1️⃣ 어플리케이션 계층 1

Application layer

최상단에 있는 계층으로 현재도 계속 우리와 interaction중

ex) HTTP 기반의 web browsing

network application → end system에서 돌아가며 network를 통해서 데이터를 주고받음

어플리케이션 구조

- Client - Server architecture

  client → 뭔가 필요할 때만 액션을 취함

  server → 매일 같은 자리에서 기다림

  ⇒ server는 고정된 IP 주소를 가짐

  클라이언트는 서로 직접적으로 통신하지 않음

  확장성을 위해 데이터베이스에 IP주소를 저장해둔다

- P2P architecture

  server가 되기도 하고 client가 되기도 함

  임의의 end system끼리 통신함 (어플리케이션은 peer라는 간헐적으로 연결된 호스트 쌍이 직접 통신하게 함)

  self scalability → 자기 확장성 ex. 파일 공유 어플리케이션에서 각 피어들이 파일을 요구해 작업 부하 X 각 피어들은 파일을 다른 피어들에게 분배해 시스템에 서비스 능력을 갖춤

Process Communicating

프로세스들 사이 통신 (메세지를 네트워크를 통해 주고받음)

→ IPC (InterProcess Communication) : OS가 socket이라 불리는 interface를 제공 (호스트의 어플리케이션 계층과 트랜스포트 계층간의 인터페이스)

socket ⇒ OS가 제공하는 system call의 한 종류

⇒ 프로세스는 **소켓**을 통해 네트워크로 메세지를 보내고 받음

Address Process

프로세스가 메세지를 받으려면 자기만의 식별자를 가지고 있어야 함

→ IP주소 + port 번호 필요 (한 IP주소의 어떤 프로세스인지 확인하기 위해)

트랜스포트 프로토콜이 어플리케이션에게 제공할 수 있는 서비스

- Data integrity (신뢰적 데이터 전송)
  - 프로토콜이 보장된 데이터 전송 서비스를 제공
  - 데이터가 오류없이 다 정확하게 도착할 것이라는 확신
  - 이 서비스를 제공하지 않으면 데이터 전혀 도착하지 않을 수 있음 → 이런 어플리케이션을 손실 허용 어플리케이션이라고 함
- Timing (시간)
  - 시간 보장 제공
- Throughput (처리율)
  - 명시된 속도에서 보장된 가용 처리율
  - 처리율 요구사항을 갖는 어플리케이션 : 대역폭 민감 어플리케이션
  - elastic app(탄력적 어플리케이션) → 요구사항이 없는 어플리케이션
- Security (보안)
  - 송신 호스트에서 모든 데이터 암호화할 수 있고 수신 호스트에서 트랜스포트 프로토콜은 모두 해독할 수 있음
  - TCP를 어플리케이션 계층에서 강화해 TLS로 보안 서비스 제공

인터넷 전송 프로토콜이 제송하는 서비스

TCP

데이터가 잘 받아졌나 확인함

- flow-control

  sender가 reciever를 overwhelm하지 않음 → reciever가 감당하지 못할 정도로 보내지 않음
- congestion control

  네트워크가 oveload되지 않도록 sender가 조절해서 데이터를 보냄

⇒ hand-shake 과정 (클라이언트와 서버에 패킷이 곧 도달할테니 준비하라고 알려주는 역할)

UDP

reliability를 보장하지 않음 → 데이터 잘 받았나 확인하지 않음

- 비연결형으로 hand-shake 과정이 없고 비신뢰적인 데이터 전송 서비스 제공
- connection setup을 하지 않아서 빠름 → UDP 서버가 열려있으면 그냥 보냄

  but 혼잡으로 인해 종단 간 처리율이 낮아져서 속도가 오히려 더 낮아질 수 있음

TCP와 UDP 모두 encryption 제공하지 않아서 어플리케이션 레벨에서 SSL을 사용해서 security 제공

둘 다 처리율, 시간 보장 서비스를 제공하지 않는다

TCP는 못 받은 패킷이 있으면 재전송, UDP는 그러지 않음

ex) 친구와 전화할 때 일부 못 받은 패킷이 있음 → 재전송한다고 해서 중요한 패킷이 아님

⇒ 이러한 이유로 TCP 말고 UDP도 사용

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

어플리케이션 계층 프로토콜

→ 어플리케이션의 프로세스가 서로 메세지를 보내는 방법을 정의

- 주고받는 메세지의 유형 → request, response 등
- 메세지 문법 → 메세지에 어떤 field가 있고 어떻게 구분이 되는지
- 메세지의 의미 → field들의 의미
- 메세지 보내고 어떻게 반응할지를 결정하는 규칙

Web

object로 이루어지며 client-server model에서 사용됨

→ Object : 단순히 단일 URL로 지정할 수 있는 하나의 파일(HTML, JPEG 이미지, 자바스크립트 등)

