2 어플리케이션 계층2

SMTP

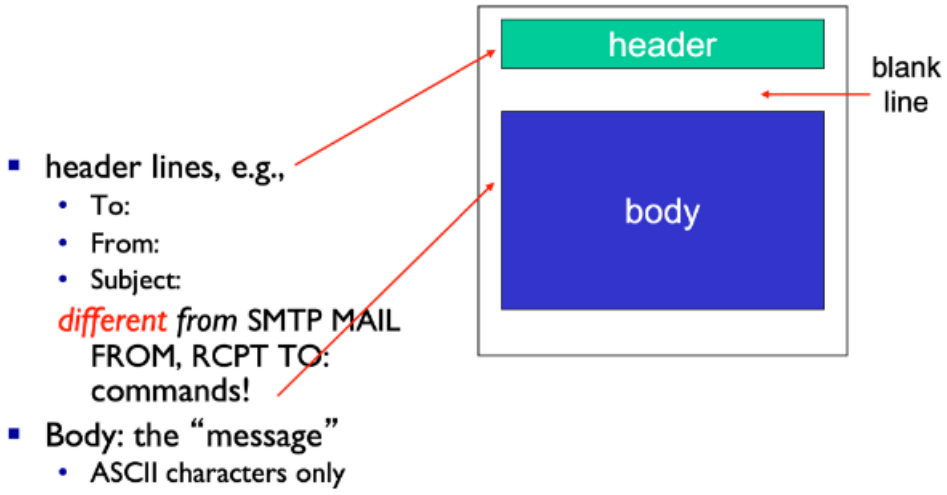
: 전자메일을 서버들 간에 메세지 주고받을 때 쓰는 프로토콜

→ persistent connection (connection을 열고 여러 개의 request를 전달)

- SMTP 서버가 메세지의 끝을 이야기할 때는 CRLF . CRLF
- HTTP는 pull 방식으로 요청해서 가져오는 반면 SMTP는 push 방식으로 새로운 이메일 메세지가 있을 때 수신 서버쪽으로 push

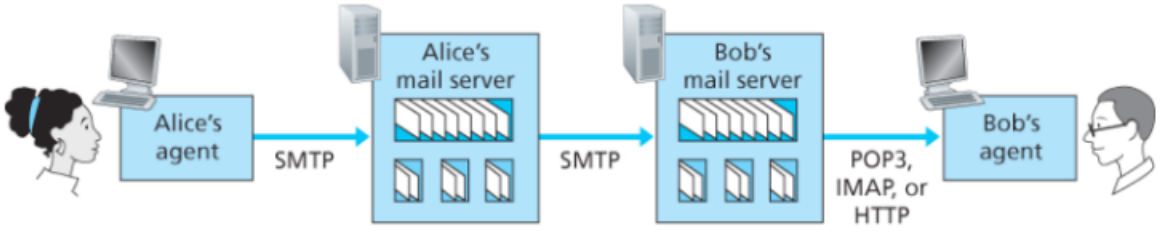
Mail message format

⇒ RFC B22



- 헤더와 바디는 CRLF로 분리됨
- 모든 헤더는 From:헤더라인과 To: 헤더라인을 반드시 가져야함

Message accress protocol



Mail accress protocol → 받는 사람 입장에서 내 메일에 접근하고 싶음

~> 서버에 접근하는 방식을 의미

수신자가 자신의 ISP 내부의 메일 서버에 있는 메세지 얻는 방법

- POP (Post Office Protocol) → 인증, 다운로드
- IMAP (Internet Mail Access Protocol) → 서버에 저장된 메일들 폴더 만들기 등 제공
- HTTP

POP3 Protocol

- authorization phase (인증 phase)

  → 아이디, 비밀번호 입력 등
- Transaction phase

  → list, retr, dele, quit

⇒ download와 delete 모드를 선택할 수 있음

~> download하고 현재 컴퓨터 서버에서 delete하면 다른 PC에서 읽기 불가능 ⇒ 문제점

IMAP

: 서버에 모든 메세지를 다 담고 있음 → user가 그때그때 다운로드해서 읽음

- 사용자가 서버에 있는 메세지들을 organize할 수 있음 (ex. 보낸 사람별 폴더)
- state를 keep 해야 함 → 그래야 다음에 또 접속해도 organize한 것을 다시 볼 수 있음

DNS (Domain Name System)

hostname (호스트 이름)

→ 인터넷 호스트 식별자 중 하나 www.facebook.com

- 호스트 이름은 인터넷에서의 호스트 위치에 대한 정보를 거의 제공하지 않음 ⇒ 호스트는 IP 주소로 식별됨

IP 주소

→ 계층구조를 가짐

- 계층구조여서 왼쪽에서 오른쪽으로 조사함으로써, 그 호스트가 인터넷의 어디에 위치하는지에 대한 자세한 정보를 얻을 수 있다.

