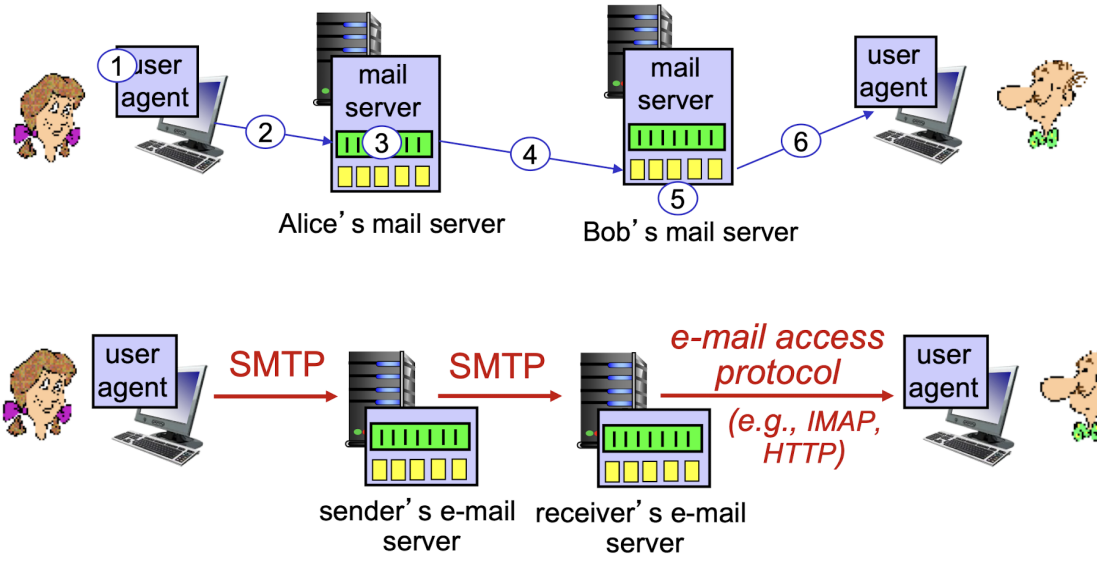
Chapter02

E-MAIL(SMTP)

Email에는 3개의 component가 존재

1. user agents: client
2. mail servers: server
3. SMTP



여기서 user agent가 각각 mail server를 갖고

mail server 간의 통신을 진행

여기서 email 내용을 받아올 때는 프로토콜이 달라져

HTTP vs SMTP

HTTP: pull

client가 데이터를 요청해서 가져옴

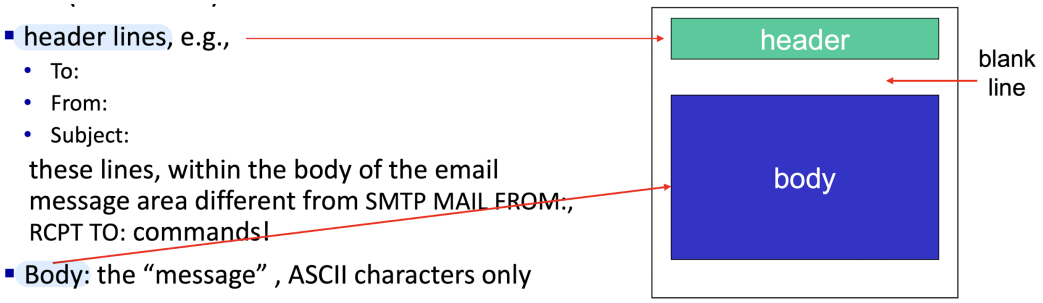
persistent connection

SMTP: push

보내는 쪽에서 connection을 열어서 받는쪽으로 집어넣음

persistent connection

end of message 여부를 CRLF .CRLF 기준으로 함



SMTP에서

header에는 meta data 정보를 저장 (누구에게, 어디서, 제목 ...)

body에 message 내용을 담아서 전송

DNS(Domain Name System)

