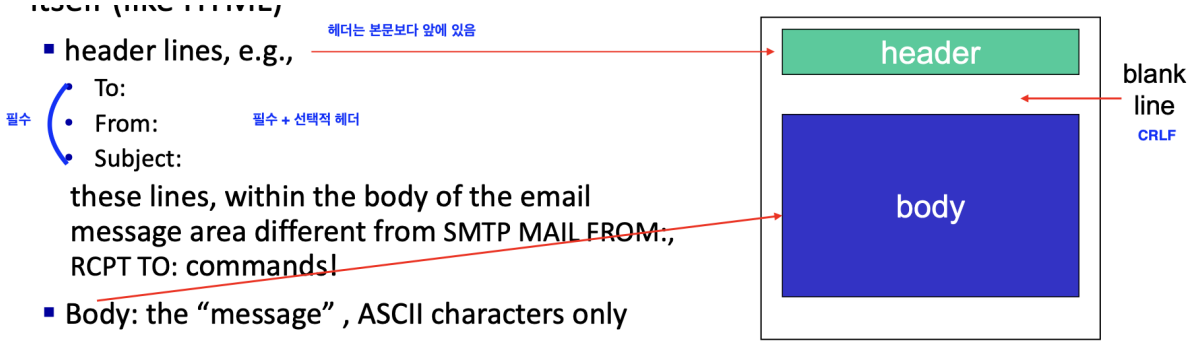
2주차_어플리케이션 계층

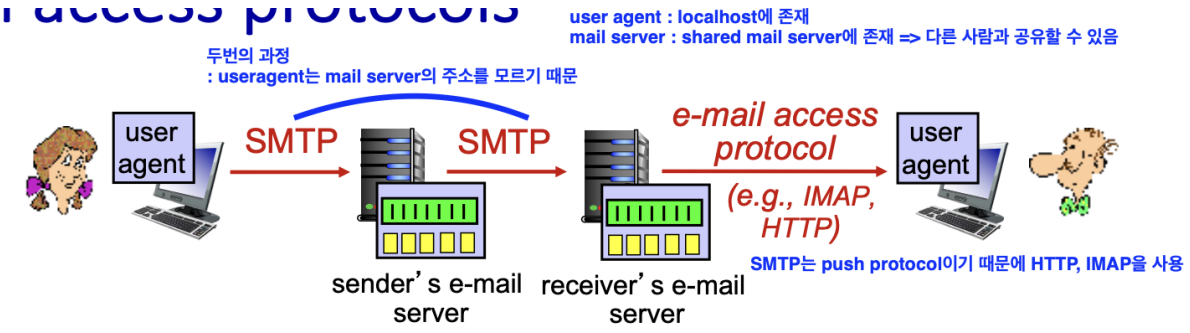
상태	진행 중
강의	CS 스터디
작성일	@2024년 3월 18일

2.3.2 메일 메시지 포맷



- header : body를 보내기 위한 정보들
- body : 실제로 보내고자 하는 내용

2.3.3 메일 접속 프로토콜(Mail access protocols)



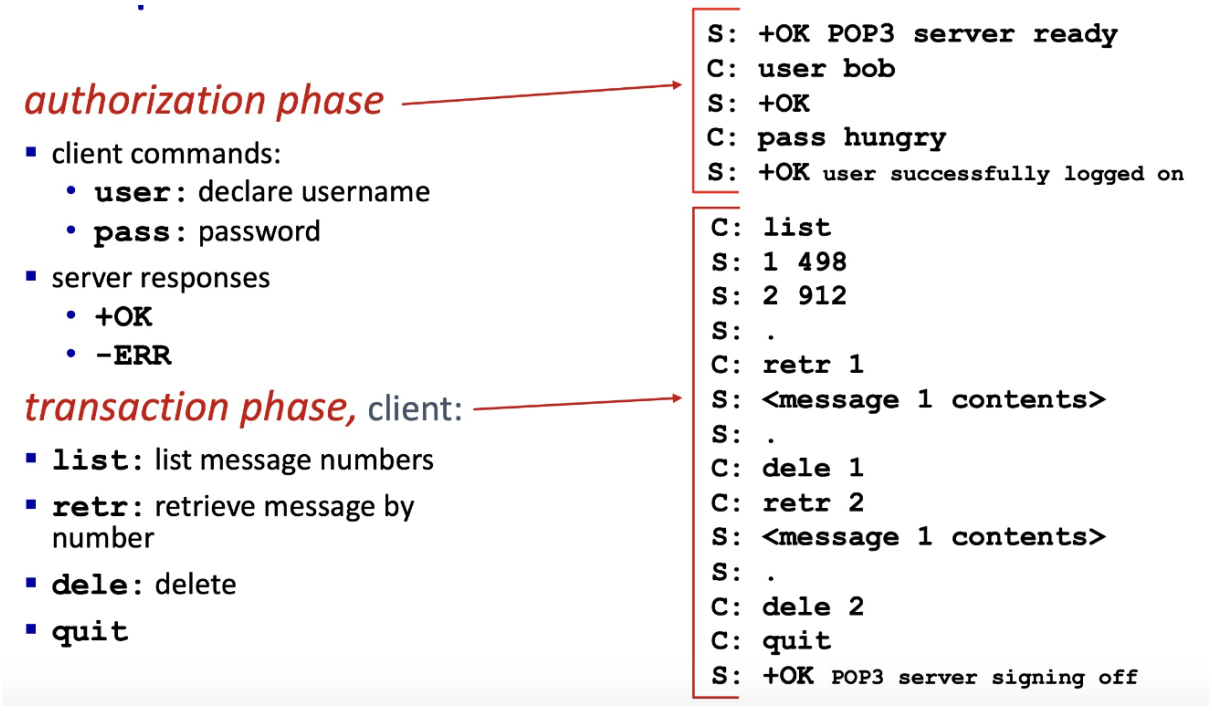
- SMTP : delivery / storage to reciever's server
- Mail access protocol : retrieval from server

→ 메일 서버가 메일 박스를 관리하고, SMTP의 클라이언트와 서버 측 모두를 수행한다.

- POP : Post Office Protocol[RFC 1939] : authorization download
- IMAP : Internet Mail Access Protoco; [RFC 1730] : messages stored on server
- HTTP : gmail, Hotmail, Yahoo! Mail etc. Provides web-based interface on top of SMTP(to send), IMAP(or POP) to returieve e-mail messages

POP3 protocol

authorization phase : 인증 phase



POP3와 IMAP

<POP3>

- download and delete 모드를 사용한 경우 : 만약 내 랩탑에서 download한 다음에 서버에서 delete를 하면, 사무실에 가서 PC에서 서버로 가서 그 메시지를 읽고 싶다? 그럼 불가능한거지!
- download-and-keep 모드를 사용한 경우 : download를 한 경우에소 서버에 그대로 메시지를 갖고 있는 것

<IMAP>

- 서버에 모든 메시지를 갖고 있음
- 메시지를 폴더로 재구성할 수 있어
- user state를 갖고 있어

2.4 DNS: 인터넷의 디렉터리 서비스

2.4.1 DNS(Domain Name System)가 제공하는 서비스

- 사람 구별자 : 주민번호, 이름 등.
- 인터넷 호스트, 라우터 :
  - IP address(32bit) : (ex) 205.251.180.193
  - name : (ex) www.google.com
  - IP는 길이 고정, name은 길이가 가변적이므로 IP주소를 사용하는 것이 속도차이가 크다.
- 그러면 www를 치면 IP주소로 어떻게 변환? → 이것이 DNS의 주요 역할

