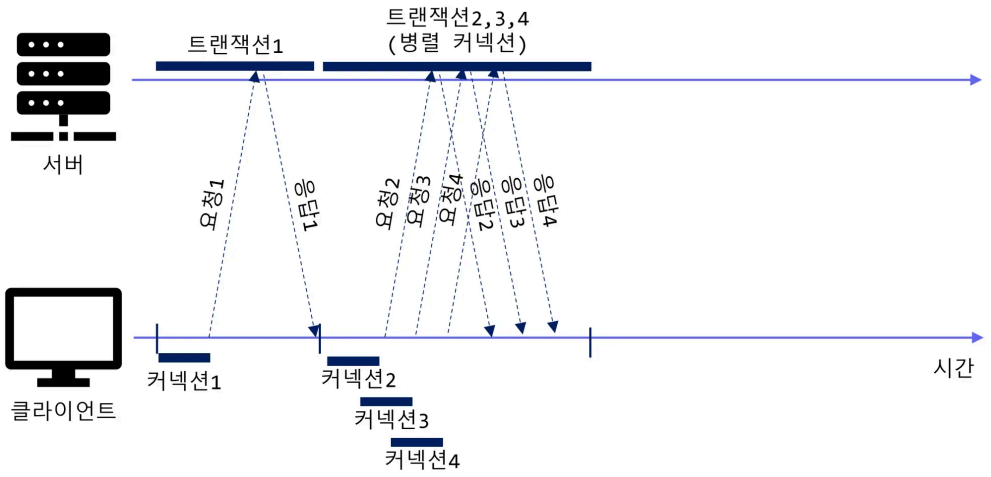
HTTP/2.0

HTTP/2.0은 왜 생겼을까

HTTP는 응답을 받아야 요청을 보낼 수 있기때문에 심각한 지연을 피할 수 없었다

이 성능 문제를 해결하고자 병렬 커넥션, 파이프라인 커넥션이 도입

병렬 커넥션



병렬 커넥션은 클라이언트가 여러 개의 커넥션을 맺음으로써 여러 개의 HTTP 트랜잭션을 병렬로 처리할 수 있게 하는 방법. 위의 예에서는 3개의 이미지를 할당받은 각 TCP 커넥션상의 트랜잭션을 통해 병렬로 내려받습니다.

병렬 커넥션의 한계

- **동시 커넥션 제한**

  브라우저는 같은 도메인에 대해 병렬로 만들 수 있는 커넥션 수에 제한 (예: 6개 정도). 이 제한 때문에 너무 많은 리소스를 동시에 요청불가

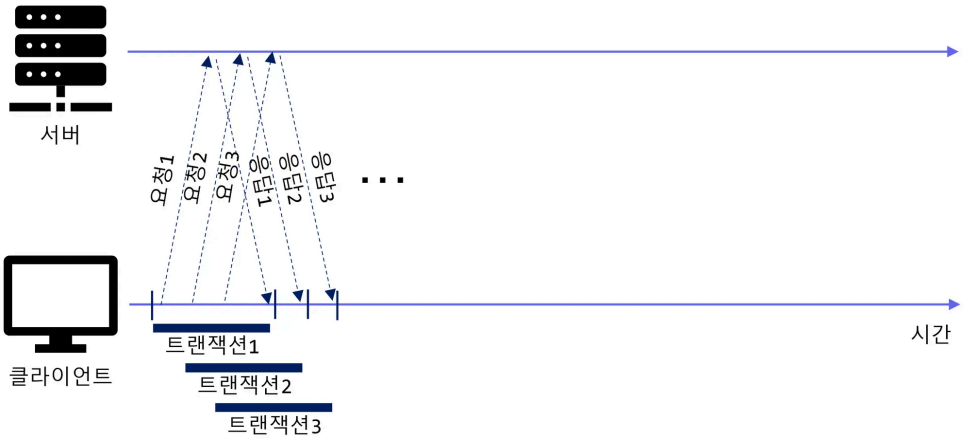
- **TCP 커넥션 오버헤드**

  병렬 커넥션을 많이 만들면 각각 TCP 연결을 새로 맺어야 하고, 그 과정(TCP 3-way handshake + SSL handshake 등)이 **시간과 리소스를 소모**함.

