

# 지능형 IoT 네트워크

- 2주차 응용계층 -

2024.09.11.

# 1. 네트워크 애플리케이션의 원리

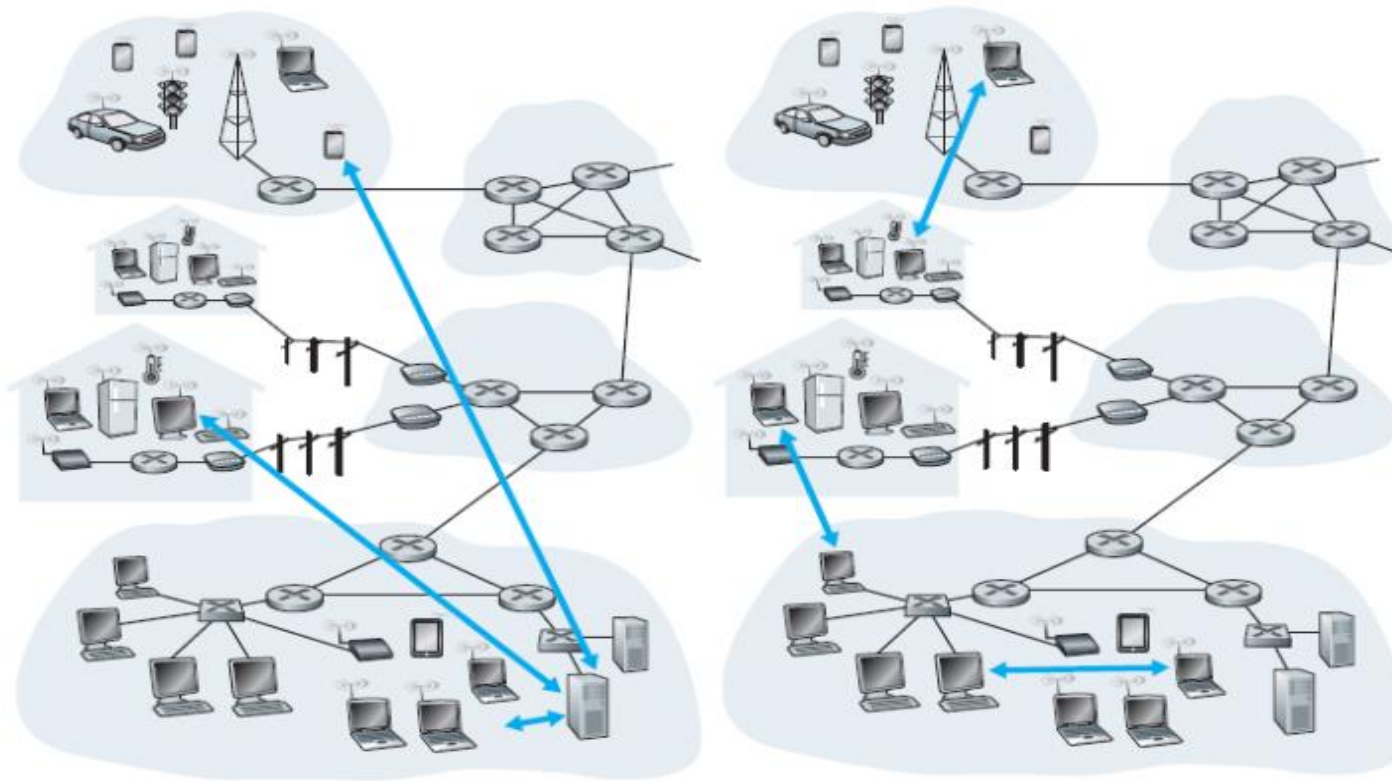
- **네트워크 애플리케이션:**

- 다른 위치의 종단 시스템에서 동작하고 네트워크를 통해 서로 통신하는 프로그램 작성
- Ex) 웹 애플리케이션에서는 서로 통신하는 서버(웹 서버 프로그램)와 클라이언트(사용자 호스트에서 실행되는 브라우저 프로그램)
- 라우터나 링크 계층 스위치처럼 네트워크 코어 장비에서 실행되는 소프트웨어까지 작성할 필요가 없다

# 1.1 네트워크 애플리케이션 구조

## 애플리케이션 구조와 네트워크 구조 비교

- 네트워크 구조는 고정되어 있고 애플리케이션의 특정 서비스 집합을 제공
- 애플리케이션 구조는 애플리케이션 개발자가 설계하고 애플리케이션이 여러 종단 시스템에서 어떻게 조직되어야 하는지 알려줌



a. Client-server architecture

b. Peer-to-peer architecture

## Client-Server 구조

- 서버는 항상 동작하고 있고 클라이언트라는 다른 호스트들로부터 서비스 요청을 받음
- 클라이언트들은 서로 직접적으로 통신하지 않음.
- 서버는 고정 IP주소로 설정
- 서버가 클라이언트로부터 오는 모든 요청에 더 응답하는 것이 불가능 할때, 많은 수의 호스트를 갖춘 데이터 센터(IDC) 가 강력한 가상 서버를 생성하는 역할로 사용됨. 예) 10만개 정도의 서버를 갖춘.

## P2P 구조

- 항상 커져 있는 Infrastructure 서버에 최소로 의존한다.(혹은 의존하지 않는다)
- 애플리케이션은 Peer라는 간헐적으로 연결된 호스트 쌍이 서로 직접 통신하게 한다.
- Peer는 클라이언트(개인 PC나 laptop 등등)
- 자기 확장성을 가짐. 예) 파일 공유 애플리케이션에서는 각 피어들이 파일을 요구하여 작업부하가 생기지만 각 피어들은 파일을 다른 피어들에게 분배하여 시스템에 서비스 능력을 갖춘
- 데이터 센터 등이 필요 없으므로 비용 효율적이다.

## 1.2 프로세스 간 통신

실제 통신하는 것은 프로그램이 아니라, 실행되고 있는 프로그램인 프로세스이다.

2개의 종단 시스템에서 프로세스는 컴퓨터 네트워크를 통한 메시지 교환으로 서로 통신한다.

송신 프로세스는 메시지를 만들어서 보내고 수신 프로세스는 메시지를 받고 역으로 메시지를 보냄으로써 응답한다.

### 클라이언트와 서버 프로세스:

- **클라이언트(Client):** 통신을 초기화(다른 프로세스와 세션을 시작하려고 접속을 초기화)하는 프로세스(요청발송)
- **서버(Server):** 세션을 시작하기 위해 접속을 기다리는 프로세스(요청수신)

※ P2P구조에서는 클라이언트도 서버 프로세스가 될 수 있고, 서버도 클라이언트 프로세스가 된다.

### 프로세스와 컴퓨터 네트워크 사이의 인터페이스:

- 프로세스는 소켓(socket)을 통해 네트워크로 메시지를 발송/수신

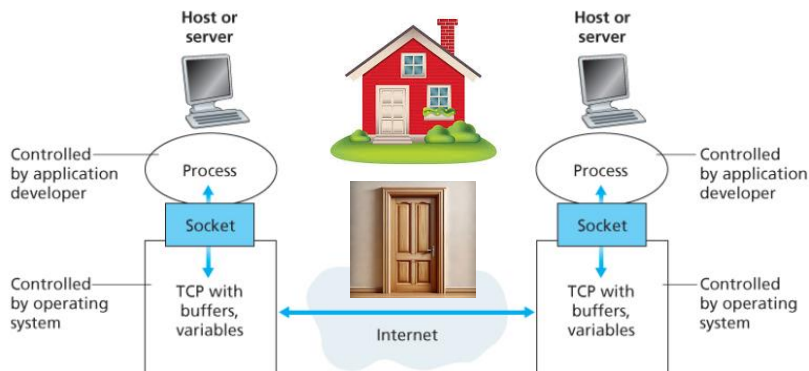


Figure 2.3 Application processes, sockets, and underlying transport protocol

네트워크 애플리케이션이 인터넷에 만든 프로그래밍 인터페이스이므로, 애플리케이션과 네트워크 사이의 API

### 애플리케이션 개발자의 Transport 계층 통제:

- Transport Protocol 선택(TCP/UDP)
- 호스트 내 수신 프로세스를 명시하는 식별자(Port Number)

### 프로세스 주소 배정:

프로세스가 다른 수행되고 있는 다른 프로세스에 패킷을 보내기 위해서는 수신 프로세스가 주소를 갖고 있어야 됨.

- 호스트 주소(IP 주소)
- 호스트 내의 수신 프로세스를 명시하는 식별자 즉, Port 번호가 필요함.

# 1.3 애플리케이션 이용 가능한 transport service

- 송신 측의 애플리케이션은 소켓을 통해 메시지를 보내고, 트랜스포트 프로토콜은 네트워크를 통해 그 메시지를 수신 프로세스의 소켓으로 이동시킬 책임이 있다.
- **Transport Protocol → Application (4 가지 서비스)**
  - **신뢰성 데이터 전송(data integrity)**
    - 신뢰적인 데이터 전송(TCP), 데이터 손실 허용 데이터전송(UDP) 실시간 비디오/오디오
  - **처리율(throughput)**
    - 다른 세션들이 네트워크 경로를 따라 대역폭을 공유하고, 이 세션들이 생겼다 없어졌다 하기때문에 가용한 처리율은 시간에 따라 변한다.
    - 대역폭 민감 멀티미디어 애플리케이션/ 탄력적 애플리케이션(elastic apps)
  - **시간(timing)**
    - 시간 보장을 제공(주로 실시간 애플리케이션에서 사용됨)
  - **보안(security)**
    - 송신 호스트에서 transport protocol은 모든 데이터를 암호화할 수 있고, 수신 호스트에서 트랜스포트 프로토콜은 모두 해독할 수 있다.  
예) TCP를 애플리케이션 계층에서 TLS로 보안 서비스를 제공

## TCP 서비스

- 연결 지향형 서비스
  - 핸드 셰이킹(handshaking)
    - 애플리케이션 계층 메시지를 전송하기전에 TCP는 Client/Server 서로 전송제어 정보를 교환
  - 전이중 연결
    - TCP 연결이 두 프로세스의 소켓 사이에 존재. 이 연결은 두 프로세스가 서로에게 동시에 메시지를 보낼 수 있게 됨.
- 신뢰적인 데이터 전송 서비스
  - 모든 데이터를 오류없이 올바른 순서로 전달하기 위해 TCP에 의존
  - TCP는 애플리케이션의 한 쪽이 바이트 스트림을 소켓으로 전달하면, 그 바이트 스트림이 손실되거나 중복되지 않게 수신 소켓으로 전달
- 혼잡 제어 방식
  - 네트워크가 혼잡 상태에 이르면 프로세스의 속도를 낮추는 방법

## UDP 서비스

- 최소의 서비스 모델을 가진 간단한 전송 프로토콜
- 비연결형으로 핸드셰이킹 과정이 없고, 비신뢰적인 데이터 전송 서비스를 제공하여 데이터 전달되는 것을 보장하지 않는다
- 혼잡 제어 방식을 포함하지 않아 프로세스의 속도 저하없이 네트워크 사용

### ※ 인터넷 트랜스포트 프로토콜이 제공하지 않는 서비스

- TCP/UDP 처리율 혹은 시간 보장 서비스를 제공하지 않는다. 시간 민감 애플리케이션 같은 경우에는 처리율 및 시간 지연에 잘 대처할 수 있도록 설계되어 있다. 그러나 지연이 과도할 때는 보장이 없기때문에 한계가 있다.

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube)	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

Figure 2.5 Popular Internet applications, their application-layer protocols, and their underlying transport protocols

애플리케이션 계층 프로토콜은 다른 종단 시스템에서 실행되는 애플리케이션의 프로세스가 서로 메시지를 보내는 방법을 정의

- 교환 메시지의 타입
  - 여러 메시지 타입의 문법(syntax)
  - 필드의 의미, 즉 필드에 있는 정보의 의미(semantics)
  - 언제, 어떻게 프로세스가 메시지를 전송하고 메시지에 응답하는지를 결정하는 규칙
- 예) HTTP, 웹 애플리케이션(문서 포맷, 웹브라우저, 웹서버 등 여러 요소로 구성됨).