Domain Name System

- 분산 데이터베이스
  - 많은 name 서버에 구현되어있는 hierarchy

- 애플리케이션 계층 프로토콜
    - 호스트와 name 서버들 간의 address/name translation

DNS : services, structure

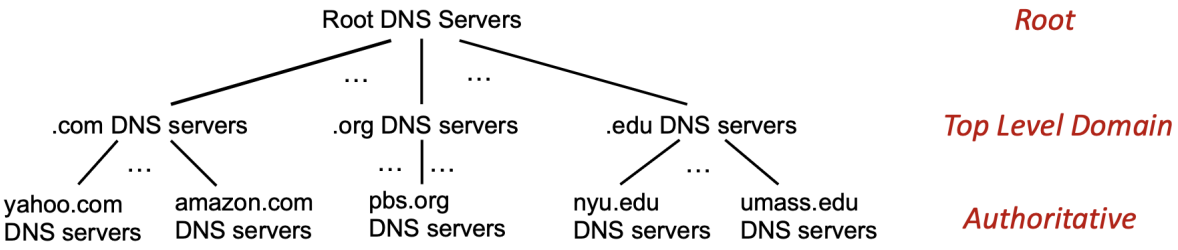
- hostname to IP 주소 변환
- host aliasing
- mail server aliasing
- **부하 분산 (load distribution)**
    - 같은 내용을 가지고 있지만 다른 서버로 traffic이 갈 수 있게끔

→ 왜 DNS를 centralize 하지 않는가?

- 서버의 고장 : 이 네임 서버가 고장 나면, 전체 인터넷이 작동하지 않는다. (single point of failure)
- 트래픽 양의 과부하 : 단일 DNS 서버가 모든 질의를 해결해야 한다.
- 먼 거리의 중앙 집중 데이터베이스: 단일 DNS 서버가 모든 질의 클라이언트로부터 '가까울' 수만은 없다. 즉, 멀면 멀수록 모든 질의가 느려진다.
- 유지 관리
    - 단일 네임 서버는 모든 인터넷 호스트에 대한 레코드를 유지해야 한다.
    - 모든 새로운 호스트를 반영하기 위해 자주 갱신되어야 하고, 사용자에게 호스트를 등록할 수 있도록 허용하는 것과 관련된 인증 문제가 있다.

⇒ 중앙 집중 데이터베이스는 **확장성(scalability)이 전혀 없고**, 결과적으로 DNS는 분산되도록 설계되어있다.

분산(distributed) 계층(hierarchial) 데이터베이스



계층 DNS 서버의 종류

루트(root) name DNS 서버

- 정확한, 인증된 mapping을 갖고 있음
- mapping을 가져와서 local name server에 가져온다.

TLD(최상위 레벨 도메인) DNS 서버

- Top-Level Domain, TLD
- com, org, net 같은 상위 레벨 도메인과 kr, uk 같은 모든 국가의 상위 레벨 도메인에 대한 TLD 서버가 있다.
- Authoritative(책임) DNS 서버에 대한 IP 주소를 제공한다.

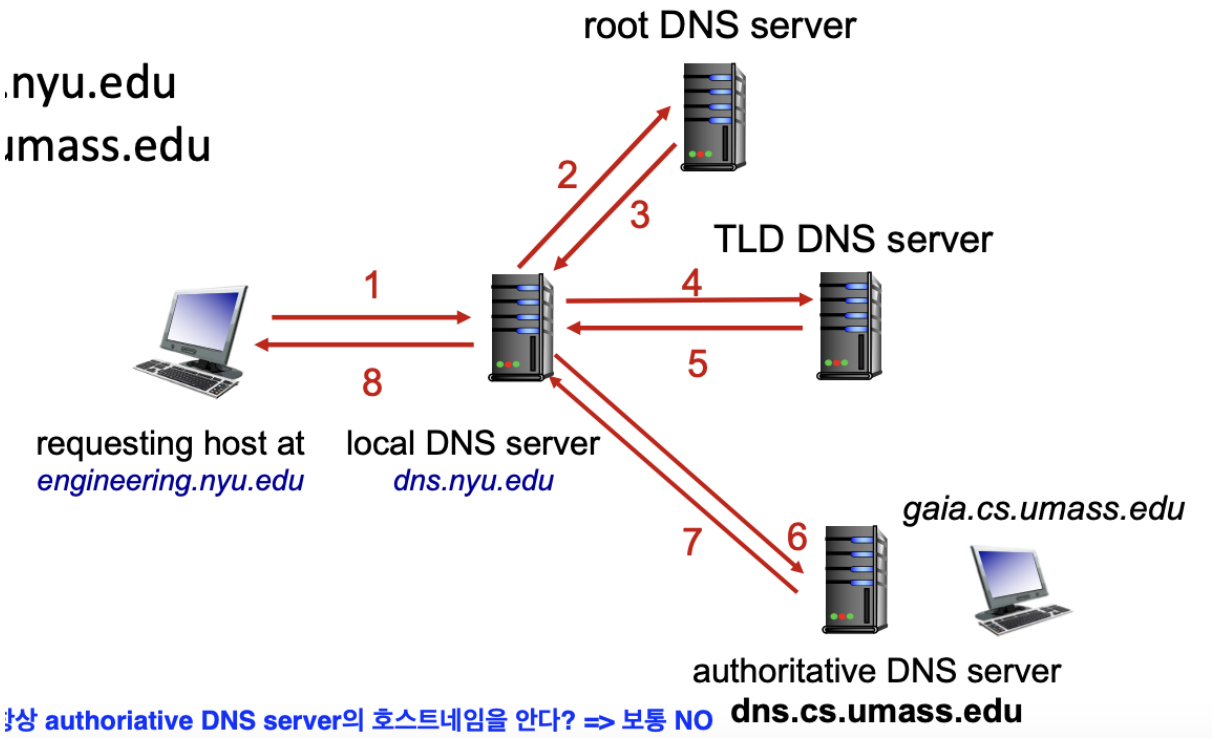
Authoritative(책임) DNS 서버

- 가장 정확한 주소의 mapping을 가시고 있는
- 인터넷에서 접근하기 쉬운 호스트를 가진 모든 기관은 호스트 이름을 IP 주소로 매핑하는 공개적인 DNS 레코드를 제공해야 한다.
    - 기관의 책임 DNS 서버는 이 DNS 레코드를 갖고 있다.