Domain name을 IP로 바꿔주는 거 or 그 역과정도 가능

name server는 tree 구조를 이루고 있어

⇒ 그에 따라 DNS look up 방식이 2개가 존재

DNS가 할 수 있는 일

1. hostname to IP address
2. 이름, 별칭 지정 가능
3. load distribution

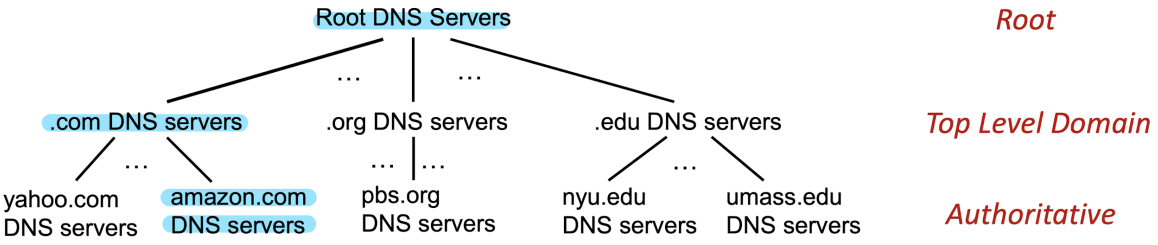
   ⇒ 서버에 여러개의 DNS 서버가 존재한다고 할 때 한 서버가 일을 다 하지 않고 부하를 분산해서 진행

   ⇒ round robin 방식으로 서버를 찾음

Why not centralize DNS?

1. single point failure: 중앙 서버에 문제가 생기면 전체 서버 싹 다 문제 생김
2. traffic volume: traffic이 몰리게 되면 처리하는데 시간이 오래걸림
3. distance: 실제 서버와 client의 물리적 거리가 멀어지게 되면 전송하는데 시간이 오래걸리니까
4. maintenance: 새로 생기거나 줄어드는 경우

> 💡 이거 doen’t scale 이라고 나와있는데 강의 중에 maintenance가 확장에 유리하다고 했는데 뭔 말인지 모르겠음;;



DNS가 위에서 트리구조로 되어있다고 했는데 이게 그 경우

당연하겠지만 root server가 터지면 하위 서버 싹 다 문제 생김

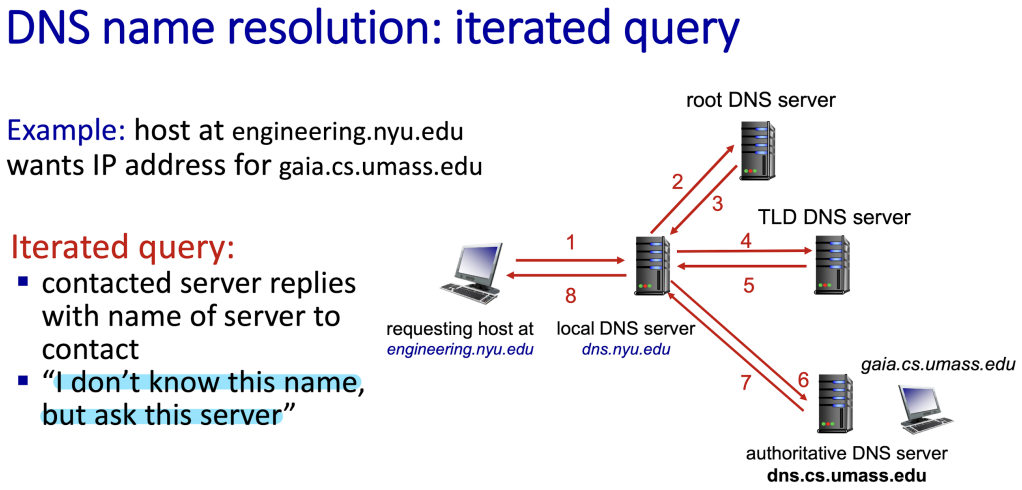
그래서 root server의 경우 복제본을 여러개 만들어놓고 전세계에 싹 뿌림

