Application layer2

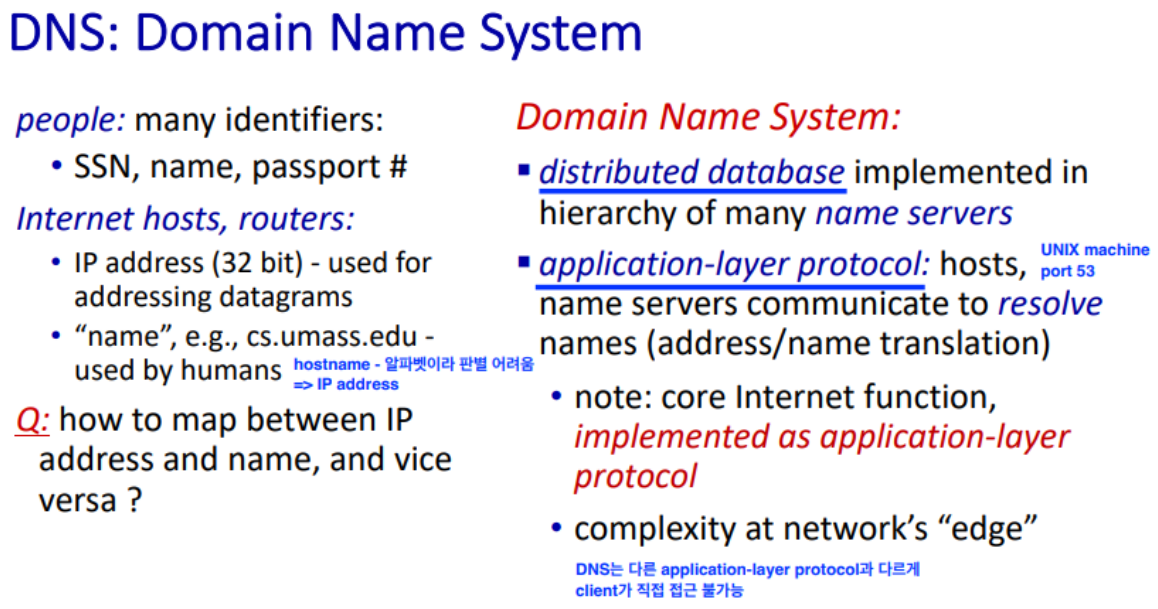
The Domain Name System DNS

호스트 이름

- 인터넷 호스트의 식별자 중 하나는 호스트 이름
- 호스트 이름은 인터넷의 호스트 위치에 대한 정보를 제공하지는 않는다.
- 라우터가 처리하는 데 어려움을 겪는다.
- 호스트는 IP 주소로 식별됨