- **혼잡 제어**

  TCP는 네트워크 혼잡을 방지하기 위해 한 연결당 윈도우 크기를 조절해. 커넥션을 많이 만들면 각각이 독립적으로 혼잡 제어를 받으면서 전체 효율이 떨어질 수 있어.

파이프라인 커넥션



파이프라인 커넥션은 먼저 보낸 요청이 완료되기 전에 송신측에서 요청을 계속해서 보냄으로써 다음 응답까지의 대기시간을 없애 네트워크 기능을 향상시킵니다. Keep-Alive를 전제로 하며 서버는 요청이 들어온 순서대로 응답을 반환

파이프라인 커넥션의 한계

- **헤드-오브-라인 블로킹(Head-of-Line Blocking)**

  파이프라이닝은 요청을 여러 개 보내되 **응답 순서는 요청 순서대로** 받아야 해.

  이 때문에 **앞 요청이 느리면 뒤 요청도 기다려야 하는 병목**이 생겨.

  예를 들어 첫 응답이 지연되면 나머지도 줄줄이 대기함.

- **중간 장비 호환 문제**

  오래된 프록시나 서버가 파이프라이닝을 잘 지원하지 않아서 **비호환 이슈**가 많았음.

  그래서 브라우저들도 파이프라이닝을 실질적으로 비활성화했어.

HTTP 2.0 개요

성능 개선 대한 근본적인 해결책 안됨.

2009년 구글은 웹을 더 빠르게 하겠다는 목표로 SPDY(스피디) 프로토콜을 내놓음

이 초안을 가져와 HTTP/2.0 초안을 만듬

HTTP 2.0

HTTP/2.0 서버와 클라이언트 사이 TCP 커넥션 위에서 동작

Multiplexed Streams

하나의 커넥션위에 여러개의 스트림으로 즉 여러개의 요청과 응답 가능

즉 여러개의 TCP 연결이 아니라 단일 TCP에서 여러개 데이터가 섞이지않게 보내는 기법

체인에서 링크는 다른 색으로 보여준다

리소스간 우선순위 설정가능 , 파이프라인의 순서지킴으로 인한 지연이 사라짐

Server push

서버는 클라이언트에게 능동적으로 리소스를 보내줌

예를 들어 클라이언트가 index.html을 요청하면

```
GET /index.html
```

서버는 그 요청을 받고,

이 html이 로드되면 style.css랑 main.js도 필요하겠네?

그래서 클라이언트가 요청하기 전에 미리 그 리소스를 push로 보내줌

```
HTTP/2 200 OK
(PUSH_PROMISE for /style.css)
(PUSH_PROMISE for /main.js)
```

