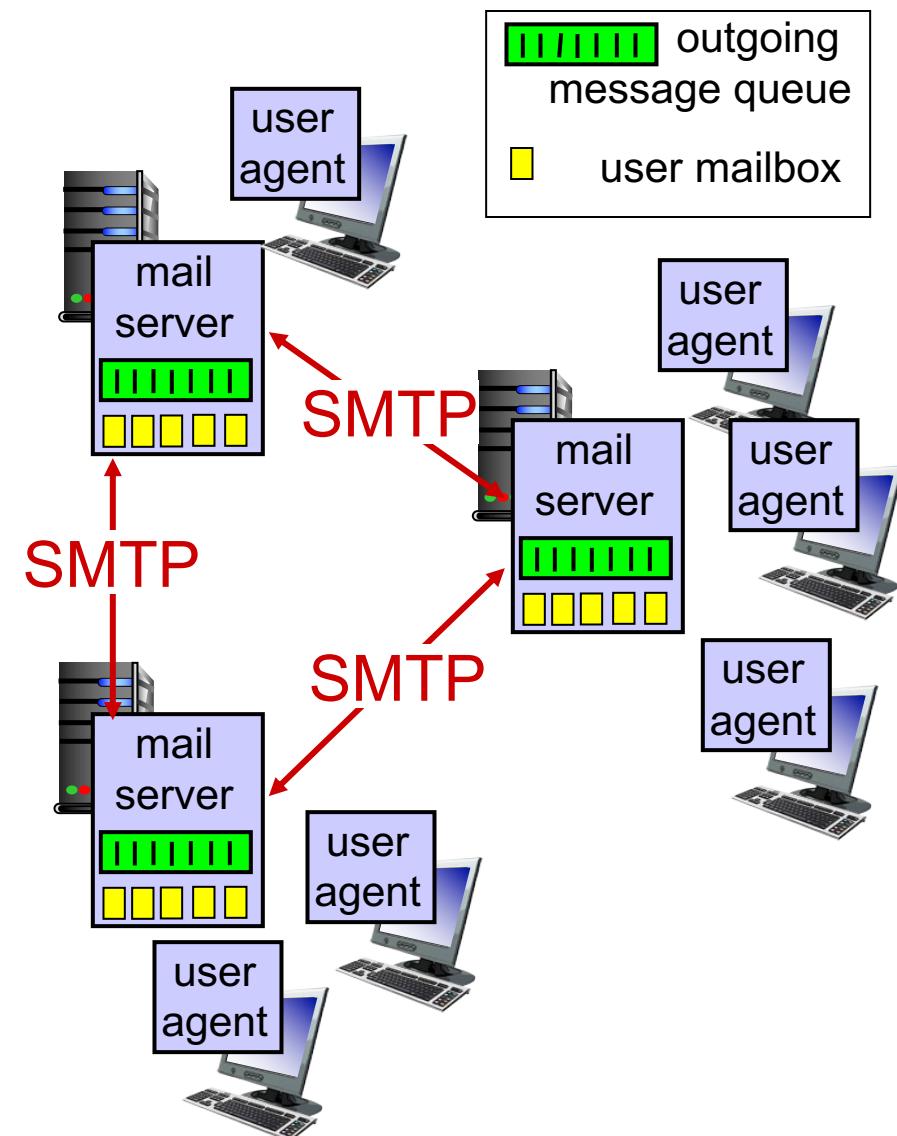
# 전자메일 Electronic Mail (E-Mail)

## ■ 3가지 주요 구성요소

- 사용자 에이전트 user agent
- 메일 서버 mail server
- SMTP Simple Mail Transfer Protocol

## ■ 사용자 에이전트

- 메일 리더 mail reader라고도 함
- 메시지를 읽고, 작성하고, 보냄
- 예) Outlook, 아이폰 메일 앱, gmail 앱 등
- 송수신 메시지는 서버에 저장



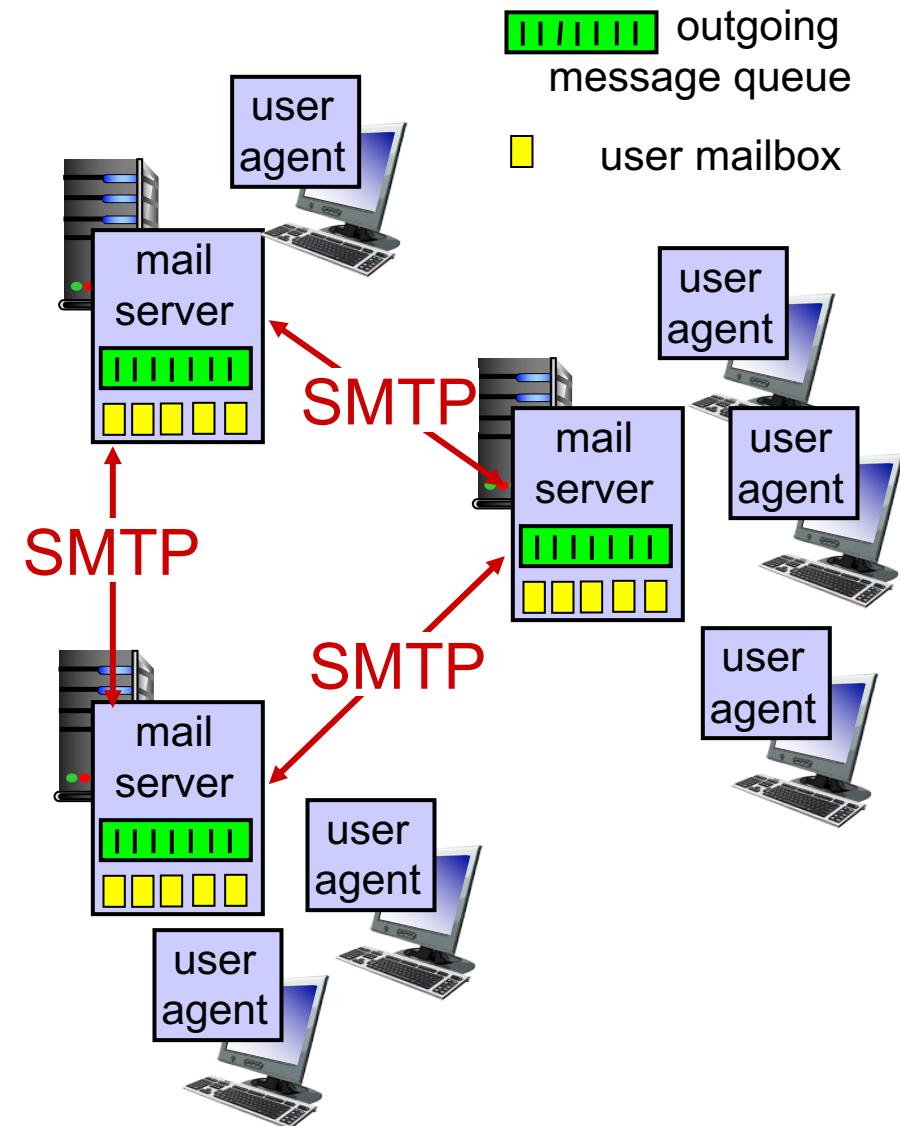
# 전자메일: 메일 서버

## ■ 메일 서버 mail server

- 사용자 별 받은 메시지를 저장하는 **메일박스** mailbox
- 보낼 메시지를 저장하는 **메시지 큐** message queue
- 이메일 메시지를 보내기 위한 메일 서버들 간의 **SMTP 프로토콜**
  - ✓ 클라이언트: 메일을 보내는 메일 서버
  - ✓ 서버: 메일을 수신하는 메일 서버

예) alice@sch.ac.kr 사용자가 bob@gmail.com 사용자에게 메일을 보내는 경우

  - ✓ 클라이언트: [smtp.sch.ac.kr](mailto:smtp.sch.ac.kr)
  - ✓ 서버: [smtp.gmail.com](mailto:smtp.gmail.com)



# 전자메일: SMTP [RFC 5321]

- 클라이언트에서 서버로 신뢰성 있게 이메일 메시지를 전송하기 위해 TCP 사용 (포트 25)

- 보내는 서버와 받는 서버 간 직접 전송

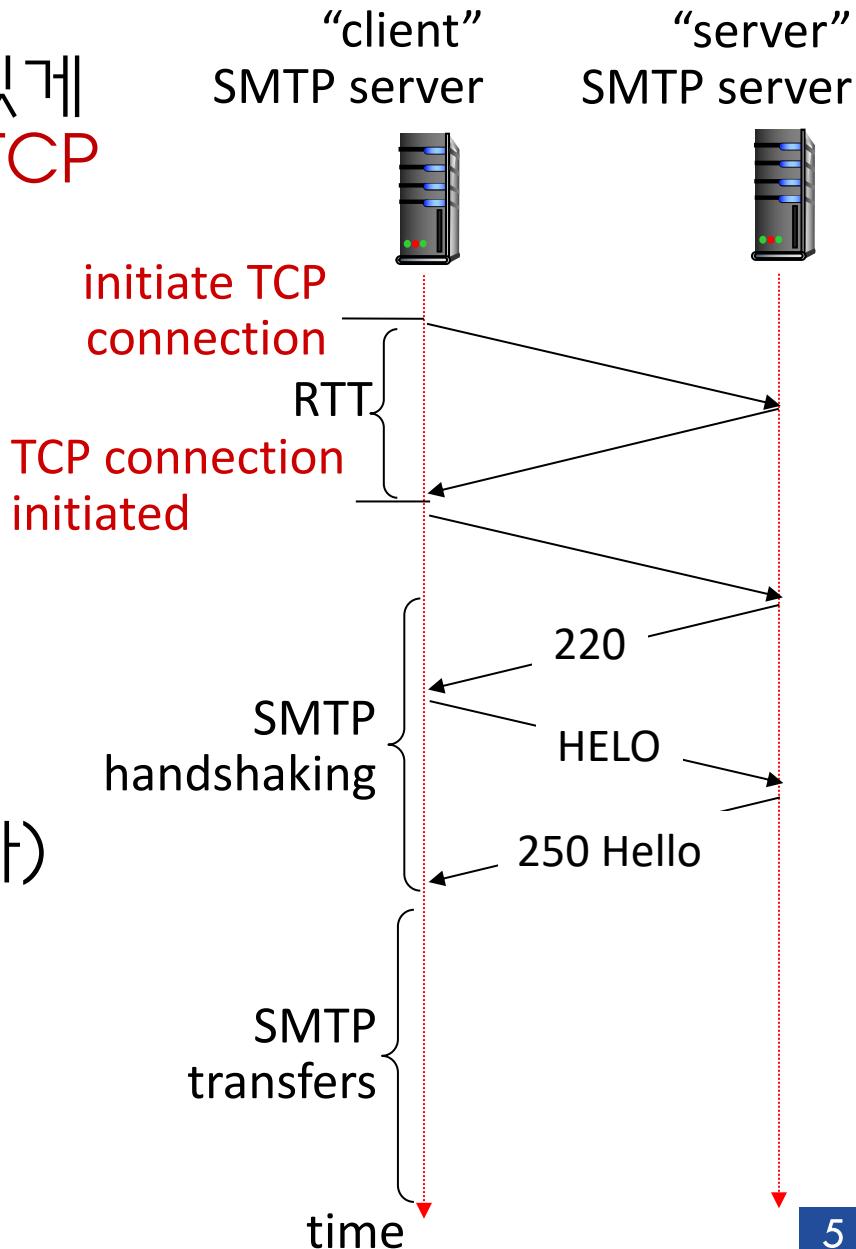
- 3단계 전송 단계

- 핸드쉐이킹 handshaking
- 메시지 전송
- 종료

- 명령/응답 상호작용 (HTTP와 유사)

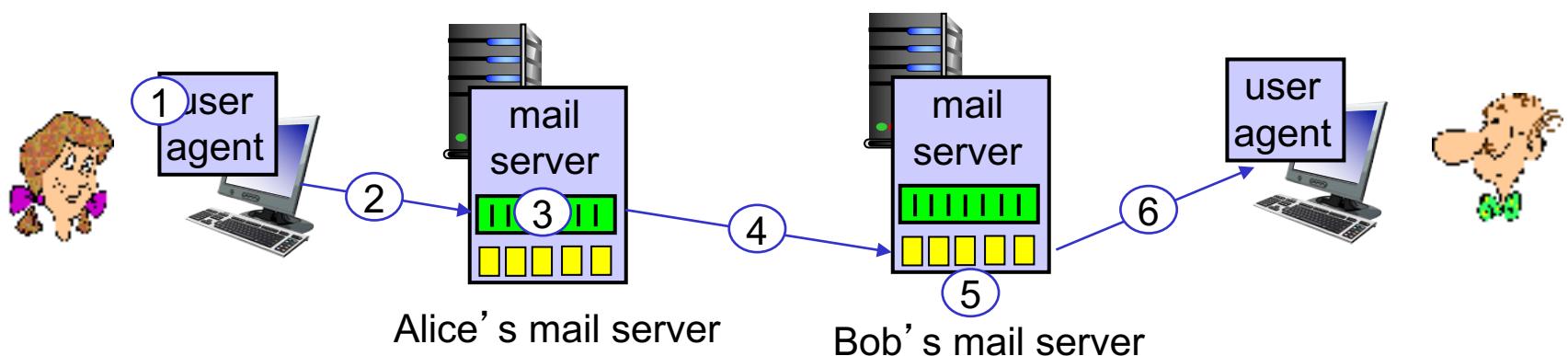
- 명령: ASCII 문자
- 응답: 상태 코드와 문장

- 메시지는 7-bit ASCII이어야 함



# 시나리오: 앤리스가 밥에게 메시지 전송

- 1) 앤리스는 **사용자 에이전트**를 이용하여 **bob@sch.ac.kr**에게 보낼 **메시지를 작성**
- 2) 앤리스의 사용자 에이전트는 앤리스의 **메일 서버**로 메시지를 전송. 메시지는 **메시지 큐**에 저장됨
- 3) **SMTP 클라이언트**(앤리스의 메일 서버)는 밥의 메일 서버와 **TCP 연결 설정**
- 4) SMTP 클라이언트는 설정된 TCP 연결을 통해 앤리스의 **메시지를 전송**
- 5) 밥의 메일 서버는 수신한 메시지를 밥의 **메일 박스**에 저장
- 6) 밥은 **사용자 에이전트**를 이용해서 **메시지를 읽음**