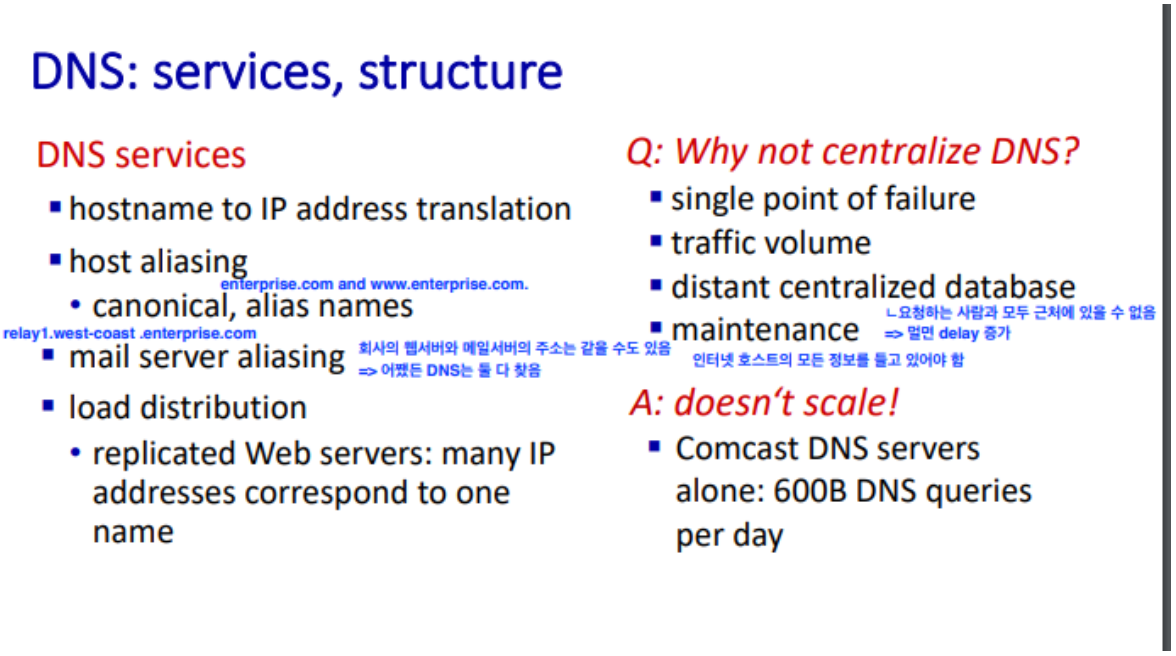
IP 주소

- IP 주소는 4바이트로 구성, 계층구조 갖음
- IP주소를 통해 호스트가 인터넷 어디에 위치하는지 정보를 얻을 수 있음



도메인 네임 시스템

- 다양한 이름 서버의 계층적으로 구현된 분산 데이터베이스
- 응용 계층 프로토콜: 호스트, 이름 서버가 이름을 해결하기 위해 통신함 (주소/이름 변환)
    - 참고: 핵심 인터넷 기능, 응용 계층 프로토콜로 구현됨
    - 네트워크의 "가장자리"에서의 복잡성



DNS 서비스

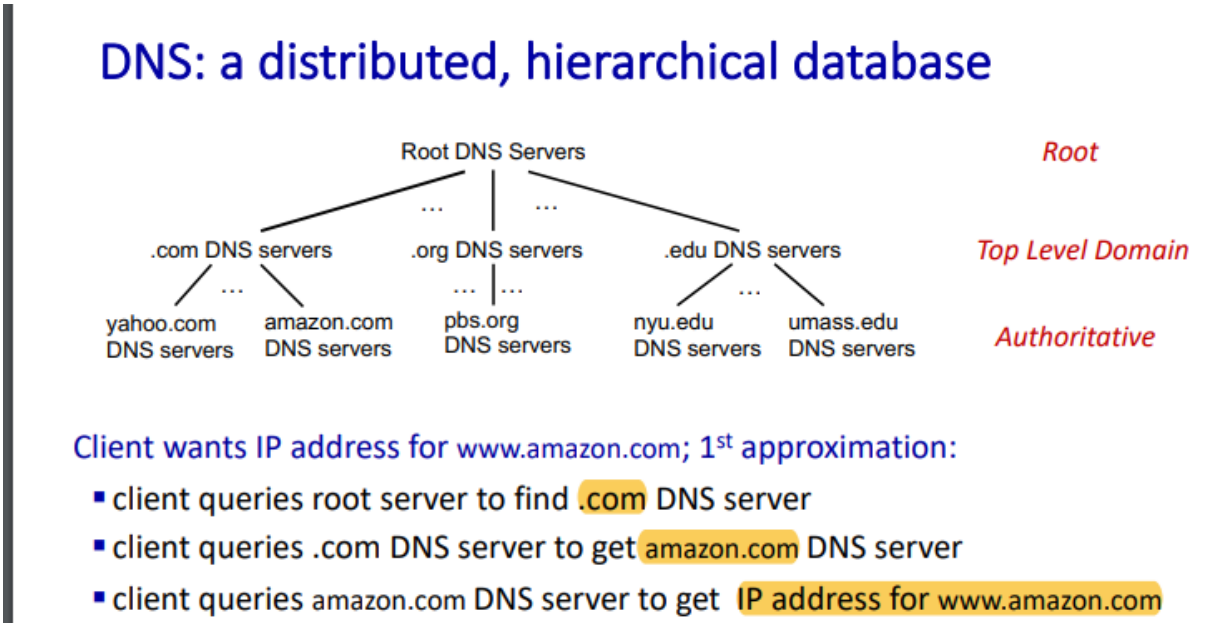
- 호스트 이름 IP 주소로 번역
- 호스트 별칭
    - 정규이름, 별칭 이름
- 메일 서버 별칭
- 부하 분산
    - 복제된 웹 서버 : 여러 IP 주소 하나의 이름에 대응

왜 DNS 중앙 집중화하지 않나?

