소켓과 HTTP

남선우

목차

웹소켓과 소켓 통신의 차이?

소켓통신

- 두 프로그램이 특정 포트를 통해, 양방향 통신할 수 있도록 하는 방식
- 네트워크 통신을 할 수 있도록 운영체제에서 제공하는 API 개념.
- 계층 : Transport layer와 application layer의 인터페이스 역할
- 특징 : 양방향 통신, 실시간성, 상태유지
- 주요 프로토콜(데이터를 운반하는 방식)
    - TCP, UDP

웹소켓 통신

- HTTP 프로토콜 위에 실시간 양방향 통신 기능을 추가한 프로토콜
- 특징
  - http에서 실시간 통신을 할 수 없다는 문제를 해결하기 위해 나온 기술
  - http의 헤더를 업그레이드해서 동작함.
- 계층 : Application layer
- 특징 : 양방향 통신, 실시간성, 상태유지

	HTTP	WebSocket
TCP 연결	**사용함**	**사용함**
TCP 활용 방식	요청-응답 **한 사이클마다 연결과 종료**를 반복 (일회용)	**최초 한 번만 연결**하고 계속 유지 (지속용)
통신 주도권	오직 **클라이언트**만 시작 가능	**클라이언트와 서버** 양쪽 모두 시작 가능
결과	실시간 통신에 부적합 (서버 푸시 불가)	실시간 양방향 통신에 최적화

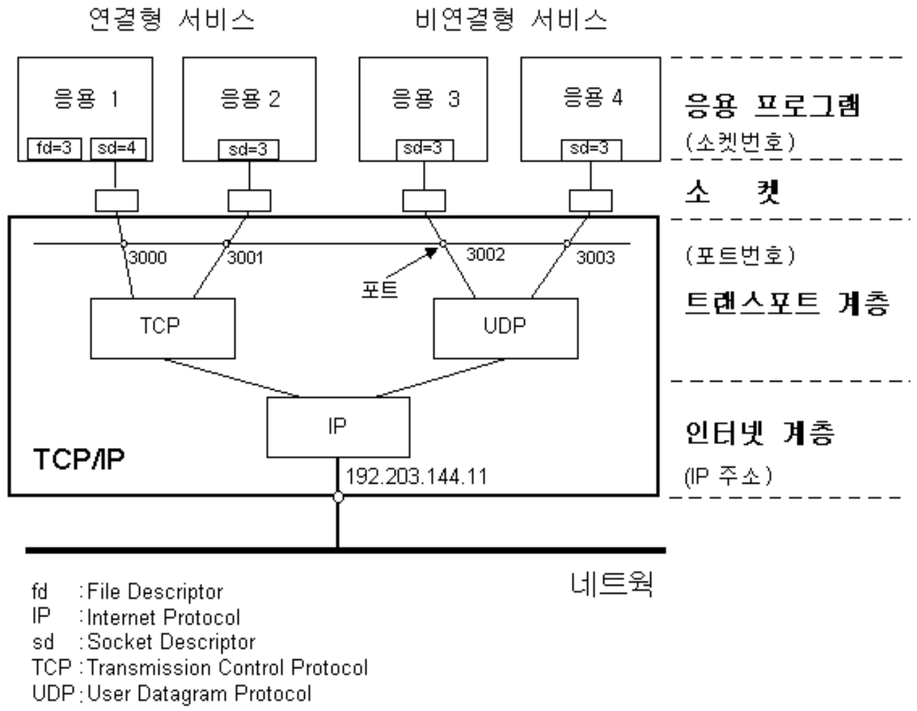
웹소켓과 소켓 통신의 차이?

- 둘이 상충되는 개념은 아니다. 둘의 역할과 목적이 다르다
- 웹 소켓
  - application layer
  - 웹 브라우저와 웹 서버 간
  - HTTP로 핸드셰이크 후, TCP 위에서 동작
- 소켓
  - Transport layer와 application layer의 인터페이스 역할
  - 모든 프로세스 간의 통신을 위한 OS의 API
  - TCP 또는 UDP 프로토콜의 수립을 보조하는 역할

소켓과 포트의 차이?