# SMTP 상호동작 예제

```
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 상호동작 실습

- telnet을 이용하여 SMTP 서버와 직접 대화
  - 사용자 에이전트를 사용하지 않고 메일을 전송

## ■ telnet servername 25

- 서버로부터 220 응답을 확인
- SMTP 명령 HELO, MAIL FROM, RCPT TO, DATA, QUIT을 사용하여 입력

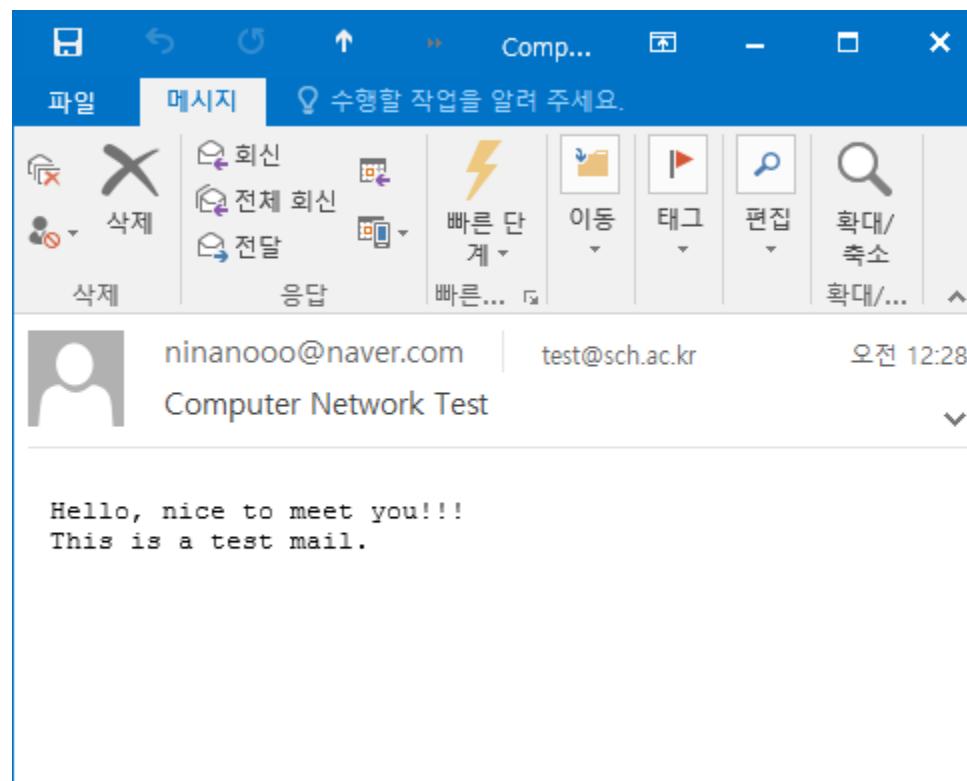
## ■ 예제)