- 서버의 고장 : 이 네임 서버가 고장 나면, 전체 인터넷이 작동하지 않는다.

- 트래픽 양 과부하 발생 : 단일 DNS 서버가 해결해야함
- 먼 거리의 중앙 집중 데이터베이스
- 유지보수
  - 단일 네임 서버는 모든 인터넷 호스트에 대한 레코드를 유지해야함
  - 모든 새로운 호스트를 반영하기 위해 자주 갱신되어야하고, 사용자에게 호스트를 등록할 수 있도록 허용하는 것과 관련된 인증 문제

> 💡 즉, 중앙 집중 데이터베이스는 확장성이 없음
> DNS는 분산되도록 설계 되어있음.



DNS는 많은 서버를 이용하고 이들을 계층 형태로 구성하며 전세계에 분산시킴

DNS 서버의 종류

루트 DNS 서버

- 1000개 이상의 루트 서버 인스턴스가 세계에 흩어져있음
- 루트 네임 서버는 TLD 서버의 IP 주소들을 제공한다.

TLD(최상위 레벨 도메인) DNS 서버

- top level domain, TLD
- com, org, net같은 상위 레벨 도메인과
- kr, uk 같은 모든 국가의 상위 레벨 도메인에 대한 tld 서버가 있다.
- Authoritative DNS 서버에 대한 IP 주소를 제공한다.

Authoritative DNS 서버

- 인터넷에서 접근하기 쉬운 호스트를 가진 모든 기관은 호스트 이름을 IP주소로 매핑하는 공개적인 DNS 레코드를 제공해야한다

로컬 DNS 서버

- 로컬 DNS 서버는 서버들의 계층 구조에 엄격하게 속하지는 않았지만 DNS 구조의 중심에 있음
- ISP는 로컬 DNS 서버를 갖고, 로컬 DNS 서버로부터 IP 주소를 호스트에게 제공한다.

DNS 쿼리과정

- 반복 쿼리
- 재귀 쿼리

Caching, Updating DNS Records

delay, DNS 메시지 개수 감소

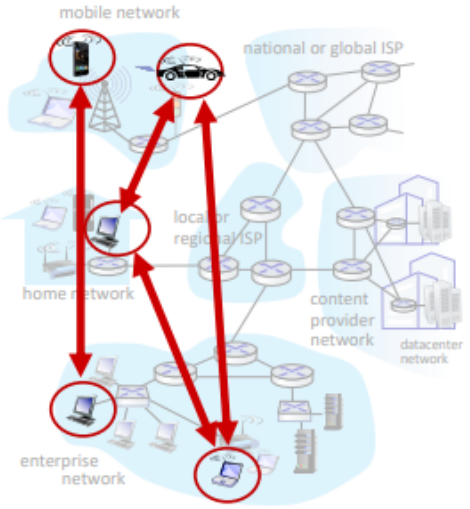
- once (any) name server learns mapping, it *caches* mapping (권한이 없더라도 줘버림)
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best-effort name-to-address translation!) hostname - IP address 매핑은 영구적이지 않음
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire!
- update/notify mechanisms proposed IETF standard
  - RFC 2136

DNS 캐싱은 DNS 쿼리 결과를 일정 기간 동안 저장하는 매커니즘
DNS 조회의 결과를 임시로 저장하여 동일한 호스트 이름에 대한 추가 쿼리의 응답 속도를 향상시킴

Peer-to-peer (P2P) architecture

Peer-to-peer (P2P) architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, and new service demands
- peers are intermittently connected and change IP addresses
  - complex management
- examples: P2P file sharing (BitTorrent), streaming (KanKan), VoIP (Skype)