local(로컬) DNS 서버

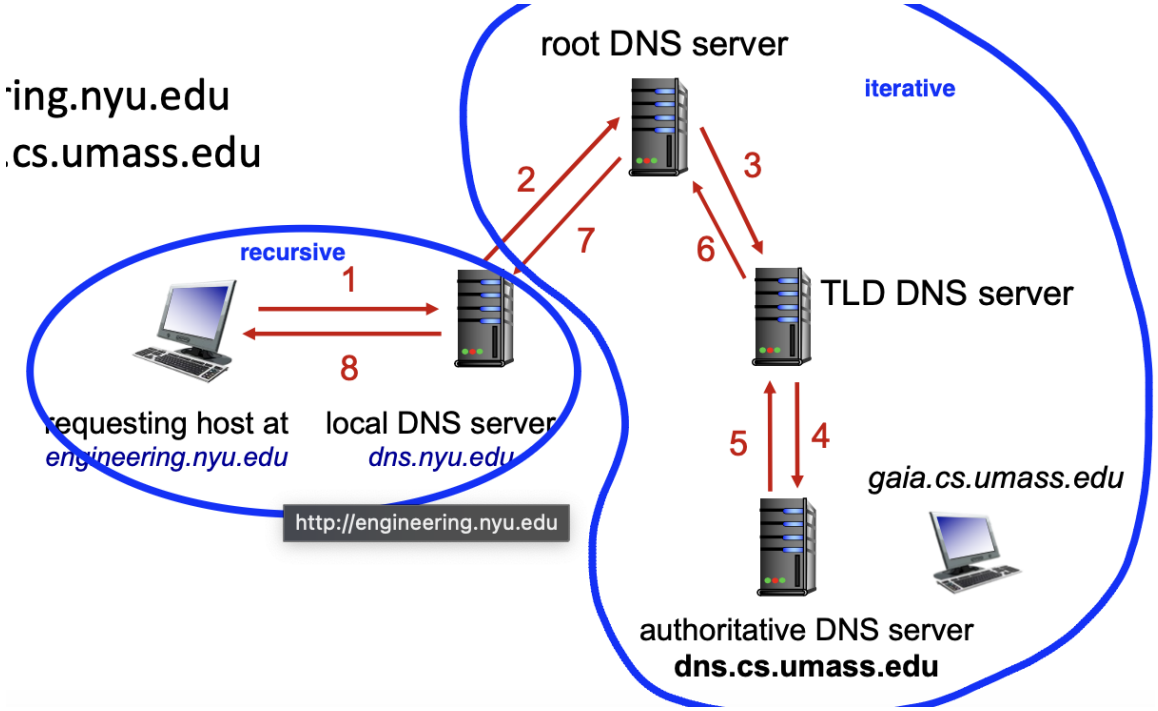
- 로컬 DNS 서버는 서버들의 계층 구조에 엄격하게 속하지는 않지만 DNS 구조의 중심에 있다.
- 각각의 ISP는 로컬 DNS 서버를 갖고, 로컬 DNS 서버로부터 IP 주소를 호스트에게 제공한다.
- 대체로 호스트에 가까이 있기 때문에 지연이 적다.

DNS name resolution 예시



cse.nyu.edu가 gaia.cs.umass.edu의 IP 주소를 원한다고 가정해보자.

1. 자신의 로컬 DNS 서버에 질의를 보낸다. 이때 변환하고 싶은 호스트의 이름을 같이 보낸다.
2. 로컬 DNS 서버는 그 질의 메시지를 루트 DNS 서버에게 전달한다.
3. 루트 DNS 서버는 edu를 인식하고, edu에 대한 책임을 가진 TLD 서버의 IP 주소 목록을 로컬 DNS 서버에 보낸다.
4. 로컬 DNS 서버는 TLD 서버에 질의를 보낸다.
5. TLD 서버는 umass.edu를 인식하고, dns.umass.edu로 이름 지어진 책임 DNS 서버의 IP 주소로 응답한다.
6. 로컬 DNS 서버는 작접 책임 DNS 서버로 질의 메시지를 다시 보낸다.
7. 최종 gaia.cs.umass.edu의 IP 주소를 응답한다.
8. 호스트에 최종 IP 주소를 응답한다.



recursive query

DNS 캐싱

- 한번 name server가 mapping을 발견하면 캐싱(저장)한다.
  - 캐시 entries가 시간이 지나면 사라져 (TTL) - soft state
  - TLD서버들은 대개 로컬 name 서버에 캐싱되어있어
- 캐시가 out-of-date 상황일 수있어 (최선을 다하지만 보장 ㄴㄴ best-effort)
  - 어떤 애는 가지고 있을 수 있고 가지고 있지 않은 애들이 있을 수있지만 잘못된 정보를 가진 애는 없다.

2.4.3 DNS 레코드와 메시지

DNS

distributed database storing rescouce records(RR

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

type=A

Address

- Name : 호스트 이름(hostname)
- Value : 호스트 이름에 대한 IP 주소

type = NS

Name Server

- Name : 도메인(domain)
- Value : 도메인 내부의 호스트에 대한 IP 주소를 얻을 수 있는 방법을 아는 책임 DNS 서버의 호스트 이름