```
dhkim@dhkim-VirtualBox:~$ telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 dhkim-VirtualBox ESMTP Sendmail 8.15.2/8.15.2/Debian-10; Tue, 2 Oct 2018 00:
27:43 +0900; (No UCE/UBE) logging access from: localhost(OK)-localhost [127.0.0.
1]
helohsch.ac.kr
250 dhkim-VirtualBox Hello localhost [127.0.0.1], pleased to meet you
mail from: <dhkim@naver.com>
250 2.1.0 <dhkim@naver.com>... Sender ok
rcpt to: <daeheekim@sch.ac.kr>
250 2.1.5 <daeheekim@sch.ac.kr>... Recipient ok
data
354 Enter mail, end with "." on a line by itself
from:ninanooo@naver.com
to:test@sch.ac.kr
subject:Computer Network Test
Hello, nice to meet you!!!
This is a test mail.
.
250 2.0.0 W91FRhiK001868 Message accepted for delivery
quit
221 2.0.0 dhkim-VirtualBox closing connection
Connection closed by foreign host.
```

# SMTP 상호동작 실습

## ■ 메일 수신 확인

- 실제 메일은 rcpt to: [daeheekim@sch.ac.kr](mailto:daeheekim@sch.ac.kr)로 수신됨
- 메시지 본문의 송신/수신 메일 주소는 data 본문의 from:/to: 필드에 적힌 것으로 표시됨



# SMTP 특징

- SMTP는 **지속 연결**을 사용
- SMTP 메시지(헤더&바디)는 **7-비트 ASCII**로 표현
- SMTP 서버는 메시지의 끝을 나타내기 위해  
“**CRLF,CRLF**”를 사용
  
- HTTP와 비교
  - HTTP: **풀(pull)** 프로토콜
  - SMTP: **푸시(push)** 프로토콜
  - HTTP, SMTP 둘 다 ASCII 요청/응답 상호 작용을 하고 응답 코드를 가짐
  - HTTP: 각 객체는 응답 메시지에 캡슐화됨
  - SMTP: 모든 객체가 하나의 메시지에 포함됨

# 메일 메시지 포맷 Mail Message Format

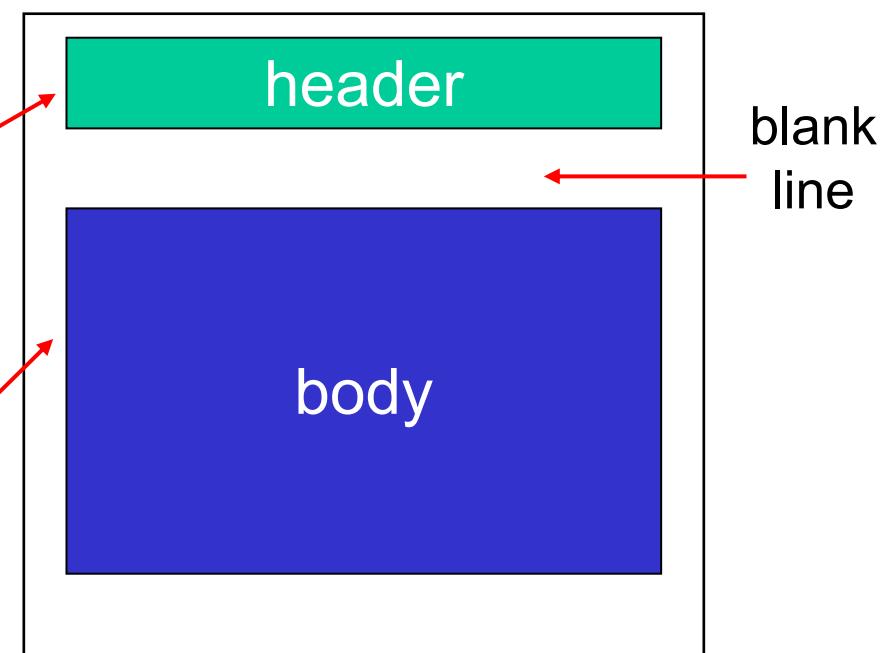
- SMTP: 전자메일 메시지를 주고 받는 프로토콜
- RFC 2822: 텍스트 메시지 포맷의 표준

## ■ 헤더 라인, e.g.,

- To:
- From:
- Subject:  
→ SMTP 명령과는 다름!

## ■ 몸체

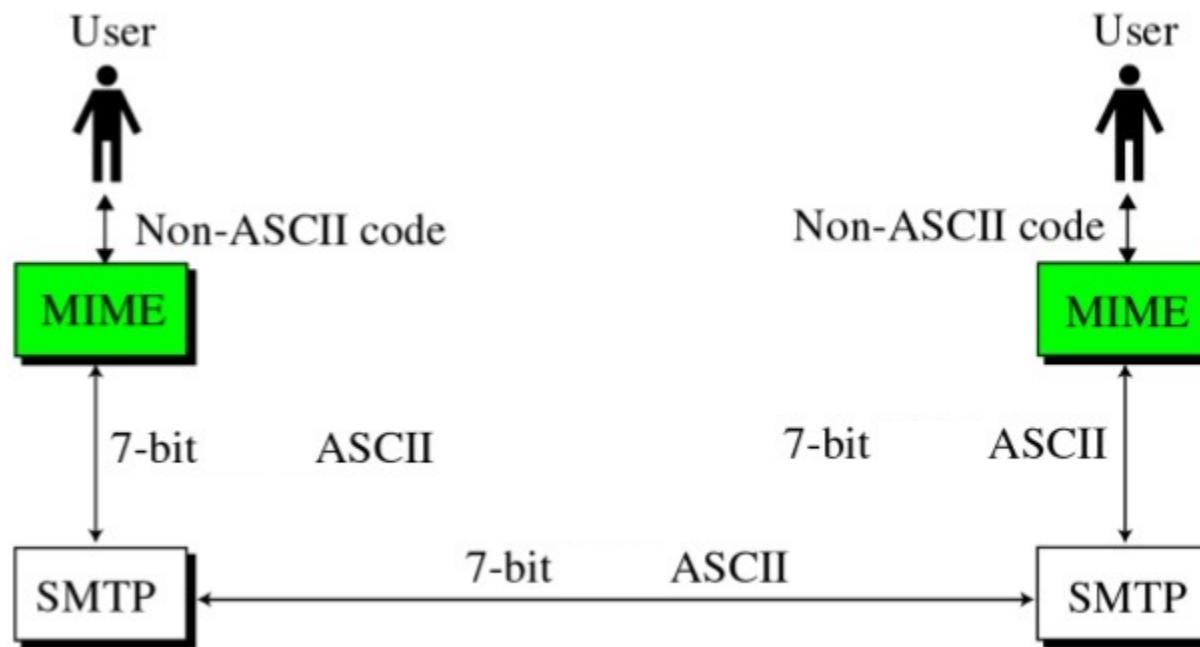
- 보내고자 하는 내용
- ASCII 문자만 가능



# MIME Multipurpose Internet Mail Extensions

## ■ ASCII가 아닌 데이터를 메일로 전송 시 사용되는 RFC 2822의 확장

- 비 ASCII 텍스트 메시지(이미지, 오디오, 비디오, 다국적 언어 등)는 SMTP가 이해하는 ASCII 포맷으로 인코딩
- RFC 2045, 2046에 추가 헤더 정의
  - ✓ Content-Transfer-Encoding: 인코딩된 코드 타입 기술
  - ✓ Content-Type: 메시지 몸체의 타입을 기술



# MIME 메시지 예

- 앤리스가 밥에게 JPEG 이미지를 전송하는 예

**From:** alice@crepes.fr

**To:** bob@hamburger.edu

**Subject:** Picture of yummy crepe.

**MIME-Version:** 1.0

**Content-Transfer-Encoding:** base64

**Content-Type:** image/jpeg

(base64 인코딩 데이터 .....)

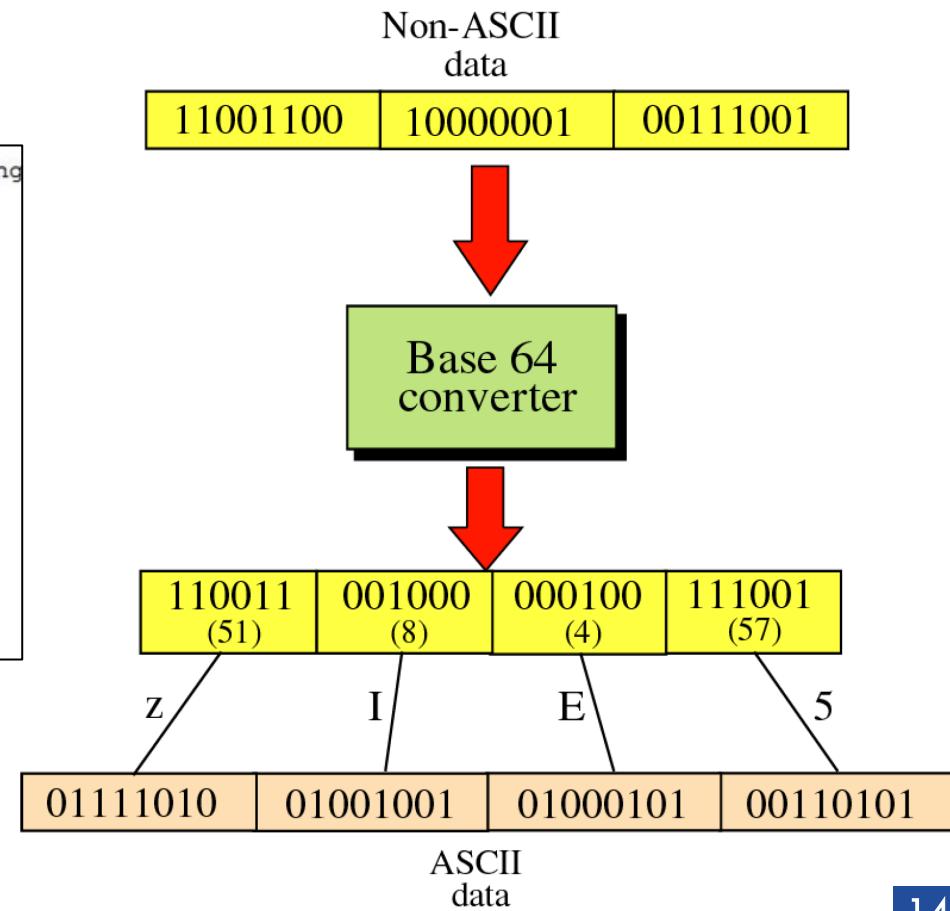
.....

..... base64 인코딩 데이터)

# Base64 인코딩

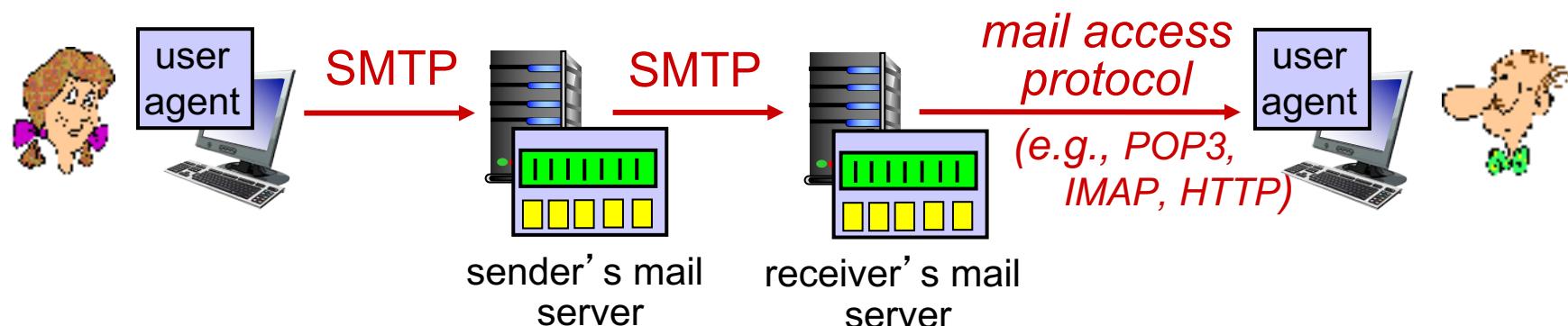
- RFC 1421
- 인코딩 절차
  - 각 6비트가 64개의 출력 가능한 문자로 인코딩
  - 8비트 3개를 6비트 4개로 변형

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0 A		17 R		34 i		51 z	
1 B		18 S		35 j		52 0	
2 C		19 T		36 k		53 1	
3 D		20 U		37 l		54 2	
4 E		21 V		38 m		55 3	
5 F		22 W		39 n		56 4	
6 G		23 X		40 o		57 5	
7 H		24 Y		41 p		58 6	
8 I		25 Z		42 q		59 7	
9 J		26 a		43 r		60 8	
10 K		27 b		44 s		61 9	
11 L		28 c		45 t		62 +	
12 M		29 d		46 u		63 /	
13 N		30 e		47 v			
14 O		31 f		48 w		(pad) =	
15 P		32 g		49 x			
16 Q		33 h		50 y			



# 메일 접속 프로토콜 Mail Access Protocol

- SMTP는 수신자의 서버에 메시지를 전송하고 저장
- 메일 접속 프로토콜
  - 메일 서버로부터 메일을 가져오기 위해 사용되는 프로토콜
  - POP3 (Post Office Protocol-Version 3, RFC 1939)
    - ✓ 사용자 에이전트와 서버간 인증과 다운로드
  - IMAP (Internet Mail Access Protocol, RFC 1730)
