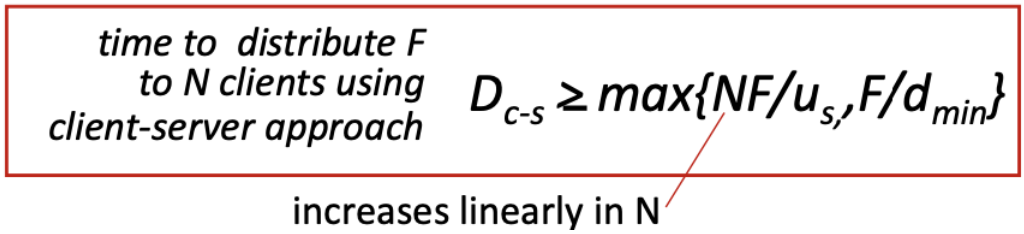
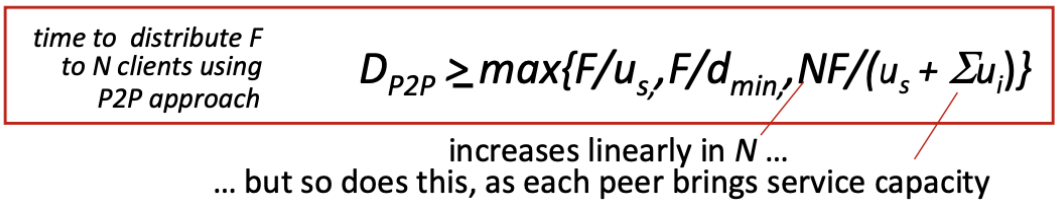
4일차

Peer-to-peer (P2P) Architecture

1. 특징
   a. 클라이언트가 서버 역할도 함
   b. 다수의 참여자가 파일을 공유(송수신)하는 형태
2. File distribution time : client-server
   a. download : client / upload : server

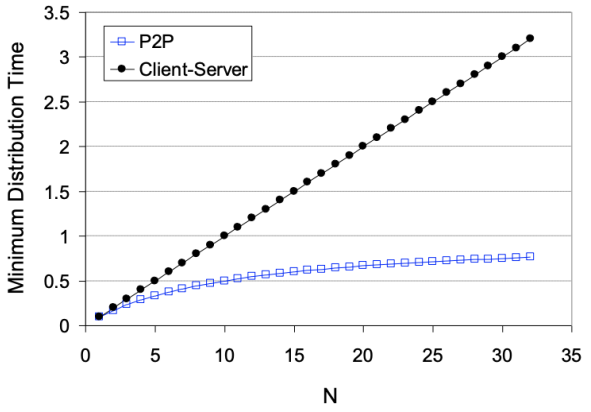


3. File distribution time : P2P
   a. 모든 유저들이 업로드를 할 수 있으므로 업로드 속도가 빠르다는 장점



   b. client 수가 늘어나면 차이가 더 많이남

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



4. P2P file distribution : BitTorrent
    a. 파일을 256Kb 단위 chunk로 나눔
    b. 멀리 있는 사람한테 파일 100프로 받는 것보다 가까운 사람한테 파일 50프로 받고 먼 사람한테 50프로 받는게 효율적
    c. peer는 file chunk를 보내고 받음
    d. torrent : peer group은 파일의 chunks를 교환
    e. tracker : torrent에 참여하는 peer 추적 (server)
    f. peer 가 파일을 모두 다운로드 받으면 자유롭게 나가거나 남을 수 있다.
    g. BitTorrent: requesting, sending file chunks
        i. 피어가 적은 것(chunk를 가진 사람이 적은 것)부터 받아야 한다 (피어가 언제 사라질지 모르므로)
    h. BitTorrent : tit-for-tat
        i. 준 만큼 받는다! 기여도에 따라서 다운로드 속도를 빠르게 해줘 형평성을 맞춘다

Video Streaming and CDNs : context

1. Stream video traffic → Netflix, Youtube, Amazon Prime, etc
    a. 어떻게 더 많은 정보를 줄 수 있을가?
        i. solution : distributed, application-level infrastructure을 통해 !!