## 네트워크 애플리케이션

- Web, 전자메일, 디렉터리 서비스, 비디오 스트리밍, P2P 애플리케이션)



## 2. 웹과 HTTP

**Web:** On-demand 방식으로 사용자가 원할 때 원하는 것을 수신한다.

### HTTP:

- 웹 애플리케이션 계층 프로토콜은(RFC 1945, RFC 7230, RFC 7540)
- 메시지의 구조 및 클라이언트와 서버가 메시지를 어떻게 교환하는지를 정의

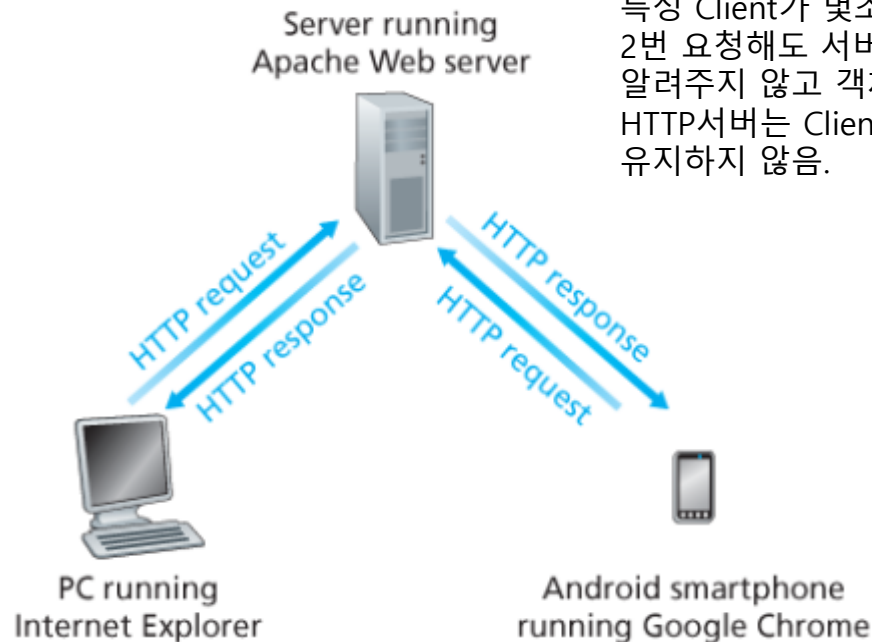
**Web page:** 객체(object)로 구성. 객체는 단순히 단일 URL로 지정할 수 있는 1개 파일(HTML, 이미지, 등)  
예) web page가 HTML 텍스트와 5개 JPEG 이미지로 구성 → 웹페이지는 6개의 객체로 구성

### Web 브라우저와 클라이언트:

- **웹서버:** URL로 각각을 지정할 수 있는 웹 객체를 갖고 있음
- **웹브라우저:** 요구한 웹페이지를 보여주고 인터넷 항해와 구성 특성을 제공

#### ※HTTP는 TCP를 전송 프로토콜로 사용:

- TCP는 신뢰적인 데이터 전송 서비스를 제공하므로 모든 HTTP 요청 메시지가 궁극적으로 서버에 도착



#### ※ stateless 프로토콜:

특정 Client가 몇초후에 같은 객체를 2번 요청해도 서버는 전에 보냈다고 알려주지 않고 객체를 또 보낸다. HTTP서버는 Client에 대한 정보를 유지하지 않음.

## 2.1 비지속 연결과 지속연결

### 비지속(non-persistent)연결

- 클라이언트-서버 상호작용이 TCP 상에서 발생할 때 각 요구/응답 쌍이 분리된 TCP 연결을 통해 보내지는 것을 말한다

웹페이지를 서버 → 클라이언트 전송하는 단계:

1. HTTP클라이언트는 HTTP 기본 포트 80을 통해 서버로 TCP 연결을 시도한다. (TCP 소켓)
2. HTTP 클라이언트는 설정된 TCP 연결 소켓을 통해 서버로 HTTP 요청 메시지를 보낸다. 이 요청에 객체 경로도 포함된다.
3. HTTP 서버는 TCP 연결 소켓을 통해 요청 메시지를 받는다. 저장 장치로부터 경로의 객체를 추출한다. HTTP 응답 메시지에 그 객체를 캡슐화 하여 소켓을 통해 클라이언트로 보낸다.
4. HTTP 서버는 TCP에게 연결을 끊으라고 한다. (클라이언트가 응답 메시지를 올바로 받을 때까지 끊지 않는다.)
5. HTTP 클라이언트가 응답 메시지를 받으면, TCP 연결이 중단된다. 메시지는 캡슐화된 객체가 HTML 파일인 것을 나타낸다. 클라이언트는 응답 메시지에서부터 파일을 추출하고 HTML파일을 조사하여 다른 객체에 대한 참조를 찾는다 (1~4 단계 반복수행)

HTTP는 통신 프로토콜만 정의할 뿐, 웹 페이지에 대한 관심은 없다.

브라우저는 여러 개의 TCP 연결을 설정하며 다중 연결상에서 웹페이지의 각기 다른 원하는 부분을 요청할 수 있다. HTTP 1.0은 비지속 연결을 지원한다.

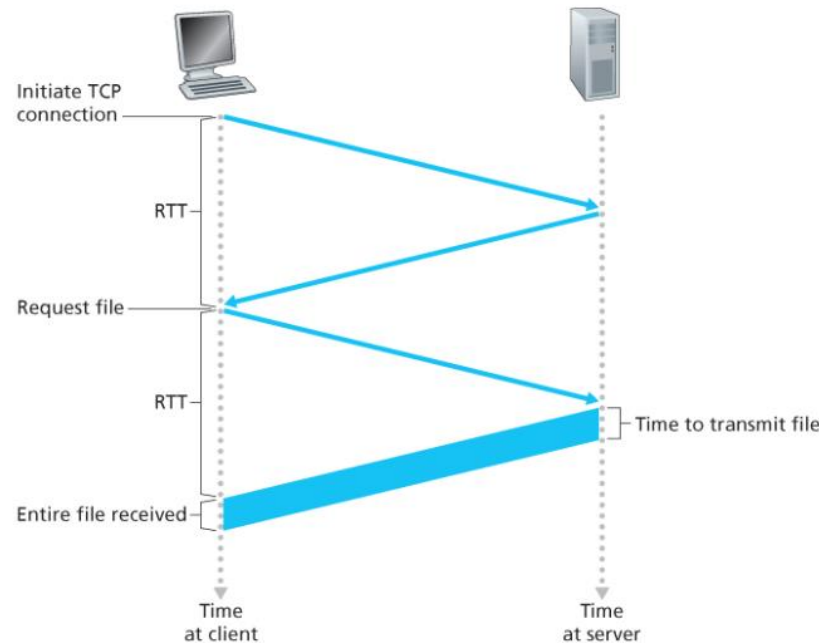
## 2.1 비지속 연결과 지속연결

**RTT(Round-trip time):** 작은 패킷이 Client로부터 Server까지 가고, 다시 Client로 되돌아오는데 걸리는 시간이다.

RTT는 패킷 전파지연, 큐일 지연, 처리 지연 등을 포함

사용자가 하이퍼링크를 클릭하면, 브라우저와 웹 서버 사이에서 TCP 연결을 시도. (3-way handshake)

- Client가 서버로 작은 TCP메시지를 보내고, 서버는 작은 메시지로 응답하고, 마지막으로 클라이언트가 다시 서버에 응답한다.



### 비지속 연결의 단점:

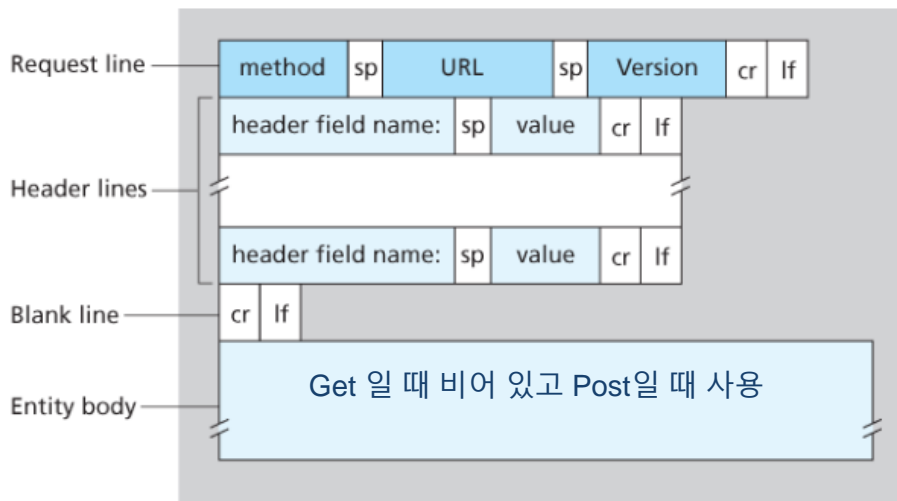
- 각 요청 객체에 대한 새로운 연결이 설정되고 유지되어야 한다.
  - TCP 버퍼가 할당되어야 하고, TCP 변수들이 Client와 서버 양쪽에 유지되어야 하는데 이는 수많은 클라이언트들의 요청을 동시에 서비스하는 웹서버에는 심각한 부하가 걸림.
- 매번 2RTT를 필요함.

## 2.1 비지속 연결과 지속연결

### 지속(persistent) 연결:

- HTTP/1.1 지속 연결에서 서버는 응답을 보낸 후에 TCP 연결을 그대로 유지(비지속 연결도 지원) 같은 클라이언트와 서버간의 이후 요청과 응답은 같은 연결을 통해 보내진다. 즉 같은 서버에 있는 여러 웹 페이지들은 하나의 지속 TCP연결을 통해 보낼 수 있다.
- 객체에 대한 요구는 진행중인 요구에 대한 응답을 기다리지 않고 연속해서 만듬( pipelining)
- 일반적으로 HTTP서버는 일정 시간 사용되지 않으면 연결을 닫는다. 디폴드 모드는 파이프라이닝을 이용한 지속 연결을 사용.

### HTTP 요청 메시지



POST, HEAD,  
PUT, DELETE

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

Post: 폼 필드 입력과 연관됨

Head: 서버가 head 요청을 받으면 HTTP메시지로 응답하는데, 요청 객체는 보내지 않는다.