    - ✓ POP3보다 다양한 특성을 가지고 복잡함
    - ✓ 예) 서버에 저장된 메시지 관리
  - HTTP
    - ✓ Gmail, 다음 mail, 네이버 mail 등



# POP3와 IMAP

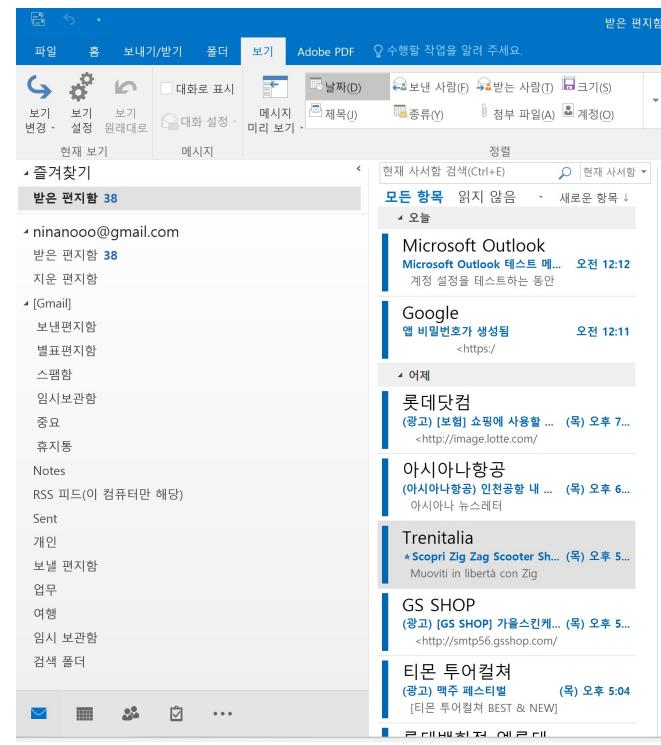
## ■ POP3

- 앞의 예에서는 “**다운로드 및 삭제 (download-delete)**” 모드 사용
  - ✓ 밥이 클라이언트를 바꾸면 (PC에서 스마트폰) 이메일을 다시 읽을 수 없음
- “**다운로드 및 유지 (download-and-keep)**” 모드를 사용하면 여러 클라이언트에 이메일의 복사본을 저장할 수 있음
- POP3는 세션 간 상태를 유지하지 않음 (stateless)



## ■ IMAP

- 모든 메시지를 **서버에 보관**
- 사용자가 폴더에 메시지를 조직화할 수 있음
- 세션 간에 사용자 상태 정보 유지
  - ✓ 폴더 이름과, 어떤 메시지(메시지 ID)와 폴더 이름 간 매핑 정보 유지



# 애플리케이션 계층

2.1 네트워크 애플리케이션의 원리

2.2 웹과 HTTP

2.3 전자메일

- SMTP, POP3, IMAP

2.4 DNS - 인터넷 디렉토리 서비스

2.5 P2P 파일 분배

2.6 비디오 스트리밍과 컨텐츠 분배 네트워크

2.7 소켓 프로그래밍: 네트워크 애플리케이션 작성

# DNS Domain Name System

## ■ 사람은 여러 방식으로 자신을 식별할 수 있음

- 이름, 주민등록번호, 여권번호, 운전면허번호 등

## ■ 호스트, 라우터는 다음과 같은 식별자를 가짐

- IP 주소 (예: 220.69.189.111, IPv4: 32bit, IPv6: 128bit)
- 호스트네임<sup>hostname</sup> (예: www.sch.ac.kr, 사람이 쉽게 사용할 수 있음)

Q. IP 주소와 호스트네임 간 맵핑은 어떻게 이루어지는가?

## ■ 도메인 네임 시스템 Domain Name System (DNS)

- 호스트네임을 IP 주소로 변환해주는 디렉터리 서비스
- 많은 서버들이 계층 구조로 구현된 분산 데이터베이스<sup>distributed database</sup>
- 애플리케이션 계층 프로토콜
  - ✓ 호스트는 주소/이름 간 변환을 위해 네임 서버와 통신함
  - ✓ Note
    - 애플리케이션 계층 프로토콜로 구현된 핵심 인터넷 기능
    - 복잡한 기능을 네트워크 엣지에서 구현하는 인터넷 철학에 부합

# DNS 서비스와 구조

## ■ DNS 서비스

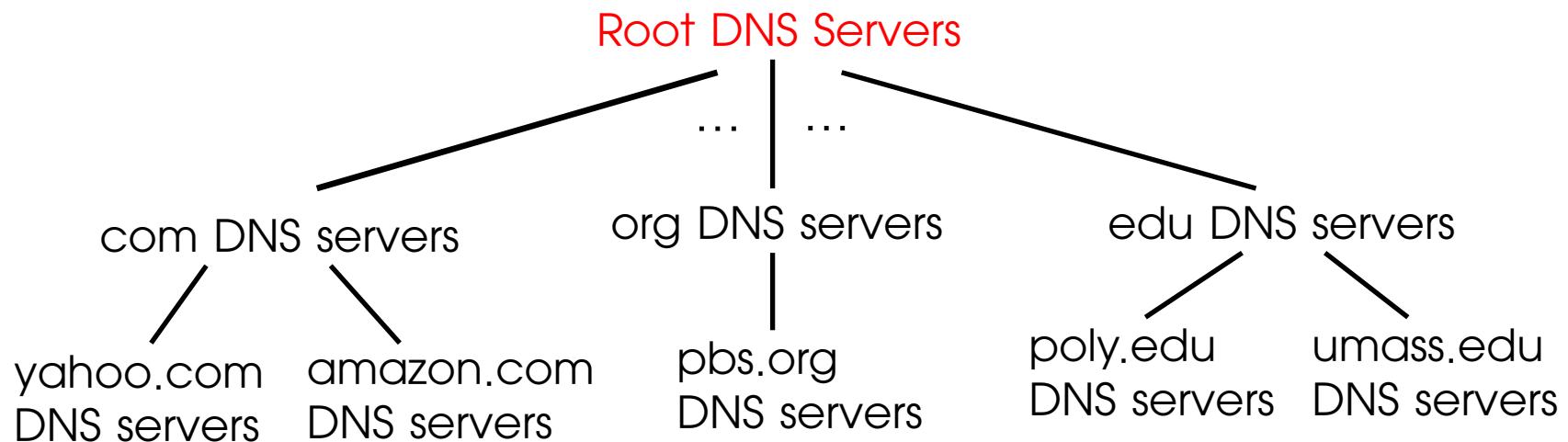
- 호스트네임을 IP 주소로 변환
- 호스트 에일리어싱<sup>host aliasing</sup>
  - ✓ 간단한 별칭 호스트네임 *alias hostname*(예: www.daum.net)을 복잡한 정식 호스트네임 *canonical hostname* (예: www.g.daum.net)으로 변환
- 메일 서버 에일리어싱
  - ✓ 메일 서버의 간단한 별칭 호스트네임(예: gmail.com)을 복잡한 정식 호스트네임(예: gmail-smtp-in.l.google.com)으로 변환
- 부하 분산<sup>load balancing</sup>
  - ✓ 중복 웹 서버: 많은 IP 주소가 1개의 호스트네임에 매핑됨
  - ✓ 예) naver.com → 210.89.160.88, 125.209.222.142, 210.89.164.90, 125.209.222.141

## Q. 단일 중앙 집중 방식의 DNS를 사용한다면?

- 서버 고장 시 전체 인터넷이 작동하지 않음 *single point of failure*
- 서버가 처리해야 할 트래픽 양이 많아짐
- 거리가 멀어 송수신 시간이 길어짐
- A: 확장성이 떨어짐

# DNS: 분산, 계층적 데이터베이스

Distributed, Hierarchical Database

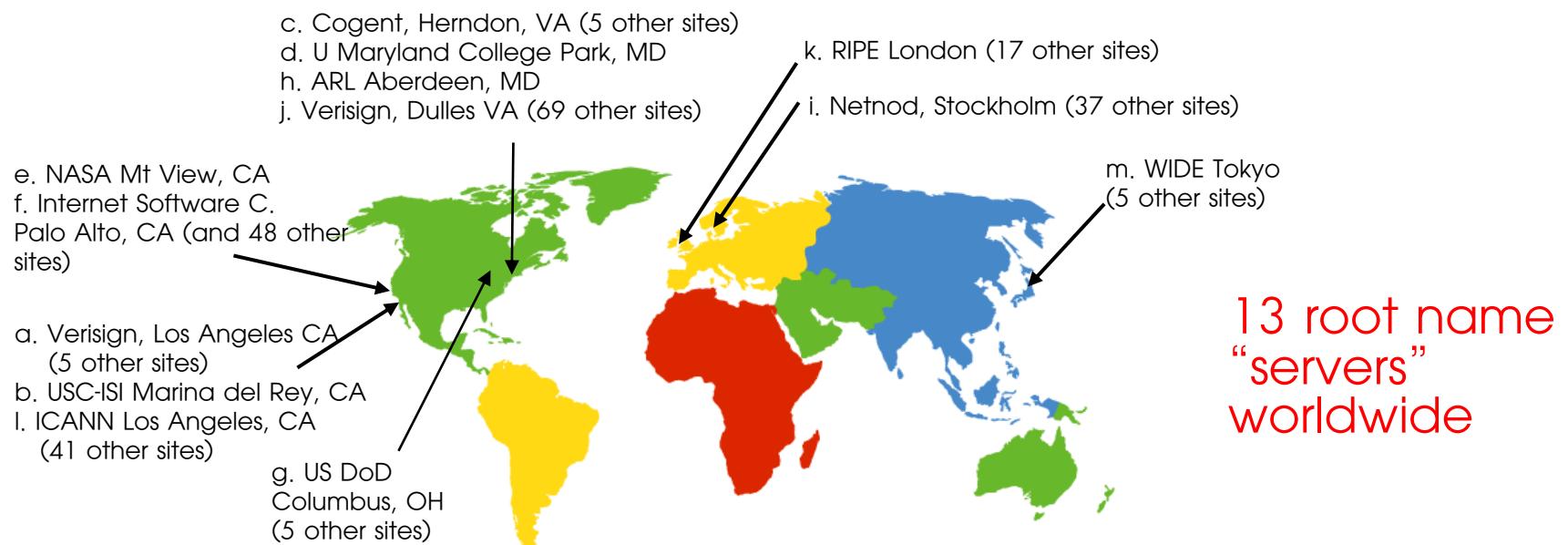


## ■ DNS 클라이언트가 [www.amazon.com](http://www.amazon.com)에 대한 IP 주소를 찾는 과정

1. 클라이언트는 .com의 DNS 서버를 찾기 위해 루트 <sup>root</sup> 서버에 질의
  - ✓ 루트 서버는 .com의 DNS 서버의 IP 주소를 클라이언트에게 전송
2. 클라이언트는 amazon.com의 DNS 서버를 찾기 위해 .com의 DNS 서버(TLD 서버, top-level domain 서버)에 질의
  - ✓ amazon.com의 DNS 서버의 IP 주소를 클라이언트에게 전송
3. 클라이언트는 [www.amazon.com](http://www.amazon.com)의 IP 주소를 찾기 위해 amazon.com의 DNS 서버(책임 서버, authoritative server)에 질의
  - ✓ 책임 서버는 [www.amazon.com](http://www.amazon.com)의 IP 주소를 클라이언트에게 전송

# DNS: 루트 네임 서버 *root name server*

- 로컬 네임 서버가 이름을 해결할 수 없는 경우 루트 네임 서버에 문의함
- 루트 네임 서버 *root name server*
  - TLD 서버의 IP 정보를 가지고 있음
  - 로컬 네임 서버로부터 질의를 받으면, 해당 TLD 서버의 IP 정보를 알려줌
  - 전 세계적으로 13개의 다른 기관에서 관리되는 400개 이상의 루트 DNS 서버가 있음



# TLD와 책임 DNS 서버

## ■ TLD 서버 Top-Level Domain Server

- .com, .org, .net, .edu 같은 상위 레벨 도메인과 .kr, .uk, .fr, .ca 등과 같은 국가의 최상위 레벨 도메인을 책임지는 서버
- Network Solutions는 .com, .net의 TLD 서버를 관리
- Educause는 .edu의 TLD 서버를 관리
- 한국인터넷진흥원은 .kr의 TLD 서버를 관리