type = CNAME

Canonical NAME

- Name : 정식 호스트 이름의 alias name
- Value : 별칭 호스트 이름 Name에 대한 정식 호스트 이름

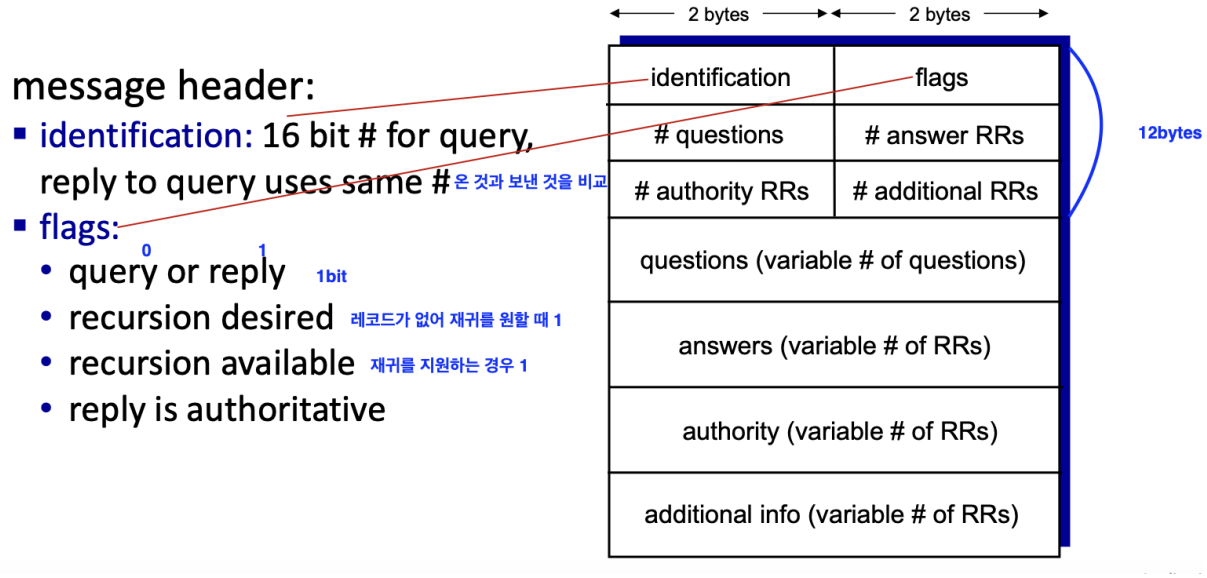
type=MX

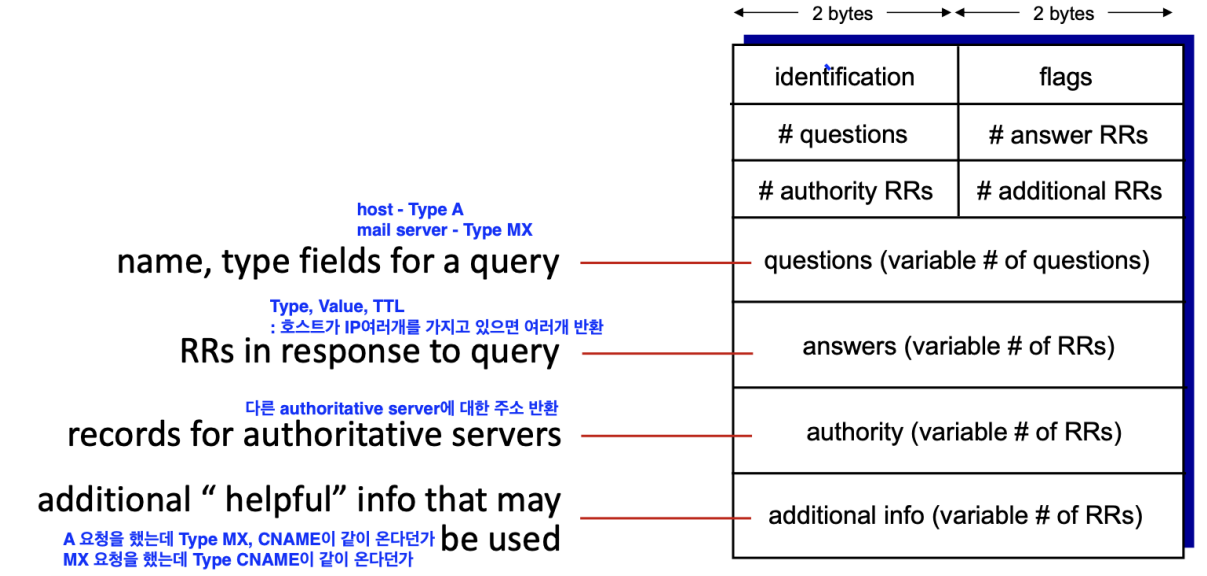
Mail eXchange

- Value : 별칭 호스트 이름 Name을 갖는 메일 서버의 정식 이름
- MX 레코드는 메일 서버의 호스트 이름이 간단한 별칭을 갖는 것을 허용한다.

DNS 프로토콜, 메시지

query(요청)와 reply(응답)메시지는 같은 메시지 형식을 가진다.





DNS 데이터베이스에 레코드 삽입

도메인 네임 networkutopia.com을 등록 기관(DNS registrar)에 등록한다고 가정해보자.

이전에는 작은 등록기관이 독점했었지만, 이제는 많은 기관이 경쟁하고 ICANN이 이러한 여러 등록기관을 승인해준다.

도메인 네임을 어떤 등록기관에 등록할 때 등록 기관에 주책임 서버와 부책임 서버의 이름과 IP 주소를 등록기관에 제공해야 한다.

- 주책임 서버 : `dns1.networkutopia.com` / 주책임 서버 IP : 212.2.212.1
- 부책임 서버 : `dns2.networkutopia.com` / 부책임 서버 IP : 212.2.212.2

위와 같다고 가정하자.

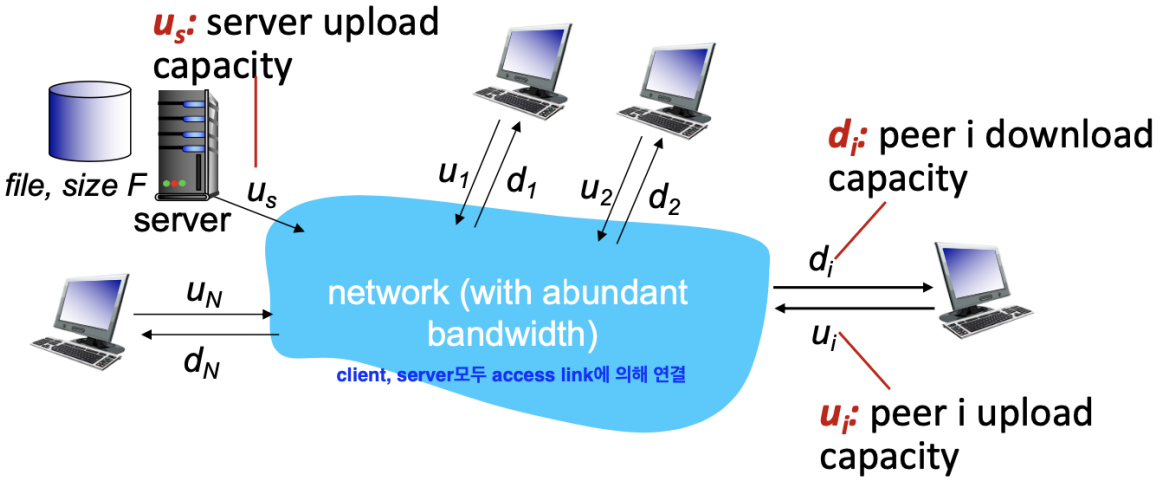
이 두 책임 DNS 서버 각각에 대해 등록 기관은 `Type NS`와 `Type A` 레코드가 **TLD com 서버에 등록되도록 확인한다.**

특히 주책임 서버의 경우 다음 두 개의 자원 레코드를 DNS 서버에 삽입한다.

2.5 P2P 파일 분배

- P2P구조는 항상 켜져있는 인프라스트럭쳐 서버에 최소한으로 의존하고, 간헐적으로 연결되는 쌍들(피어, peer)이 서로 직접 통신한다.

Q. 하나의 서버에서 N개의 peer들까지 얼마나 많은 시간들이 걸리는가?



<client-server 구조 분배 시간>

- 서버 transmission : 별도의 connection을 열어서 n개의 copy를 보내야해
  - 1개의 copy를 보내는 시간 : F/u_s
  - N개의 copy를 보내는 시간 : NF/u_s

- client : 각각의 클라이언트들은 하나의 파일을 다운로드하기만 하면됨
  - d_{min} = 클라이언트들의 다운로드 시간 최소값
  - 최소 다운로드 시간 : F/d_{min}