Put: 웹 서버에 업로드할 객체를 필요로 하는 애플리케이션

Delete: 사용자 또는 애플리케이션 웹서버에 있는 객체 지우기

Figure 2.8 General format of an HTTP request message

## HTTP 응답 메시지

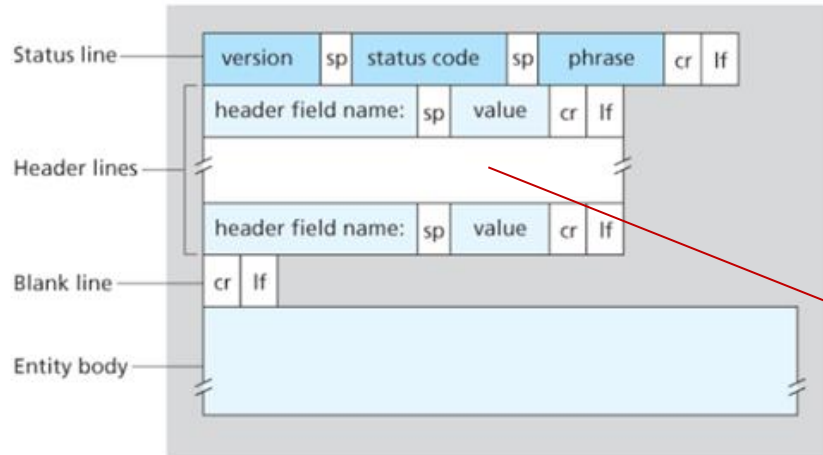


Figure 2.9 General format of an HTTP response message

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)
```

### 헤더 라인

1. Connection : 클라이언트에게 메시지를 보낸 후 TCP 연결을 닫을지 말지 결정한다.
2. Date : HTTP 응답이 서버에 의해 생성되고 보낸 날짜와 시간을 나타낸다.
3. Server : 메시지가 어떤 웹 서버에 의해 만들어졌는지 나타낸다.
4. Last-Modified : 객체가 생성되거나 마지막으로 수정된 시간과 날짜를 나타낸다.
5. Content-Length : 송신되는 객체의 바이트 수를 나타낸다.
6. Content-Type : 개체 몸체 내부(Entity body)의 객체가 어떤 타입인지 나타낸다.

### 상태 라인과 상태 코드

상태 라인(status line)은 버전 필드와 상태 코드, 해당 상태 메시지를 갖는다.

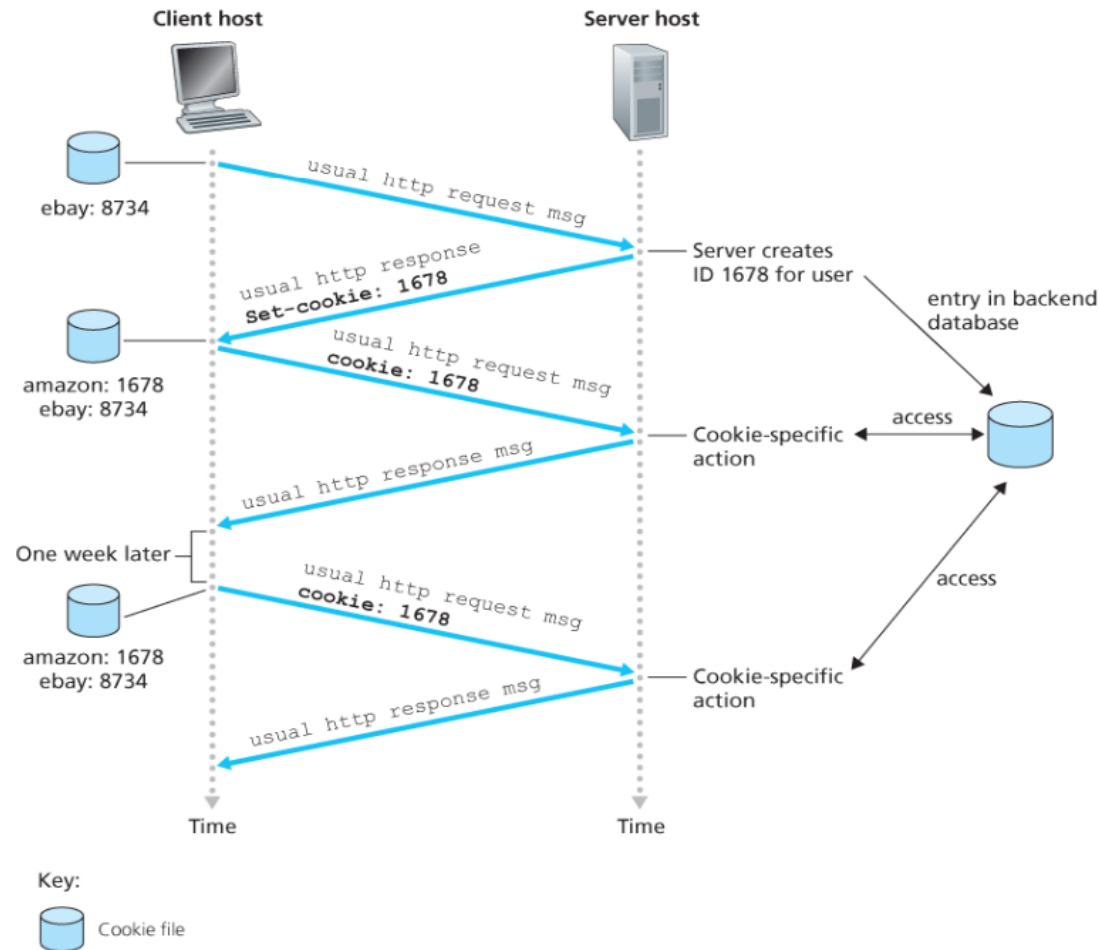
#### 상태 코드와 메시지

- 200 OK: 요청이 성공했고, 정보가 응답으로 보내졌다.
- 301 Moved Permanently: 요청 객체가 영원히 이동되었다. 이때, 새로운 URL은 응답 메시지의 Location 헤더에 나와있다.
- 400 Bad Request : 서버가 요청을 이해할 수 없다.
- 404 Not Found : 요청한 문서가 서버에 존재하지 않는다.
- 505 HTTP Version Not Supported : 요청 HTTP 프로토콜 버전을 서버가 지원하지 않는다.

## 2.2 사용자와 서버 간의 상호 작용: 쿠키

### Cookies:

서버가 사용자 접속을 제한하거나 사용자에게 따라 콘텐츠를 제공하기 원하므로 사용자를 확인할 때 사용



### Web Cache(proxy server):

기점 웹 서버를 대신하여 HTTP 요구를 충족시키는 개체이다.

웹 캐시는 자체의 저장 디스크를 갖고 있어, 최근 호출된 객체의 사본을 저장 및 보존한다.

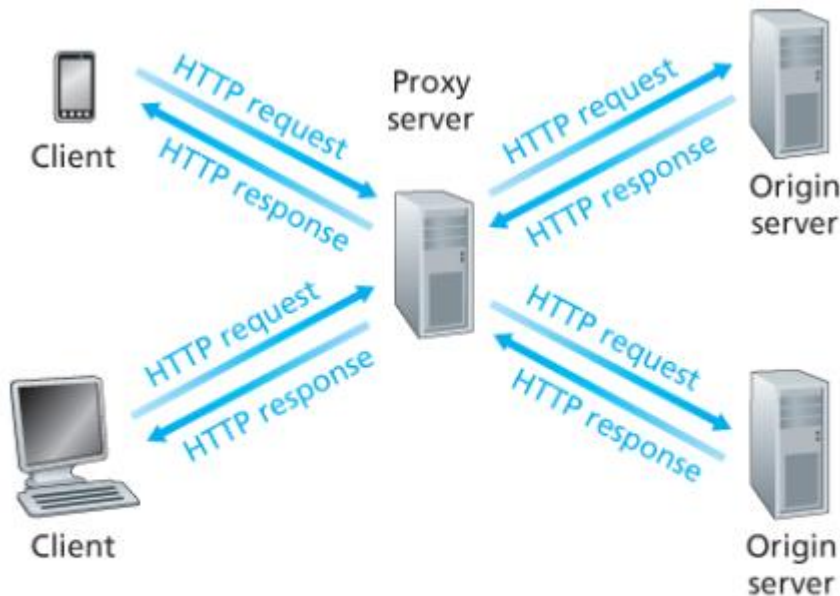


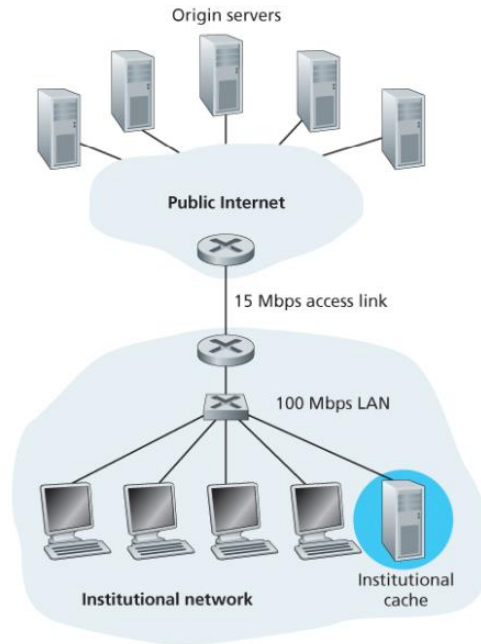
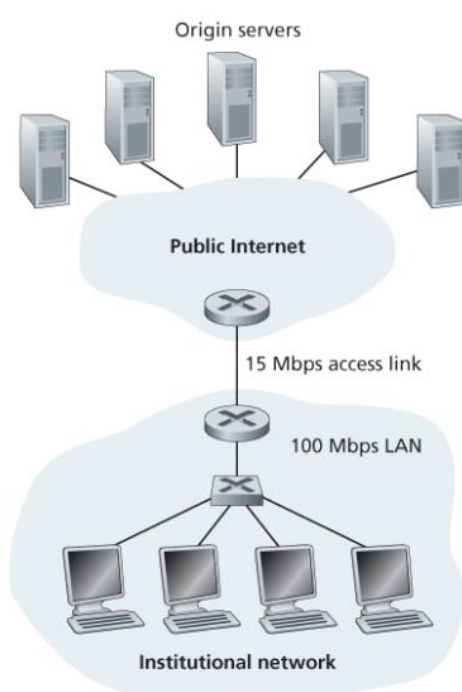
Figure 2.11 Clients requesting objects through a Web cache

### -웹 캐싱의 사용 이유:

- 1). Client 요구에 대한 응답시간을 줄인다.  
(병목현상 방지)
- 2). 한 기관에서 인터넷으로의 접속하는 링크 상의 웹 트래픽을 대폭 줄인다.
- 3). 인터넷 전체의 웹 트래픽을 실질적으로 줄여주어 모든 애플리케이션의 성능 향상.

※웹 캐시는 ISP(university, company, residential ISP)가 구입하고 설치한다.

## 2.3 웹 캐시 미사용과 사용 성능 비교



캐시가 만족시킨 요청 비율(hit rate) 0.2 ~ 0.7인데, 적중률을 0.4로 가정

→ 캐시와 클라이언트는 고속 LAN 연결되어 요청의 40%는 캐시에 의해(10ms이내) 즉시 만족, 나머지 60%의 요청은 여전히 기점 서버에 의해 만족되어야 하므로 트래픽 강도 0.6으로 감소 (<0.8이면 작은 지연)

∴ 평균 지연  
 $0.4 \times 0.01\text{초} + 0.6 \times 2.01 = 1.2...$

평균 객체 크기=1Mb/요청, 기관 브라우저로부터 기점 서버에 대한 평균 요청 비율 = 15요청/초  
 HTTP 메시지 요청이 무시할만큼 작으므로 네트워크 접속 회선에 어떤 트래픽도 발생시키지 않는다고 가정  
 접속 회선에의 인터넷 부분 라우터가 HTTP요청을 전달하고 응답을 받을 때 까지 평균 소요시간을 2초로 가정 (인터넷 지연)

총 응답시간 = LAN 지연 + 접속 지연 + 인터넷 지연 → LAN 트래픽 강도 = (15 요청/초) x (1Mb/요청) / 100Mbps = 0.15초

접속 회선(라우터와 라우터 사이)의 트래픽 강도 → (15 요청/초) x (1 Mb/요청) / 15 Mbps = 1 (접속 회선의 접속률 증가시 비용이 커짐)

LAN의 트래픽 강도는 많아야 수십 ms의 지연을 야기하므로 LAN지연 무시, 트래픽 강도가 1에 가까워지면 회선의 지연은 매우 커진다.



## 2.4 조건부(conditional) GET

문제점: 웹 캐시 내부에 있는 복사본이 새것이 아닐 수 있음. 복사본이 클라이언트에 캐싱된 이후 웹서버에 있는 객체가 갱신되었을 수도 있기 때문이다.

HTTP 요청 메시지가 (1) GET 방식을 사용하고, (2) 헤더 If-modified-since 를 포함한다면, 그것이 조건부 GET이다

### 조건부 GET 동작 과정

1. 브라우저의 요청을 대신해서 프록시 캐시는 요청 메시지를 웹 서버로 보낸다.

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

2. 웹 서버는 캐시에게 객체를 가진 응답 메시지를 보낸다.

```
HTTP/1.1 200 OK
Date: Sat, 3 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 9 Sep 2015 09:23:24
Content-Type: image/gif
(data data data data data ...)
```

3. 일주일 후에 다른 브라우저가 같은 객체를 캐시에게 요청하면 캐시에 저장되어 있다.  
이 객체는 지난주에 웹 서버에서 수정되었으므로, 브라우저는 조건부 GET 으로 조사를 수행한다.

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 9 Sep 2015 09:23:24
```

- If-modified-since 값이 일주일 전에 서버가 보낸 Last-Modified 값과 완벽히 일치한다.
- 이 조건부 GET은 서버에게 If-modified-since에 명시된 값 이후 수정된 경우에만 그 객체를 보내라고 한다.

4. 변경되지 않았다면,

```
HTTP/1.1 304 Not Modified
Date: Sat, 10 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)
(empty entity body)
```

- 위와 같은 응답을 보낸다.  
데이터가 변화가 없어도 객체를 보내는 것은 대역폭을 낭비하는 것이고, 특히 그 객체가 크다면 사용자가 느끼는 응답 시간이 증가된다.
- 위 응답 메시지는 클라이언트에게 요청 객체의 캐싱된 복사본을 사용하라는 것을 의미한다.

2020년 현재 주요 웹 사이트 천만 개의 10%가 HTTP/2를 지원하고 있다.

### HTTP/2 목표:

하나의 TCP연결상에서 멀티플렉싱 요청/응답 지연 시간을 줄이고 요청 우선순위화, 서버 푸시, HTTP 헤더 필드의 효율적인 압축 기능을 제공

HTTP/2는 Client와 서버간의 데이터 포맷 방법과 전송 방법을 변경

### 기존 HTTP/1.1:

- 지속연결시 1개 TCP연결/웹페이지
- **HOL 블로킹 문제** : 병목 구간 발생, 비디오 클립은 병목 링크 통과시 오랜 지연, 작은 객체들은 비디오 클립 뒤에서 대기 시간 증가 → 비디오 클립이 객체들을 블로킹하게 됨.
- **TCP 혼잡 제어**: 각 TCP 연결이 공정하게 병목 링크를 공유하여 같은 크기의 가용한 대역폭을 공정하게 나누게 한다.

**HTTP/2 프레이밍 (framing):** 하나의 웹 페이지를 전송하기 위한 병렬 TCP연결의 수를 줄이거나 제거

- HTTP 메시지를 독립된 프레임들로 쪼개고 인터리빙하고 반대편 사이트에서 재조립하는 것이야말로 HTTP/2의 가장 중요한 개선점이다.
- 메시지 우선순위화 (1 ~ 256 사이의 가중치를 부여함)
- 서버 푸싱: 특정 클라이언트 요청에 대해 여러 개의 응답을 보낼 수 있게 해준다. 서버는 해당 요청들을 기다리는데 소요되는 추가 지연을 없앤다.