## ■ 책임 DNS 서버 Authoritative DNS Server

- 기관(회사, 대학 등) 내 서버(웹 서버, 메일 서버 등)들의 호스트네임을 IP 주소로 변환해주는 서버
- 기관이나 ISP가 책임 DNS 서버를 관리
- 예) 순천향대학교 책임 DNS 서버
  - ✓ ns0.sch.ac.kr (220.69.193.132)
  - ✓ ns1.sch.ac.kr (220.69.193.133)
  - ✓ sch.ac.kr 도메인의 모든 호스트네임/IP 변환을 책임짐

# 로컬 DNS 서버 Local DNS Server

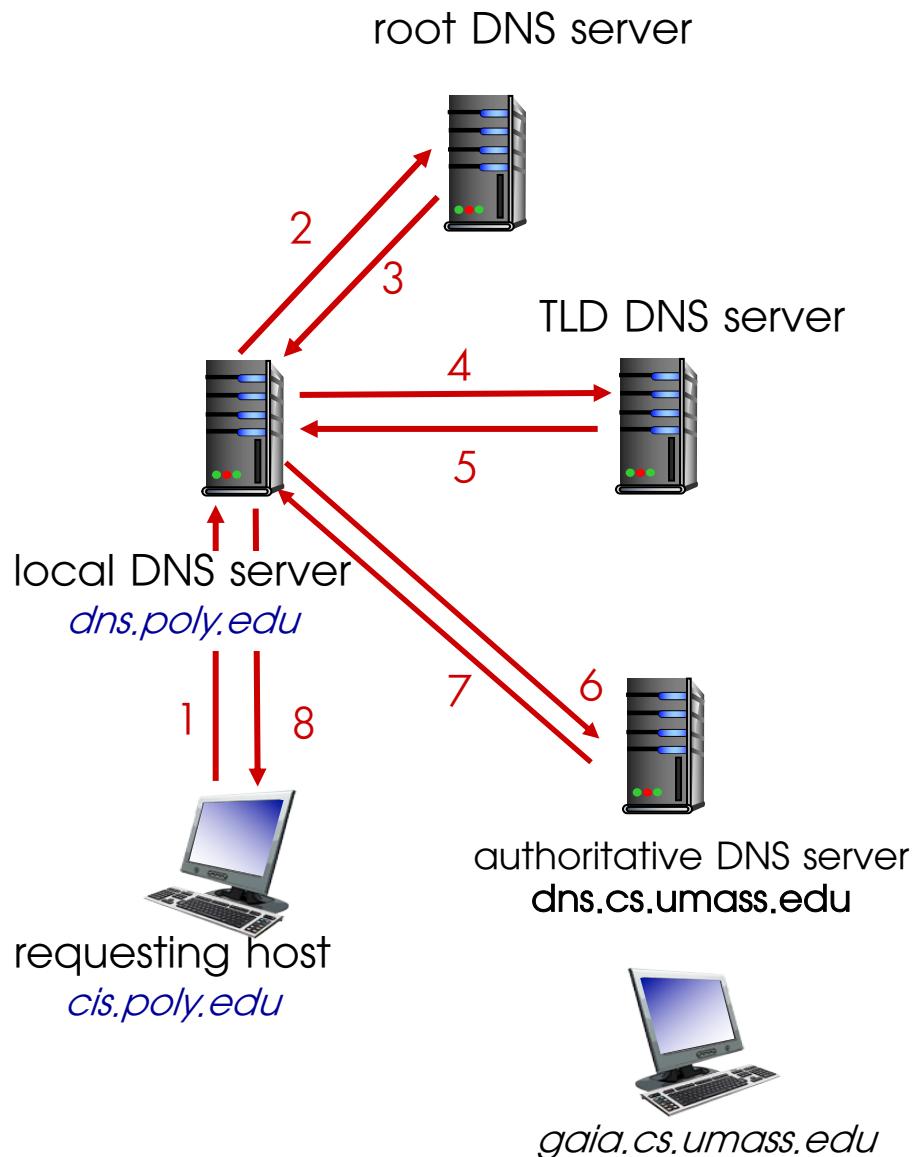
- 로컬 DNS 서버는 계층 구조에 포함되지 않음
- 각 ISP(홈 ISP, 회사, 대학)는 **최소 1개의 로컬 DNS 서버를 가지고 있음**
  - 디폴트 네임 서버(default name server)라고도 함
- 호스트가 DNS 쿼리 **query**를 하면, 해당 쿼리는 로컬 DNS 서버로 전송됨
  - 최근 수행한 **이름/주소 매핑 정보를 캐싱**하고 있음
  - **프록시와 같은 역할**을 하며, 쿼리를 DNS 계층 구조로 전달함

설명 . . . . .	:	Parallels VirtIO Ethernet Adapter
물리적 주소 . . . . .	:	00-1C-42-F2-C4-98
DHCP 사용 . . . . .	:	예
자동 구성 사용 . . . . .	:	예
IPv6 주소 . . . . .	:	fdb2:2c26:f4e4:0:845a:6609:2547:e97d(기본 설정)
임시 IPv6 주소 . . . . .	:	fdb2:2c26:f4e4:0:d052:b31c:1281:a67c(기본 설정)
링크-로컬 IPv6 주소 . . . . .	:	fe80::845a:6609:2547:e97d%8(기본 설정)
IPv4 주소 . . . . .	:	10.211.55.4(기본 설정)
서브넷 마스크 . . . . .	:	255.255.255.0
임대 시작 날짜 . . . . .	:	2022년 3월 15일 화요일 오후 9:12:11
임대 만료 날짜 . . . . .	:	2022년 3월 16일 수요일 오후 4:54:10
기본 게이트웨이 . . . . .	:	10.211.55.1
DHCP 서버 . . . . .	:	10.211.55.1
DHCPv6 IAID . . . . .	:	100670530
DHCPv6 클라이언트 DUID . . . . .	:	00-01-00-01-29-58-17-F6-00-1C-42-F2-C4-98
DNS 서버. . . . .	:	10.211.55.1

# DNS 동작 예제

## Name Resolution

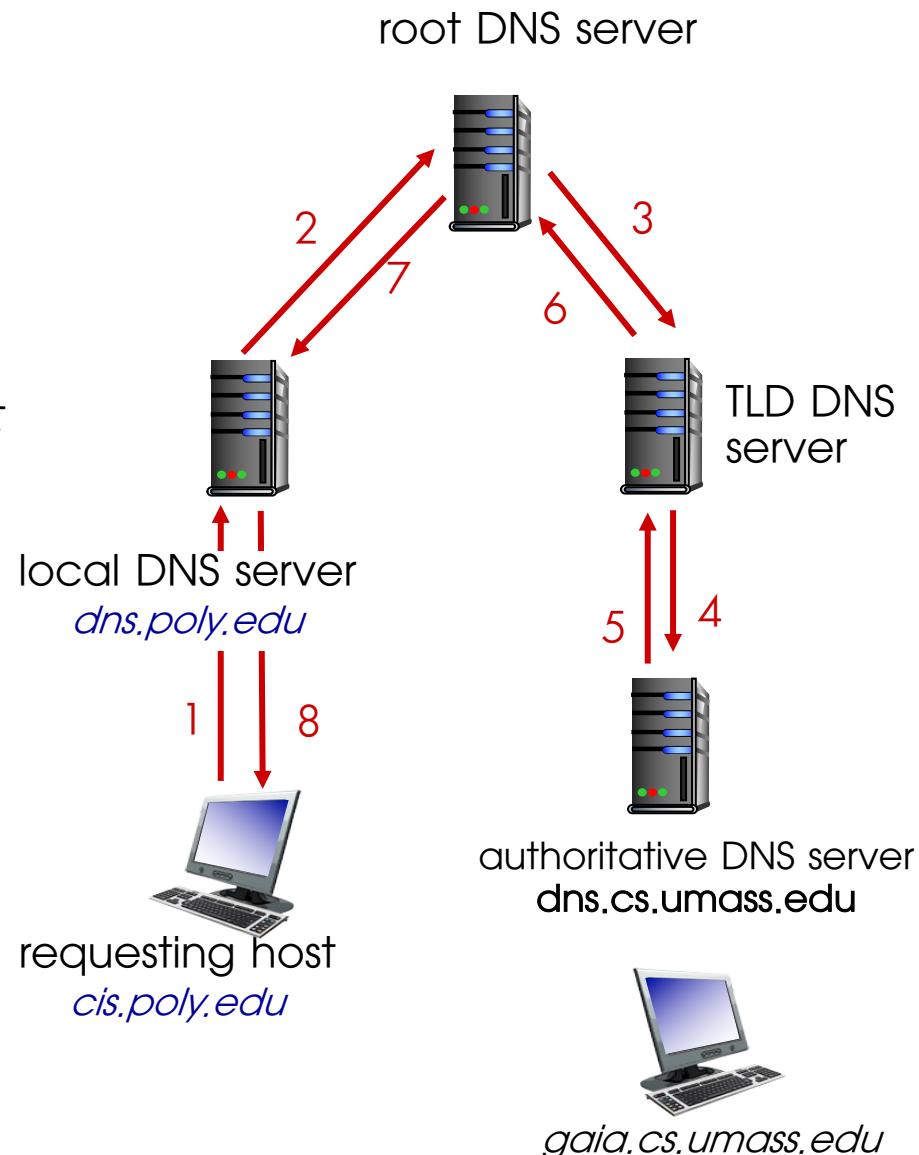
- 호스트 cis.poly.edu가 gaia.cs.umass.edu의 IP 주소를 찾고자 함
- 반복적 쿼리 Iterated query
  - 쿼리를 받은 서버는 다음에 연결할 서버의 주소를 알려줌
  - “나는 이 이름의 IP 주소를 모르니, 이 서버에 물어보세요”



# DNS 동작 예제 Name Resolution

## ■ 재귀적 쿼리 recursive query

- 이름에 대한 해결(name resolution)을 접속한 네임 서버에 맡김
- 상위 계층의 서버는 많은 부담을 가짐



# DNS 캐싱 DNS Caching

## ■ DNS 서버가 새로운 이름/IP 주소 매팅 정보를 얻으면, 그 정보를 저장함 (caching)

- 캐시 엔트리(캐싱된 정보)는 일정 시간(TTL Time To Live) 이후에 삭제됨
- 일반적으로 로컬 DNS 서버는 TLD 서버의 주소를 캐싱함
  - ✓ 따라서, 루트 DNS 서버에 자주 방문하지 않음

## ■ 캐시 엔트리는 out-of-date 정보일 수 있음

- 이름/IP 주소 매팅 정보를 얻어서 캐싱한 뒤에, 이름/IP 주소 매팅 관계가 변경된 경우
  - ✓ 예) (www.schlot.ac.kr, 10.10.10.10) → (www.schlot.ac.kr, 11.11.11.11)
- 특정 이름을 가진 호스트가 IP 주소를 변경하면, TTL이 지나기 전에는 이전 IP 주소를 가지고 있을 수 있음

## ■ RFC 2136

- 업데이트(update)/알림(notify) 메커니즘 정의

# DNS 레코드 DNS record

- DNS는 **자원 레코드** resource records (RR)를 저장하는 분산 데이터베이스

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

## type=A

- **name** : 호스트네임
- **value** : IP 주소

## type=NS

- **name** : 도메인  
(예: sch.ac.kr)
- **value** : 이 도메인에 대한 책임 DNS의 호스트네임  
(예: ns0.sch.ac.kr)

## type=CNAME

- **name** : 실제 이름에 대한 간단한 별칭(alias)
- **value** : 정식 호스트 네임
- www.ibm.com(별칭)은 실제로 severeast.backup2.ibm.com (정식 호스트네임)

## type=MX

- **name** : 메일 서버의 간단한 별칭
- **value**: 메일 서버의 정식 호스트 네임

# DNS 프로토콜, 메시지

- DNS 프로토콜은 동일 메시지 포맷을 갖는 질의 query와 응답 reply 메시지로 구성

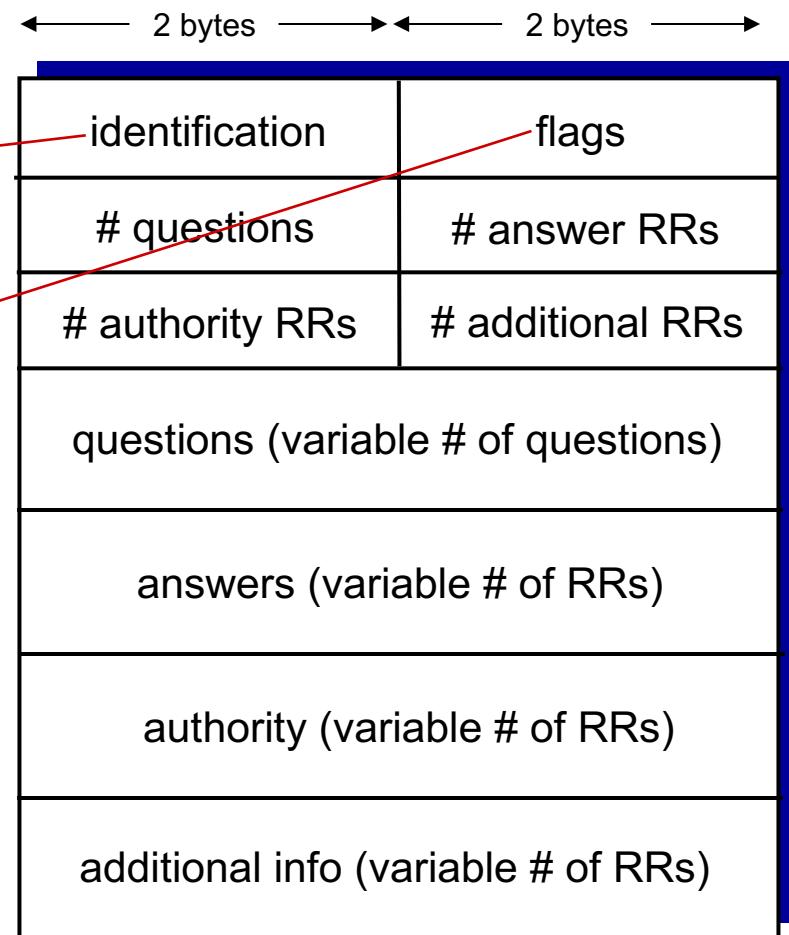
## 메시지 헤더

- 식별자 identification

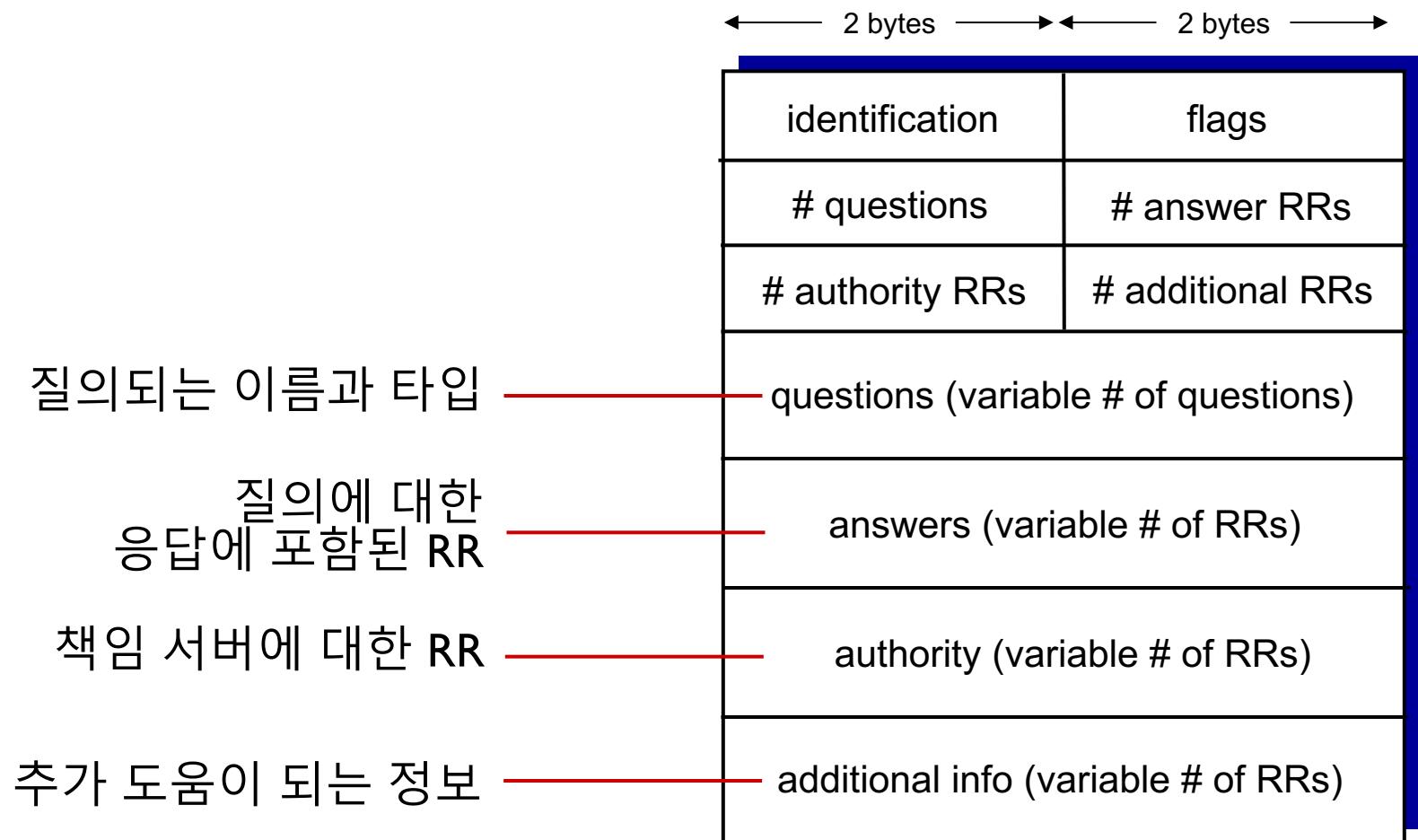
- 질의를 식별하는 16비트 식별자
- 응답 메시지도 질의와 동일 식별자를 가짐

- 플래그 flags

- 질의와 응답을 구분
- 재귀적 질의 요청
- 재귀적 질의 가능
- 책임 서버의 응답



# DNS 프로토콜, 메시지



# DNS에 레코드 삽입

- 예제: 새로운 스타트업 회사 “[Network Utopia](#)”