HTTP 1.1 과 2의 커넥션 차이

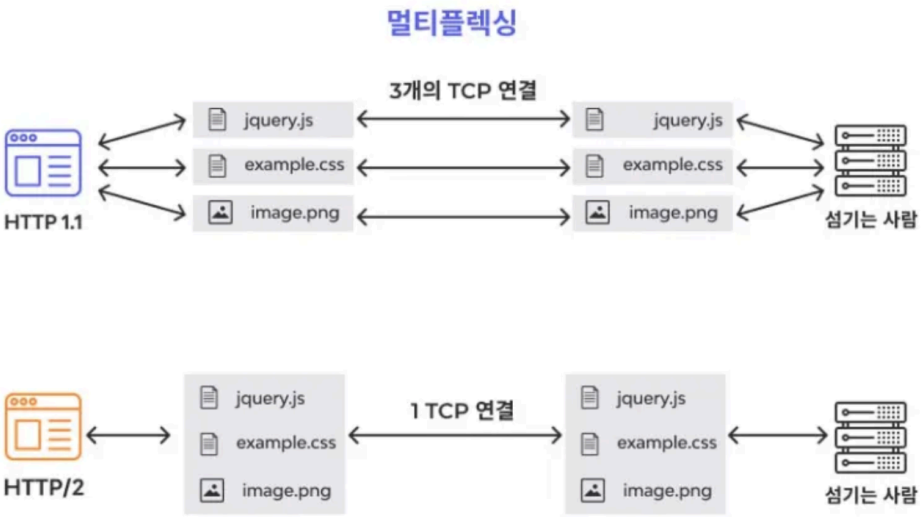
	HTTP/1.1	HTTP/2
병렬 처리 방식	커넥션 여러 개 열어서 병렬화	**1개 커넥션 안에서 스트림으로 병렬화**
커넥션 수	도메인당 보통 6개 제한	도메인당 **1개가 원칙**
병렬 처리 성능	커넥션 수에 따라 제약 있음	스트림 수는 수천 개까지도 가능
커넥션 수 많은가?	응. 진짜 병렬 커넥션	아냐. **논리적인 병렬**

HTTP 2.0 의 재밌는 예시

Alice가 친구 Bob이 쓴 소설을 읽고 싶어하지만, Alice와 Bob은 일반 메일을 통해서만 의사소통한다고 상상해 보세요. Alice는 Bob에게 편지를 보내 소설을 보내달라고 부탁합니다. Bob은 HTTP/1.1 스타일이라는 소설을 보내기로 결정합니다. Bob은 한 번에 한 장씩 우편으로 보내고, Alice로부터 이전 장을 받았다는 답장을 받은 후에만 다음 장을 우편으로 보냅니다. 이 콘텐츠 전달 방법을 사용하면 Alice가 Bob의 소설을 읽는 데는 몇 주가 걸립니다.

이제 Bob이 Alice에게 자신의 소설 HTTP/2 스타일을 보내기로 결정했다고 상상해 보세요. 이 경우 Bob은 소설의 각 장을 개별적으로 보내지만(우편 서비스의 크기 제한 내에서 유지하기 위해), 동시에 보냅니다. Bob은 또한 각 장에 1장, 2장 등의 번호를 매깁니다. 이제 Alice는 소설을 한꺼번에 받고 적당한 시간에 올바른 순서로 조립할 수 있습니다. 챕터가 누락된 경우 특정 챕터를 요청하는 빠른 답장을 보낼 수 있지만, 그렇지 않으면 프로세스가 완료되고, Alice는 단 며칠 만에 소설을 읽을 수 있습니다.

HTTP/2에서는 Bob이 Alice에게 한 번에 여러 장을 보내는 것처럼 데이터가 한 번에 전송됩니다. Bob과 마찬가지로 개발자는 HTTP/2에서 각 장에 번호를 매길 수 있습니다. 개발자는 웹 페이지의 텍스트, CSS 파일, JavaScript, 사용자 경험에 가장 중요하다고 생각되는 것 중 어느 것을 먼저 보낼지 결정할 수 있습니다.



HTTP2의 한계

- 패킷 손실에 취약

1. RTT(TCP로 통신하면 발생할 수밖에 없음)
2. TCP 자체의 HOLB
3. 패킷이 유실되거나 오류가 있을 때 재전송을 하게 됨, 재전송 과정에서 패킷 지연이 발생하면 HOLB문제가 발생
4. 중개자 캡슐화 공격
    - 2.0은 헤더 필드의 이름과 값을 바이너리로 인코딩하는데 어떤 문자열이든 사용가능하다.
    - 2.0 메세지를 proxy 서버가 http 1.1메세지를 변환할 때 메세지를 불법 위조할 수 있음
5. 길게 유지된 커넥션으로 인한 개인정보 유출 악용 위험성