### 3. 인터넷 전자메일

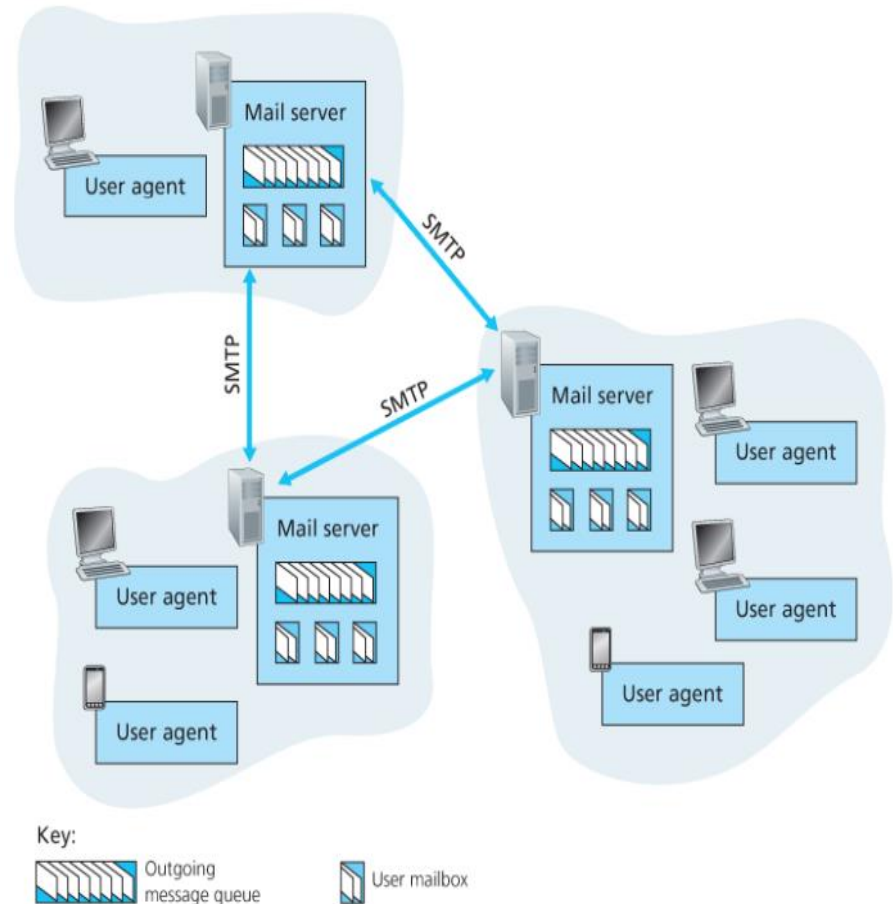
**User Agent(UA):** 사용자 메시지를 읽고, 응답/전달/저장/구성 하게 함. 예) Outlook

**Mail server:** 각 수신자는 메일 서버에 mail box를 갖고 있다. 일반 메시지는 송신자의 UA에서 전달이 시작되고, 송신자의 메일 서버를 거친후에 수신자의 메일 서버로 전달 메시지를 보려면 메일서버는 사용자 계정과 비밀번호를 이용하여 사용자 인증 필수

수신자 메일 서버로 전달할 수 없을 경우 그 메시지를 메시지 큐에 보관하고 나중에 전달을 위해 보관. 재시도는 30분마다 일어나고, 계속 실패시에 서버는 그 메시지를 제거하고 송신자에게 통보

**SMTP:** 메일을 송신자의 메일서버로부터 수신자 메일 서버로 전송하는데 TCP의 신뢰적인 데이터 전송 서비스를 이용

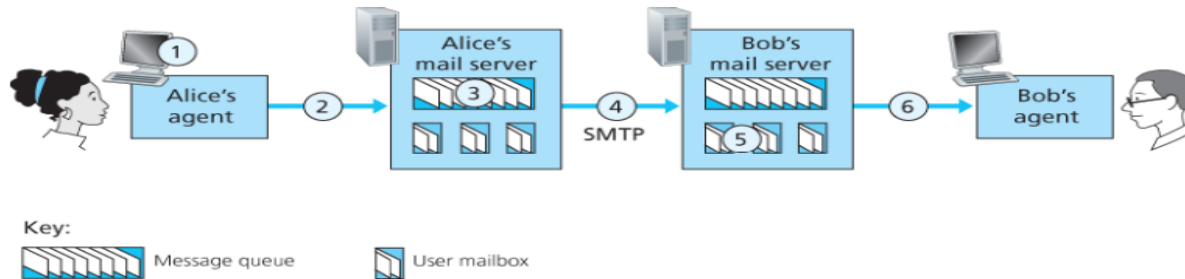
- Client -Sever 구조
- Client & Sever 모두가 메일서버에서 수행



# 3. 인터넷 전자메일

## SMTP:

모든 메일 메시지의 몸체는 단순 7-bit ASCII여야 한다. SMTP 25번 포트로 TCP연결

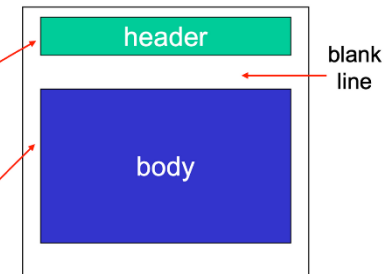


SMTP는 두 메일 서버가 먼 거리에 떨어져 있더라도 중간 메일 서버를 이용하지 않는다.

*handshaking*

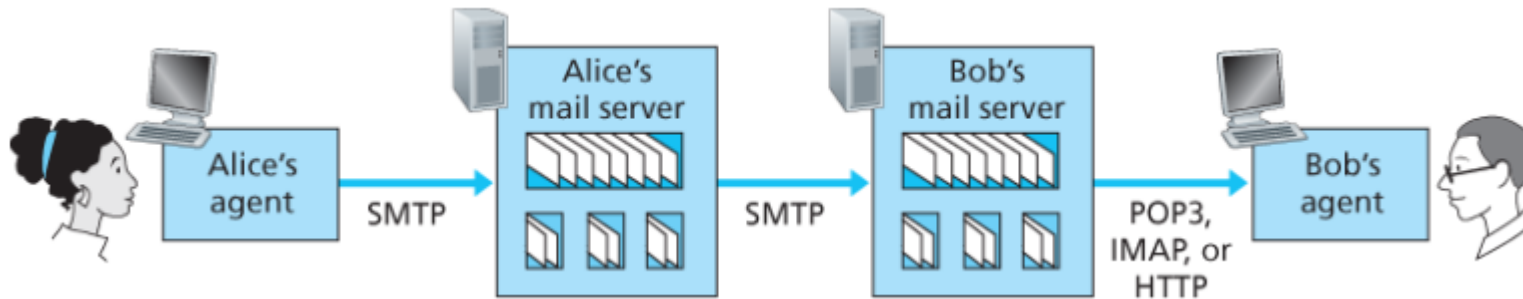
```
S: 220 hamburger.edu server's host name
C: HELO crepes.fr client's host name
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
data exchange
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: . the end of the message
S: 250 Message accepted for delivery
C: QUIT Alice가 보낸 message는 이것이 마지막이라는 의미
connection close
S: 221 hamburger.edu closing connection
```

- header lines, e.g.,
  - To:
  - From:
  - Subject:*different from SMTP MAIL FROM, RCPT TO: commands!*
- Body: the "message"
  - ASCII characters only



### 3. 인터넷 전자메일

수신자 메일 서버에서 수신자 사용자 에이전트까지



SMTP는 PUSH 프로토콜인 반면 메시지를 얻는 것은 PULL 동작이기 때문에 다른 프로토콜 사용

- HTTP:
  - 웹기반 전자메일이나 스마트폰 앱의 경우에 쓰인다.
  - 메일 서버는 SMTP 인터페이스와 HTTP 인터페이스 둘 다 가지고 있음
- IMAP:
  - RFC 3501에 정의된 인터넷 메일 접근 프로토콜

## 4. DNS: 인터넷의 디렉터리 서비스

**Hostname:** 인터넷 호스트의 식별자이지만 호스트 위치에 대한 정보를 거의 제공 안함.

**IP주소 :** IP주소는 4바이트로 구성되고, 계층구조를 구성됨. 왼쪽에서 오른쪽으로 조사함으로써, 그 호스트가 인터넷 어디에 위치하는지에 대한 자세한 정보를 취득.

**DNS(Domain Name System) :** hostname translation, address resolutions

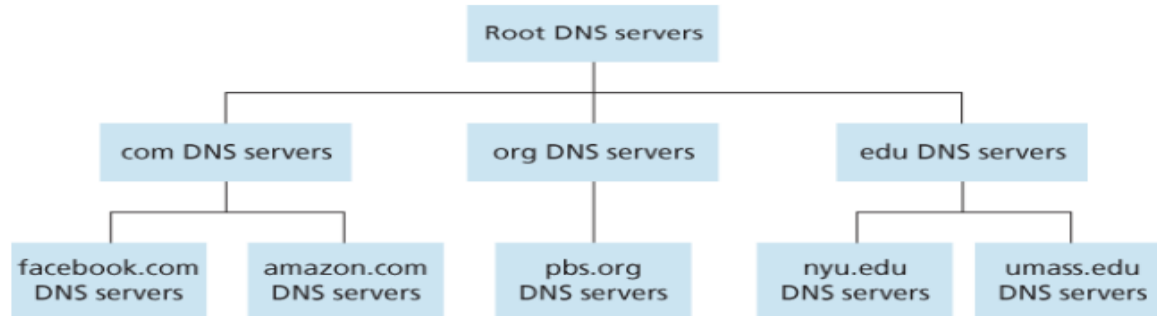
- Implemented in hierarchy of many name servers(DNS 서버들을 계층구조 구현된 분산 DB
- DNS 서버는 주로 BIND(Berkeley Internet Name Domain) 소프트웨어를 수행하는 UNIX 컴퓨터
- DNS 는 호스트가 분산 DB로 질의하도록 허락하는 애플리케이션 계층 프로토콜
- DNS 프로토콜은 UDP상에서 수행되고 포트 번호 53을 이용

**DNS 가 UDP를 사용하는 이유:**

- 빠른 속도: TCP와 달리 UDP는 연결 설정에 드는 비용이 없다.
  - 전달하는 패킷의 크기가 작기때문에 신뢰성이 보장 필요성 없음.(실패시 재전송)
- 연결 상태를 유지할 필요가 없다.
  - 연결상태를 유지하지 않고 정보 기록을 최소화할 수 있는 UDP르 채택

# 4. DNS: 인터넷의 디렉터리 서비스

## 분산 계층 데이터 베이스



- Single point of failure
- Heavy traffic load
- Remote centralized DB
- Maintenance

## 계층 DNS 서버의 종류:

### 1. Root DNS 서버:

- 1000개 이상의 루트 서버 인스턴스로 구성.
- TLD 서버의 IP주소를 제공
- 인터넷 할당 번호 관리기관 ICANN(Internet Corporation for Assignment Names and Numbers)에 의해 조정

### 2. TLD(Top Level Domain):

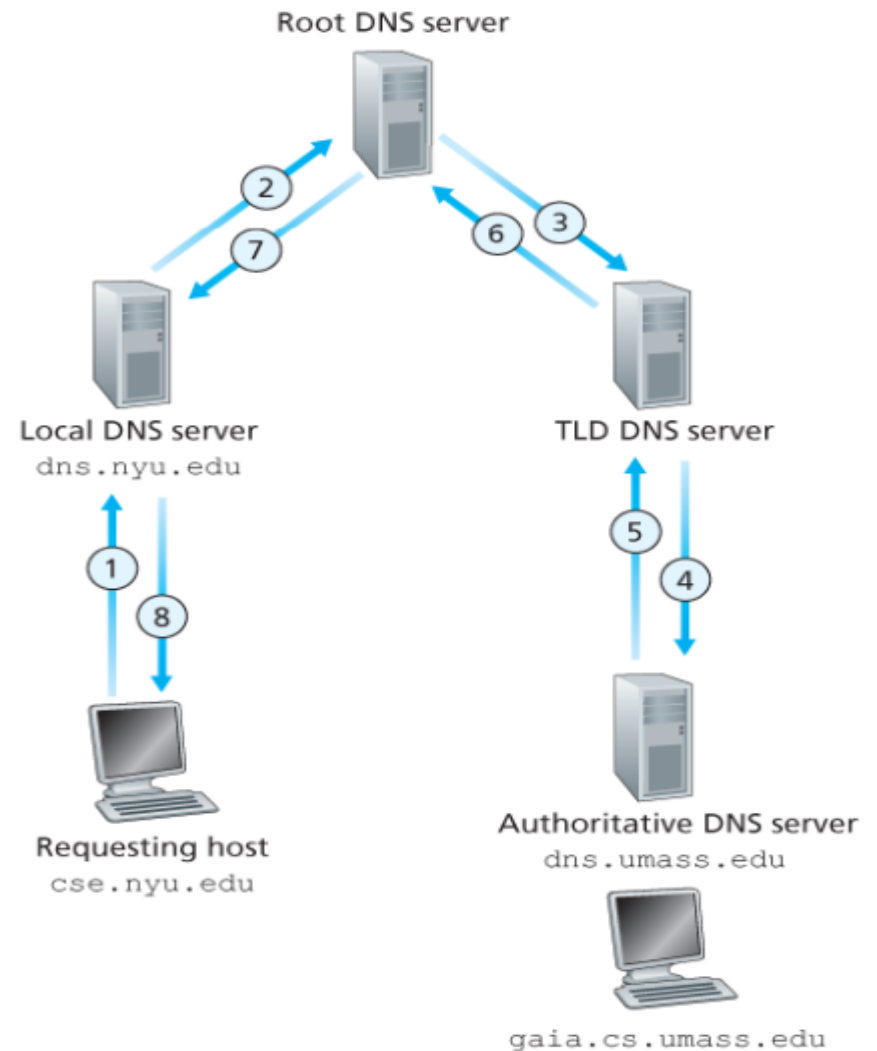
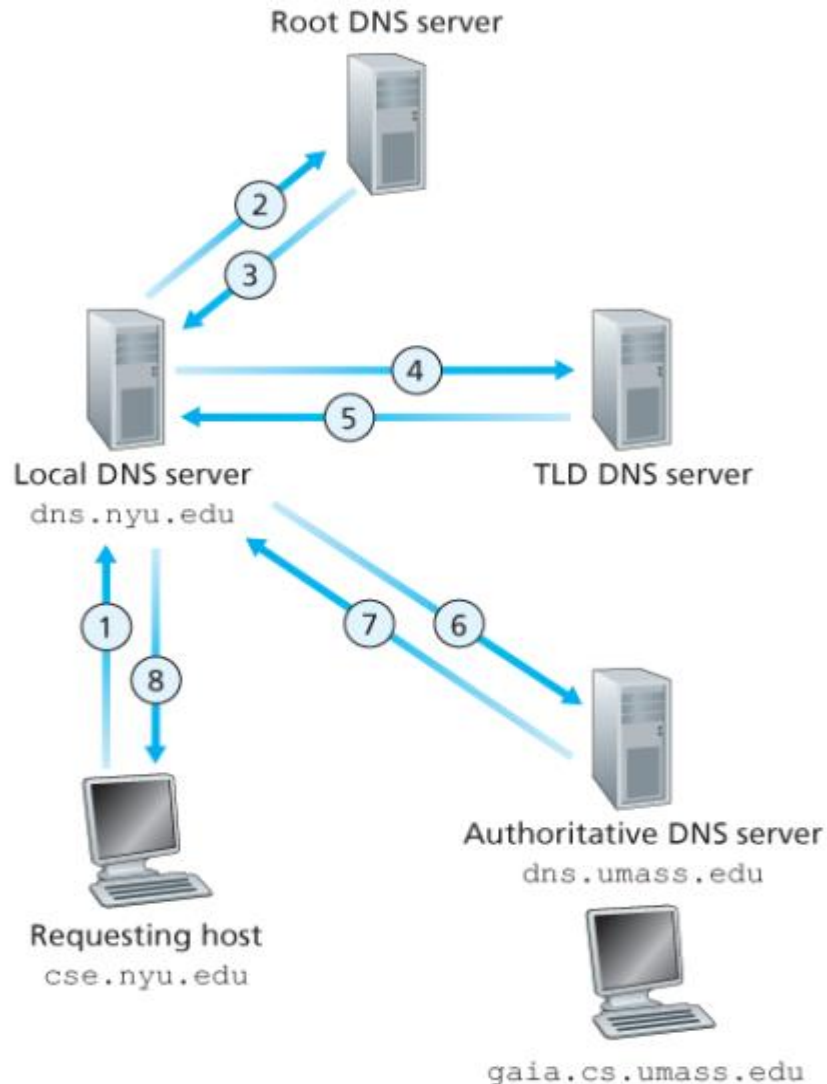
-com, org, net 같은 상위 레벨 도메인과 kr, uk 같은 모든 국가의 상위 도메인에 대한 TLD 서버가 있다.