  - 웹 서버와 메일 서버 운영이 필요함
- **DNS 등록기관**(registrar)에 도메인 네임 “[networkutopia.com](#)”을 등록
  - 책임 DNS 서버의 호스트네임과 IP 주소를 등록기관에 제공
  - 등록기관은 **두 개의 자원 레코드(RR)**를 .com TLD 서버에 삽입
    - ✓ ([networkutopia.com](#), [dns1.networkutopia.com](#), NS)
    - ([dns1.networkutopia.com](#), 212.212.212.1, A)
- **책임 DNS 서버**에 아래의 레코드 등록
  - [www.networkutopia.com](#)([웹 서버](#))을 위한 type A 레코드
    - ✓ ([www.networkutopia.com](#), 212.212.212.20, A)
  - [networkutopia.com](#)에 대한 type MX 레코드
    - ✓ ([networkutopia.com](#), [mail.networkutopia.com](#), MX)
  - [mail.networkutopia.com](#)([메일 서버](#))을 위한 type A 레코드
    - ✓ ([mail.networkutopia.com](#), 212.212.212.30, A)

# 애플리케이션 계층

2.1 네트워크 애플리케이션의 원리

2.2 웹과 HTTP

2.3 전자메일

- SMTP, POP3, IMAP

2.4 DNS - 인터넷 디렉토리 서비스

2.5 P2P 파일 분배

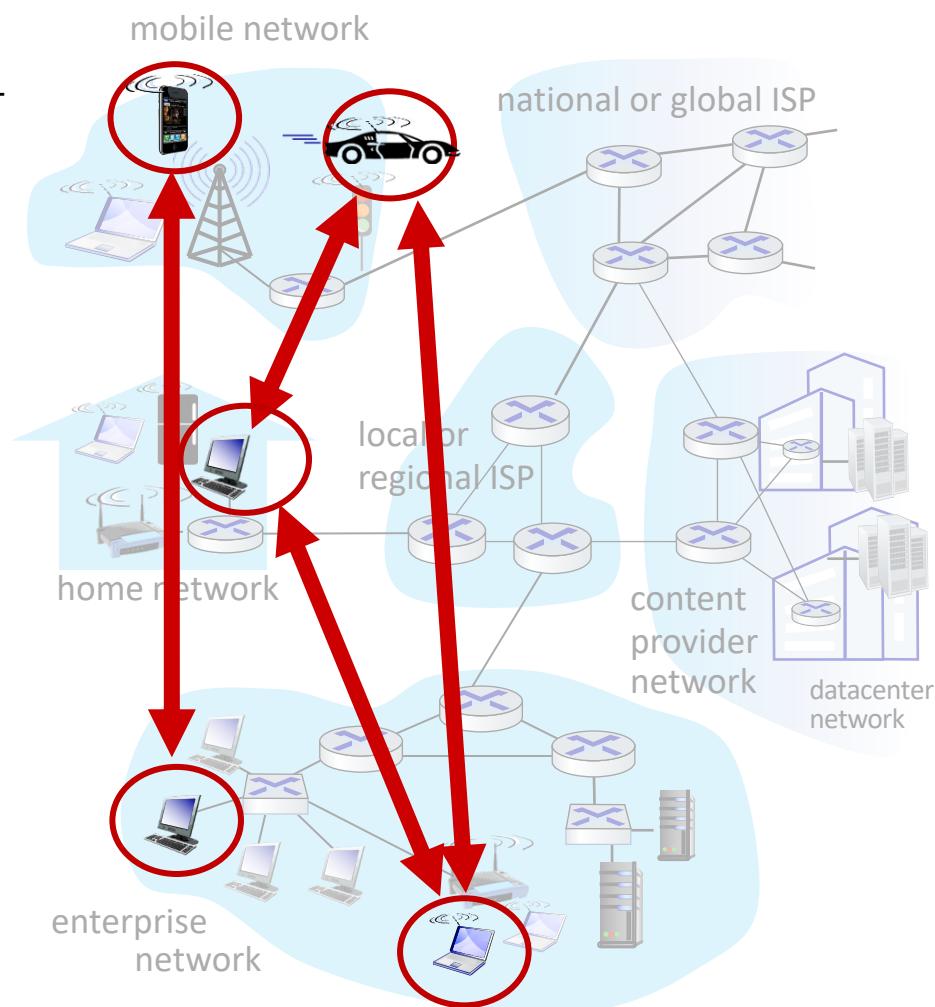
2.6 비디오 스트리밍과 컨텐츠 분배 네트워크

2.7 소켓 프로그래밍: 네트워크 애플리케이션 작성

# P2P 구조

## ■ P2P 구조

- 항상 켜져 있는(always-on) 서버가 없음
- 임의의 종단 장치들끼리  
직접 통신 수행
- 각 **피어**<sup>Peer</sup>들은 각각 서비스를  
요청하고, 서비스를 제공함
  - ✓ 높은 자기 확장성(self scalability)
    - 새로운 피어는 새로운 서비스를 제공함과  
동시에 새로운 서비스를 요청
- 피어들은 간헐적으로 연결되며  
IP 주소가 바뀔 수 있음

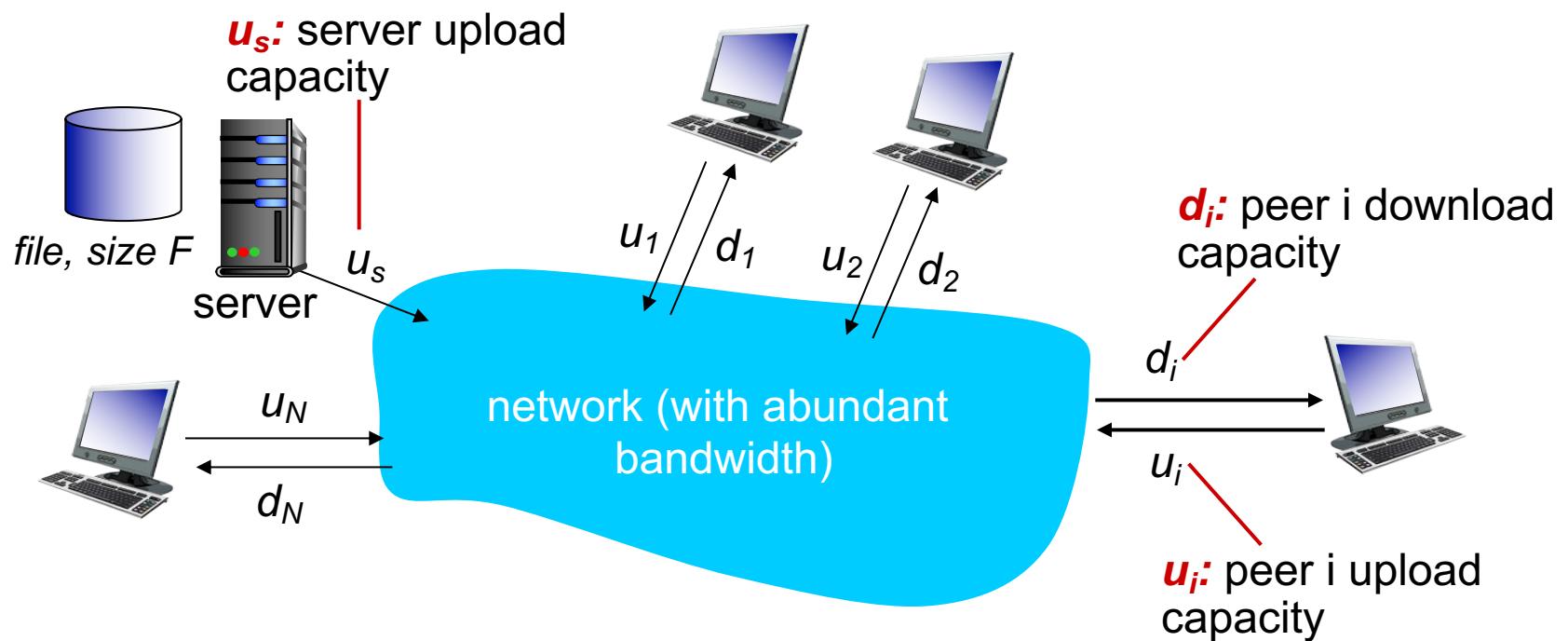


## ■ P2P 애플리케이션의 예

- 파일 분배: BitTorrent
- 스트리밍: KanKan
- VoIP: Skype

# 파일 분배: 클라이언트-서버 vs P2P

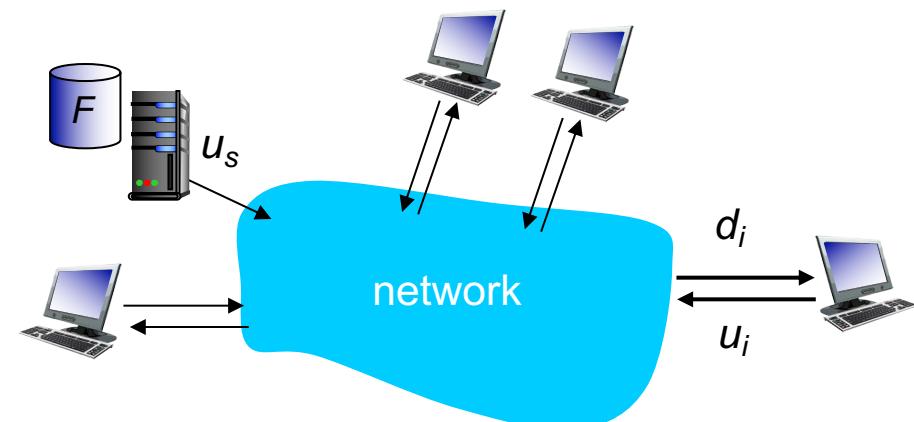
- Q: 크기  $F$ 인 파일을 1개의 서버에서  $N$ 개의 피어로 전송하는데 걸리는 시간은?
  - 피어들의 업로드/다운로드 대역폭은 그림과 같이 한정되어 있음



# 파일 분배 시간: 클라이언트-서버

## ■ 서버

- N개의 파일 복사본을 피어들에게 순차적으로 전송(업로드)해야 함
  - ✓ 1개의 복사본 전송 시간:  $F/u_s$
  - ✓ N개의 복사본 전송 시간:  $NF/u_s$



## ■ 클라이언트

- 각 클라이언트는 파일 복사본을 다운로드 해야 함
  - ✓  $d_{min}$  = 클라이언트들 중 최소 다운로드 속도
  - ✓ 클라이언트의 최대 다운로드 시간:  $F/d_{min}$

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

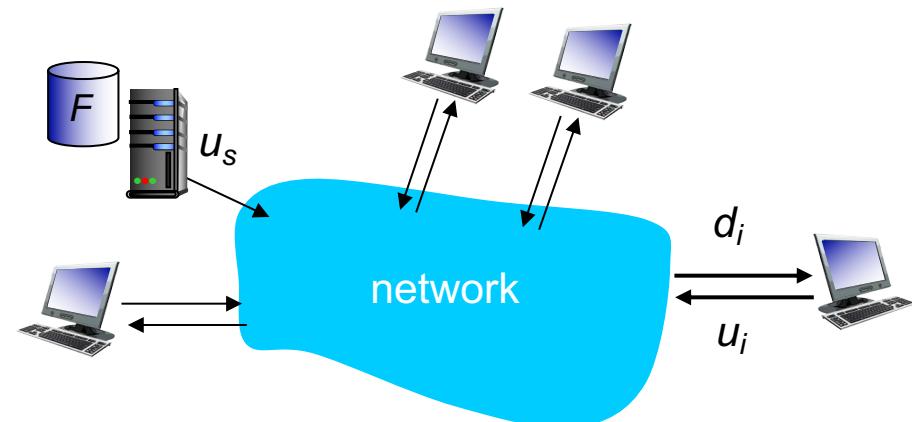
$$D_{c-s} \geq \max\{NF/u_s, F/d_{min}\}$$

피어 수 N에 따라 선형적으로 증가

# 파일 분배 시간: P2P

## ■ 서버

- 1개의 복사본만 전송
  - ✓ 1개의 복사본 전송 시간:  $F/u_s$



## ■ 클라이언트

- 각 클라이언트는 파일 복사본을 다운로드 해야 함
  - ✓ 클라이언트의 최대 다운로드 시간:  $F/d_{min}$

## ■ 클라이언트들

- 다운로드 총량은  $NF$  비트
  - ✓ 시스템 전체 업로드 속도는 서버 업로드 속도 + 각 피어 업로드 속도:  $u_s + \sum u_i$

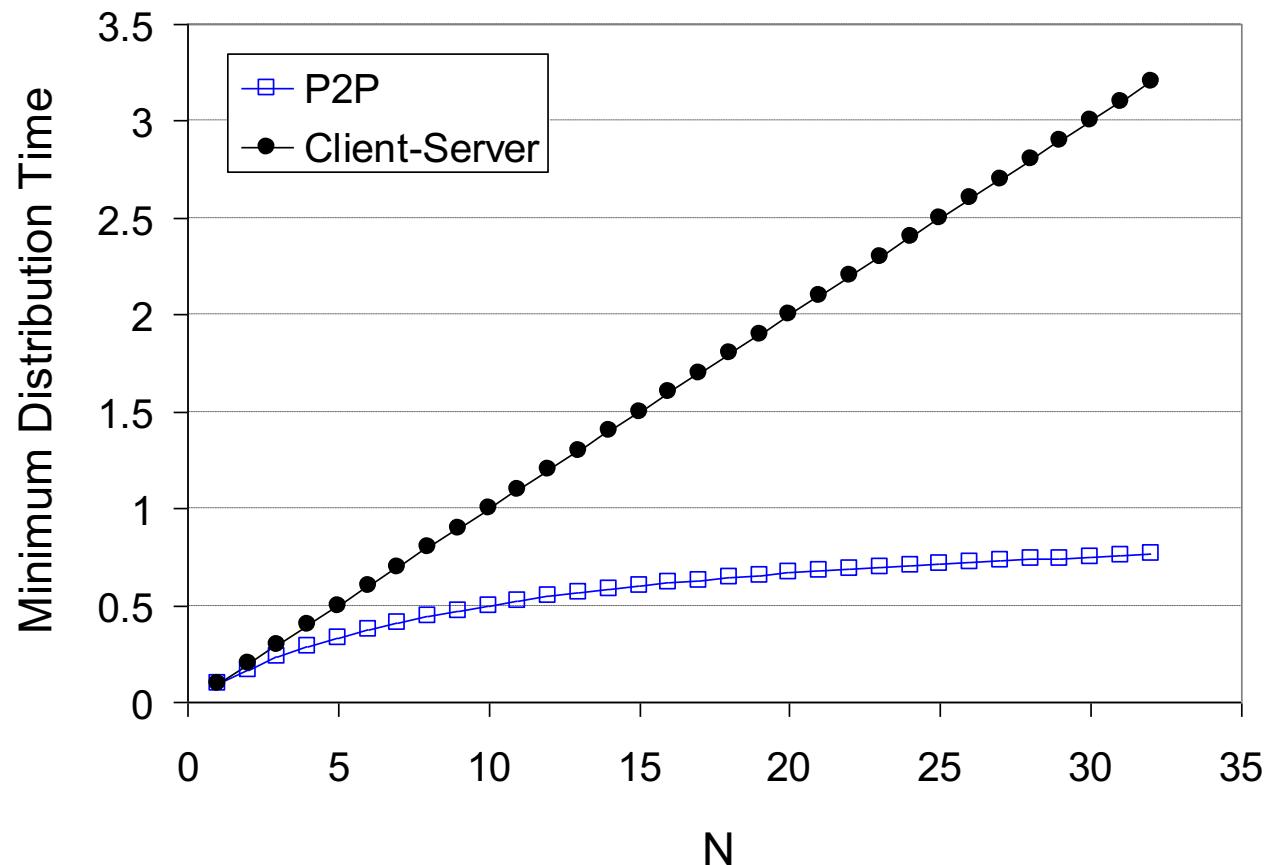
Time to distribute  $F$   
to  $N$  clients using  
P2P approach

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

피어 수  $N$ 에 따라 선형적으로 증가  
각 피어가 업로드도 수행

# 클라이언트-서버 vs P2P 비교 예제

- 클라이언트 업로드 속도 =  $u$ ,  $F/u = 1 \text{ hour}$ ,  $u_s = 10u$ ,  $d_{min} \geq u_s$

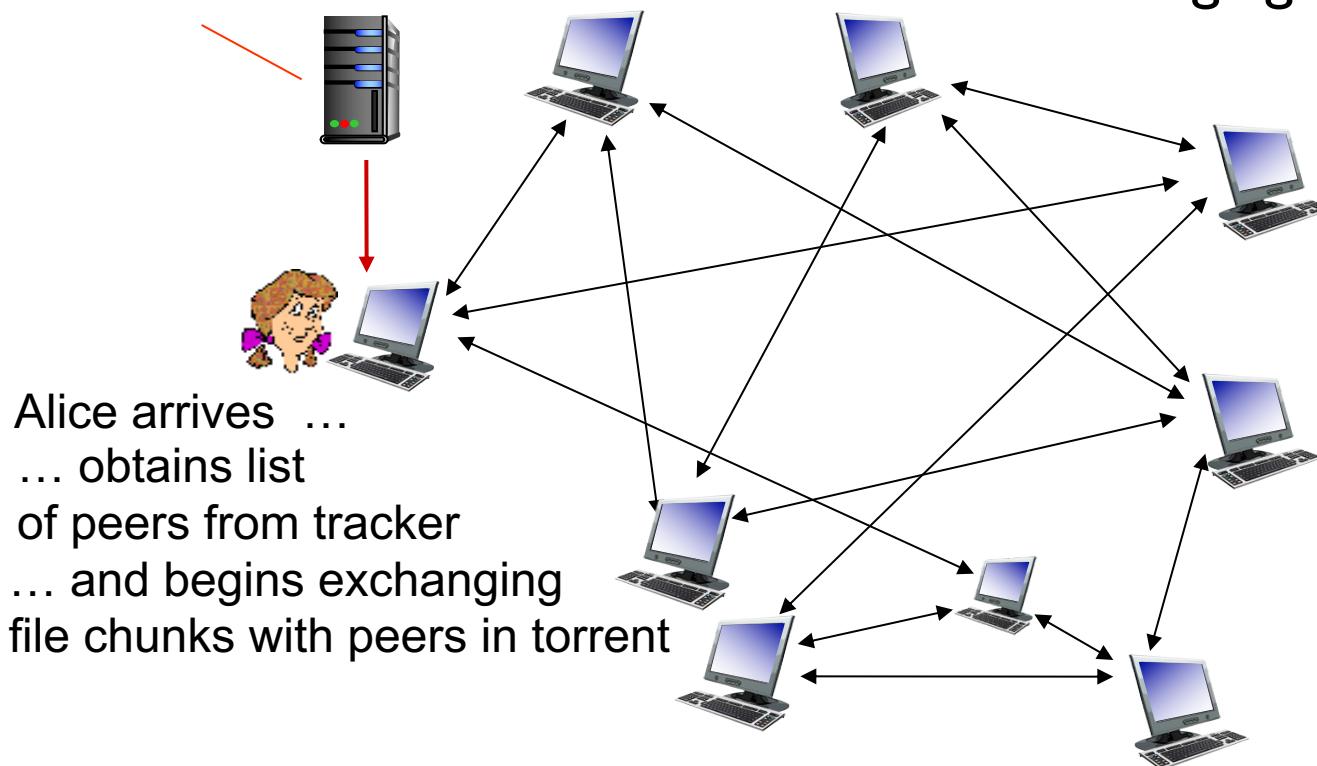


# P2P 파일 분배: BitTorrent

- 각 파일을 256KB 청크(chunk)로 분할
- 피어들은 파일 청크를 주고 받음

**tracker:** tracks peers  
participating in torrent

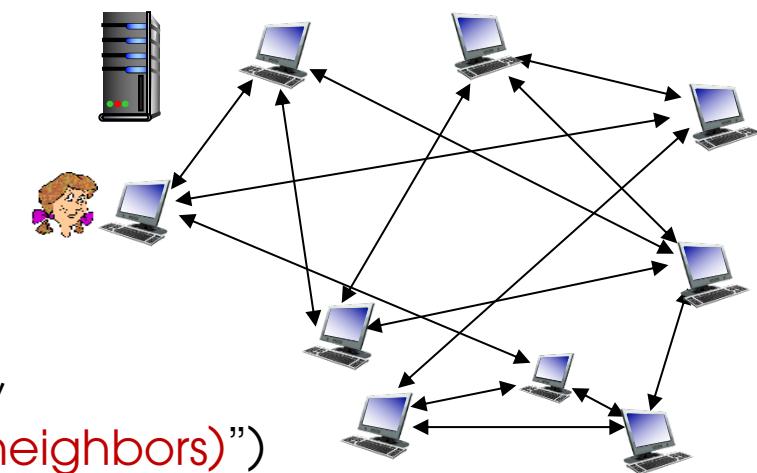
**torrent:** group of peers  
exchanging chunks of a file



# P2P 파일 분배: BitTorrent

## ■ 토렌트에 참여하는 피어는

- 처음에는 청크가 없으나, 시간이 지남에 따라 다른 피어들로부터 청크를 획득
- “트래커”에 등록한 후 피어 리스트를 얻고, 이들 중 일부 피어에 접속함 (“이웃 피어(neighbors)”)



## ■ 피어의 동작

- 피어는 청크를 다운로드하는 동안, 다른 피어들에게 청크를 업로드 함
- 피어는 청크를 교환하는 상대 피어들을 변경할 수 있음
- 피어는 전체 파일을 얻은 후, 자유롭게 토렌트를 (이기적으로) 떠나거나 또는 (이타적으로) 남아 있을 수 있음