항상 켜져 있는 서버가 없음

- 임의의 종단 시스템이 직접 통신합니다.
- 피어들은 다른 피어로부터 서비스를 요청하고, 다른 피어에 대한 서비스를 제공합니다.
  - 자가 확장성 - 새로운 피어는 새로운 서비스 용량과 새로운 서비스 요구를 가져옵니다.
- 피어들은 때때로 연결되어 있고 IP 주소가 변경됩니다.
  - 복잡한 관리
- 예시: P2P 파일 공유 (BitTorrent), 스트리밍 (KanKan), VoIP (Skype)

클라이언트-서버 구조 분배 시간

- **서버는 파일 복사본을 N개의 피어 각각에게 전송해야 한다.** 따라서 서버는 `NF 비트`를 전송해야 한다.
  - 즉, 서버가 파일을 분배하는 시간은 적어도 `NF/u(s)`이다.
- d(min)이 가장 낮은 다운로드 속도를 가진 피어의 다운로드 속도를 나타낸다고 하자.
  - 가장 낮은 속도를 가진 피어는 F/d(min)초보다 적은 시간에 파일의 모든 F 비트를 얻을 수 없다.
  - 즉 최소 분배 시간은 `F/d(min)`이다.

즉, 분배 시간을 D(cs)라고 하면 다음과 같은 수식을 얻을 수 있다.

```
D(cs) ≧ max{ NF/u(s) , F/d(min)}
```

위 식에서 충분히 큰 N에 대해 클라이언트-서버 분배 시간은 `NF/u(s)`로 주어진다는 사실을 알 수 있다. **즉, N에 따라 선형 증가한다.**

P2P 구조 분배 시간

여기서는 각 피어들이 서버가 파일을 분배하는 데 도움을 줄 수 있다.

특히 한 피어가 파일 데이터 일부를 수신할 때, 피어는 그 데이터를 다른 피어들에게 재분배하는 데 자신의 업로드 용량을 이용할 수 있다.

- 분배가 시작되면 서버만이 파일을 갖고 있다.
  - 이 파일이 피어 커뮤니티에 도달할 수 있도록 하기 위해, 서버는 적어도 한 번 접속 링크로 파일의 각 비트를 보내야 한다.
  - 따라서 최소 분배 시간은 적어도 `F/u(s)` 다.
    (서버가 한 번 보낸 비트는 서버가 다시 보낼 필요가 없는데, 이는 **피어들이 그들 사이에서 재분배할 수 있기 때문이다.**)
- 클라이언트-서버 구조와 마찬가지로 다운로드 속도가 가장 낮은 피어는 F/d(min)초보다 적은 시간 안에 파일의 모든 F 비트를 얻을 수 없다.
  - 따라서 최소 분배시간은 적어도 `F/d(min)`이다.
- 마지막으로, **시스템의 전체 업로드 용량**은 전체적으로 서버의 업로드 속도와 각 피어들의 속도를 더한 것이다. 이를 u(total)이라 하자.
  - 시스템은 각 피어들 각각에게 F 비트를 전달해야 한다. 이는 u(total)보다 빠르게 할 수 없다.
  - 따라서 최소 분배 시간은 `NF/u(total)`이다.

즉, 분배시간을 D(p2p)라고 하면 다음과 같은 수식을 얻을 수 있다.

```
D(p2p) ≧ max{ F/u(s) , F/d(min), NF/u(total)
```

비트토렌트

💡 비트토렌트는 파일 공유 프로토콜
P2P 네트워크를 기반, 사용자들간 직접적으로 파일 교환

작동 원리
1. 토렌트 파일 생성
2. 파일 공유
3. 트래커 서버에 연결하고 피어들을 찾음
4. 파일 전송 속도 향상

video streaming and content distribution networks

비디오

비디오는 이미지의 연속으로서 일반적으로 초당 24개에서 30개의 이미지로 일정한 속도로 표시된다.

압축되지 않은 디지털 인코딩된 이미지는 픽셀 단위로 구성되며, 각 픽셀은 휘도와 색상을 나타내는 여러 비트들로 인코딩된다.

비디오의 중요한 특징은 압축될 수 있다는 것인데, 비디오 품질과 비트 전송률은 서로 반비례한다.

오늘날의 상용 압축 알고리즘은 근본적으로 원하는 모든 비트 전송률로 비디오를 압축할 수 있다. (비트 전송률이 높을수록 이미지 품질이 좋다.)