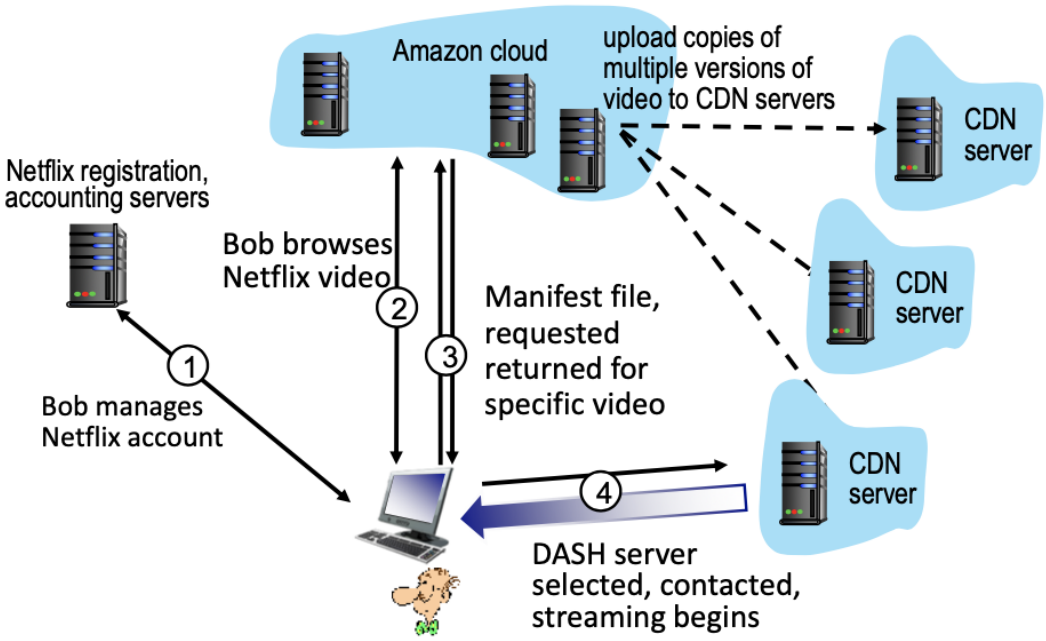
2. Multimedia : Video
    a. video : 일정한 비율로 이미지의 연속을 보여줌
       e.g. 24 images/sec
    b. digital image : 픽셀의 모임 (각 픽셀은 bit로 표현 가능)
    c. 두 가지 연속되는 frame 을 보낸다 생각할때 모든 frame을 보내면 시간, 비용이 많이 든다.
       따라서 변경되는 정보만 주거나 반복되는 정보를 어떠한 방식으로 보내주면 시간, 비용을 절약할 수 있다.
3. Streaming Stored Video : playout buffering
    a. delay가 생기는 문제를 buffering(속도가 좋을때 미리 받는 방식)을 통해 해결

Streaming multimedia : DASH

1. Dynamic, Adaptive Streaming over HTTP
2. client 가 bandwidth를 측정하고 chunk 파일을 요구하는 방식
3. Server
    a. 비디오 파일을 여러개의 chunk로 나눈다.
    b. 각 chunk들은 다른 비율로 encoding 되어 저장된다.
    c. manifestfile : 어떤 chunk가 어디에 있는지
4. Client
    a. manifest 파일 요청
    b. server와 client의 bandwidth 를 측정한다.
    c. manifest 파일을 보고 측정한 bandwidth 에 맞는 chunk 요청