# P2P 파일 분배: BitTorrent (파일 청크 요청 및 전송)

## ■ 청크 요청

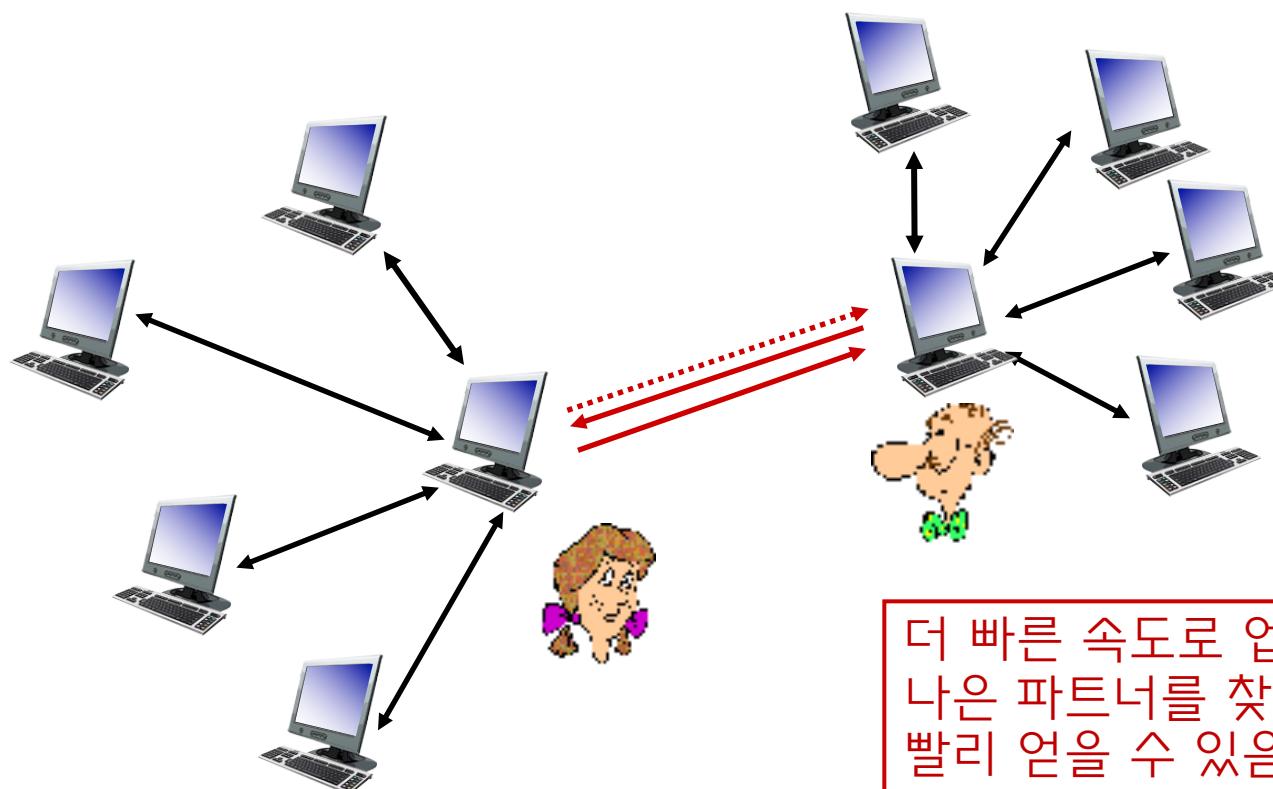
- 특정 시간에 서로 다른 피어들은 파일의 서로 다른 청크를 가지고 있음
- 주기적으로, 앤리스는 이웃 피어들에게 각자 가지고 있는 청크 리스트를 요청함
- 앤리스는 자기가 가지고 있지 않은 청크를 요청
  - ✓ 이웃들이 가지고 있는 청크 중 가장 개수가 적은 것을 먼저 요청함 (rarest first)

## ■ 청크 전송

- tit-for-tat
- 앤리스는 가장 빠른 속도로 청크를 보내는 4개의 이웃 피어들에게 청크를 전송
  - ✓ 다른 피어들은 앤리스로부터 청크를 받지 못함
  - ✓ 매 10초마다 가장 빠른 4개의 이웃 피어를 다시 선택
- 매 30초마다, 랜덤하게 1개의 다른 피어를 추가 선택하여 청크 전송
  - ✓ 특정 피어가 소외되지 않도록 낙관적으로 전송
  - ✓ 새로 선택된 피어가 가장 빠른 4개의 피어에 포함될 수 있음

# BitTorrent: tit-for-tat

- (1) 앤리스는 랜덤하게 낙관적으로 밥을 선택하여 전송 (optimistically unchoke)
- (2) 앤리스는 밥의 가장 빠른 4개의 제공자 중 하나가 됨. 밥이 앤리스에게 전송
- (3) 밥도 앤리스의 가장 빠른 4개의 제공자 중 하나가 됨.



더 빠른 속도로 업로드하면, 더  
나은 파트너를 찾고, 파일을 더  
빨리 얻을 수 있음

# 애플리케이션 계층

2.1 네트워크 애플리케이션의 원리

2.2 웹과 HTTP

2.3 전자메일

- SMTP, POP3, IMAP

2.4 DNS - 인터넷 디렉토리 서비스

2.5 P2P 파일 분배

2.6 비디오 스트리밍과 컨텐츠 분배 네트워크

2.7 소켓 프로그래밍: 네트워크 애플리케이션 작성

# 비디오 스트리밍과 CDN Contents Distribution Network

## ■ 비디오 트래픽이 인터넷 대역폭의 주요 소비자가 되고 있음

- Netflix, YouTube, Amazon Prime: 가정용 ISP 트래픽의 80% 차지 (2020)
- ~2B YouTube 사용자, ~160M Netflix 사용자

## ■ 해결 과제

- 확장성scalability: 10억 명 이상의 사용자에게 비디오 스트리밍 서비스를 제공할 수 있어야 함
  - ✓ 하나의 초대형 비디오 서버로 운영할 수 없음
- 이질성heterogeneity: 서로 다른 환경의 서로 다른 사용자에게 서비스를 제공할 수 있어야 함
  - ✓ 무선/유선 사용자, 높은 대역폭/낮은 대역폭의 사용자



## ■ 해결책

- 분산화된 애플리케이션 레벨의 인프라스트럭처

# 멀티미디어: 비디오

## ■ 비디오

- 일정 속도로 보여지는 연속적인 **이미지**
- 예) 24 images/sec

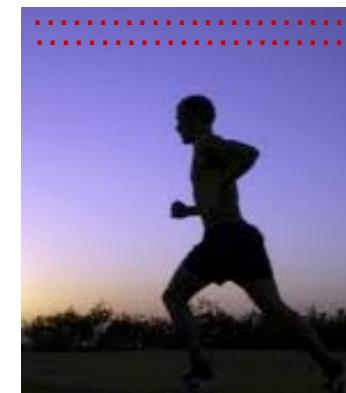
## ■ 디지털 이미지: **픽셀의 배열**

- 각 픽셀은 일정한 수의 비트로 표현

## ■ 압축(coding)

- 이미지 인코딩 시 이미지 크기를 줄이기 위해 **이미지 내 또는 이미지 간의 중복**<sup>redundancy</sup>된 내용을 활용
- 공간적 중복 <sup>spatial redundancy</sup>
  - ✓ 이미지 내의 중복
- 시간적 중복 <sup>temporal redundancy</sup>
  - ✓ 연속된 이미지 간의 중복

*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  $i$



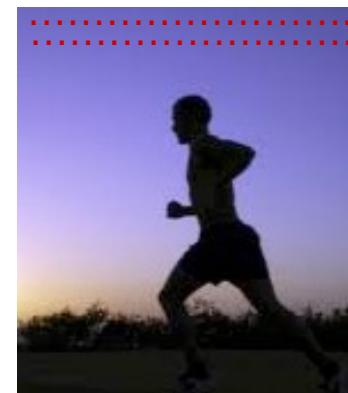
frame  $i+1$

# 멀티미디어: 비디오

## ■ CBR constant bit rate

- 고정 비트레이트
- 단위 시간 당 출력하는 데이터 크기가 고정

*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 ( $N$ )



## ■ VBR variable bit rate

- 가변 비트레이트
- 비디오 인코딩 속도가 공간, 시간적 코딩 변화에 따라 가변

## ■ 예제

- MPEG 1 (CD-ROM) 1.5 Mbps
- MPEG 2 (DVD) 3~6 Mbps
- MPEG 4 (인터넷에서 자주 사용, 64 Kbps~12 Mbps)

frame  $i$

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



frame  $i+1$

# 멀티미디어: 비디오

## ■ 네트워크 측면에서 비디오의 가장 큰 특징

→ “**높은 비트레이트**”

- 예) 4K(UHD) 비디오 스트리밍은 10Mbps 이상의 비트레이트

## ■ 인터넷 비디오는 일반적으로 고화질 스트리밍을 위해 100kbps에서 3Mbps의 비트레이트로 인코딩됨

- 비트레이트가 높을수록 고화질 비디오임

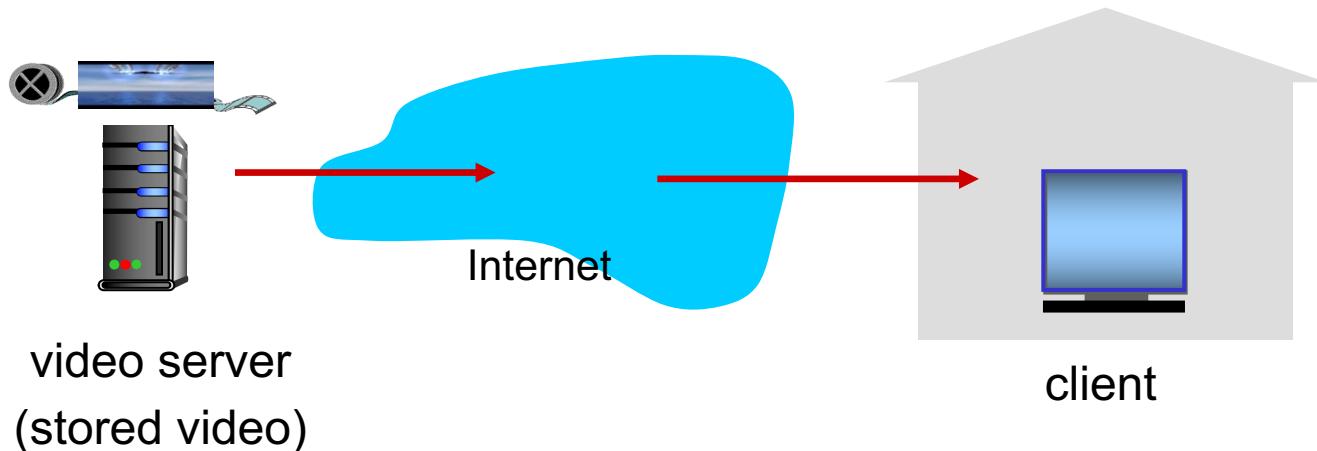
→ 끊김없는 재생을 위해 네트워크는 비트레이트 이상의 평균 처리 속도를 제공해야 함

## ■ 동일한 비디오를 여러 버전의 품질로 인코딩

- 예) “펭수”를 300kbps, 1Mbps, 3Mbps의 세가지 버전으로 생성
- 네트워크 속도에 따라 적절한 버전을 전송
  - ✓ 초고속 인터넷 연결을 가진 사용자는 3Mbps 버전을 선택할 수 있으며, 스마트폰으로 3G를 통해 동영상을 시청하는 경우 300kbps 버전 선택

# 저장 비디오 스트리밍 Streaming Stored Video

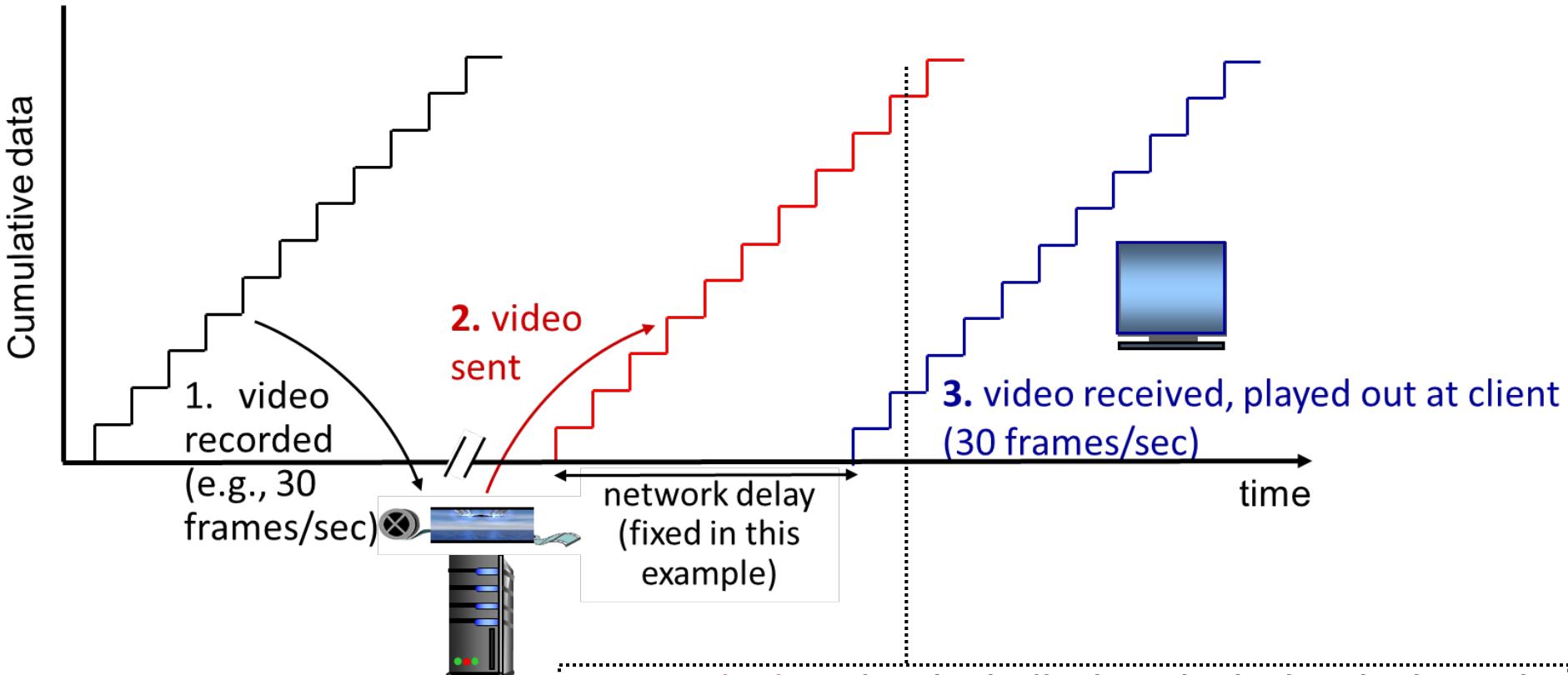
## ■ 간단한 시나리오



## ■ 주요 이슈

- 서버-클라이언트 간 대역폭이 시간에 따라 바뀜
  - ✓ 집, 액세스 네트워크, 네트워크 코어, 비디오 서버 사이의 네트워크 혼잡 수준이 변함
- 혼잡에 따른 패킷 손실, 지연으로 인해 비디오 재생이 끊어지거나 비디오 품질이 나빠짐