네트워킹 측면에서 비디오의 가장 두드러진 특성은 높은 비트 전송률이다.

인터넷 비디오는 일반적으로 고화질 동영상을 스트리밍 하기 위해 100 kbps에서 4 Mbps 이상으로 구성된다.

4K 스트리밍은 10 Mbps 이상의 비트 전송률로 예상된다. 이는 하이엔드 동영상의 경우 트래픽과 스토리지 용량이 엄청나게 필요함을 의미한다.

연속 재생을 제공하기 위해, 네트워크는 압축된 비디오의 전송률 이상의 스트리밍 애플리케이션에 대한 평균 처리량을 제공해야 한다.

→ 압축을 사용하여 동일한 비디오를 여러 버전의 품질로 만들 수 있다.

HTTP 스트리밍 & DASH

http 스트리밍에서의 비디오

- http 서버 내 특정 url을 갖는 일반적인 파일로 저장
- 클라이언트는 서버에서 tcp연결을 설립, url에 대한 http get 요청 발생
- 서버는 트래픽 허용되는 대로 비디오 파일 전송
- 어플리케이션 버퍼에 전송된 바이트가 저장
- 버퍼의 바이트 수가 미리 정해진 임계값을 초과하면 재생 시작

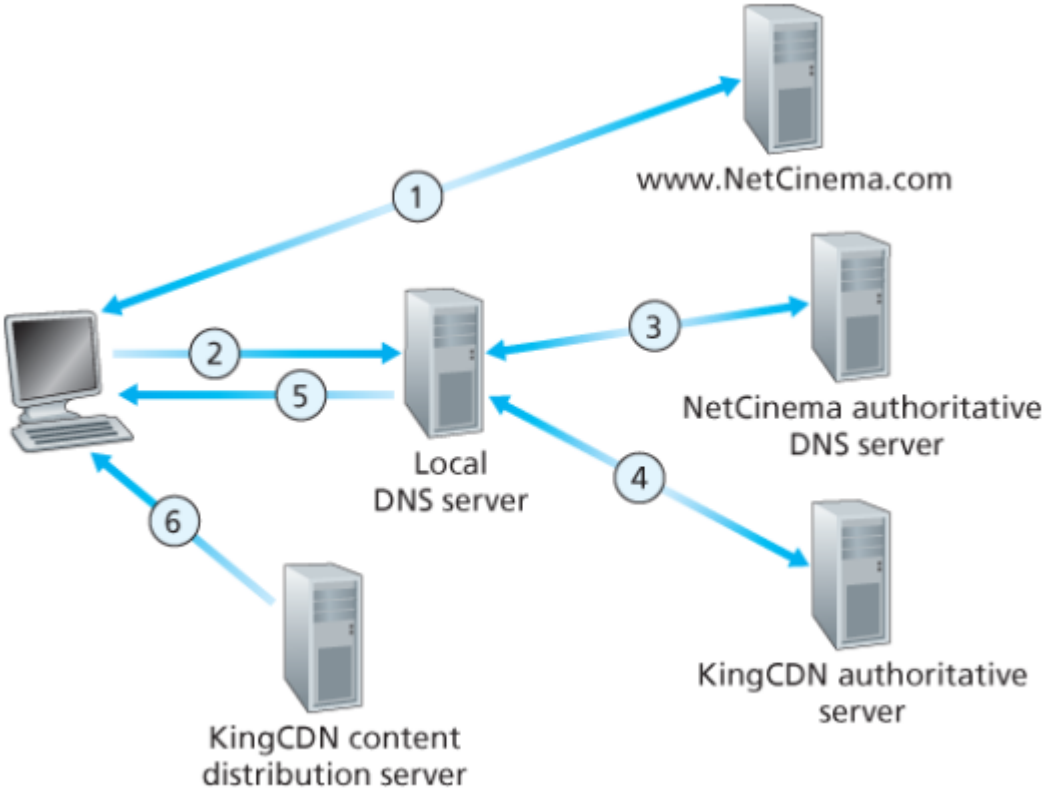
DASH

http 스트리밍에서의 비디오는 가용 대역폭이 달라도 똑같이 인코딩된 비디오를 전송받음

이로 인해 DASH 개발

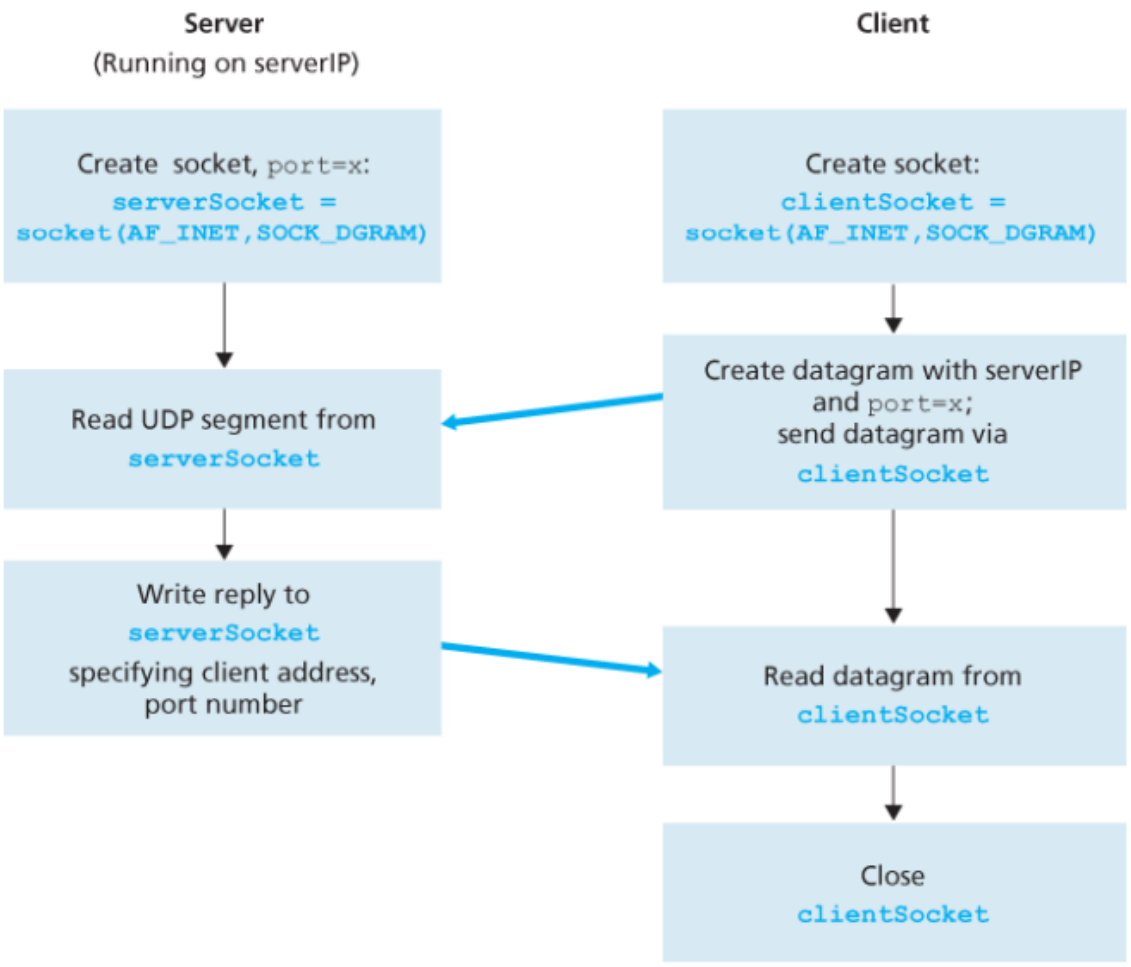
- 클라이언트는 자신의 상황에 맞는 비디오 버전 요청
- 비디오 버전은 자신의 상황에 맞는 비디오 버전 요청
- http 서버는 비트율에 따른 각 버전의 url을 제공하는 매니페스트 파일
- dash는 클라이언트가 서로 다른 품질 수준 가질 수 있게

콘텐츠 분배 네트워크 (CDN)