대부분의 웹 페이지는 기본 HTML과 여러 참조 객체로 구성됨

- Client → request, HTTP protocol을 사용해 받음
- Server → receive

HTTP

메세지의 구조 및 클라이언트와 서버가 메세지를 어떻게 교환하는지에 대해 정의하고 있으며 TCP를 사용함

1. HTTP 클라이언트는 먼저 서버에 TCP 연결을 시작한다.
2. 연결이 이루어지면, 브라우저와 서버 프로세스는 각각의 소켓 인터페이스를 통해 TCP로 접속한다.

3. 클라이언트는 HTTP 요청 메세지를 소켓 인터페이스로 보내고, 소켓 인터페이스로부터 HTTP 응답 메세지를 받는다.

→ TCP를 통해 메세지를 보내기 때문에 모든 HTTP 요청 메세지가 궁극적으로 서버에 도착한다.

client는 port번호 80번 사용 (server)

server는 client로부터 TCP connection을 받아들임

stateless ⇒ client가 어떤 요청을 했었는지 기억하고 있지 않음

HTTP connections

HTTP response를 보내기 위해 TCP connection이 사전에 맺어짐

→ 사전에 맺은 TCP connection을 계속 사용할 것인가 = persistent HTTP

→ 사전에 맺는 TCP connection을 계속 사용하지 않고 계속 새로 생성한다 = non-persistent HTTP

non-persistent connection (비지속 연결)

→ 각 request / response 쌍이 분리된 TCP 연결을 통해 보내지는 것

connection을 열고 **최대 하나의 object**만 보낼 수 있음

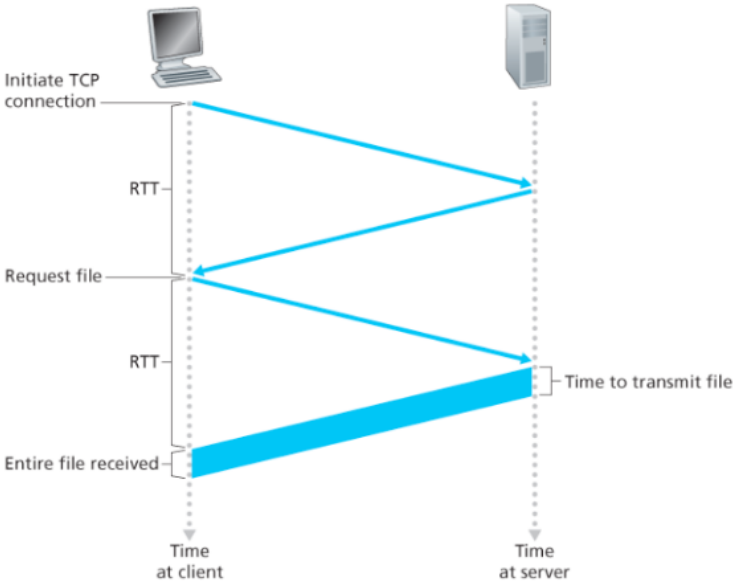
브라우저는 여러 개의 TCP 연결을 설정하며 다중 연결 상에서 웹 페이지의 각기 다른 원하는 부분을 요청할 수 있음

클라이언트가 파일을 요청하고 그 파일이 클라이언트로 수신될 때까지의 시간

⇒ 2RTT + time to transmission time

RTT(round trip time) : connection request를 보내고 response 오기까지의 시간

time to transmission time : response로 받는 파일의 크기가 크기 때문에



~> 각 요청 객체에 대한 새로운 연결이 설정되고 유지되어야 함 = 수많은 클라이언트들의 요청을 동시에 서비스하는 웹 서버에게는 심각한 부담

~> 매번 2RTT를 필요로 함

persistent connection (지속 연결)

→ TCP 연결을 그대로 유지해 같은 클라이언트와 서버 간의 이후 요청과 응답은 같은 연결을 통해 보내짐

여러 개의 object를 connection 한 번 열고 이 connection에 계속해서 보냄

진행 중인 요구에 대한 응답을 기다리지 않고 연속해서 만듦 → 파이프라이닝 (pipelining)

object가 많아지면 persistent가 non-persistent의 절반만큼 시간 걸릴 듯

HTTP 메세지 포맷

메세지 형태가 ASCII 형태로 되어있음

HTTP 요청 메세지의 첫 줄은 요청 라인이라고 부르고 이후의 줄은 헤더 라인이라고 부름

- 요청 라인

  방식 필드, URL 필드, HTTP 버전 필드를 가짐

  방식 필드 → GET, POST, HEAD, PUT, DELETE 등의 값을 가짐

HTTP 응답 메세지

상태 라인은 버전 필드, 상태 코드, 해당 상태 메세지를 가짐

- 상태 코드와 메시지
  - 200 : OK → 요청이 성공했고 정보가 응답으로 보내짐
  - 301 : Moved Permanently → 요청 객체가 영원히 이동됨
  - 400 : Bad Request → 서버가 요청을 이해할 수 없음
  - 404 : Not Found → 요청한 문서가 서버에 존재하지 않음
  - 505 : HTTP Version Not Supported → 요청 HTTP 프로토콜 버전을 서버가 지원하지 않음