### 3. Authoritative DNS 서버

-접근이 쉬운 호스트를 가진 모든 기관은 호스트 이름을 IP주소로 매핑하는 공개적인 DNS레코드를 제공

4. 로컬(Local) DNS서버: ISP는 로컬 DNS서버를 갖고, 로컬 DNS서버로부터 IP주소를 호스트에게 제공

## 4. DNS: 인터넷의 디렉터리 서비스

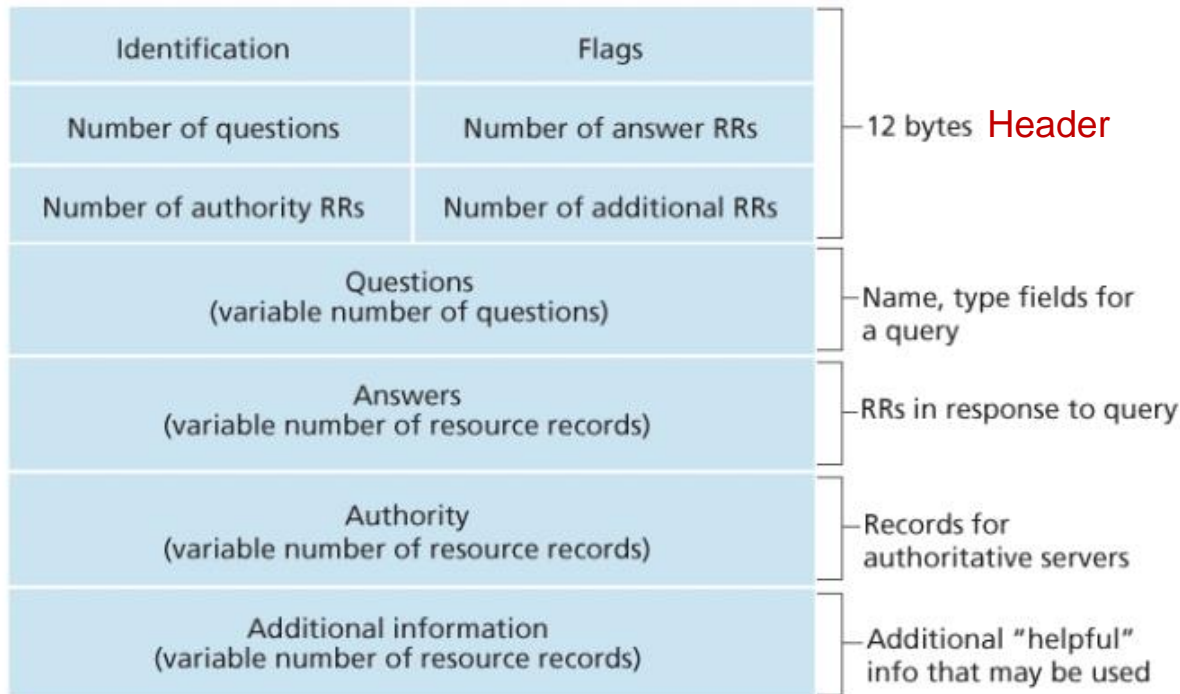




## 4. DNS: 인터넷의 디렉터리 서비스

### DNS 레코드와 메시지:

- 각 DNS는 하나 이상의 자원 레코드를 가진 메시지로 응답한다. (Name, Value, Type, TTL)
- Name과 Value는 Type을 따른다.
- Type = A (Address), NS(name server), CNAME(Canonical Name), MX(Mail eXchange)



### DNS 취약점:

- DDoS 대역폭 플러딩 공격
- DNS중독 공격

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

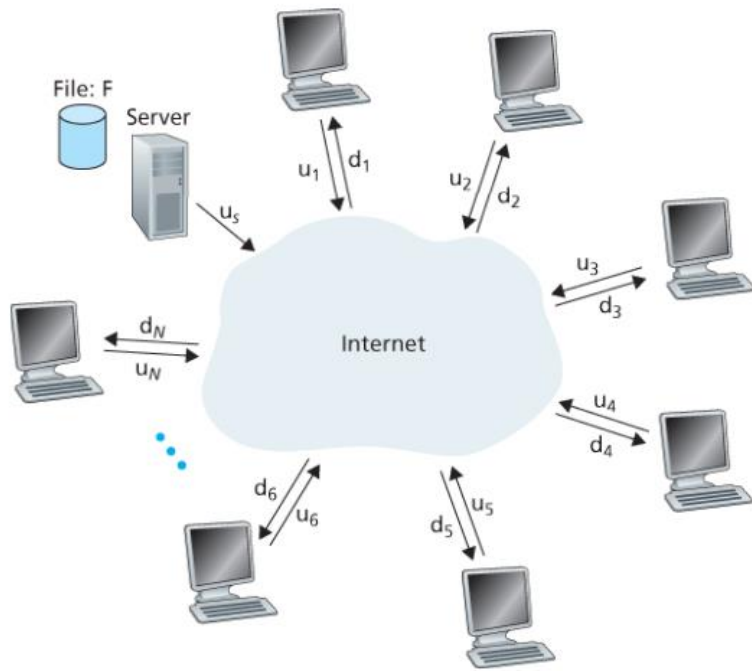
- 주책임 서버 : dns1.networkutopia.com / 주책임 서버 IP : 212.2.212.1
- 부책임 서버 : dns2.networkutopia.com / 부책임 서버 IP : 212.2.212.2

# 5. P2P 파일 분배

## P2P 구조의 확장성:

P2P 구조는 항상 켜져 있는 Infrastructure 서버에 최소한으로 의존하고, 간헐적으로 연결되는 호스트 쌍들(Peer)이 서로 직접 통신한다.

- 클라이언트-서버 파일 분배에서 서버는 파일 복사본을 각 클라이언트에게 보내려면 서버에게 커다란 부하를 주고, 많은 양의 서버 대역폭을 소비한다.
- P2P 파일 분배에서 각 피어는 수신한 파일의 임의의 부분을 다른 피어들에게 재분배할 수 있어서 서버의 분배 프로세스를 도울 수 있다. ( 예: BitTorrent)



### 클라이언트-서버 구조 분배 시간

- 서버는 파일 복사본을  $N$ 개의 피어 각각에게 전송해야 한다. 따라서 서버는  $NF$  비트를 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.

$$D(cs) \geq \max\{ NF/u(s) , F/d(\min) \}$$

$$D_{CS} \geq \max\left(\frac{F \times N}{u_s}, \frac{F}{d_{\min}}\right)$$

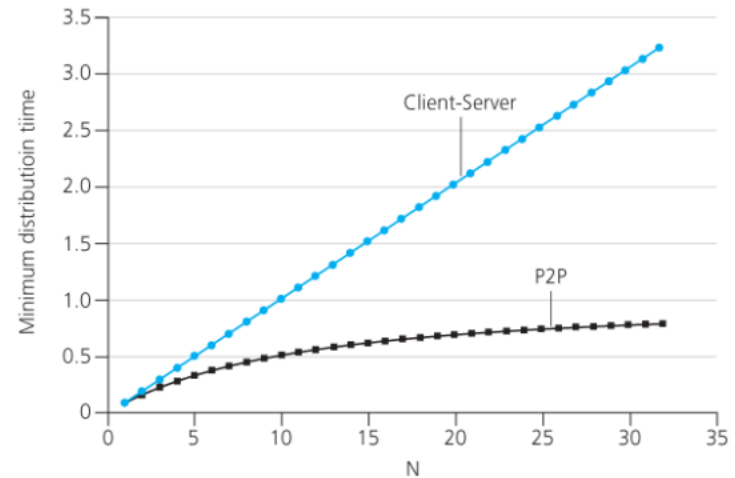
서버 성능이 낮을 때      클라이언트 성능이 낮을 때

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

# 5. P2P 파일 분배

## P2P 분배 시간:

- 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 분배 시간은 적어도  $F/d(\min)$  이다.
- 마지막으로, 시스템의 전체 업로드 용량은 전체적으로 서버의 업로드 속도와 각 피어들의 속도를 더한 것이다. 이를  $u(\text{total})$ 이라 하자.
  - 시스템은 각 피어들 각각에게  $F$  비트를 전달해야 한다. 이는  $u(\text{total})$ 보다 빠르게 할 수 없다.
  - 따라서 최소 분배 시간은  $NF/u(\text{total})$  이다.



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

$$D(p2p) \geq \max \{ F/u(s), F/d(\min), NF/u(\text{total}) \}$$

$$D_{P2P} \geq \max \left( \frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{N \times F}{u_s + \sum_{i=1}^N u_i} \right)$$

서버 성능이 낮을 때

클라이언트 성능이 낮을 때

서버와 Peer들이 업로드하는 성능이 낮을 때

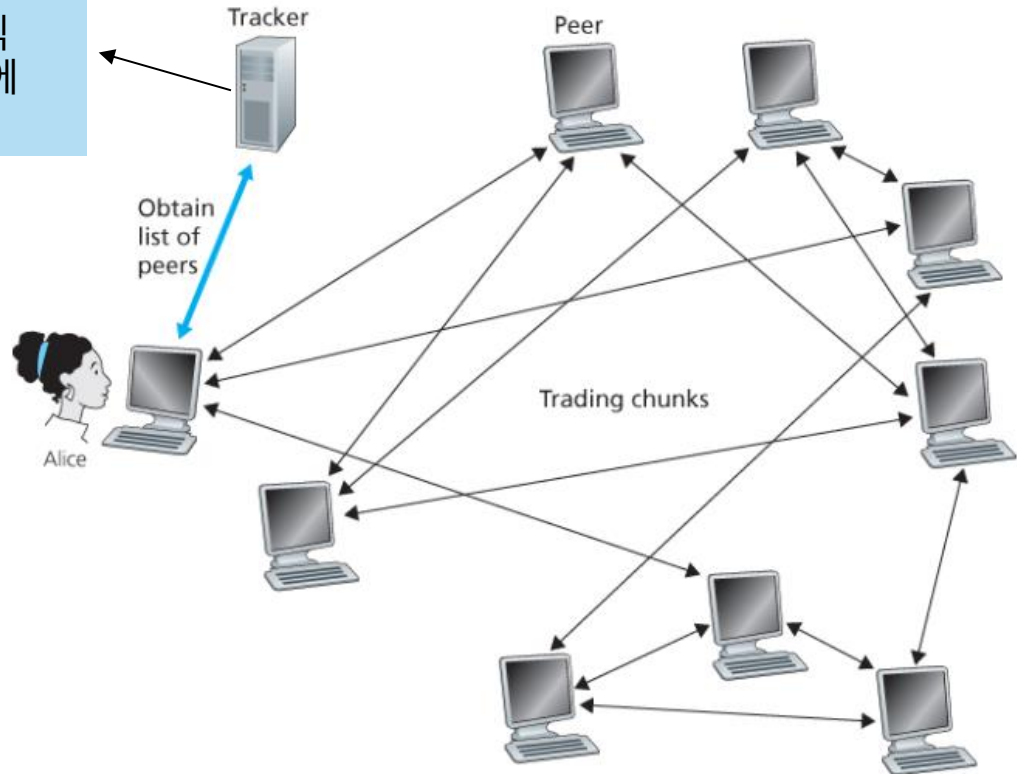
**P2P = 자기 확장성**

# 5. P2P 파일 분배

## 토렌트(Torrent): 특정 파일의 분배에 참여하는 모든 피어의 모임

- 피어들은 서로에게서 같은 크기의 Chunk를 다운로드 한다. (- 256KB)

한 피어가 토렌트에 가입할때 트래커에 자신을 등록하고 주기적으로 자신이 아직 토렌트에 있음을 알려, 트래커는 토렌트에 있는 피어들을 추적 가능



이웃 가운데 가장 드문 청크를 결정하고 이를 먼저 요구하는 것이다.