TLD(Top Level Domain)의 경우 .com, .org, .edu 이런 애들을 싹 관리

⇒ 실제로 얘가 일을 하진 않고 authoritative server의 위치를 알려주는 용도

authoritative server가 실제 도메인에 대한 DNS 정보를 관리

DNS name resolution: iterated query



DNS를 찾는 방식이 2가지가 있는데 이 방식을 가장 많이 사용

root로 request 요청을 보내서 확인

root가 이걸 받고 있으면 알려주는데 없으면 나는 모르는데 TLD 라는 애가 알 거 같으니 얘한테 가서 물어봐 함

그래서 TLD 로 가서 얘한테 다시 물어봄

당연히 얘도 있으면 바로 알려주는데 없으면 나는 모르는데 authoritative server가 알 거 같으니 얘한테 가서 물어봐 함

authoritative는 무조건 저장하고 있으니 얘가 알려줌

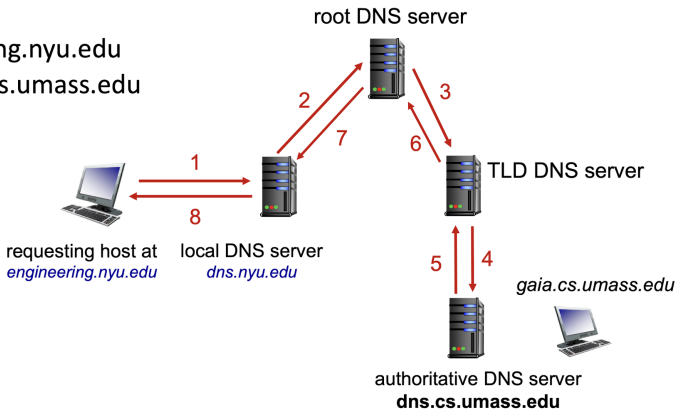
⇒ 장점: 일단 root(상위 level)에 부하가 적게 걸림

DNS name resolution: recursive query

Example: host at engineering.nyu.edu
wants IP address for gaia.cs.umass.edu

Recursive query:

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



이거는 재귀적으로 타고 들어가는데 root부터 쭉 내려감

당연히 마지막에서 루트로 반환해줘야 해서 계속 경로에 존재했던 서버들에 대한 정보를 갖고 넘어가야됨

⇒ 부하가 커서 잘 사용 안함

DNS caching

dns에 전달된 데이터를 캐싱함

TTL(Time To Live)이 지나면 캐싱된 데이터를 삭제

⇒ 실제 서버의 데이터가 변경되면 캐싱된 데이터가 의미가 없어지니까

DNS record

1. type = A

   name: hostname

   value: IP address

2. type = NS

   name: domain

   value: hostname에 대한 authoritative server name

3. type = CNAME

   name: alias name

   value: canonical name

4. type = MX

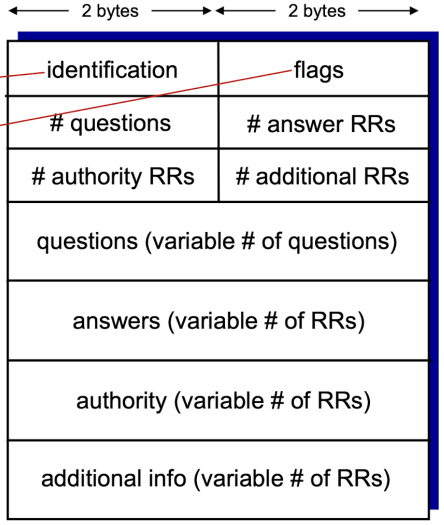
   value: mailserver의 name

DNS protocol messages

DNS *query* and *reply* messages, both have same *format:*

message header:

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



header: 처음 12byte

query에 대한 meta data 저장

identification: reply에도 동일한 값을 사용

flag:

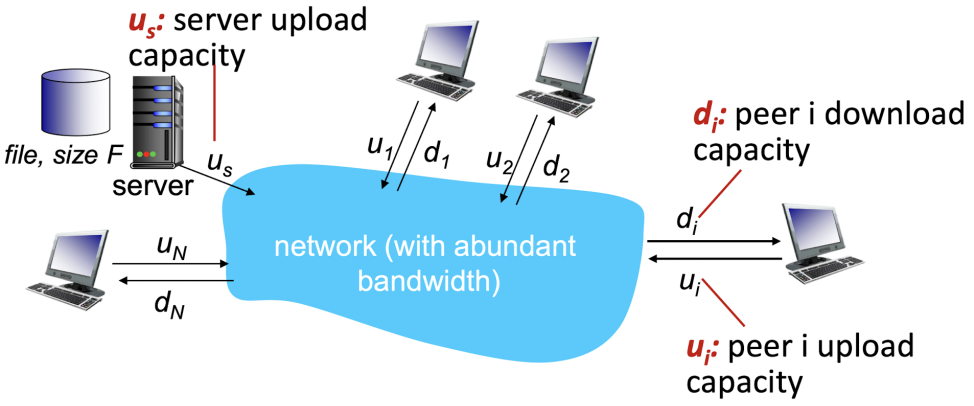
1. query인지 reply인지 식별
2. recursion query를 요구(기본이 iterated query)
3. recursion query가 가능한지에 대해 응답
4. authoritative server일 때 응답 메시지에 설정

P2P(Peer to Peer)

Peer가 client이자 server를 담당

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

- peer upload/download capacity is limited resource



여기서 file을 다운로드 혹은 업로드 하는데 어느정도의 시간이 걸리는지 보자(upper bound or lower bound)

server transmission

time to send one copy: F / u

time to send N copies: N * F / u

u(upload): 파일을 upload 하는 속도

그러면 client가 파일 하나를 다운 받을 수 있는 하한선은

server가 upload 하는 시간 vs client가 download 하는 시간으로 결정됨

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

근데 주로 client 보다는 server가 문제되는 경우가 많아

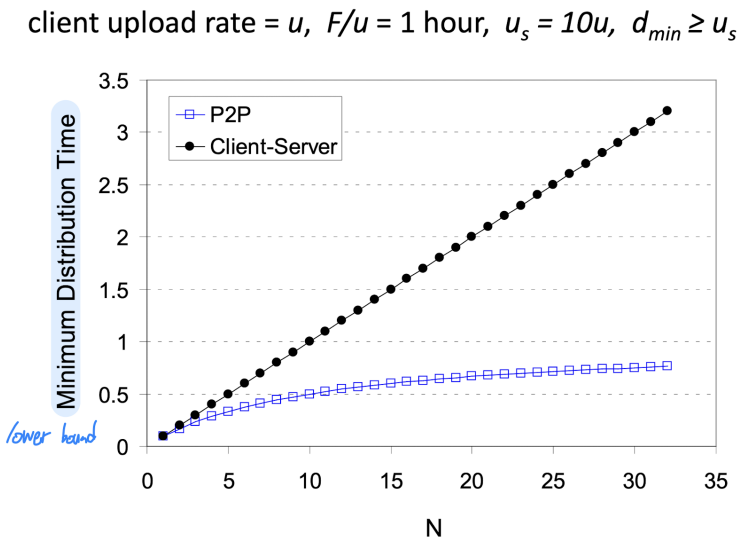
p2p transmission

client 하나가 다운 받는 속도: F/u_s (만약 client가 충분히 빠르게 다운로드 받을 수 있다면)