호스트 이름을 IP 주소로 변환해주는 디렉토리 서비스가 필요 → 이 역할을 DNS가 함

: 분산 데이터베이스 → 많은 name server들의 hierarchy의 구현

- application-layer protocol → host와 name server 간의 통신할 때 사용

DNS service

- hostname을 IP address로 변환
- host aliasing → 복잡한 호스트 이름을 가진 호스트는 하나 이상의 별명을 가질 수 있음

  ⇒ DNS는 호스트의 IP 주소 뿐만 아니라 제시한 별칭 호스트 이름에 대한 정식 호스트 이름을 얻기 위해 이용될 수 있음
- email server aliasing (전자 메일 주소는 간단하지만 그 서버 호스트 네임은 일반적으로 더 복잡함)
- load distribution → load를 분산시켜줌

  → 여러 개의 IP주소를 가지는데 다른 IP 주소로 갈 수 있게 분산시켜줌

  → 인기 있는 사이트는 여러 서버에 중복되어 있어서, 각 서버가 다른 종단 시스템에서 수행되고 다른 IP 주소를 가짐

centralize DNS 안쓰는 이유

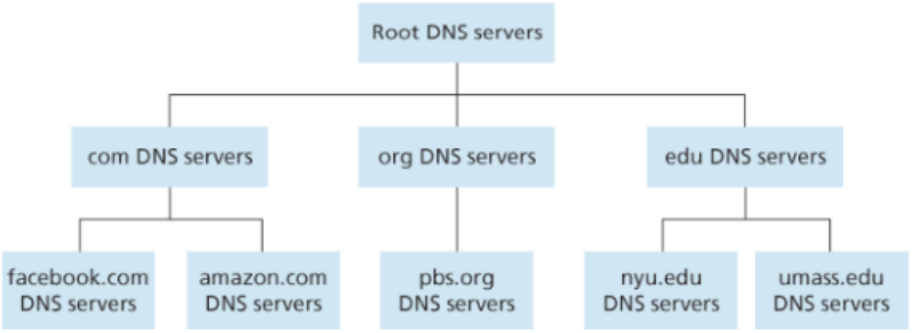
- single point of failure (서버의 고장)

  → 하나의 서버에 모든 정보 → 그 server가 failure면 모두 먹통
- 트래픽 양의 과부하 → 단일 DNS 서버가 모든 질의를 해결해야 함
- distane centralized database

  → 멀리 떨어져 있으면 hostname 알 때까지 시간이 좀 걸림

⇒ 확장성이 없음

분산(distributed) 계층(hierarchical) 데이터베이스



DNS는 많은 서버를 이용하고 이들을 계층 형태로 구성하며 전세계에 분산시킴

계층 DNS 서버의 종류

DNS : root name server

: TLD 서버의 IP 주소들을 제공

→ name을 resolve할 수 없을 때 local name server로 연결

= IP 주소 mapping을 가지고 있지 않을 때

- root name server는 mapping을 가지고 있지 않으면 authoritative name server를 컨택

  → mapping 가져와서 local name server로 mapping을 return

TLD(Top-level domain) Server

→ domain을 책임지고 있는 server(ex. com, org ...)