즉 분배시간 D_{C-S}의 lower bound,

> 💡 $D_{C-S} \geqq$ max{ NF/u_s , F/d_{min}}

<P2P 구조 분배 시간>

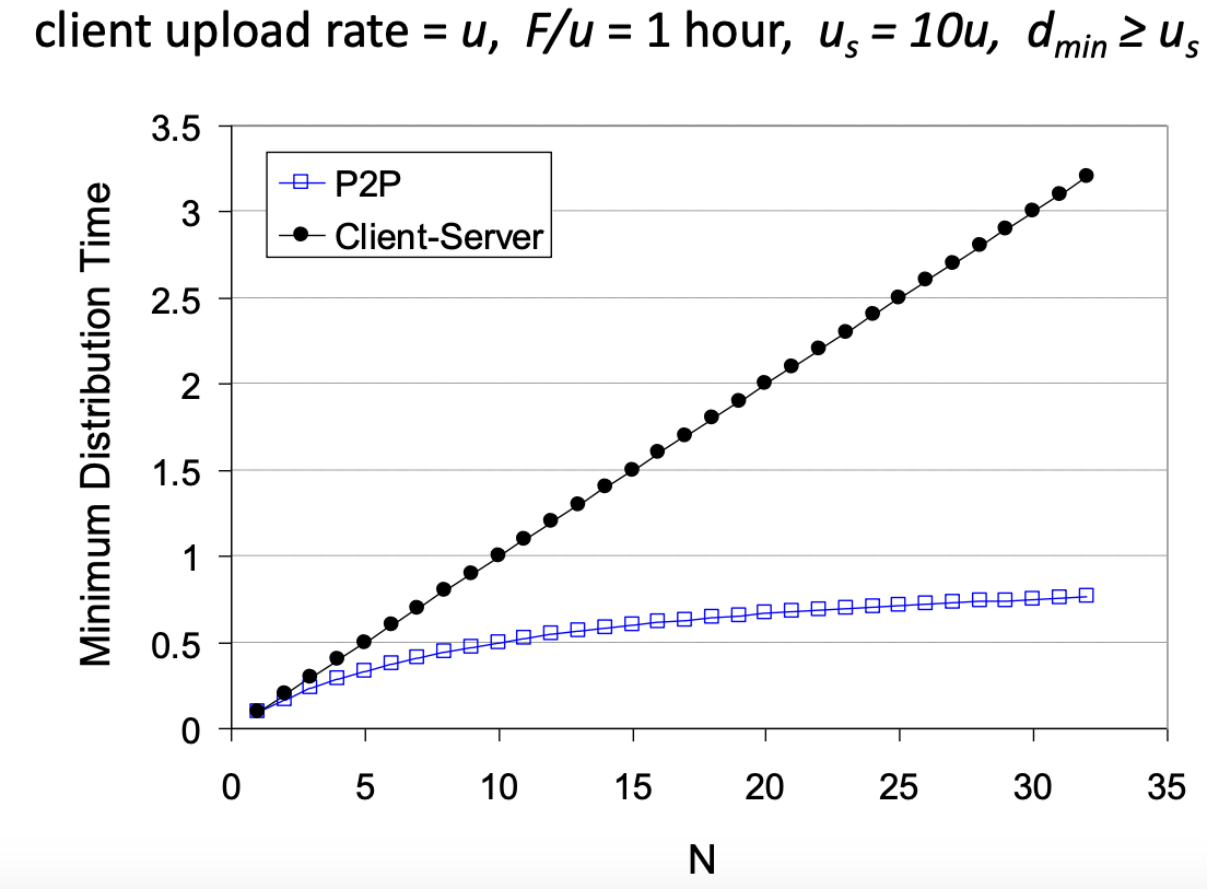
- 서버 transmission :최소 하나의 copy만 업로드하면 됨
  - 1개의 copy를 보내는 시간 : F/u_s

- client : 각각의 클라이언트들은 하나의 파일을 다운로드하기만 하면됨
  - 최소 다운로드 시간 : F/d_{min}
- N명의 클라이언트 전체로 봤을 때, N명의 클라이언트가 F bit씩 받아야하니까 전체는 NF bit
  - 최대 업로드 속도는 u_i + Σu_i ⇒ 거의 N가까이 커짐

즉 분배시간 D_{P2P}의 lower bound,

> 💡 $D_{P2P} \geqq$ max{ F/u_s , F/d_{min}, NF/(u_s+ Σu_i) }

예시

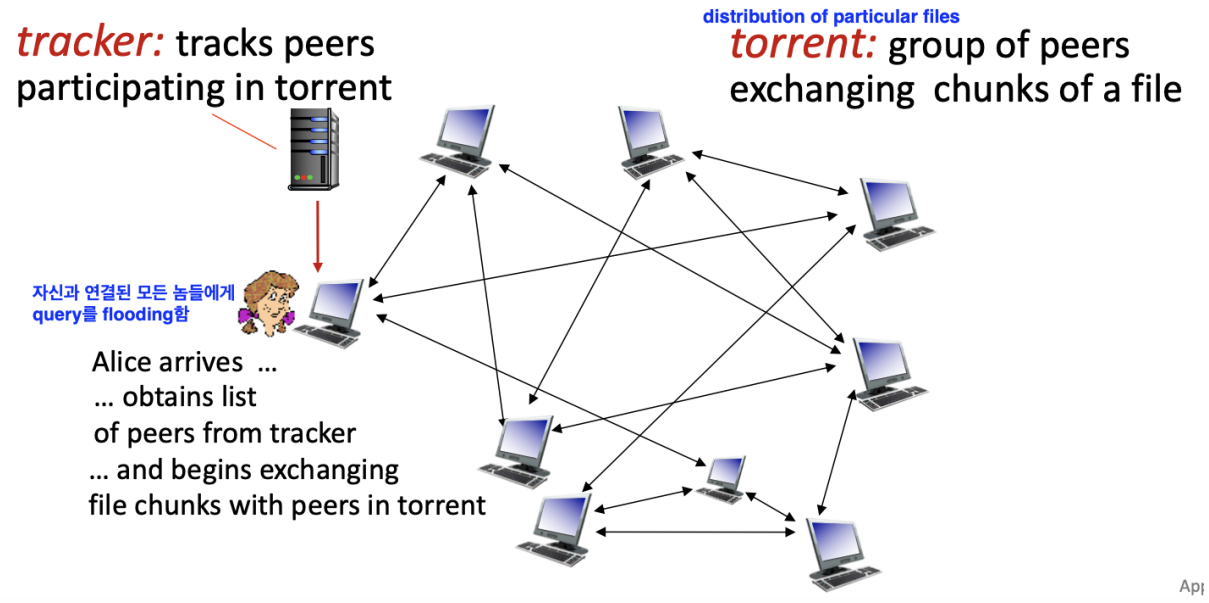


client-server는 linear하게 증가, P2P는 유저수가 늘어나면 늘어날수록 확장성을 가짐(?)

단, 위의 그래프는 lower bound를 비교한 것임을 잊지마시길~

비트 토렌트(BitTorrent)

- 파일들이 256Kb 청크로 나눠져있음
- peer들이 서로에게서 파일 청크들을 주고받아



- 처음으로 가입하면 그 피어에는 청크가 없지만, 시간이 지나면 점점 많은 청크를 쌓을 수 있다.
- 일단 한 피어가 전체 파일을 얻으면 토렌트를 떠나거나, 토렌트에 남아서 다른 피어들로 청크를 계속해서 업로드할 수 있다.

requesting chunks

- 어느 임의의 시간 안에 서로 다른 피어(peer)는 서로 다른 청크의 일부를 가질 것이고
- 앨리스는 이웃들이 어느 청크를 가지고 있는지 자신이 가지고 있는 청크의 목록을 받을 것이고
- 앨리스는 가장 희귀한 청크부터 요청하기 시작

sending chunks : tit-for-tat (눈에는 눈, 이에는 이)

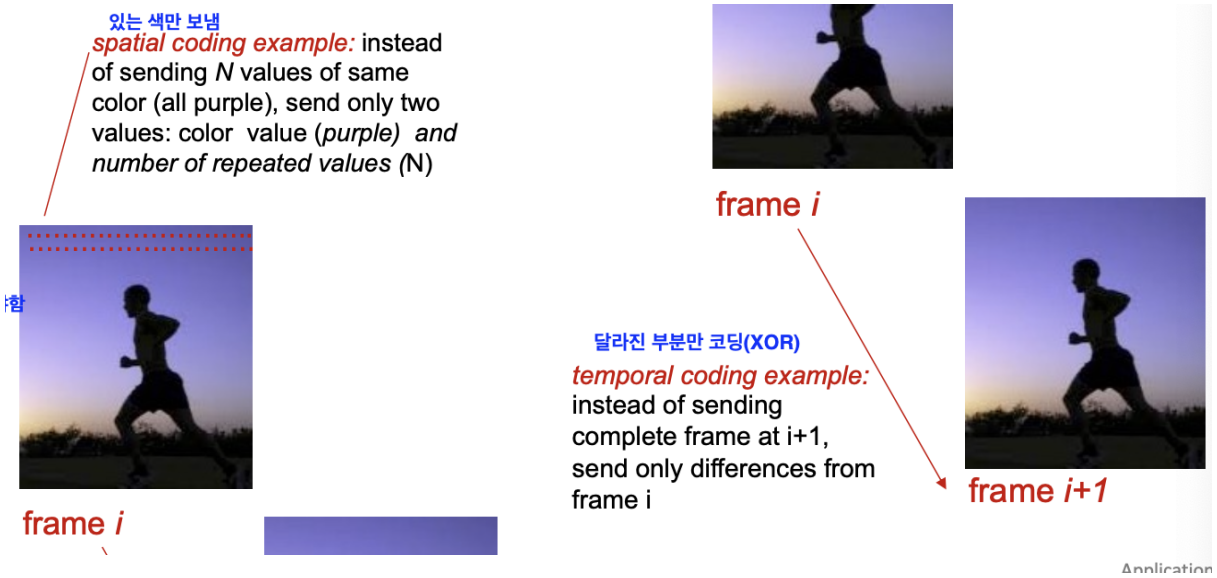
- 네가 나한테 많이 주면 나도 많이 주겠다!
- 가장 높은 속도로 자신에게 청크를 보내는 peer들에게 청크를 보냄
- 계속해서 비트를 수신하는 속도를 측정하고 **가장 빠르게 전송하는 4개의 피어**를 결정하고, 이 4개의 피어에게 청크를 보냄으로써 보답한다.
- 이는 10초마다 계산하여 집합을 수정한다.