가장 빠른 속도로 그녀에게 데이터를 제공하는 이웃에게 우선순위를 주는 것이다

## 2.6 비디오 스트리밍과 콘텐츠 분배 네트워크

### HTTP 스트리밍:

HTTP 스트리밍에서 비디오는 HTTP 서버 내의 특정 URL을 갖는 일반적인 파일로 저장된다.  
문제점: 가용 대역폭이 달라도 똑같이 인코딩된 비디오를 전송 받는 문제가 있다.

### DASH(Dynamic Adaptive Streaming over HTTP)

비디오는 여러가지 버전으로 인코딩 되며, 각 버전은 비트율과 품질 수준이 서로 다르다.

- ➔ HTTP서버는 비트율에 따른 각 버전의 URL을 제공하는 매니페스트(manifest)파일을 갖고 있다.
- ➔ 클라이언트는 이 매니페스트 파일을 제공받고, 이에 따라 매번 원하는 버전의 비디오 조각 단위를 선택하여 자신의 상황에 알맞은 비디오 버전을 요청/자유롭게 서로 다른 품질 수준을 변화시킴.

### 콘텐츠 분배 네트워크(Content Distribution Network):

단일 거대 데이터 센터를 구축하고 모든 비디오 자료를 데이터 센터에 저장한 뒤, 전 세계의 사용자에게 비디오 스트림을 데이터 센터로부터 직접 전송한다

문제점:

1. 클라이언트가 데이터 센터로부터 먼 지점에 있는 경우 다양한 통신 링크와 ISP를 거치게 되고, 이 링크들 중 하나라도 비디오 소비율 보다 낮은 전송 용량을 갖는 경우 병목현상이 발생한다.
2. 인기 있는 비디오는 같은 통신 링크를 통해 여러 번 반복적으로 전송될 것이다. 동일한 바이트를 전송하는 데에 반복 비용을 지불하게 된다.
3. 한 번의 장애로 전체 서비스가 중단될 수 있다.

## 2.6 비디오 스트리밍과 콘텐츠 분배 네트워크

CDN은 다수의 지점에 분산된 서버들을 운영하며, 비디오 및 다른 형태의 웹 콘텐츠 데이터의 복사본을 이러한 분산 서버에 저장한다.

사용자는 최적의 사용자 경험을 제공받을 수 있는 지점의 CDN 서버로 연결된다.

CDN은 구글처럼 사설 CDN일 수도 있으며, 제 3자가 운영하는 CDN을 통해 서비스될 수도 있다.

### CDN서버 위치의 2가지 철학:

- Enter Deep
- Bring Home

#### Enter Deep

push CDN servers deep into many access networks

Akamai에 의해 주창된 것으로서 서버 클러스터를 세계 곳곳의 접속 네트워크에 구축함으로써 ISP의 접속 네트워크로 깊숙이 들어가는 것이다.

즉, 최대한 서버를 사용자 근처에 위치시켜 링크 및 라우터를 거치는 횟수를 줄여 지연시간 및 처리율을 개선하는 것이다.

#### Bring Home

smaller number (10's) of larger clusters in POPs near (but not within) access networks

Limelight와 다른 회사들에 의해 적용된 것으로, 좀 더 적은 수의 핵심 지점에 큰 규모의 서버 클러스터를 구축하여 ISP를 Home으로 가져오는 개념이다.

접속 ISP에 연결하는 대신, 일반적으로 CDN들은 그들의 클러스터를 인터넷 교환 지점(IXP)에 배치한다.

이에 따라 Enter Deep보다 처리율(throughput)은 더 낮고 delay가 더 걸릴 수 있지만, 회사의 입장에서는 유지 보수하기에 편하며, 비용이 적게 든다.

CDN은 콘텐츠의 복사본을 이들 클러스터에 저장하는데 모든 복사본을 유지하지는 않는다.

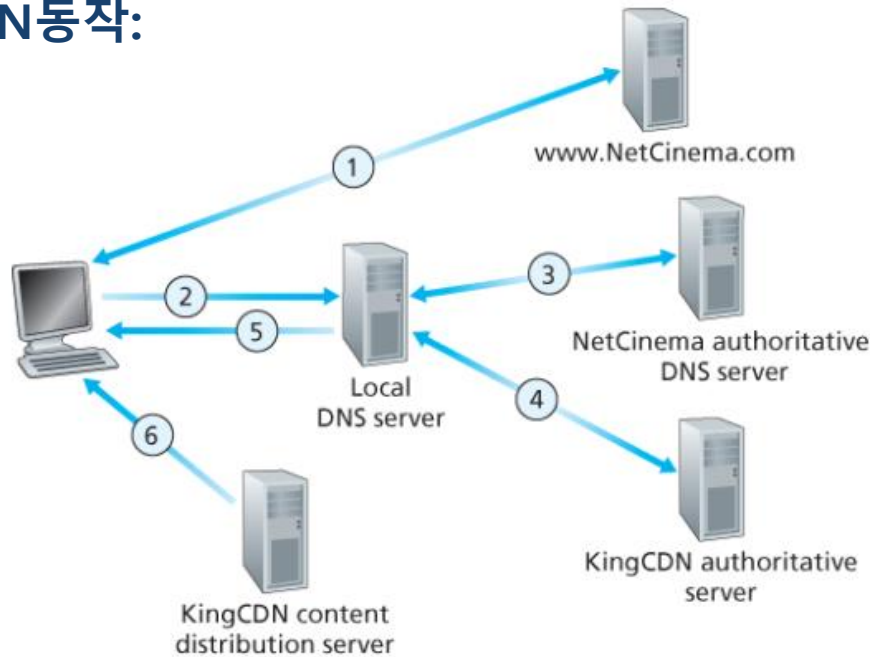
어떤 비디오는 인기가 거의 없거나 특정 국가에서만 인기가 있을 수 있기 때문이다.

실제로 CDN은 클러스터에 대해 사용자의 요청이 오면 중앙 서버나 다른 클러스터로부터 전송받아 사용자에게 서비스하는 동시에 복사본을 만들어 저장하는 pull 방식을 이용한다.

저장 공간이 가득 차게 되면 자주 사용되지 않는 비디오 데이터는 삭제된다.

## 2.6 비디오 스트리밍과 콘텐츠 분배 네트워크

### CDN동작:



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
4. 로컬 DNS는 CDN 서버의 책임 DNS로 질의를 보내고, CDN 콘텐츠 서버의 IP 주소를 로컬 DNS  
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 받아

### Netflix, Youtube



#### 넷플릭스의 아마존 클라우드의 기능

- 콘텐츠 수집: 영화를 수집하고 처리한다. 영화의 스튜디노 마스터 버전을 받아서 아마존 클라우드 시스템의 호스트에 업로드한다.
- 콘텐츠 처리: 아마존 클라우드 시스템의 기기에서는 데스크톱 컴퓨터, 스마트폰, TV에 연결된 게임 콘솔 등 고객들의 다양한 플랫폼에 기기 사양에 적합하도록 각 영화의 여러가지 형식의 비디오를 생성한다. DASH를 이용하여 각 형식별로 다양한 비트율의 여러가지 버전을 생성한다.
- CDN으로 버전 업로드: 영화의 다양한 버전이 생성되면 아마존 클라우드 시스템의 호스트는 이러한 버전을 CDN으로 업로드할 수 있다.

자체 CDN을 구축하기 위해 넷플릭스는 IXP 및 거주용 ISP 자체에서 서버 (rack)를 설치했다.

현재 IXP 위치에 200대 이상의 서버 rack과 서버 rack을 수용하는 수백 개의 ISP 장소도 보유하고 있다.

각각의 rack 서버에는 10 Gbps 이더넷 포트와 100 테라바이트 이상의 스토리지가 있다.

넷플릭스는 푸시 캐싱(push caching)을 사용하여 IXP 및 ISP CDN 서버를 채운다.

전체 라이브러리를 보유할 수 없는 위치의 경우, 매일매일 가장 많이 결정되는 비디오만 푸시한다.

1. 사용자가 재생할 영화를 선택한다.
2. 아마존 클라우드에서 실행 중인 넷플릭스 소프트웨어가 영화 사본을 갖고 있는 CDN 서버를 결정한다.
3. 영화가 있는 서버 중에서 클라이언트 요청에 대한 최적의 서버를 결정한다.
4. 일반적으로 로컬 ISP가 CDN 서버 rack이 있다면 해당 CDN을 사용하거나, 근처 CDN 서버가 있는 IXP를 사용한다.
5. 클라이언트는 요청된 영화의 다른 버전에 대한 URL을 가진 메타데이터 파일과 특정 서버의 IP 주소를 보낸다.
6. 클라이언트와 해당 CDN 서버는 독점 버전의 DASH를 이용하여 직접 상호작용한다.

넷플릭스는 자체 CDN을 사용하고 있기 때문에 DNS redirection을 사용할 필요가 없다.

대신 아마존 클라우드에서 실행되는 것처럼 넷플릭스 소프트웨어는 클라이언트에게 특정 CDN 서버를 사용하도록 알려준다.

#### 유튜브

넷플릭스와 마찬가지로 자체 CDN을 사용한다.

대부분의 경우 클러스터 선택 정책은 클라이언트와 클러스터간의 RTT가 가장 적은 곳을 선택하는 것이다.

때로는 작업 부하를 위해 더 멀리 있는 CDN을 선택하기도 한다.

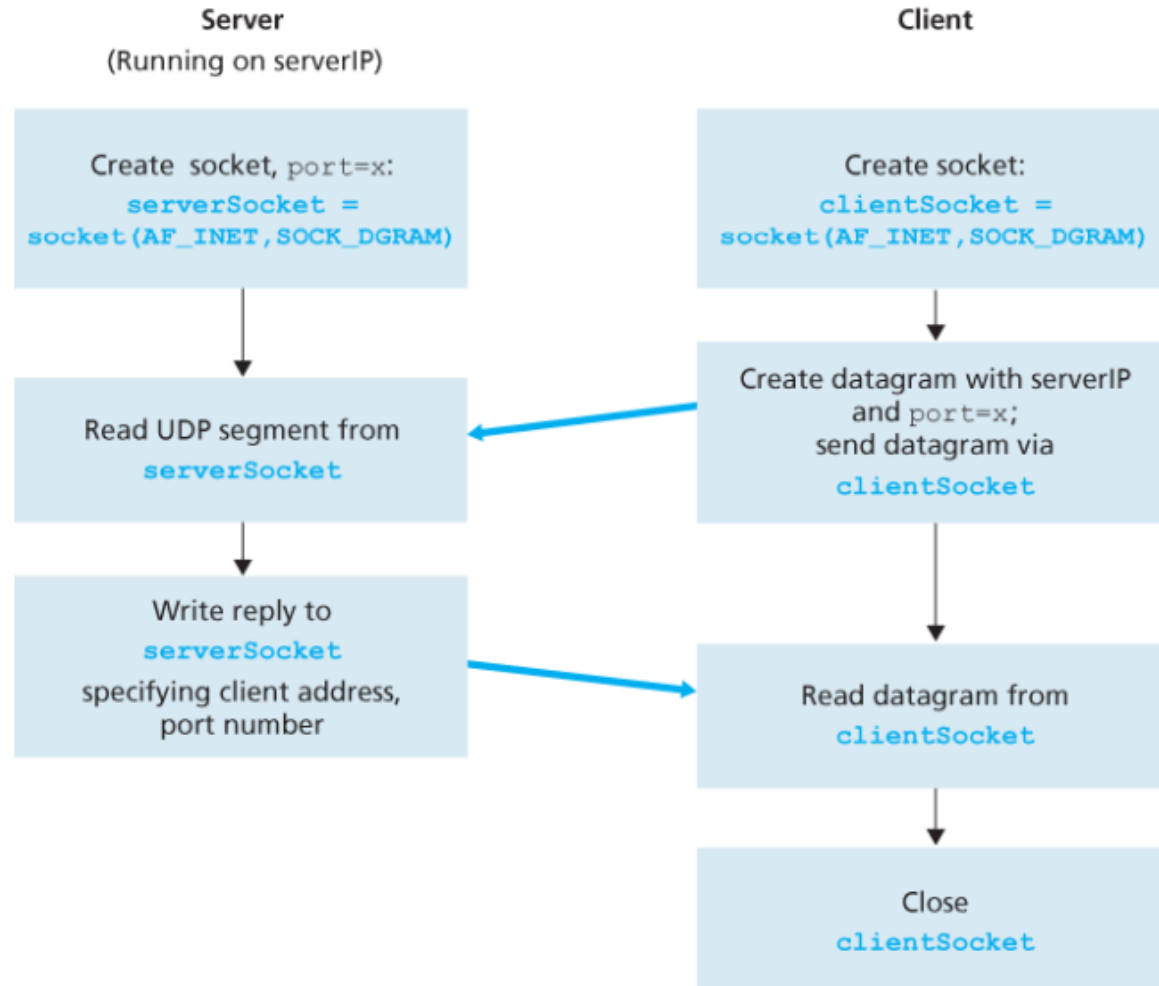
유튜브는 HTTP 스트리밍을 채용하여 사용자가 직접 버전을 선택하게 했다.