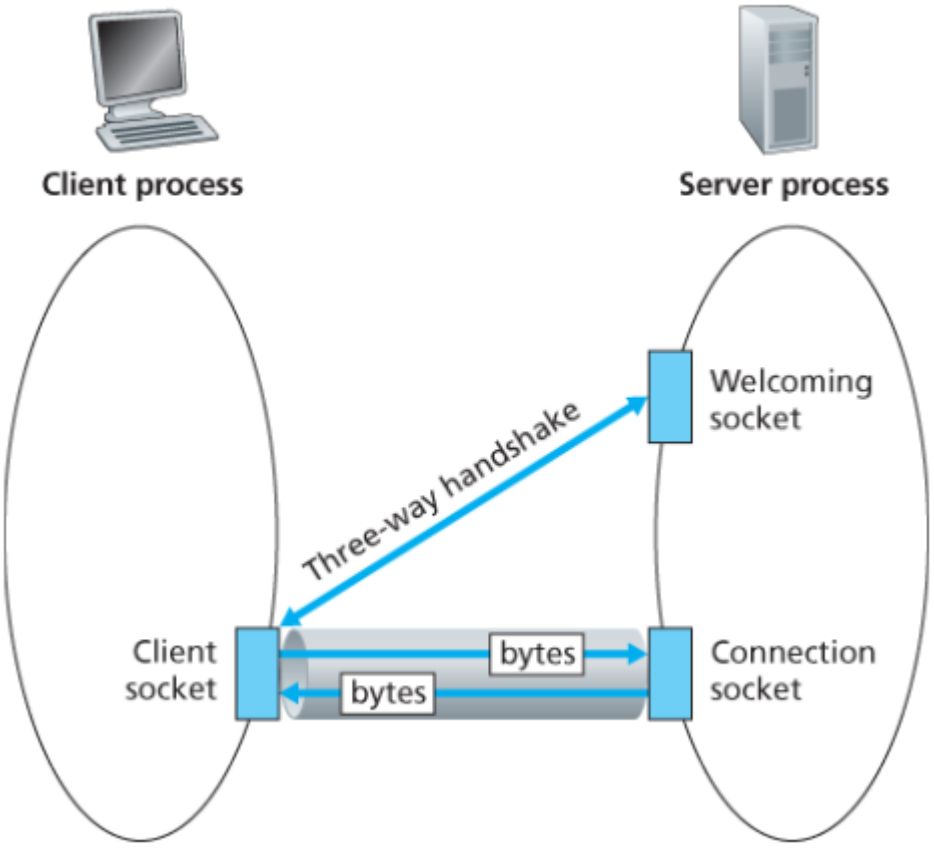
1. 사용자가 URL을 입력한다.
2. 사용자의 호스트는 URL의 host name에 대한 질의를 로컬 DNS로 보낸다.
3. 로컬 DNS는 host name의 책임 DNS 서버로 질의를 전달한다.책임 DNS 서버는 해당 질의를 CDN 서버로 연결하기 위해 CDN 서버의 책임 DNS 서버의 IP를 전달한다.
4. 로컬 DNS는 CDN 서버의 책임 DNS로 질의를 보내고, CDN 콘텐츠 서버의 IP 주소를 로컬 DNS 서버로 응답한다.이때 클라이언트가 콘텐츠를 전송받게 될 서버가 결정된다.
5. 로컬 DNS 서버는 사용자 호스트에게 CDN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