min client download time: F/d_{min} (시간이 최소가 되려면 속도는 당연히 제일 빨라야 함

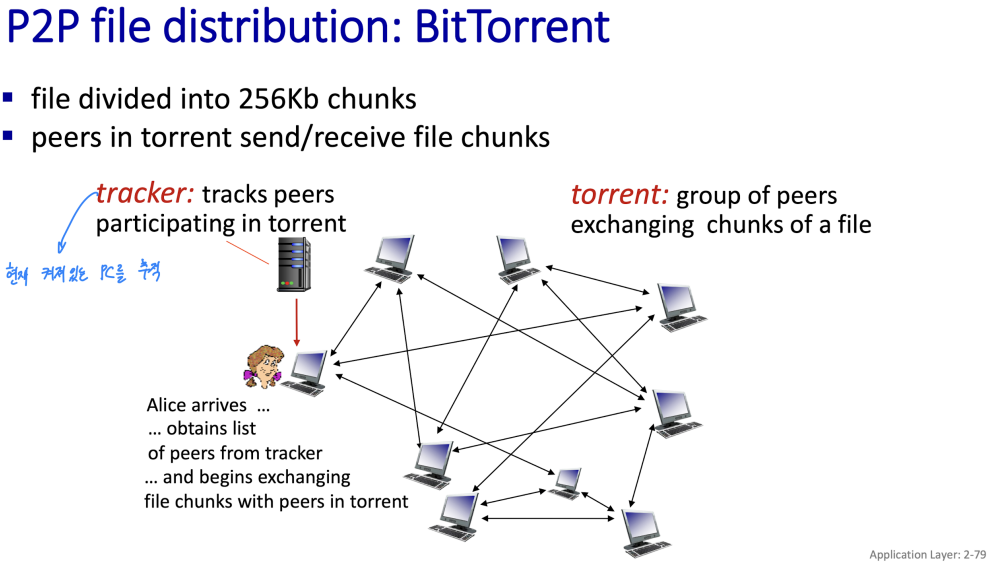
⇒ d_{min}: 다운로드 최대 속도

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



P2P랑 client-server를 비교해보면 P2P가 시간이 더 적게 증가하는 걸 볼 수 있다

BitTorrent



tracker: 현재 p2p에 참여한 PC를 추적 ⇒ 서버 역할을 하는 PC를 찾아야 돼

torrent: peer끼리 file의 일부분을 교환

file을 chunk로 쪼개서 관리하다보니 파일 전체를 하나의 peer가 없을 수도 있음

그래서 다른 peer와 교환해야 하는데 그 peer한테 나한테 없는 chunk를 요구해야돼 (rarest first)

peer 끼리 chunk가 겹치지 않아서 많이 주면 많이 받게 됨(tit-for-tat)

Video Streaming and CDNS

2017년 기준으로 넷플릭스가 새로웠다고 하네요 ㄷㄷㄷㄷ

아무튼 2020년 기준으로 넷플릭스, 유튜브, 아마존 프라임이 ISP 트래픽의 80%를 점유할 정도로 네트워크를 많이 쓰는 업체? 회사?가 고정됨

그러다보니 어떻게 모든 사용자들에게 데이터를 보낼수 있을까? (CDN)

그리고 사용자별로 받아들일 수 있는 양이 다른데 이거를 어떻게 조절하지?(DASH)

Multimedia: video

일단 video는 image file의 연속

이미지 크기가 1024 * 1024 개의 픽셀로 이뤄져있다고 가정

근데 동영상이 초당 24개의 image 전송한다고 할 때 ⇒ 보내야 되는 byte가 너무 많아

⇒ 그래서 이걸 압축하는 기술을 구현

뭔가 locality 느낌이 나긴 하지만 일단 보면

1. spatial 압축

   이미지가 연속적으로 이어질 때 비슷한 이미지가 되게 많음

   그래서 달라지는 부분에 대해서만 이미지를 원본으로 보내고 비슷하거나 같은 부분은 압축해서 보냄
2. temporal 압축

   데이터를 보낼 때 frame[i] 와 frame[i + 1]이 다른 부분만 보냄

Streaming multimedea: DASH

(Dynamic, Adaptive Streaming over Http)