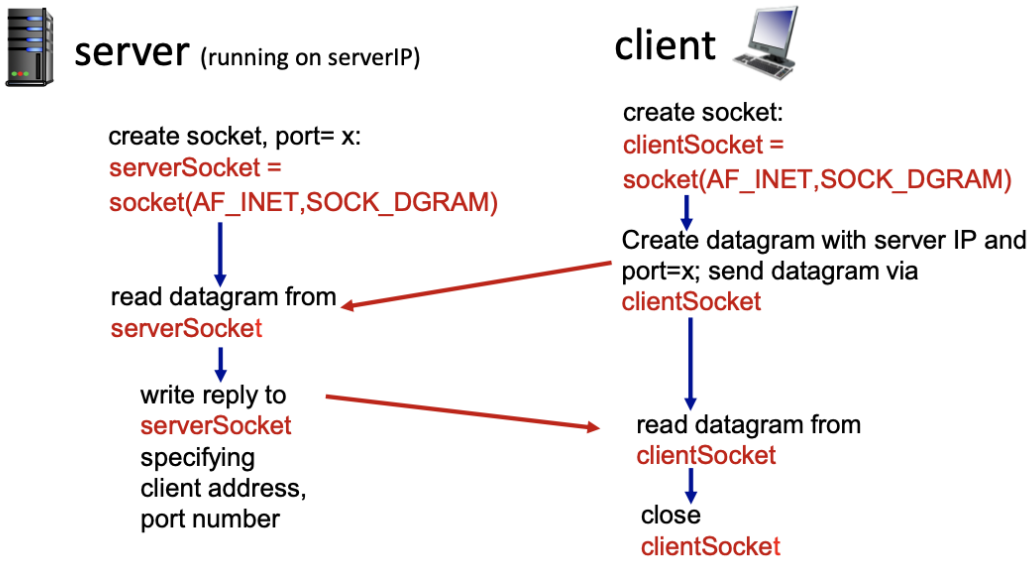
Content Distribution Networks (CDNs)

1. Case study : Netflix
    a. 나한테 가까운 CDN server에서 받도록



Socket programming

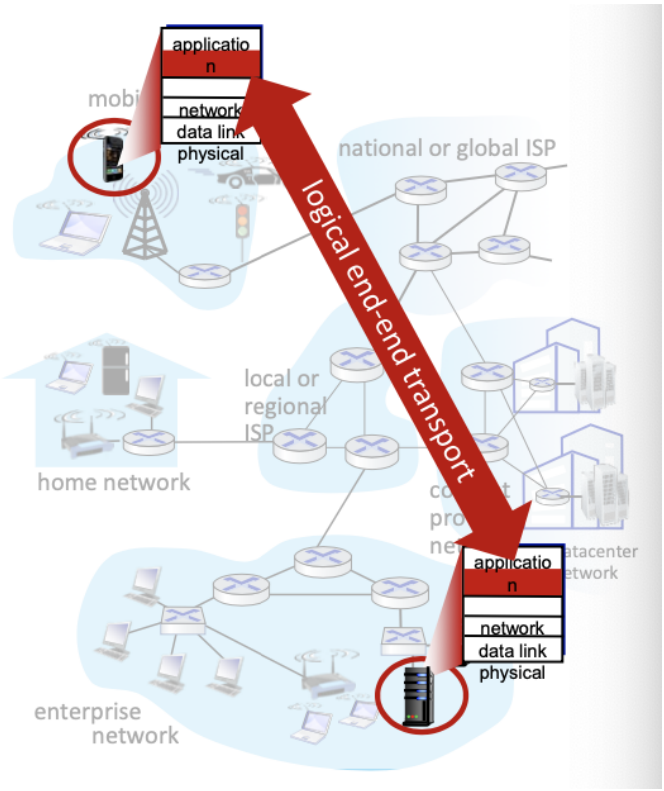
1. UDP : 클라이언트 서버 사이 connection이 없음 (unreliable)
    a. handshaking x
    b. ip address and port 가 있으면 걔한테만 보내줌
    c. 순서가 바뀔 수 있음
    d. data may be lost
    e. SOCK_DGRAM → UDP를 말함



2. TCP
    a. 서버가 먼저 동작
    b. IP, port 알아야함
    c. connection setup
    d. SOCK_STREAM → TCP를 말함
    e. UDP와는 함수차이가 있음 (bind, listen, accept, send 등)