- 포트
  - 컴퓨터(서버)내에서 실행되고 있는 어떤 프로세스로 가야하는가를 식별하기 위한 숫자

- 소켓
  - 실제로 네트워크를 통해 데이터가 오고 가는 창구이자, 통신의 주체
  - 서버 클라이언트 구조에서는 둘의 소켓끼리 연결되어 데이터를 주고 받게 된다.
  - 구성요소 : IP주소 + 포트번호 + 프로토콜 정보



소켓을 써볼일이 없던 이유 - 추상화

Python

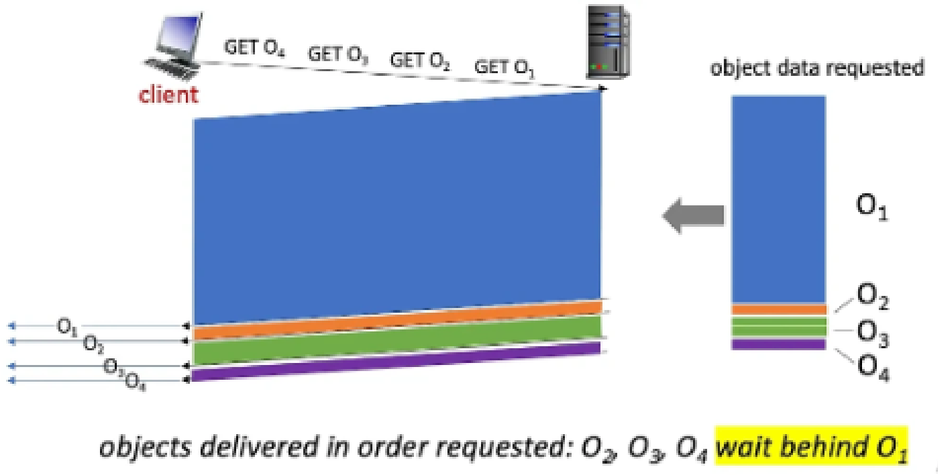
```
import requests

response = requests.get('https://api.google.com/search?q=gemini')
print(response.text)
```

```
[requests.get()]
      |
      V
[requests/adapters.py] 에서 urllib3 의 PoolManager.urlopen() 호출
      |
      V
[urllib3/connection.py] 에서 http.client.HTTPConnection 클래스 사용
      |
      V
[cpython/Lib/http/client.py] 에서 socket.create_connection() 호출
```

HOL Blocking

HOL Blocking

- 선두의 지연이 전체를 막는 현상
- 패킷 전송 관점
  - 데이터가 여러 개의 패킷 조각으로 순차적으로 전송될 때, 앞 순서의 패킷이 어떤 이유로 지연되거나 손실되면, 그 뒤에 도착한 패킷들이 먼저 처리되지 못하고 무작정 대기하는 문제
- 큐잉 시스템 관점
  - 스위치/라우터의 출력 큐에서 패킷이 못 빠져나가 뒤에 있는 패킷들도 대기되는 문제