- 이런 방식으로 이기적인 사용자 방지

2.6 비디오 스트리밍과 콘텐츠 분배 네트워크

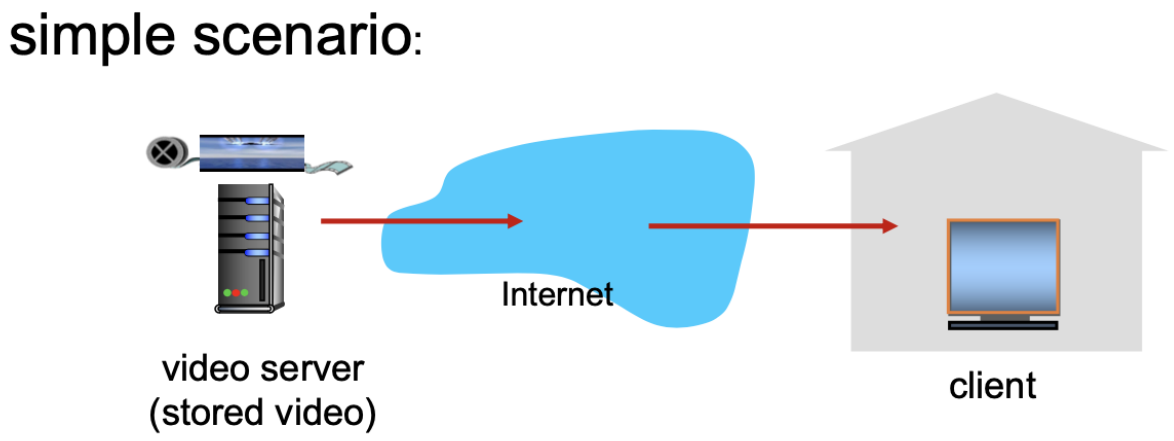
2.6.1 멀티미디어 : 비디오

- 비디오 : 일정한 속도로 일련의 이미지들이 표시되는 것
  e.g. 24images/sec
- 디지털 이미지: 픽셀의 모음
- raw image가 그대로 전송되는 것이 아니라 encoding(압축)을 해서 보내진다.



- CBR(constant bit rate) : 인코딩 속도가 고정 되어있는
- VBR(variable bit rate) : 인코딩 속도가 (인코딩 방법에 따라) 변할 수 있는

Streaming stored video



Streaming multimedia : DASH

- DASH : Dynamic, Adaptive, Streaming over HTTP
- 서버
  - HTTP 서버
  - 비디오 파일을 여러 청크로 나눠서 각기 다른 속도로 인코딩
  - manifest file : 나눠진 청크들의 url을 갖고 있음
- 클라이언트
  - 자기 네트워스 상황(Bandwidth)에 맞게 HTTP를 가져와서 play
- “Intelligence”는 다 클라이언트 쪽에 있다. 결국 모두 클라이언트가 결정하는 것

어려움

어떻게 수많은 유저들에게 컨텐츠를 스트리밍 할 것인가?

옵션 1

단일, 거대 “메가서버” 구축해서 모든 비디오 자료 저장해서 전세계로 모든 사용자에게 비디오 전송

- 한번 잘못되면 끝장
- network conjection

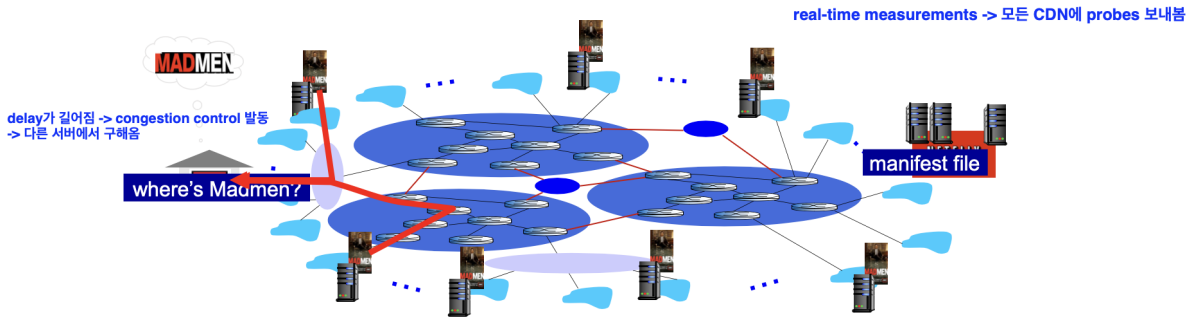
- 멀리 보내면서 네트워크 자원 낭비
- 하나의 비디오 데이터가 여러번 지나는 비효율

⇒ 사용하지 말자^^

옵션 2

Content distribution networks (CDN)

- 다수의 지점에 분산된 서버들을 운영하며, 비디오 및 다른 형태의 웹 콘텐츠 데이터의 복사본을 이러한 분산 서버에 저장한다.
- enter deep
- bring home

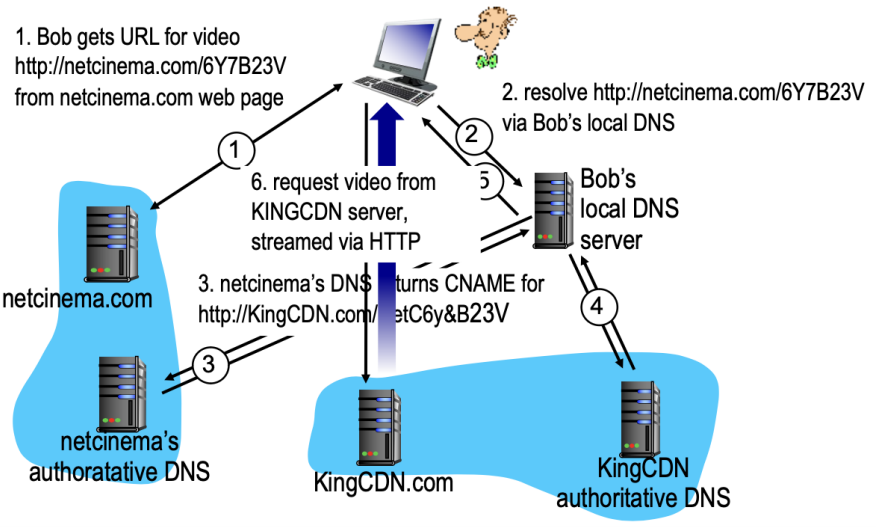


- OTT challenges :
  - 어떤 노드로부터 컨텐츠를 가져올 것인가?
  - 어떤 CDN 노드에 어떤 동영상들을 저장해놓을 것인가?