⇒ com, org와 같은 상위 레벨 도메인과 kr, uk와 같은 모든 국가의 상위 레벨 도메인에 대한 TLD서버가 있음

authoritative DNS 서버에 대한 IP 주소를 제공

Authoritative Server

조직 소유의 server ⇒ 가장 정확한 IP 주소 매핑을 가지고 있음

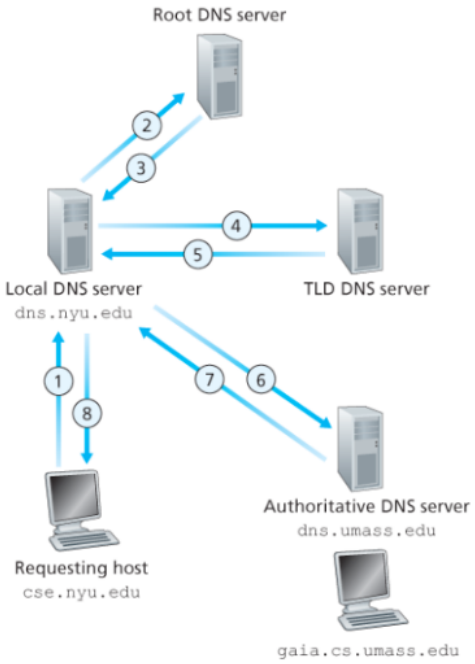
인터넷에서 접근하기 쉬운 호스트를 가진 모든 기관은 호스트 이름을 IP 주소로 매핑하는 공개적인 DNS 레코드를 제공해야 함

Local DNS name server

각각의 ISP가 하나씩 가지고 있을 수도 있음

ISP는 local DNS server를 갖고, local DNS server로부터 IP 주소를 호스트에게 제공

host가 DNS 쿼리를 만들면 이게 local DNS server로 보내짐



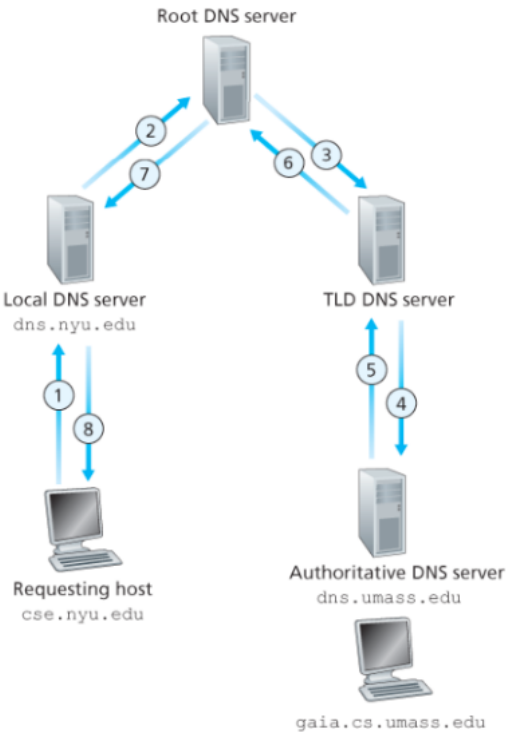
EX) cis.poly.edu에서 gaia.cs.umass.edu의 IP주소를 알고싶을 때

1. 자신의 로컬 DNS 서벙 질의를 보냄 → 변환하고 싶은 호스트의 이름을 같이 보냄
2. 로컬 DNS 서버는 그 질의 메세지를 루트 DNS 서버에게 전달함
3. 루트 DNS 서버는 edu를 인식하고 edu에 책임을 가진 TLD 서버의 IP 주소 목록을 DNS 서버에게 보냄
4. 로컬 DNS 서버는 TLD 서버에 질의를 보냄
5. TLD 서버는 umass.edu를 인식하고 dns.umass.edu로 이름 지어진 책임 DNS 서버의 IP 주소로 응답함
6. 로컬 DNS 서버는 직접 책임 DNS 서버로 질의 메세지를 다시 보냄
7. 최종 gaia.cs.umass.edu의 IP 주소를 응답함
8. 호스트에 최종 IP 주소를 응답합

→ local DNS server로 보내서 안가지고 있으면 root DNS server에 요청

→ iterated query(반복적 질의) 계속 반복적으로 server에 contact → cse.nyu.edu로부터 dns.nyu.edu로 보내는 질의는 자신을 필요한 매핑을 대신하여 얻도록 dns.nyu.edu에 요구

recursive query(재귀적 질의) → 갔던 server가 계속 다시 방문하는 것



→ 모든 질의가 재귀적인 DNS 질의 사슬

DNS : caching, updating records

→ DNS 지연 성능 향상과 네트워크의 DNS 메세지 수를 줄이기 위해 캐싱 사용

어떤 name server가 mapping을 시켜서 발견하면 cache에 일단 저장

- cache는 TTL 시간이 지나면 timeout으로 사라짐

  → 호스트 DNS와 IP 사이의 매핑과 호스트는 영구적이지 않기 때문에 TTL 이후에 저장된 정보 제거
- TLD 서버는 대부분 local name server cache에 저장되어 있음
- cached entry가 out-of-date되는 경우도 있음