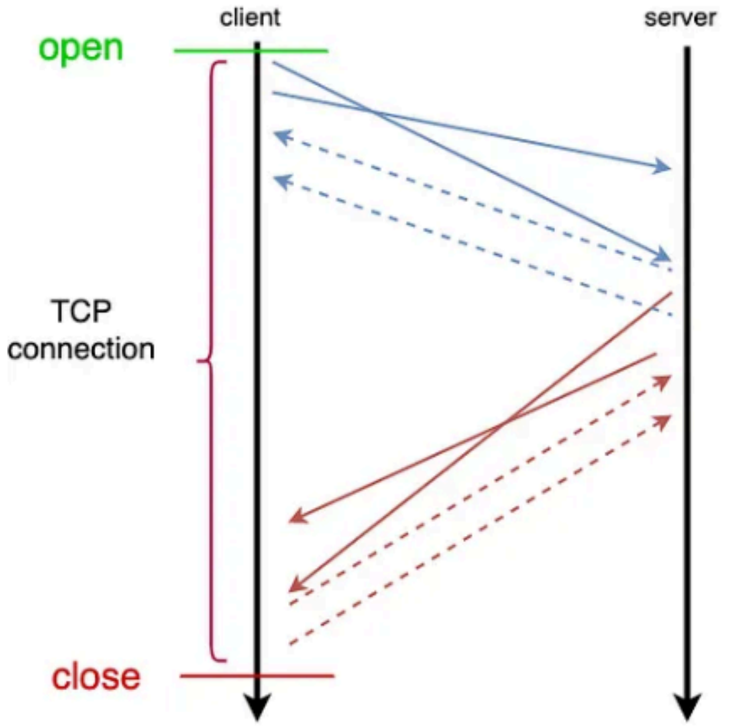
HTTP/1.1, HTTP/2, HTTP/3.0

HTTP/1.1

- HTTP 1.0 이하는 비지속연결 방식으로 구현
  - 왜 이렇게 구현?
    - 당시 환경에서는 보통 데이터가 작고, 한 HTML 문서가 있으면 끝이기 때문에 연결 유지할 필요가 없었다.
  - TCP를 사용하긴 하지만 요청에 대한 값 받으면 끊는 방식으로 구현
- HTTP 1.1
  - 지속연결 방식으로 구현
    - 지속 연결을 통해 비지속 연결보다 빠른 속도로 여러 HTTP 요청과 응답 처리 가능
  - keep-alive
  - 파이프라이닝
  - 한계
    - HOL Blocking