CDN 동작

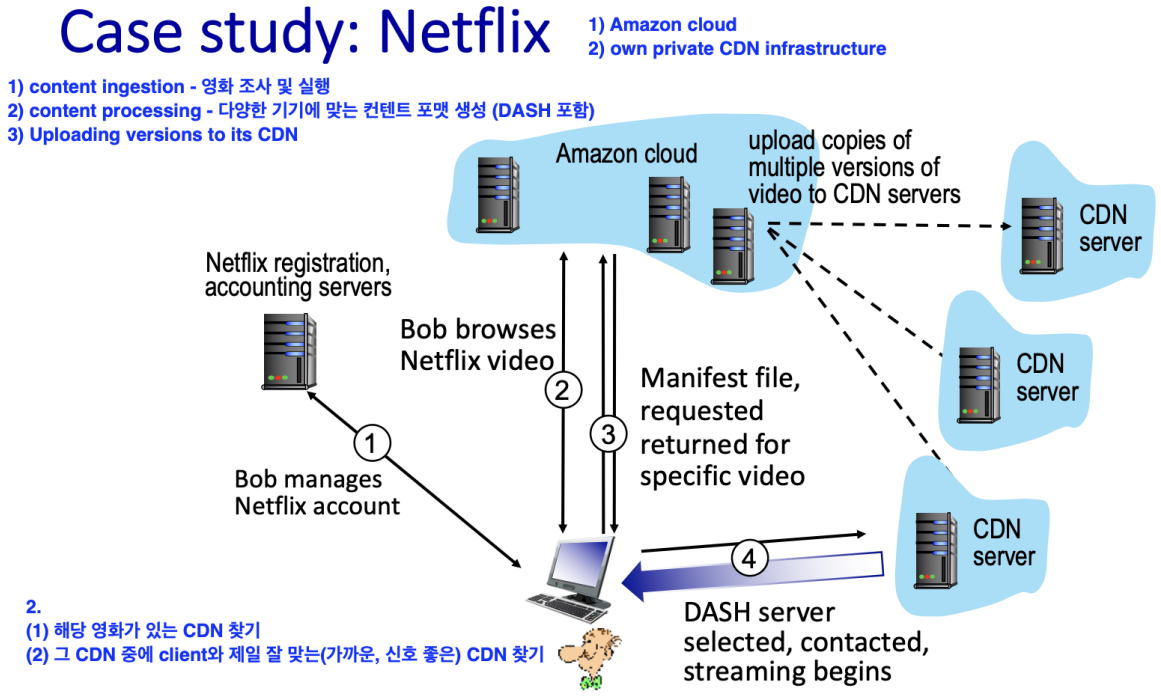
(1) intercept -> DNS의 이점 이용
(2) client와 가장 맞는 CDNserver 결정
(3) client의 요청을 그 서버로 리다이렉트

6. DASH가 사용중이라면,
서버는 manifest file을 response



1. 사용자가 URL을 입력한다.
2. 사용자의 호스트는 URL의 host name에 대한 질의를 로컬 DNS로 보낸다.
3. 로컬 DNS는 host name의 책임 DNS 서버로 질의를 전달한다.책임 DNS 서버는 해당 질의를 CDN 서버로 연결하기 위해 CDN 서버의 책임 DNS 서버의 IP를 전달한다.
4. 로컬 DNS는 CDN 서버의 책임 DNS로 질의를 보내고, CDN 콘텐츠 서버의 IP 주소를 로컬 DNS 서버로 응답한다.이때 클라이언트가 콘텐츠를 전송받게 될 서버가 결정된다.
5. 로컬 DNS 서버는 사용자 호스트에게 CDN 서버의 IP 주소를 알려준다.
6. 클라이언트는 호스트가 알게된 IP 주소로 HTTP 혹은 DASH 프로토콜을 통해 비디오를 받아온다.

2.6.4 Case Study : Netflix

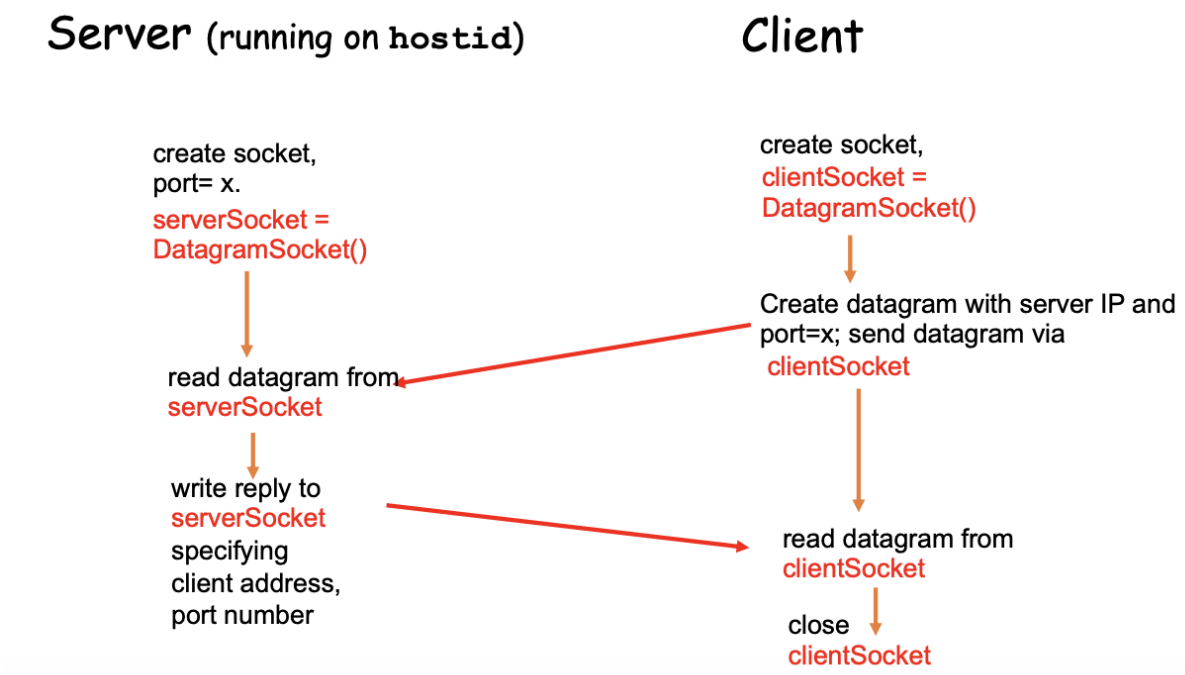


2.7 소켓 프로그래밍 : 네트워크 애플리케이션 생성

- socket : 애플리케이션 프로세스와 end-to-end프로토콜 사이의 문과 같은 역할
- 두가지 소켓 타입
  - UDP : unreliable datagram
  - TCP : reliable byte stream-oriented

2.7.1 UDP 소켓 프로그래밍

- 클라리언트와 서버 사이에 connection이 없다!
  - TCP와 같이 hand shaking 과정이 없고 서버가 열려있다 생각하고 정해진 IP주소로 그냥 보내는 거
  - 데이터를 잃어버릴수도, 순서가 바뀌어 들어올 수도 있음



1. 클라이언트는 키보드로부터 한 줄의 문자를 읽고 그 데이터를 서버로 보낸다.
2. 서버는 그 데이터를 수신하고 문자를 대문자로 변환한다.
3. 서버는 수정된 데이터를 클라이언트에게 보낸다.
4. 클라이언트는 수정된 데이터를 수신하고 그 줄을 화면에 나타낸다.