DNS records

각 DNS는 하나 이상의 자원 레코드를 가진 메세지로 응답

→ DNS 서버들은 호스트 이름을 IP 주소로 매핑하기 위한 자원 레코드를 저장

> 💡 (Name, Value, Type, TTL)

→ Name과 Value는 Type을 따름 → Type에 따라서 해석법이 달라짐

- Type = A

  표준 호스트 이름의 IP 주소 매핑을 제공

Name : 호스트 이름

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

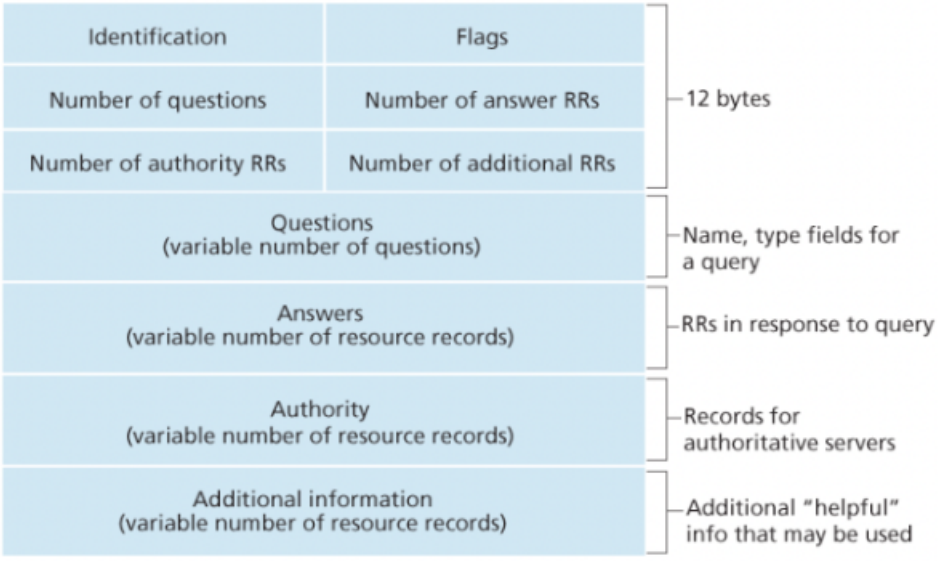
- Type = MX

  Mail eXchange

  Value : 별칭 호스트 이름 Name을 갖는 메일 서버의 정식 이름

  → 메일 서버의 호스트 이름이 간단한 별칭을 갖는 것을 허용

DNS message



query와 reply 메세지는 동일한 message format을 가짐

- identification : query와 query에 대한 응답이 똑같은 숫자로 정의됨
- flags
  - 질의/응담 플래그 : 메세지가 질의인지 응답인지 구분
  - 책임 플래그 : DNS 서버가 질의 이름에 대해 책임 서버일 때 응답 메세지에 설정됨
  - 재귀 요구 플래그 : DNS 서버가 레코드를 갖지 않을 때 재귀적 질의를 수행하기를 클라이언트가 원할 때 설정됨

- 재귀 가능 필드 : DNS 서버가 재귀 질의를 지원하면 응답에 설정
- 나머지 4개의 개수 필드 → 헤더 다음에 오는 데이터 영역의 네 가지 타입의 발생 횟수

DNS에 record를 삽입하는 방법

DNS registrar에 name을 등록

name, authoritative name server의 IP주소를 넘김

두 개의 RR를 RLD 서버에 저장함 (type=NS, type=A)

Pure P2P Architecture

- always-on server가 없음 → client가 server가 되기도 하고 server가 client가 되기도 함
- end system이 직접 통신 (서버와 하지 않음)
- peer끼리 connected되기도 하고 안되기도 함 (컴퓨터 꺼지고 켜지는 것에 따라서)

> ❓ size F인 파일을 한 개의 서버에서 N개의 peer에게 전달하는데 걸리는 시간은?
> (access 부분에서만 delay가 있다고 가정할 때)