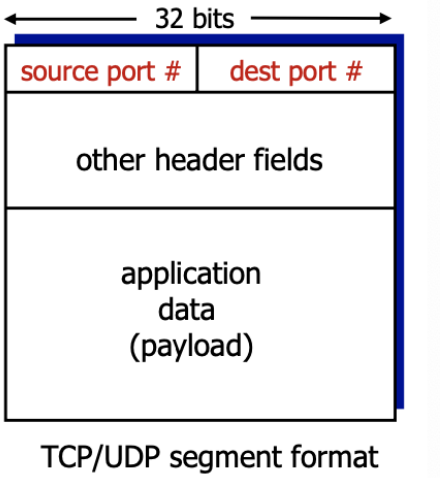
Transport-Layer Services

1. Transport Services and Protocols
    a. 서로 다른 host들에서 실행중인 application process들 사이에서의 logical communication 제공
    * Logical communication: network의 수많은 path를 고려하지 않고 end-to-end communication만 고려하는 것



b. segment : 데이터를 보내는 최소단위

2. Internet Transport-layer Protocols
   a. TCP
   b. UDP
3. Multiplexing / demultiplexing
   a. Multiplexing (sender) : sender 측에서 여러 socket 에서 오는 데이터들에 transport header 를 추가해서 packet으로 보내는 작업
   b. demultiplexing (receiver) : receiver 가 해당 segment의 header 정보를 이용해 적절한 socket 으로 보내는 작업
   c. How Demultiplexing Works



TCP/UDP segment format

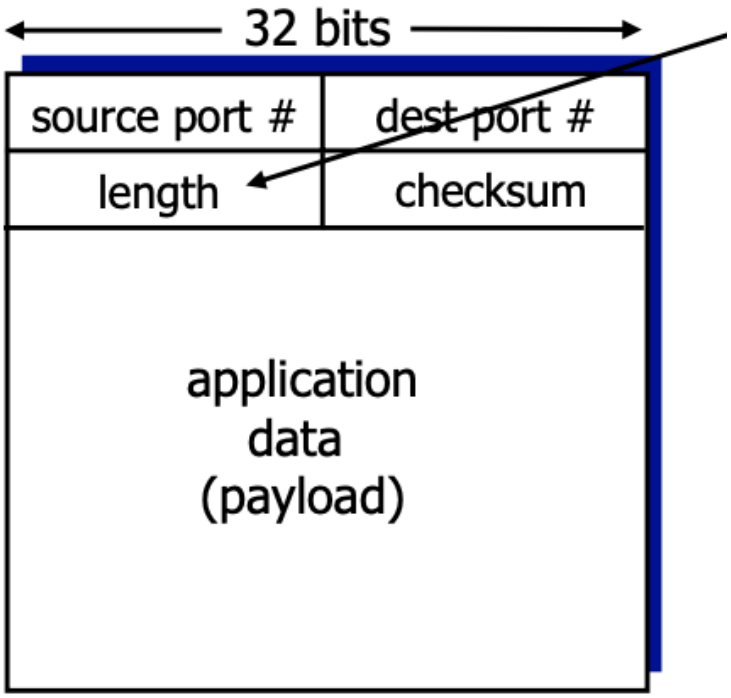
i. Host 가 IP datagram 을 받음
   i. 각 datagram은 source ip 와 destination ip 를 가짐
   ii. 각 datagram은 transport-layer segment 를 운반
   iii. TCP/UDP 모두 source, destination port 를 가지고 있음
ii. Host 는 IP 주소와 port를 사용해서 segment를 적절한 socket 으로 보낸다.

4. Connectionless Demultiplexing (UDP)
   a. 경로 : source → destination port 만을 보고 감
5. Connection-oriented Demultiplexing (TCP)
   a. TCP socket identified by 4 tuple : source IP, port / destination IP, port
      → 이걸로 길을 만들어줌
   b. 목적지의 ip, port가 같더라도 source가 다르면 길이 다르다.

UDP

1. 특징
   a. 뼈대가 없다
   b. lost, out of order
   c. connectionless
   d. usage : DNS, SNMP, streaming multimedia apps
2. UDP Segment Header

3. checksum : 비트 단위 데이터 유효성검사
4. checksum을 뺀 나머지를 가지고 checksum 값을 구함
    a. 16진수 숫자를 모두 더해서 (source, destination length, data) 이를 1의 보수 취해줌
    b. 나중에 receiver 에서도 똑같이 하여 checksum 이 같은지 확인을 통해 유효한 데이터인지 판단.



UDP segment format