UDPClient.py

```
# socket module이다. 이 module을 통해 소켓을 생성할 수 있다.
from socket import *

#서버의 IP 혹은 서버의 호스트 이름을 할당한다.
serverName = 'hostname'

# 목적지 port 번호를 나타낸다.
serverPort = 12000

# 클라이언트 소켓을 생성한다. AF_INET은 IPv4를 사용하고 있음을 나타내고,
clientSocket = socket(AF_INET, SOCK_DGRAM)

# 보낼 메시지를 입력 받는다.
```

```
message = Input('Input lowercase sentence:')

# 소켓으로 바이트 형태를 보내기 위해 먼저 encode()를 통해 바이트 타입으로
# sendTo() 메서드는 목적지 주소를 메시지에 붙이고 그 패킷을 프로세스 소켓
# 클라이언트 주소도 같이 보내지는데 이는 자동으로 수행된다.
clientSocket.sendto(message.encode(),(serverName, serverPort)

# 패킷 데이터는 modifiedMessage에 저장되고, 패킷의 출발지 주소(IP, por
# recvfrom() 메서드는 2048의 버퍼 크기로 받아들인다.
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

# 출력
print(modifiedMessage.decode())

# 소켓 닫기
clientSocket.close()
```

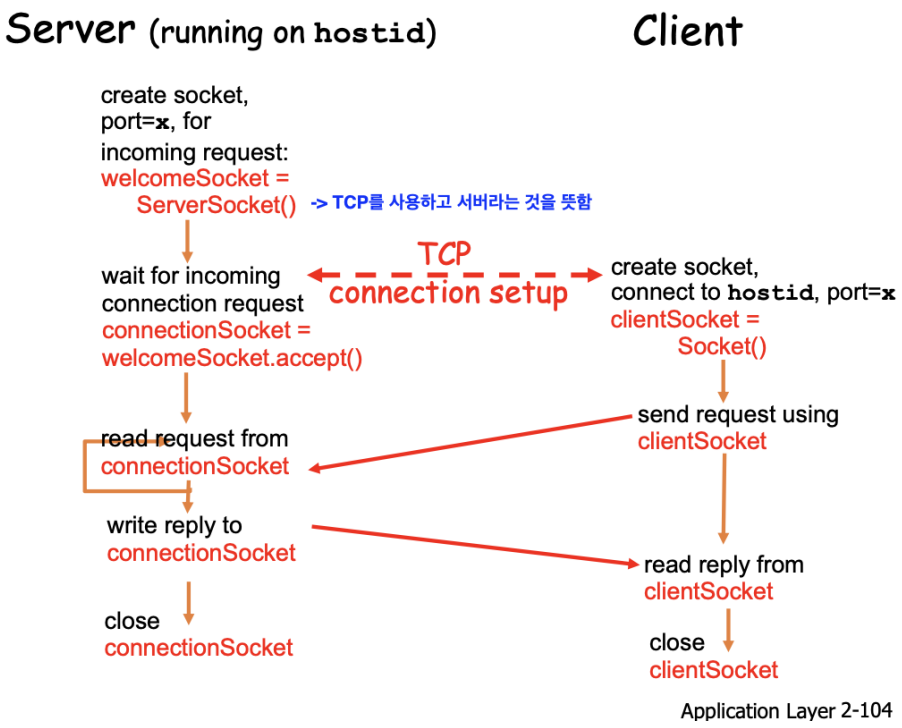
UDPServer.py

```
from socket import *

# 포트 번호
serverPort = 12000

# UDP 소켓 생성
serverSocket = socket(AF_INET, SOCK_DGRAM)

# 12000 포트 번호를 소켓에 할당한다. 이를 통해 서버 IP 주소의 12000 포트
serverSocket.bind(('', serverPort))

print("The server is ready to receive")

while True:
    # 패킷이 서버에 도착하면 데이터는 메세지에 할당되고 패킷의 출발지 주소
    # 해당 주소로 서버는 응답을 어디에 보내야할지 알 수 있다.
    message, clientAddress = serverSocket.recvfrom(2048)

    # 바이트 데이터를 decode()하고 대문자로 변환한다.
```

```
    modifiedMessage = message.decode().upper()

    # 클라이언트 주소를 대문자로 변환된 메시지에 붙이고, 그 결과로 만들어진
    # 서버의 주소도 같이 보내지는데 이는 자동으로 수행된다.
    serverSocket.sendto(modifiedMessage.encode(), clientAddre
```

2.7.2 TCP 소켓 프로그래밍

- 클라이언트는 서버에게 contact을 해야해(Handshaking)
  - 서버 프로세스는 돌아가고 있어야해
  - 서버는 소켓을 만들어서 듣고 있어야해(?)
- 클라이언드가 서버에 contact하려면
  - TCP소켓을 만들어서 서버에게 connection을 열고 싶다고 연락



TCPClient.py

```
from socket import *

serverName = 'servername'
serverPort = 12000

# 클라이언트 소켓을 의미한다. SOCK_STREAM으로 TCP 소켓임을 명시했다.
# UDP 때와 마찬가지로 따로 출발지 주소를 명시하지 않는다. (운영체제가 대신
clientSocket = socket(AF_INET, SOCK_STREAM)

# 클라이언트가 TCP 소켓을 이용하여 서버로 데이터를 보내기 전에 TCP 연결이
# 해당 라인으로 TCP 연결을 시작하고, connect() 메서드의 파라미터는 연결의
# 이 라인이 수행된 후에 3-way handshake가 수행되고 클라이언트와 서버 간
clientSocket.connect((serverName, serverPort))

sentence = raw_input('Input lowercase sentence:')

# 클라이언트 소켓을 통해 TCP 연결로 보낸다. UDP 소켓처럼 패킷을 명시적으로
# 대신 클라이언트 프로그램은 단순히 문자열에 있는 바이트를 TCP 연결에 제공
clientSocket.send(sentence.encode())

# 서버로부터 바이트를 수신하기를 기다린다.
modifiedSentence = clientSocket.recv(1024) #1024는 buffer size
print('From Server: ', modifiedSentence.decode())

# 연결을 닫는다. 이는 클라이언트 TCP가 서버의 TCP에게 TCP 메시지를 보내게
clientSocket.close()
```

TCPServer.py

```
from socket import *

serverPort = 12000

# TCP 소켓 생성
serverSocket = socket(AF_INET, SOCK_STREAM)
```

```
# 서버의 포트 번호를 소켓과 연관시킨다.
serverSocket.bind((’’, serverPort))

# 연관시킨 소켓은 대기하며 클라이언트가 문을 두드리기를 기다린다.
# 큐잉되는 연결의 최대 수를 나타낸다.
serverSocket.listen(1)
print(’The server is ready to receive’)

while True:
    # 클라이언트가 TCP 연결 요청을 하면 accept() 메소드를 시작해서 클라
    # 그 뒤 클라이언트와 서버는 핸드셰이킹을 완료해서 클라이언트의 소켓과
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.encode())

    # 응답을 보내고 연결 소켓을 닫는다. 그러나 환영소켓인 serverSocket
    connectionSocket.close()
```