# 저장 비디오 스트리밍

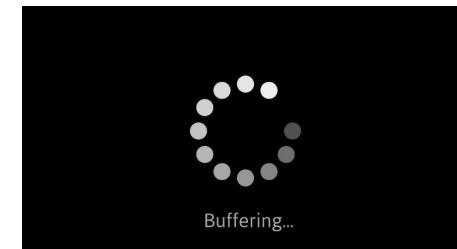


**스트리밍**: 이 시점에서, 서버가 비디오의 후반 부분을 보내는 동안, 클라이언트는 비디오의 후반 부분을 재생하고 있음

# 저장 비디오 스트리밍

## ■ 제약사항

- 클라이언트가 재생을 시작하면, 원래의 타이밍과 맞게 재생되어야 함
- 하지만, 네트워크 지연 delay이 일정하지 않고 가변적임
  - ✓ 이를 **지터 jitter**라고 함
- Q. 어떻게 해결할 수 있을까? A. **클라이언트 쪽 버퍼**

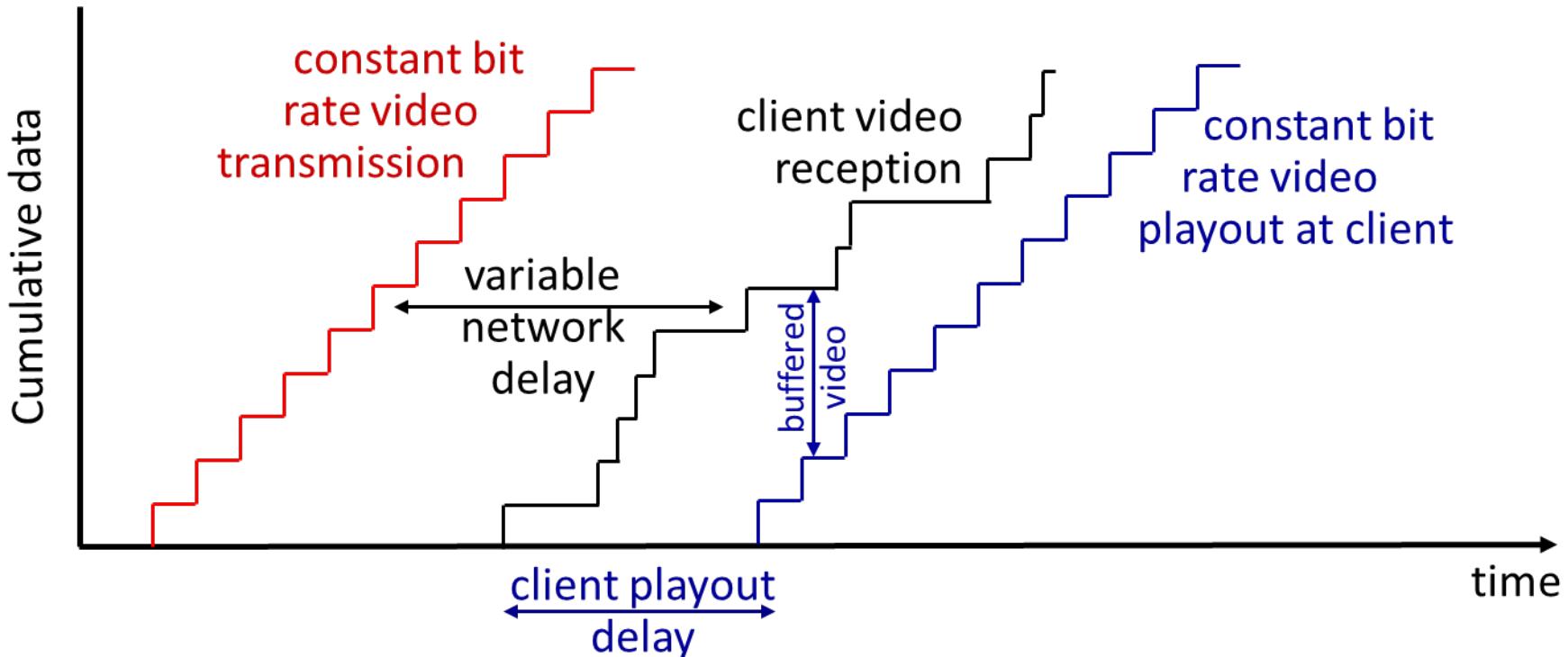


## ■ 기타 고려사항

- 클라이언트와의 상호작용 interaction
  - ✓ pause, fast-forward, rewind, jump 등
- 비디오 패킷들은 손실되고, 재전송될 수 있음

# 저장 비디오 스트리밍: 버퍼링

- 클라이언트 쪽 버퍼링 client-side buffering과 재생 지연 playout delay



# 적응적 스트리밍과 DASH

## ■ DASH Dynamic, Adaptive Streaming over HTTP

- 클라이언트들의 사용 대역폭 차이에 따라 서로 다른 품질 수준의 비디오 인코딩을 선택

## ■ 서버

- 비디오 파일을 여러 개의 비디오 조각(chunk)으로 분할
- 각 조각은 각기 다른 비트레이트로 인코딩되어 저장
- 매니페스트 manifest 파일: 각 비트레이트를 가지는 조각에 대한 URL을 제공

## ■ 클라이언트

- 주기적으로 서버와 클라이언트 간 대역폭을 측정
- 매니페스트 파일을 참조하여, 한 번에 한 조각을 요청
  - ✓ 현재 사용한 대역폭 하에서 가능한 최대 비트레이트의 비디오 조각을 선택
  - ✓ 각 시간대에서 대역폭의 변화에 따라 다른 비트레이트의 비디오 조각을 선택할 수 있음

# 적응적 스트리밍과 DASH

## ■ “지능적”인 클라이언트

- 클라이언트는 버퍼가 비거나, 버퍼가 꽉 차지 않도록 조각의 요청 시점(when)을 결정
- 가용 대역폭을 보면서 요청할 비디오의 비트레이트(what)를 결정
  - ✓ 더 많은 대역폭이 가용하면, 더 높은 품질의 비디오를 요청
- 클라이언트와 가깝거나, 높은 가용 대역폭을 가진 URL 서버(where)를 선택하여 비디오 조각을 요청

Streaming video = encoding + DASH + playout buffering

# 컨텐츠 분배 네트워크 CDN, Contents Distribution Network

## ■ 해결 과제

- 어떻게 동시에 수십만의 사용자들에게 (수백만 개의 비디오 중 선택된) 비디오 스트리밍 서비스를 제공할 것인가?

## ■ 옵션 1: 단일 초고성능의 서버 구축

- 한 번의 장애로 인해 전체 서비스 중단 가능성 single point of failure
- 서버가 네트워크 혼잡의 주요 지점이 됨
- 지역적으로 먼 클라이언트들은 긴 경로 및 긴 전송 시간 소요
- 인기 비디오는 같은 통신 링크 상에서 여러 번 반복 전송

→ 확장성이 없음(doesn't scale)

# 컨텐츠 분배 네트워크

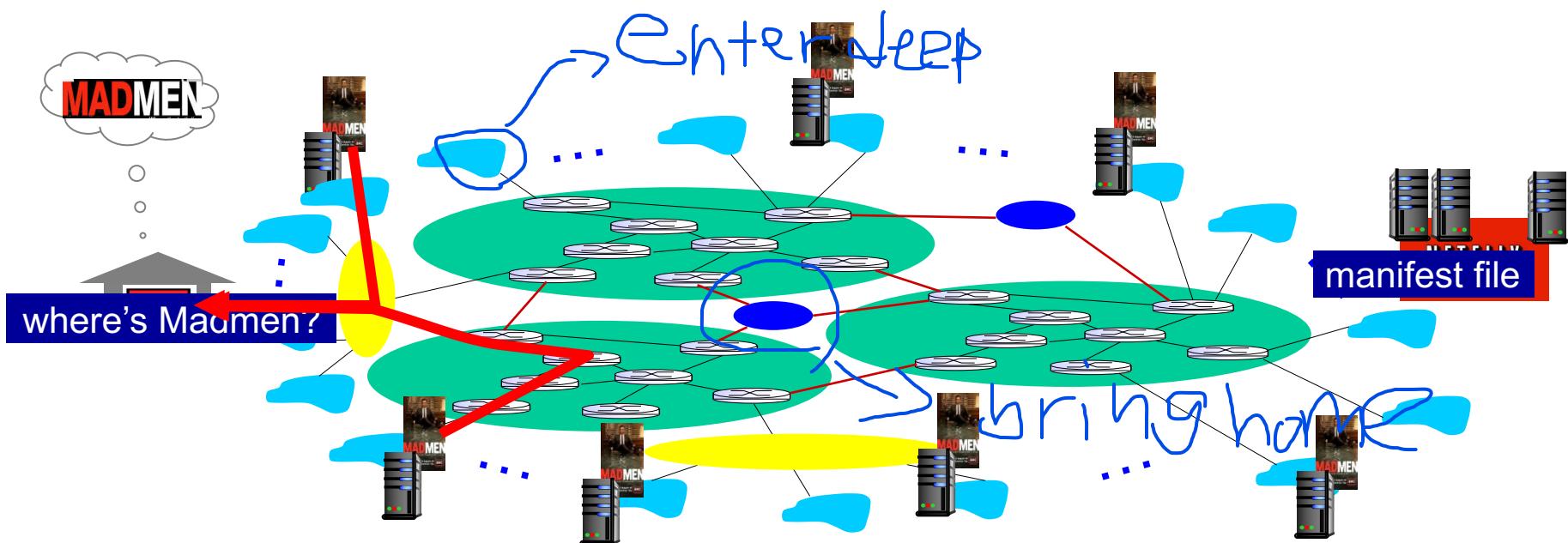
## ■ 해결 과제

- 어떻게 동시에 수십만의 사용자들에게 (수백만 개의 비디오 중 선택된) 비디오 스트리밍 서비스를 제공할 것인가?
- 
- 옵션 2: 다수의 지리적으로 분산된 지점에 비디오들의 복사본을 분산 저장
    - *enter deep*: CDN 서버들이 세계 곳곳의 액세스 네트워크에 깊숙이 들어가 있도록 배치
      - ✓ 최대한 사용자 가까이 위치
      - ✓ Akamai는 120개국 240,000개 이상의 지점에 서버 클러스터를 구축
    - *bring home*: 접속 네트워크의 (내부가 아닌) 근처 POP에 적은 수의 큰 규모 서버 클러스터 구축
      - ✓ Limelight에 의해 사용



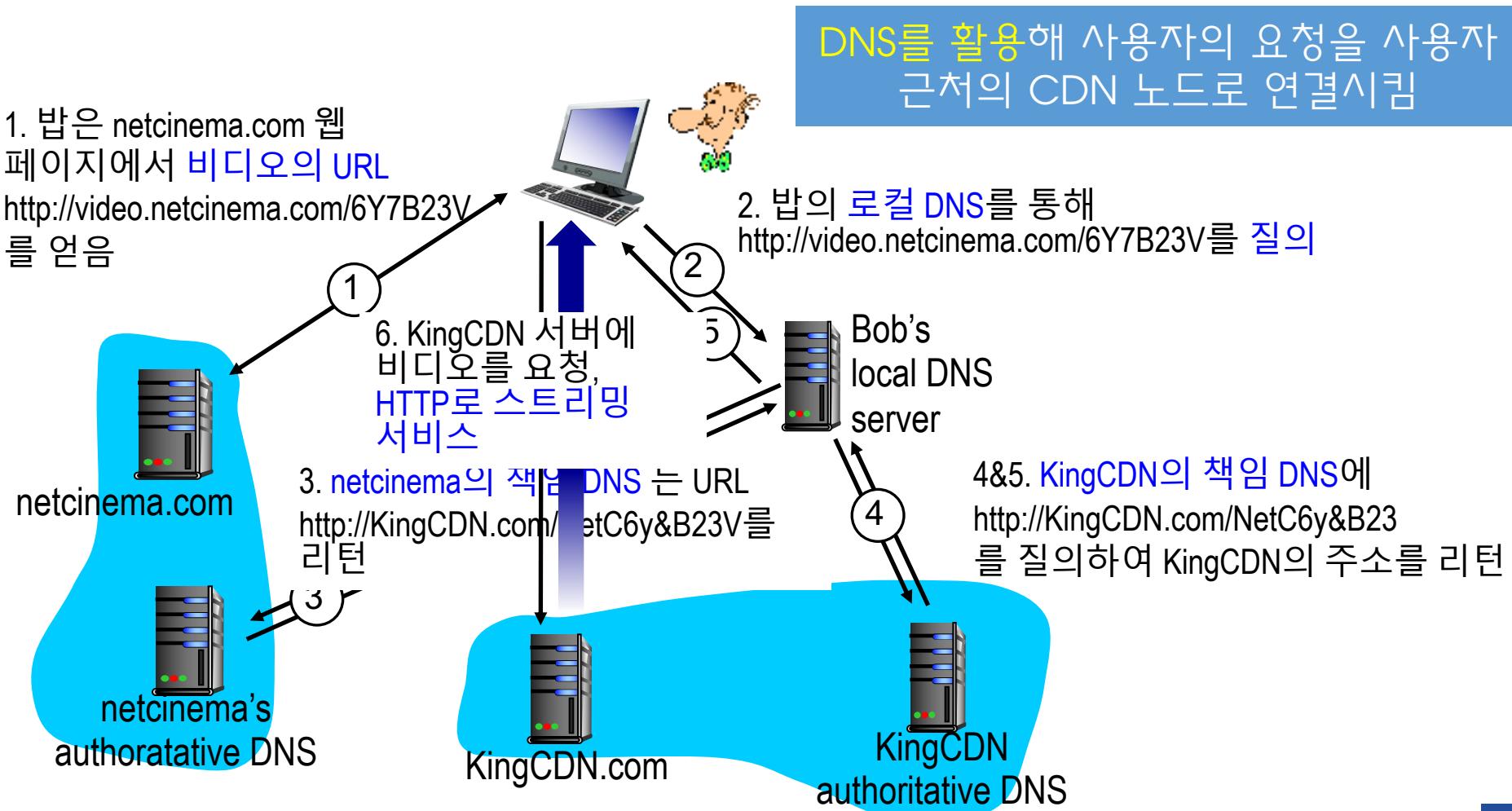
# 컨텐츠 분배 네트워크

- CDN은 CDN 노드들에 컨텐츠의 복사본들을 저장
  - 예) Netflix는 “MadMen”의 복사본을 각 분산 CDN 노드에 저장
- 가입자는 CDN의 컨텐츠를 요청
  - 가입자에 가까운 CDN 노드로 연결되어 컨텐츠를 수신
  - 네트워크의 경로가 혼잡하면 다른 CDN 노드를 선택할 수 있음



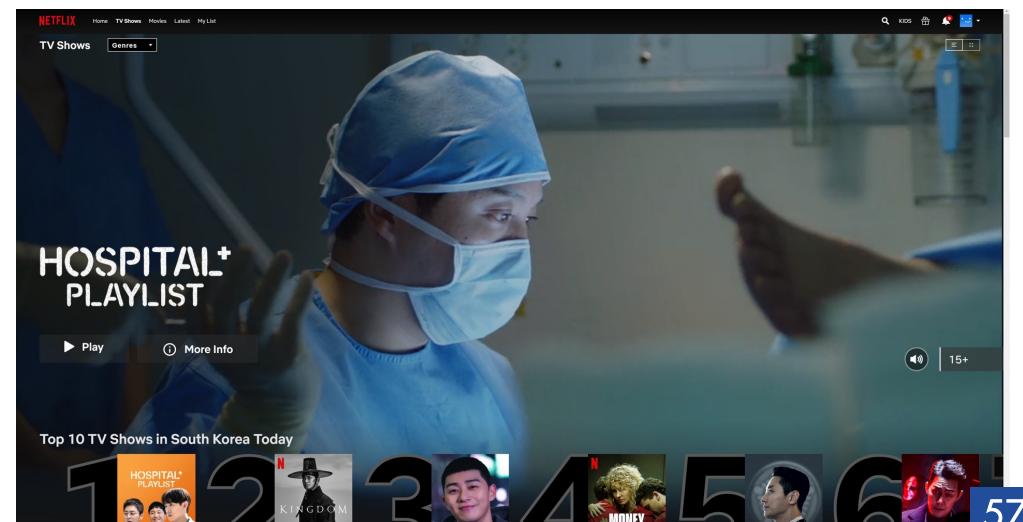
# CDN 컨텐츠 접근 상세 동작

- 밥은 <http://video.netcinema.com/6Y7B23V>에 있는 비디오를 요청
  - 비디오는 CDN <http://KingCDN.com/NetC6y&B23V>에 저장

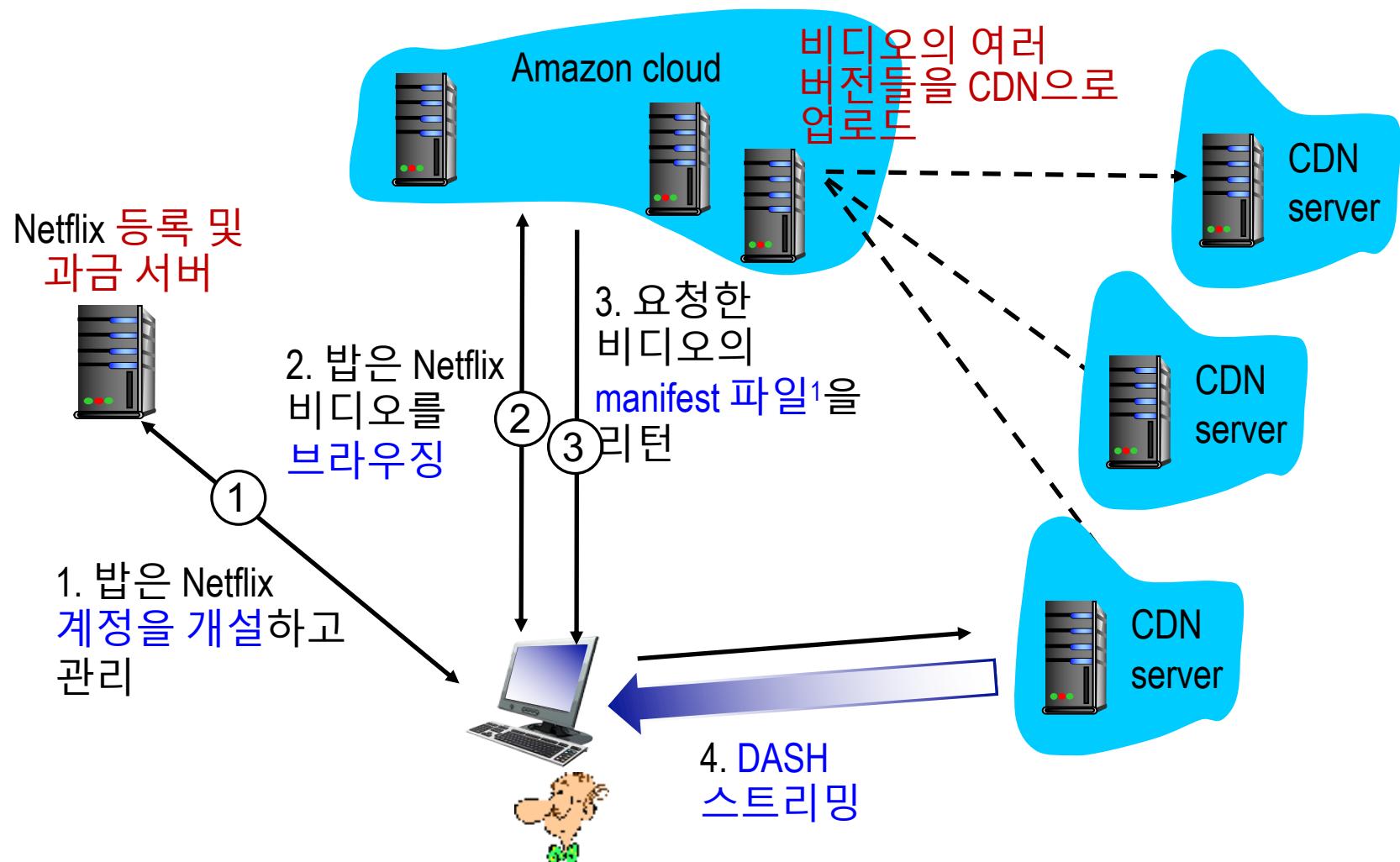


# 사례 연구: Netflix

- 미국의 선도적인 온라인 영화 및 TV 서비스 공급자
- 자신의 고유한 인프라스트럭처를 소유하지 않고 제3자의 서비스(서버, 대역폭, 저장공간, 데이터베이스) 이용
  - 자신의 등록 및 과금 서버 운용
  - Amazon (3<sup>rd</sup> party) 클라우드 서비스 이용
    - ✓ Netflix 영화의 스튜디오 마스터 버전을 Amazon 클라우드에 업로드
    - ✓ 다른 인코딩 비트율을 갖는 여러 가지 형식의 영화를 클라우드에 생성
    - ✓ 클라우드에서 CDN으로 여러 버전의 영화를 업로드
  - 제3자 CDN 서비스 동시 활용
    - ✓ Akamai, Limelight, Level-3



## 사례 연구: Netflix



<sup>1</sup> manifest 파일: CDN 순위 리스트. DASH의 다양한 버전 비디오의 URL 정보 등을 포함

# 요약

## ■ 대표적인 응용 계층 프로토콜

- SMTP, POP3, IMAP
- DNS
- P2P: BitTorrent

## ■ CDN

- 비디오의 특징
- DASH
- CDN 개념 및 사례 연구