1. Client-Server
   - 각각의 client에게 서버가 각각 보내야 함 → N개의 copy upload ⇒ NF/u_s
   - client는 각각의 파일을 download ⇒ F/d_{min}
   - $D_{C-S} >= MAX(NF/u_s, F/d_{min})$ ⇒ N에 따라 선형적으로 증가
2. P2P
   - 최소 하나의 copy만 upload하면 됨 → 자기들끼리 파일을 공유해서 upload하면 되기 때문 ⇒ F/u_s
   - client는 각각 파일을 download ⇒ F/d_{min}
   - clients들 전체로 보면 NF bit은 받아야 함 - 최대 upload 속도 ⇒ $u_s + \sum u_i$

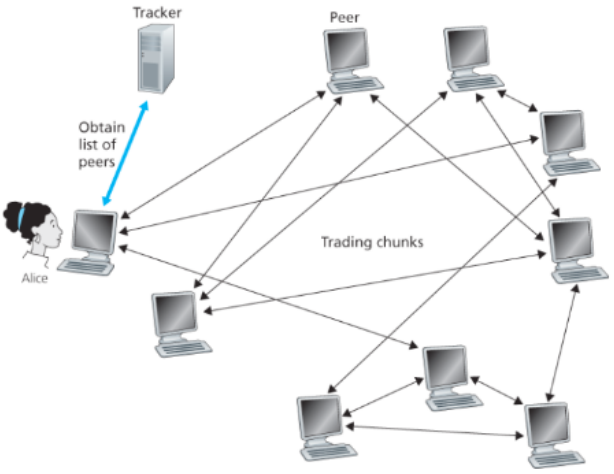
     → 최대 upload 속도가 최대 download 속도의 upperbound
   - $D_{p2p} >= MAX(F/u_s, F/d_{min}, NF/(u_s + \sum u_i))$

     → N이 증가한다고 해서 선형적으로 증가하는건 아님

P2P file distribution : BigTorrent

: 파일 분배를 위한 인기 있는 P2P 프로토콜

토렌트 → 특정 파일의 분배에 참여하는 모든 피어의 모임



- 파일이 256KB chunk로 이루어짐

  → peer들이 파일 chunk를 보내고 받음
- tracker : 켜져있는 컴퓨터들의 list

  한 피어가 토렌트에 가입할 때 트래커에 자신을 등록하고 주기적으로 자신이 아직 토렌트에 있음을 알려, 트래커는 토렌트에 있는 피어들을 추적할 수 있음
- 다운로드를 하면서 업로드도 동시에 함
- 나한테서 다운로드 해 가는 peer를 볼 수 있음

requesting chunk

→ 어떤 시간에서 서로 다른 peer는 서로 다른 chunk를 가지고 있을 수 있음

sending chunk (tit-for-tat)

→ 가장 높은 속도로 chunk를 보낸 사람에게만 chunk를 보냄

인터넷 비디오

비디오 : 이미지들이 연속해서 보여지는 것

→ 하나의 이미지는 픽셀로 구성, 각 픽셀은 여러 비트들로 encoding됨

만약 다음 장면과 변경된게 별로 없을 때는 frame 간의 차이만 보내줌 → temporal coding

- CBR (Constant Bit Rate)

  encoding 속도가 고정
- VBR (Variable Bit Rate)

  encoding 속도가 가변적

→ 어떤 화면에서는 압축이 심하고 어떤 화면에서는 압축이 덜 함

⇒ 압축을 사용하여 동일한 비디오를 여러 버전의 품질로 만들 수 있음

HTTP 스트리밍은 가용 대역폭이 달라도 똑같이 인코딩된 비디오를 전송받는 문제가 있음 → DASH로 해결

Streaming multimedia : DASH

= Dynamic, Adaptive Streaming over HTTP

→ HTTP위에서 서비스 ⇒ transport layer는 TCP를 사용

- 비디오는 여러 가지 버전으로 인코딩 되며, 각 버전은 비트율과 품질 수준이 서로 다름
- 클라이언트가 자신의 상황에 알맞는 비디오 버전을 요청함

- Server
- 비디오 파일이 여러 개의 chunk로 나눠짐
- 각 chunk는 각기 다른 encoding rate로 저장됨
- inteligence는 다 client한테 있음

→ 언제 chunk를 request할지, 어떤 encoding rate로 request할지, 어디로 chunk를 request할지

> 💡 어떻게 수많은 사용자에게 컨텐츠를 스트리밍할 것인가?