재생 위치 조정과 초기 종료로 인한 대역폭과 서버 자원의 낭비를 줄이기 위해, 유튜브는 HTTP byte-range 헤더를 이용해 목표한 분량의 선인출 데이터 이후에 추가로 전송되는 데이터 흐름을 제한한다.

클러스터 선택 정책: Geographically Closest, RT Measurements

## 2.7 소켓 프로그램: 네트워크 애플리케이션 생성

### UDP 소켓 프로그램





## UDPClient.py

소켓을 생성할 때는 따로 소켓의 포트 번호를 명시하지 않아도 되며, 운영체제가 이 작업을 대신 수행한다.

```
# 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, SOCK_DGRAM은 UDP 소켓임을 의미한다.
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 변환한다.
# sendto() 메서드는 목적지 주소를 메시지에 붙이고 그 패킷을 프로세스 소켓인 clientSocket으로 보낸다.
# 클라이언트 주소도 같이 보내지는데 이는 자동으로 수행된다.
clientSocket.sendto(message.encode(),(serverName, serverPort))

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

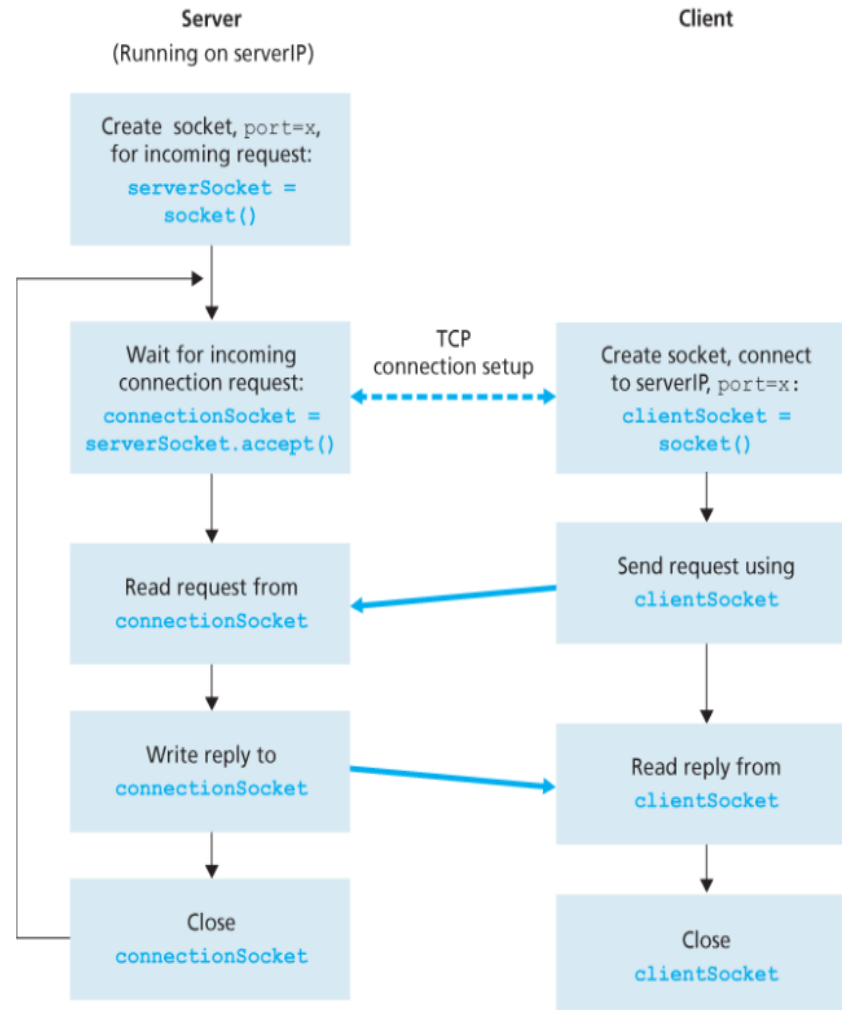
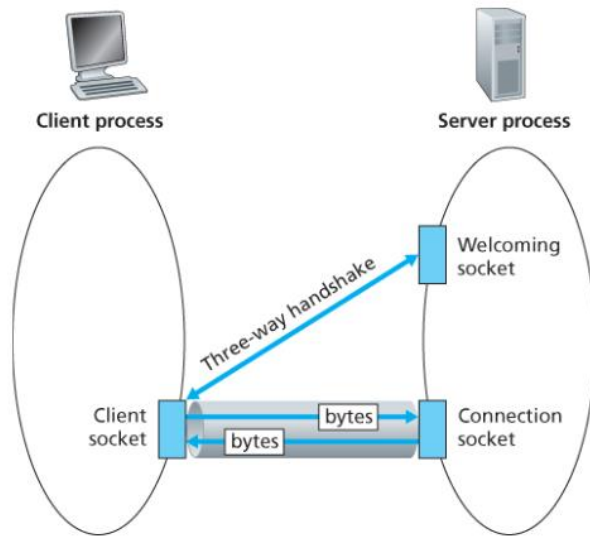
# 출력
print(modifiedMessage.decode())

# 소켓 닫기
clientSocket.close()
```

## UDPServer.py

while 문을 통하여 한 번의 통신 이후에도, 계속 다음 UDP 패킷이 도착하기를 기다린다.

```
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 포트로 패킷을 보내면 해당 소켓으로 패킷이 전달된다.  
serverSocket.bind(('', serverPort))  
  
print("The server is ready to receive")  
  
while True:  
    # 패킷이 서버에 도착하면 데이터는 메시지에 할당되고 패킷의 출발지 주소는 clientAddress에 저장된다.  
    # 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 패킷을 서버에 보낸다.  
    # 서버의 주소도 같이 보내지는데 이는 자동으로 수행된다.  
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```



## TCPClient.py

주석은 UDP와 다른 부분을 위주로 작성하였다.

```
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 해준다.)  
clientSocket = socket(AF_INET, SOCK_STREAM)  
  
# 클라이언트가 TCP 소켓을 이용하여 서버로 데이터를 보내기 전에 TCP 연결이 먼저 클라이언트와 서버 사이에 설정되어야 한다.  
# 해당 라인으로 TCP 연결을 시작하고, connect() 메서드의 파라미터는 연결의 서버 쪽 주소이다.  
# 이 라인이 수행된 후에 3-way handshake가 수행되고 클라이언트와 서버 간에 TCP 연결이 설정된다.  
clientSocket.connect((serverName, serverPort))  
  
sentence = raw_input('Input lowercase sentence:')  
  
# 클라이언트 소켓을 통해 TCP 연결로 보낸다. UDP 소켓처럼 패킷을 명시적으로 생성하지 않으며 패킷에 목적지 주소를 붙이지 않는다.  
# 대신 클라이언트 프로그램은 단순히 문자열에 있는 바이트를 TCP 연결에 제공한다.  
clientSocket.send(sentence.encode())  
  
# 서버로부터 바이트를 수신하기를 기다린다.  
modifiedSentence = clientSocket.recv(1024)  
print('From Server: ', modifiedSentence.decode())  
  
# 연결을 닫는다. 이는 클라이언트 TCP가 서버의 TCP에게 TCP 메시지를 보내게 한다.  
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 클라이언트를 위한 연결 소켓을 서버에 생성한다.
    # 그 뒤 클라이언트와 서버는 핸드셰이킹을 완료해서 클라이언트의 소켓과 연결 소켓 사이의 TCP 연결을 생성한다.
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.encode())

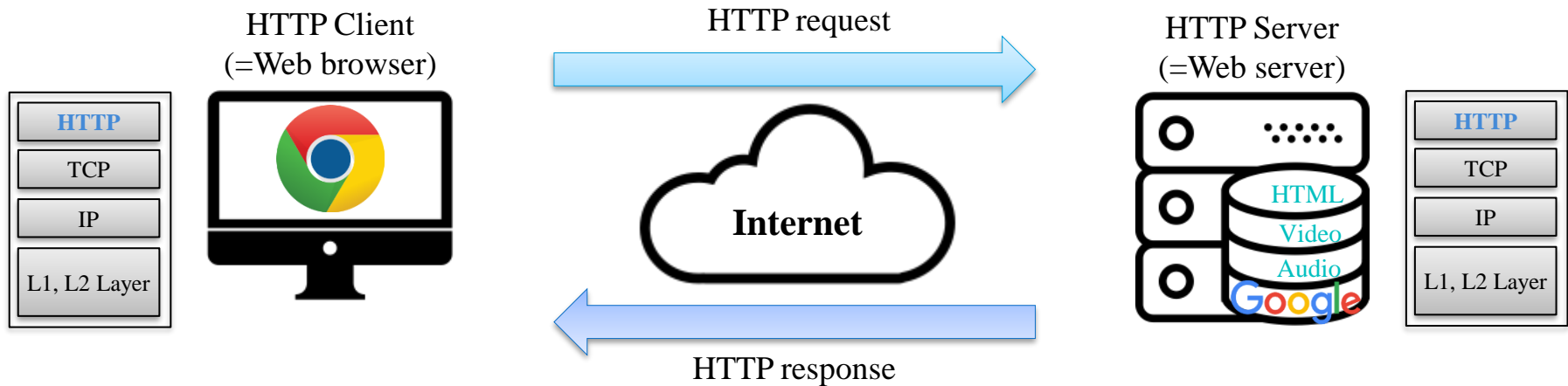
    # 응답을 보내고 연결 소켓을 닫는다. 그러나 환영소켓인 serverSocket이 열려있어 다른 클라이언트가 서버에 연결을 요청할
    connectionSocket.close()
```

# 실습. HTTP



## ■ HyperText Transfer Protocol

- ✓ 인터넷을 위한 텍스트 기반 클라이언트/서버 프로토콜
- ✓ HTML 파일, 그림, 스타일 등과 같은 리소스를 전송하기 위함
- ✓ 클라이언트가 요청(request)하고 서버가 응답(response) 방식으로 동작



## ■ Uniform Resource Locator

✓ 컴퓨터 네트워크 상에서 리소스가 어디 있는지를 알려주기 위한 주소





## REQUEST

**GET** http://127.0.0.1:5500/styles/navigation.css **HTTP/1.1**

HTTP request  
line

HOST: **127.0.0.1:5500**

Accept: text/css,\*/\*;q=0.1

Accept-Language: en-GB,en;q=0.5

Accept-Encoding: gzip, deflate, br

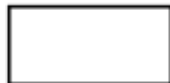
User-Agent: Mozilla/5.0 (Windows NT 10.0;

Win64; x64; rv:102.0) Gecko/20100101 Firefox/102.0

Connection: keep-alive

**<CRLF>**

HTTP headers



HTTP body  
(empty)

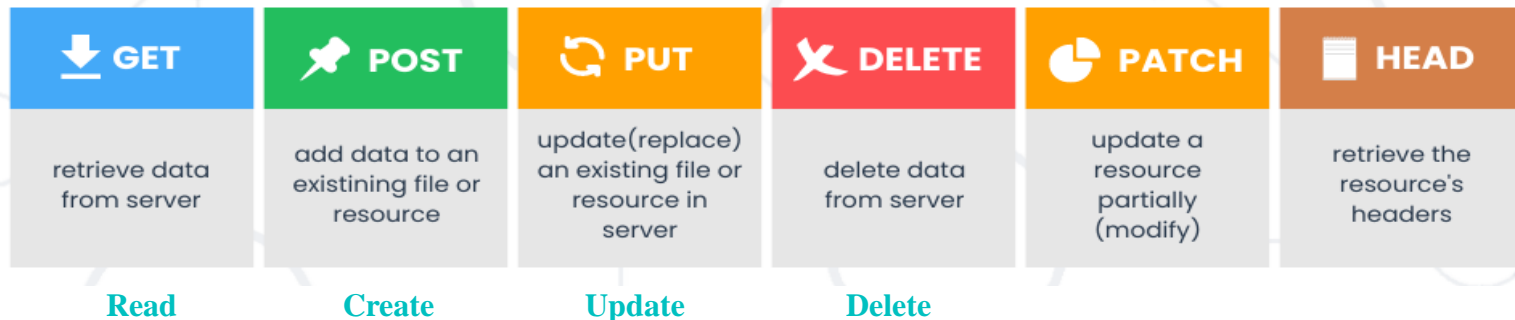
## ■ GET – 서버로부터 데이터 요청

- ✓ URL에 쿼리스트링(=파라미터)을 붙여서 전송
- ✓ 요청 정보가 캐싱됨(idempotent – 항상 같은 응답 보장)
- ✓ 예: 게시판 목록 보여주기, 게시판 글 보여주기 등
- ✓ <https://jsonplaceholder.typicode.com/users?id=3>

## ■ POST – 서버의 리소스를 생성/변경

- ✓ HTTP body에 데이터를 담아서 전송(대용량 전송 가능)
- ✓ 요청 정보가 캐싱되지 않음
- ✓ 예: 게시판 글 등록 등
- ✓ <https://jsonplaceholder.typicode.com/users>

## HTTP Request Methods



## RESPONSE