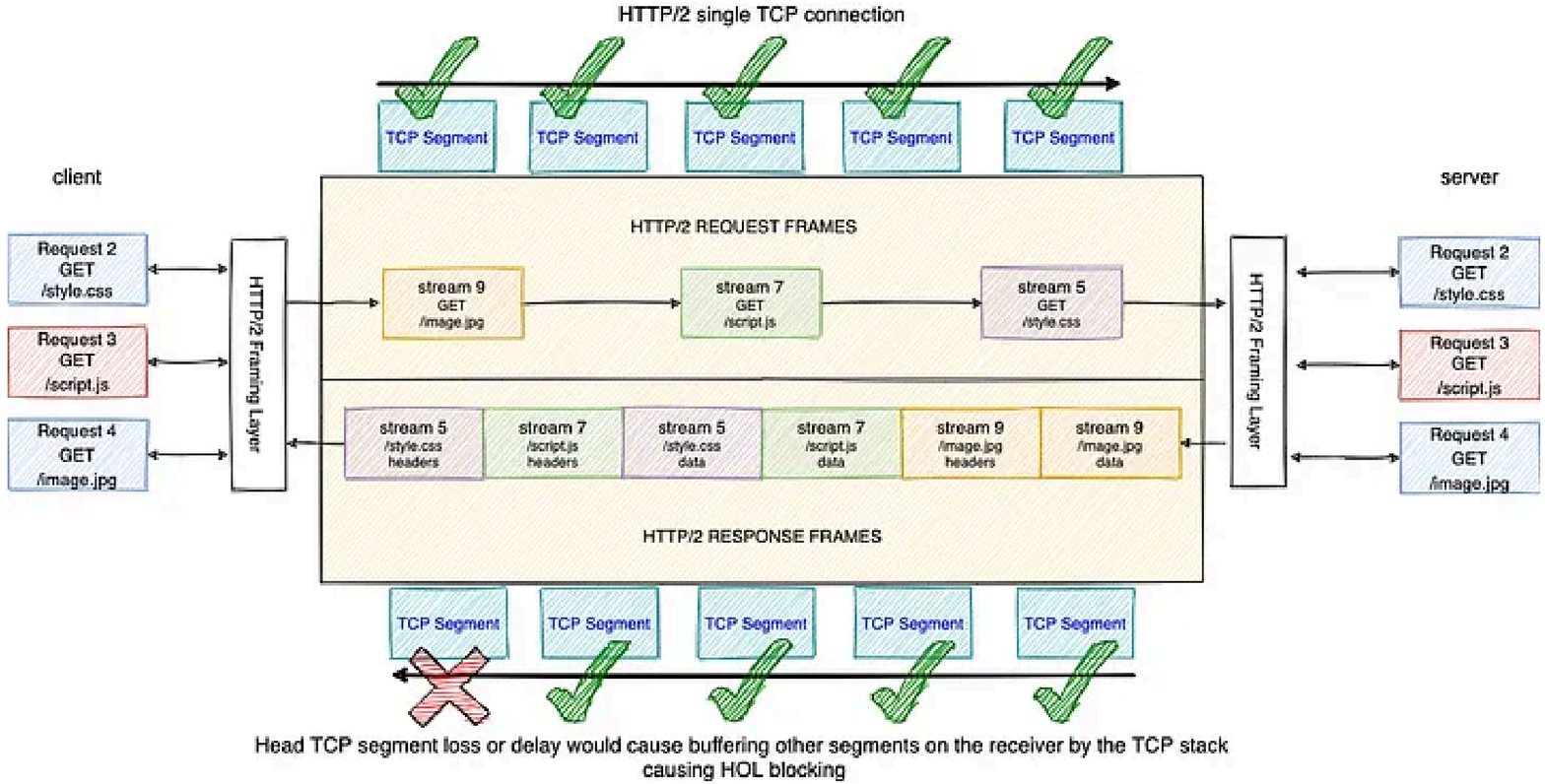
HTTP/2

- 이전까진 데이터가 텍스트였는데, 바이너리 형태로 바꿔서 전송
  - 파싱 효율 증가
- 멀티플렉싱 도입
  - 같은 연결에서 요청들을 '병렬로 동시에 처리하는 것'
  - stream을 활용해 다수개의 요청을 병렬적으로 처리가 가능해짐
- 서버푸시
  - 미래에 필요할 것 같은 자원 함께 응답



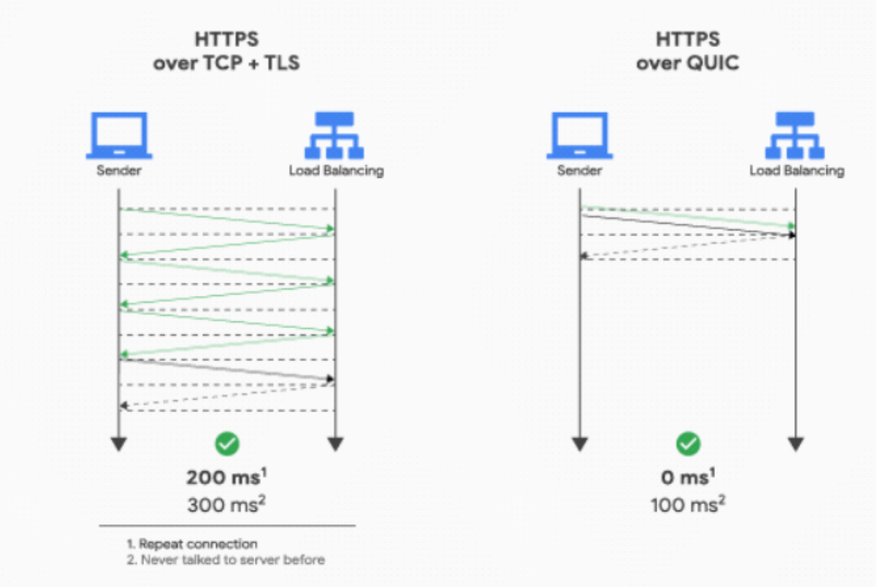
한계

- HTTP/2 자체는 TCP 위에서 동작하기 때문에 애플리케이션 계층에서는 멀티플렉싱이 제공되나, 전송계층에서는 순서보장을 위해 앞선 패킷이 손실되면 뒤 패킷도 모두 대기.



HTTP/3

- 원초적으로 TCP가 문제다
- UDP + QUIC
- QUIC
  - TCP의 기능들(Congestion, flow control, TLS...)을 UDP 위에서 소프트웨어 적으로 구현
- HOL Blocking과 Handshake 같은 오버헤드 해결



지금은 뭘 사용할까?

- HTTP/2가 사실상 표준
  - 브라우저들 기본 연결을 HTTP/2로 시도
- HTTP/3 점진적 확산 중
  - 모바일/대규모 트래픽 서비스에서 확대 중
  - 영상회의에서 표준처럼 사용
  - 아직 도입 원활히 안된 이유
    - 네트워크 장비들이 기존 TCP 중심
    - 서버/클라이언트 모두 QUIC 스택 구현 필요
      - (웹 서버 운영자가 구현 해야함.. - Nginx, Apache...)