소켓 프로그래밍: 네트워크 애플리케이션 생성



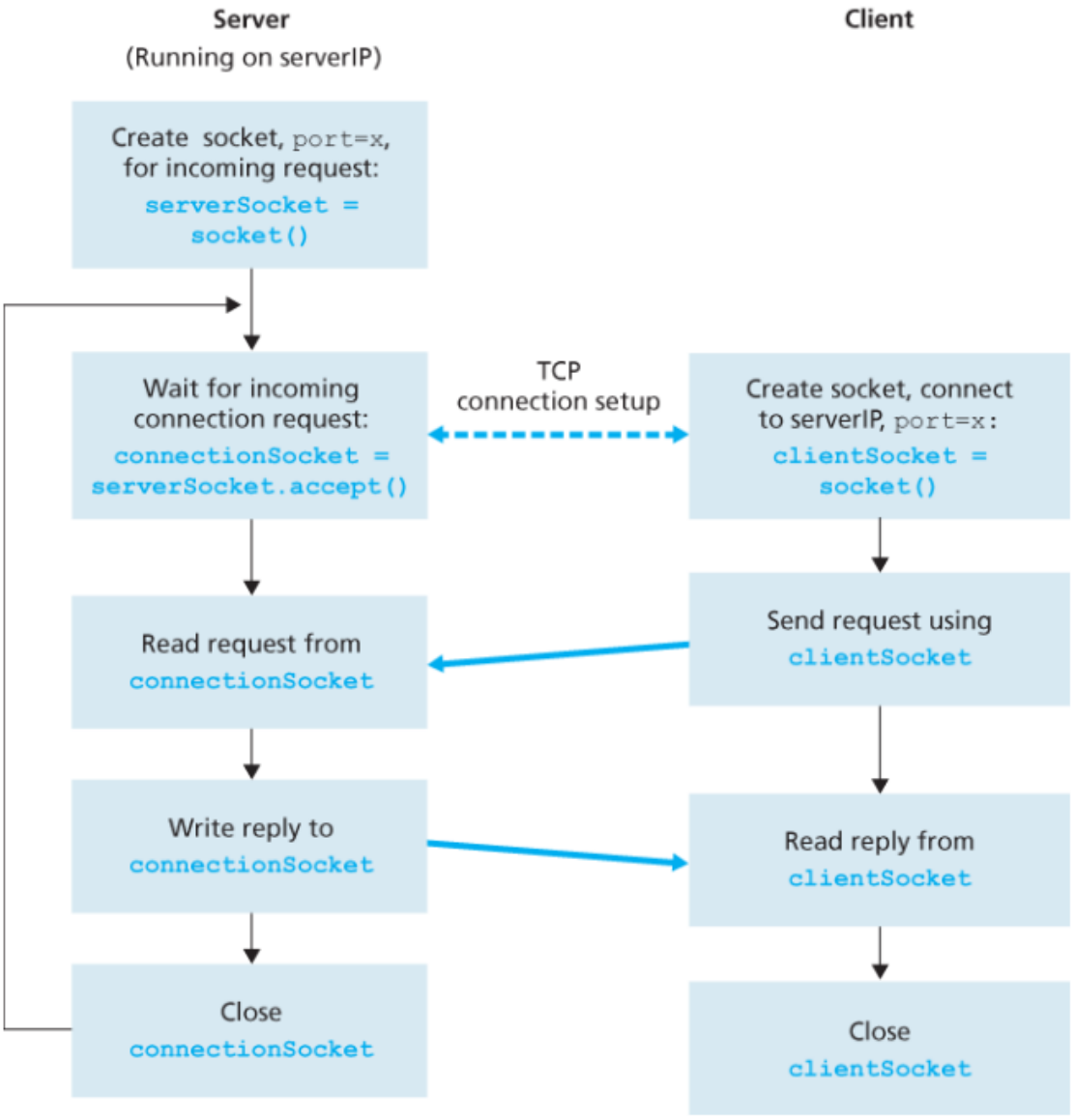
1. 클라이언트는 키보드로부터 한 줄의 문자를 읽고 그 데이터를 서버로 보낸다.
2. 서버는 그 데이터를 수신하고 문자를 대문자로 변환한다.
3. 서버는 수정된 데이터를 클라이언트에게 보낸다.
4. 클라이언트는 수정된 데이터를 수신하고 그 줄을 화면에 나타낸다.

TCP 소켓 프로그래밍



tcp 소켓

- 환영 소켓의 주소 명시
- 핸드셰이킹 동안 서버는 **해당 클라이언트에게 지정되는 새로운 소켓**을 생성


Server
(Running on serverIP)
Client
Create socket, port=x, for incoming request:
serverSocket = socket()
Wait for incoming connection request:
connectionSocket = serverSocket.accept()
TCP connection setup
Create socket, connect to serverIP, port=x:
clientSocket = socket()
Read request from
connectionSocket
Send request using
clientSocket
Write reply to
connectionSocket
Read reply from
clientSocket
Close
connectionSocket
Close
clientSocket