사용자별로 받아들일 수 있는 양이 다른데, 이거를 조절할 수 있게 함

server에서 video를 chunk 단위로 분리해서 저장

client가 rate를 설정하고 그거에 맞게 chunk를 불러옴

만약 rate가 높으면 bitrate가 높은 비디오 chunk를 요청

낮으면 반대

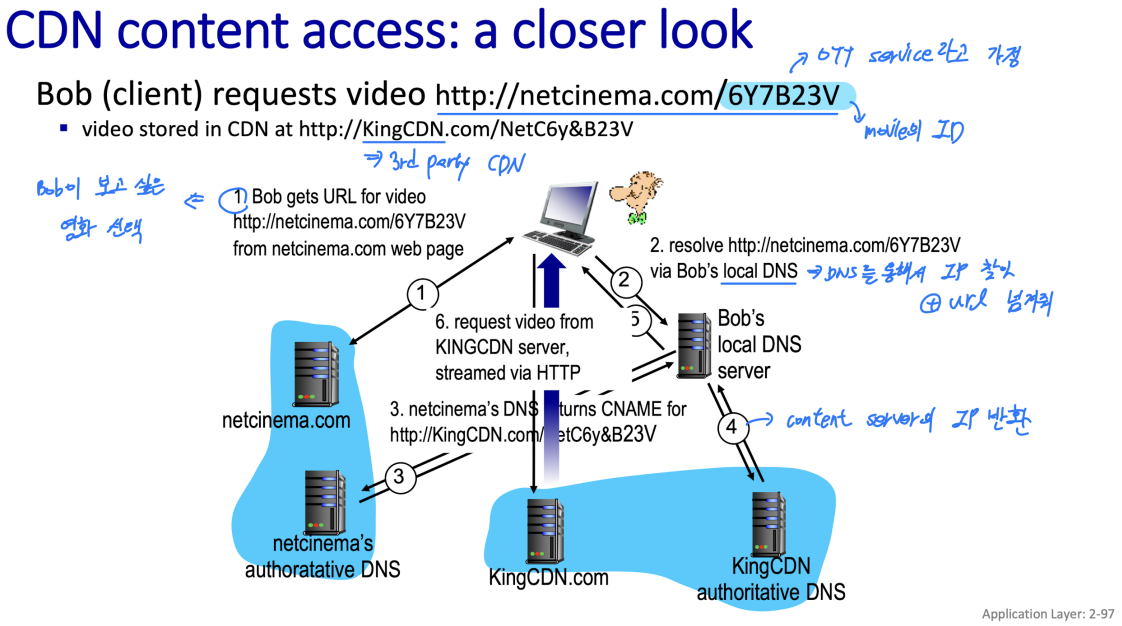
그래서 데이터의 요청에 대한 속도, 장소, 시점을 결정하는 건 client

CDN(Contents Distribution Network)

어떻게 모든 사용자들에게 동시에 데이터를 전송할 수 있을까에 대한 대답

1. 하나의 큰 mega server
   a. single point failure
   b. point of network congestion (속도 저하의 원인)
   c. long distance (거리가 멀 수밖에 없어)
   d. client할 때 보낼 때 video copy해서 보내야돼

2. 그래서 분산된 server 사용 (CDN)
    a. enter deep
        최대한 사용자(local ISP)와 가깝게 CDN 서버를 설치
        당연히 속도가 빨라, but copy server의 개수가 너무 많아
        ⇒ 그러다보니 server를 일치시키는데에도 비용이 많이 듦
    b. bring home
        좀 더 큰 ISP에 가깝게 설치
        속도가 느린 대신에 유지 보수, 관리 면에서 유리



1. Bob이 url을 통해 보고 싶은 비디오를 가져오려고 함
2. 해당 url이 있는 root DNS를 통해서 IP를 찾아
3. root DNS에서 authoratative DNS로 접속
4. authoratative DNS에서 실제 비디오 데이터가 있는 CDN DNS 주소(authoratative)를 반환
5. KingCDN authoratative에서 실제 주소를 찾고 거기서 비디오를 보내줌

Socket Programming

UDP와 TCP가 이걸로 동작

어차피 상세한 정보는 다음 chapter 때 볼 예정 (application layer가 아님)

TCP

1. 신뢰성을 보장(reliable transfer ⇒ 데이터가 손실되면 다시 보냄 + 순서 보장)

2. byte stream(데이터를 연속적으로 보냄)

UDP

1. 신뢰성을 보장하지 못함(데이터를 다시 보내지 않음 + 순서도 보장되지 않음)
2. Datagram 단위로 전송