1. single, large “mega-server”

   → 데이터센터로부터 먼 지점에 클라이언트가 있으면 다양한 통신 링크, ISP 거치게 됨, 링크 중 하나라도 낮은 전송 용량이면 병목현상

   → 인기있는 비디오는 동일한 바이트를 전송하는 데에 반복 비용을 지불해야 함

   → 한 번의 장애로 전체 서비스가 중단될 수 있음
2. 컨텐츠 분배 네트워크 CDN (지리적으로 떨어져있는 위치에 서버를 둠)

CDN

→ 인터넷 스트리밍 서비스를 제공하는 가장 단순한 방법

: 다수의 지점에 분산된 서버들을 운영하며, 비디오 및 다른 형태의 웹 콘텐츠 데이터의 복사본을 이러한 분산 서버에 저장함

서버 위치의 철학

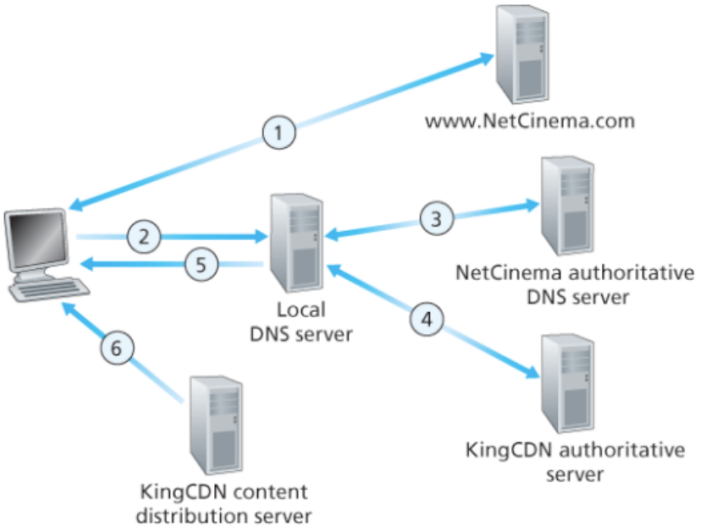
- Enter Deep

  : 최대한 서버를 사용자 근처에 위치시켜 링크 및 라우터를 거치는 횟수를 줄여 지연시간 및 처리율을 개선하는 것
- Bring home

: 좀 더 적은 수의 핵심 지점에 큰 규모의 서버 클러스터를 구축하여 ISP를 Home으로 가져오는 개념

→ 접속 ISP를 연결하는 대신, 일반적으로 클러스터를 인터넷 교환 지점에 위치

CDN 동작



1. 사용자가 URL을 입력
2. 사용자의 호스트는 URL의 host name에 대한 질의를 로컬 DNS로 보냄
3. 로컬 DNS는 host name의 책임 DNS 서버로 질의를 전달, 책임 DNS 서버는 해당 질의를 CDN 서버로 연결하기 위해 CDN 서버의 책임 DNS 서버의 IP를 전달
4. 로컬 DNS는 CDN 서버의 책임 DNS로 질의를 보내고, CDN 콘텐츠 서버의 IP 주소를 로컬 DNS 서버로 응답
5. 로컬 DNS 서버는 사용자 호스트에게 CDN 서버의 IP 주소를 알려줌
6. 클라이언트는 호스트가 알게된 IP 주소로 HTTP 혹은 DASH 프로토콜을 통해 비디오를 받아봄

CDN content access : a closer look

지리적으로 가까운 클러스터를 할당

Socket programming

프로세스가 소켓으로부터 읽고 쓰기를 통해 통신

- application 특성에 따라 transport layer를 결정해야 함

UDP socket programming

→ unreliable datagram

- client와 server 사이에 connection이 없음

- 송신 프로세스가 IP 주소와 port number를 각각의 패킷에 붙여넣음
- 수신 프로세스는 받은 패킷으로부터 IP주소와 port number를 빼냄
- 내가 보낸 데이터를 잃어버릴 수도 있고 순서가 바뀌어서 전달될 수도 있음

TCP socket programming

→ reliable, byte stream-oriented

- client가 먼저 server에 contact를 해야 함
- 클라이언트 소켓 주소와 서버 소켓 주소를 연결과 연관시킴
- 연결이 설정된 후에 소켓을 통해 데이터를 TCP 연결로 보내면 됨