Cookie - 사용자와 서버 간의 상호 작용

HTTP의 stateless 단점을 극복하기 위해 나온 것

→ request전 디스크 안의 cookie file에 있는지 찾아봄

⇒ 없으면 일반적인 HTTP request (response갈 때 cookie 번호를 알려줌)

⇒ 있으면 그 cookie 번호로 감

1. 웹 서버에 HTTP 요청 메세지를 전달
2. 웹 서버는 유일한 식별 번호를 만들고 이 식별 번호로 인덱싱 되는 백엔드 데이터베이스 안에 엔트리를 만듦
3. HTTP 응답 메시지에 set-cookie : 식별 번호의 헤더를 포함해 전달
4. 헤더를 보고 관리하는 특정한 쿠키 파일에 그 라인을 덧붙임
5. 다시 동일한 웹 서버에 요청을 보낼 때는 쿠키 파일을 참조해 식별번호를 발췌해 cookie: 식별번호의 헤더를 요청과 함께 보냄

Web cache (proxy server)

client에서 나가는 request는 반드시 proxy server를 거침

web cache는 자체의 저장 디스크를 가지고 있어서 최근 호출된 객체의 사본을 저장, 보존함

→ 이미 proxy server에 request에 대한 응답이 들어있으면 proxy server가 건네줌)

⇒ 빠름 (클라이언트와 캐시 사이에 높은 속도의 연결이 설정되어 있어 웹서버 캐시에 객채를 가지고 있으면 병목 현상을 줄일 수 있음)

⇒ 인터넷 access link에 대한 traffic도 줄일 수 있음

❗원본에 대한 복사본을 가지고 있기 때문에 일관성 문제에 대한 고려를 해야 함

Conditional GET

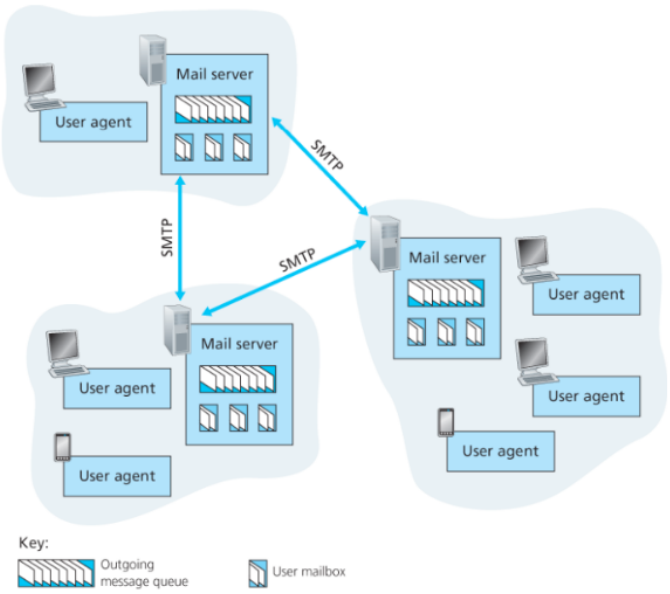
일관성 문제의 해결방안

: 브라우저로 전달되는 모든 객체가 최신의 것임을 확인하면서 캐싱

→ GET 방식 사용 + 기존에 있던 HTTP request message에 if-modified-since를 사용

파일이 이 시간 이후로 수정되었는지를 물어보는 것 → 수정되었으면 담아 보내고 아니면 안 보냄 (not modified 메시지만 보냄)

Electronic mail



세 개의 구성요소

- user agent
- mail server
- SMTP (simple mai transfer protocol)

  → user agent와 mail server간에 데이터를 어떻게 주고받을지 정의

user agent

사용자 에이전트는 사용자가 메시지를 읽고, 응답하고, 전달하고, 저장하고, 구성하게 해줌

mail server

각 수신자는 메일 서버에 메일 박스를 갖고 있음

- 특정 사용자한테 오는 메세지를 mailbox에 저장
- 보낸 메세지가 쌓이는 message queue가 존재
- 메일 서버들 사이에 주고받는 프로토콜인 SMTP

SMTP

→ TCP를 사용

클라이언트와 서버를 갖고있음

SMTP의 클라이언트와 서버 모두가 모든 메일 서버에서 수행되고, 상대 메일로 송신할 때는 클라이언트가 되고 수신할 때는 서버가 됨