```
HTTP/1.1 200 OK
```

HTTP response  
status line

```
Date: Wed, 06 Jul 2022 09:30:28 GMT  
Accept-Ranges: bytes  
Content-Length: 2005  
Content-Type: text/css; charset=UTF-8  
<CRLF>
```

HTTP response  
headers

```
nav.navbar {  
    ...some style  
}
```

HTTP response  
body



404. That's an error. HTTP 응답: GET 메서드

The requested URL /nonexistent was not found on this server. That's all we know.

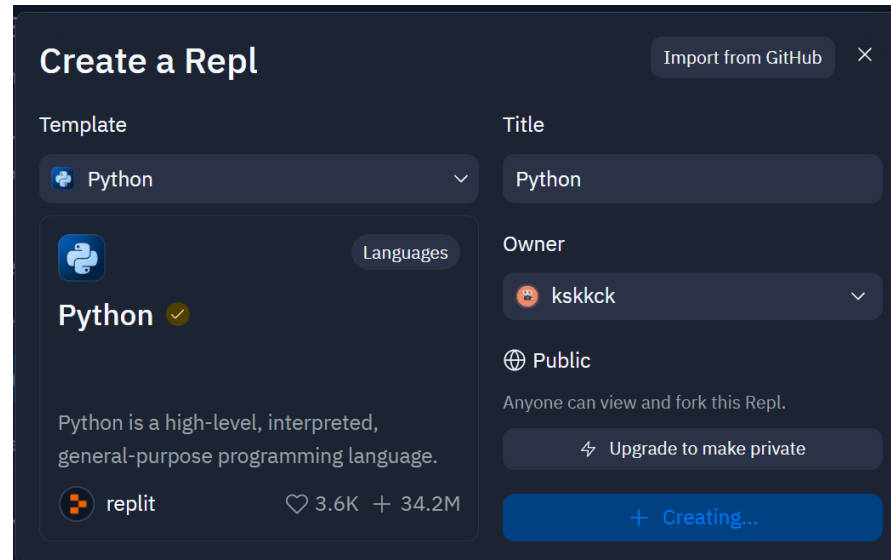
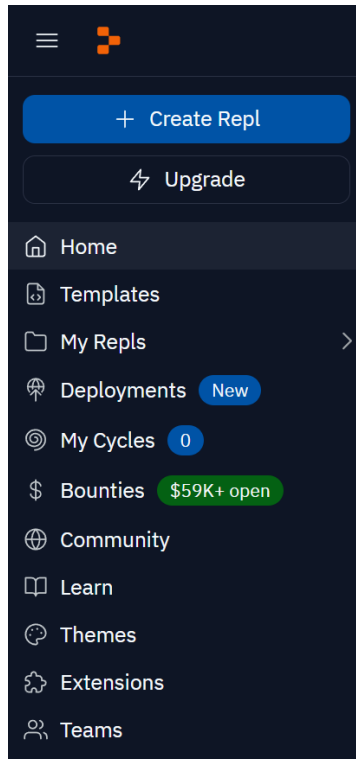


## HTTP Response Status Codes

Status Code	Action	Description
200	OK	Successfully retrieved resource
201	Created	A new resource was created
204	No Content	Request has nothing to return
301 / 302	Moved	Moved to another location (redirect)
400	Bad Request	Invalid request / syntax error
401 / 403	Unauthorized	Authentication failed / Access denied
404	Not Found	Invalid resource was requested
409	Conflict	Conflict was detected, e.g. duplicated email
500 / 503	Server Error	Internal server error / Service unavailable

## ■ replit

✓ <https://replit.com/>



## ■ HTTP 테스트 서비스

✓ <https://jsonplaceholder.typicode.com/>

### {JSON} Placeholder

Free fake API for testing and prototyping.

Powered by JSON Server + LowDB. Tested with XV.

Serving ~2 billion requests each month.

## ■ <https://jsonplaceholder.typicode.com/users>

```
main.py > ...  
1 import requests  
2  
3 url = 'https://jsonplaceholder.typicode.com/users'  
4  
5 response = requests.get(url)  
6  
7 print(response.text)  
8 print(response.status_code)  
9
```

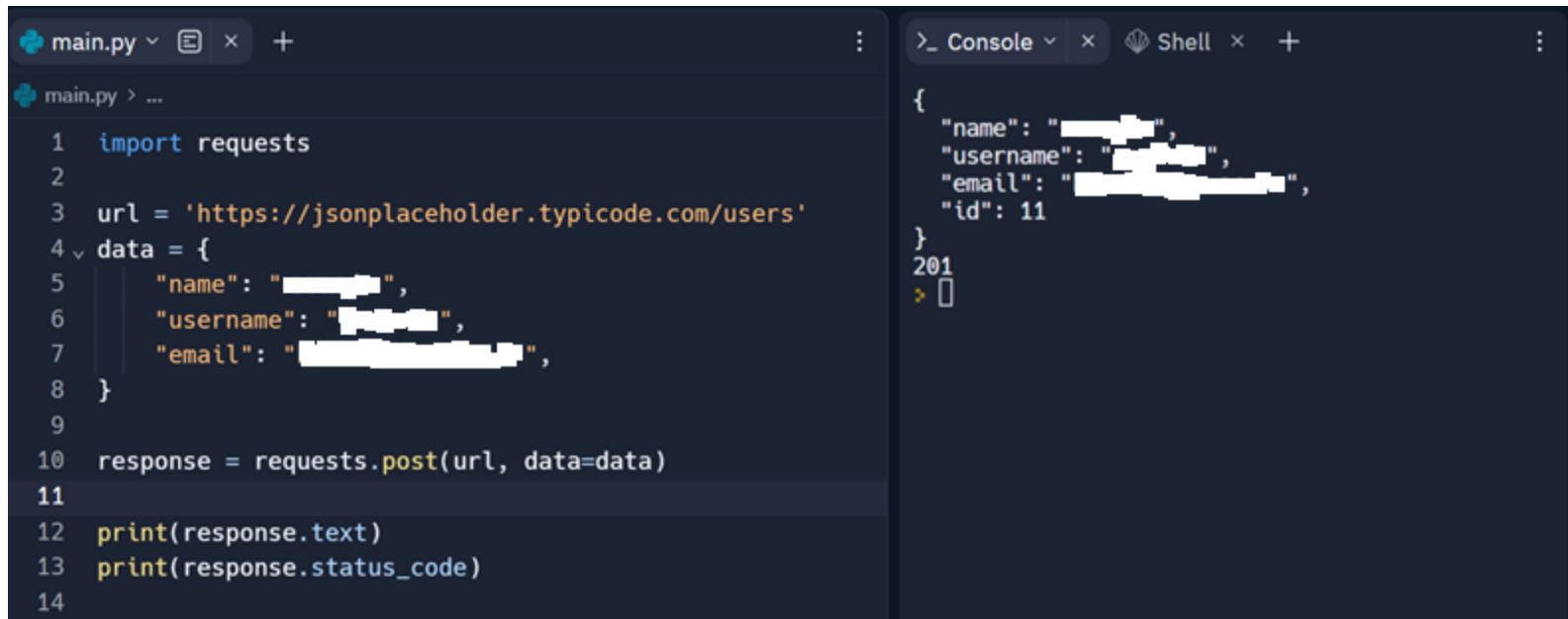
```
>_ Console > ...  
[  
  {  
    "id": 1,  
    "name": "Leanne Graham",  
    "username": "Bret",  
    "email": "Sincere@april.biz",  
    "address": {  
      "street": "Kulas Light",  
      "suite": "Apt. 556",  
      "city": "Gwenborough",  
      "zipcode": "92998-3874",  
      "geo": {  
        "lat": "-37.3159",  
        "lng": "81.1496"  
      }  
    },  
    "phone": "1-770-736-8031 x56442",  
    "website": "hildegard.org",  
    "company": {  
      "name": "Romaguera-Crona",  
      "catchPhrase": "Multi-layered client-serv  
er neural-net",  
      "bs": "harness real-time e-markets"  
    }  
  },  
  {  
    "id": 2,  
    "name": "Ervin Howell",  
    "username": "Antonette",  
    "email": "Shanna@melissa.tv",  
    "address": {  
      "street": "Victor Plains"
```

## ■ <https://jsonplaceholder.typicode.com/users?id=3>

```
main.py > ...  
1 import requests  
2  
3 url = 'https://jsonplaceholder.typicode.com/users'  
4  
5 response = requests.get(url, params={"id": 3})  
6  
7 print(response.text)  
8 print(response.status_code)  
9  
[  
  {  
    "id": 3,  
    "name": "Clementine Bauch",  
    "username": "Samantha",  
    "email": "Nathan@yesenia.net",  
    "address": {  
      "street": "Douglas Extension",  
      "suite": "Suite 847",  
      "city": "McKenziehaven",  
      "zipcode": "59590-4157",  
      "geo": {  
        "lat": "-68.6102",  
        "lng": "-47.0653"  
      }  
    },  
    "phone": "1-463-123-4447",  
    "website": "ramiro.info",  
    "company": {  
      "name": "Romaguera-Jacobson",  
      "catchPhrase": "Face to face bifurcated i  
nterface",  
      "bs": "e-enable strategic applications"  
    }  
  }  
]  
200  
>
```



- <https://jsonplaceholder.typicode.com/users>



The screenshot shows a code editor with a Python script in the left pane and a console output in the right pane. The script uses the 'requests' library to perform a POST request to the URL 'https://jsonplaceholder.typicode.com/users' with a JSON body containing user details. The console shows the response as a JSON object with a 201 status code.

```
main.py x +
main.py > ...
1 import requests
2
3 url = 'https://jsonplaceholder.typicode.com/users'
4 data = {
5     "name": " ",
6     "username": " ",
7     "email": " ",
8 }
9
10 response = requests.post(url, data=data)
11
12 print(response.text)
13 print(response.status_code)
14
```

```
>_ Console x Shell x +
{
  "name": " ",
  "username": " ",
  "email": " ",
  "id": 11
}
201
> []
```

## ■ 한국환경공단\_에어코리아\_대기오염정보 Open API를 활용하여 미세먼지 측정기를 만드세요.

- ✓ <https://www.data.go.kr/index.do>
- ✓ 측정소별 실시간 측정정보 조회 -> 미세먼지(PM10)농도
- ✓ xml/json 파싱 방법을 검색해서 확인 필요

검색할 지역을 입력하세요 :

검색할 지역을 입력하세요 : 종로구  
9um  
▶

- 한국환경공단\_에어코리아\_대기오염정보 Open API를 활용하여 미세먼지 측정기를 만드세요.

```
import requests

location = input("검색할 지역을 입력하세요: ")
dataUrl = 'http://apis.data.go.kr/B552584/ArpltnInforInquireSvc/getMsrstnAcctoRltmMeasureDnsty?serviceKey=K7PE0BxurvKDd49AYQelwY2cdYElg0it7vmtlb96y%2FfU5an7v3KNzJ5qQEDL1FjGKCPFJCZcwGhkR0Kde66XBg%3D%3D'
params = {
    'returnType': 'json',
    'numOfRows': '100',
    'pageNo': '1',
    'stationName': location,
    'dataTerm': 'DAILY',
    'ver': '1.0',
}

response = requests.get(dataUrl, params=params)
print(response.json()['response']['body']['items'][0]['pm10Value'] + 'um')
```

## ■ 국립중앙도서관 Open API를 활용하여 ISBN 검색기를 만드세요.

✓ <https://www.nl.go.kr/NL/contents/N31101010000.do>

✓ xml/json 파싱 방법을 검색해서 확인 필요

ISBN을 입력하세요 :

ISBN을 입력하세요 : 9791185475158

컴퓨터 네트워킹 : 하향식 접근



- 국립중앙도서관 Open API를 활용하여 ISBN 검색기를 만드세요.

```
isbn = input("ISBN을 입력하세요: ")
libraryUrl = 'https://www.nl.go.kr/NL/search/openApi/search.do'
params = {
    'key': '1ca63415483ca538f09a8fbdf6a6c8cc27a810f03e13258be1b9324dbe6da050',
    'apiType': 'json',
    'detailSearch': True,
    'isbnOp': 'isbn',
    'isbnCode': isbn
}
```