HTTP 완벽가이드

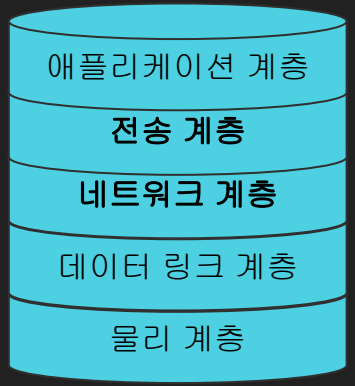
by David Gourley, Brian Totty

1부 HTTP: 웹의 기초

4장 커넥션 관리

HTTP는 어떻게 TCP커넥션을 사용할까?

- 전 세계 모든 HTTP 통신은 TCP/IP를 통해 이루어진다.
- TCP/IP(Transmission Control Protocol/Internet Protocol)란? 컴퓨터 사이의 통신 및 네트워크 상호연결에 대한 규칙을 지정하는 프로토콜로, 패킷 교환 및 스트림 전송으로 작동한다.



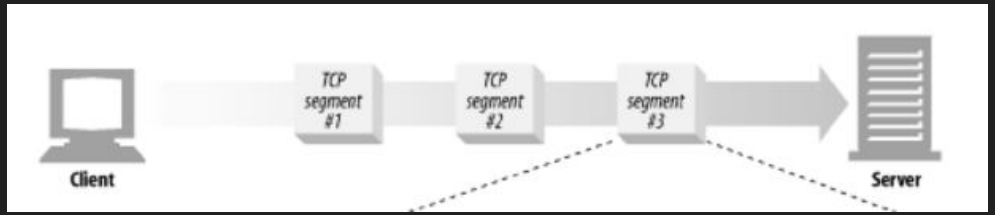
- 애플리케이션 계층 : HTTP, FTP, DNS, SMTP …
- **전송 계층 : TCP, UDP, DCCP …**
- **네트워크 계층 : IP …**
- 데이터 링크 계층 :
- 물리 계층 : 이더넷, wi-fi …

- HTTP는 네트워크 통신의 세부사항을 TCP/IP에게 맡긴다.
  (∵ TCP/IP는 신뢰성이 있기 때문이다)

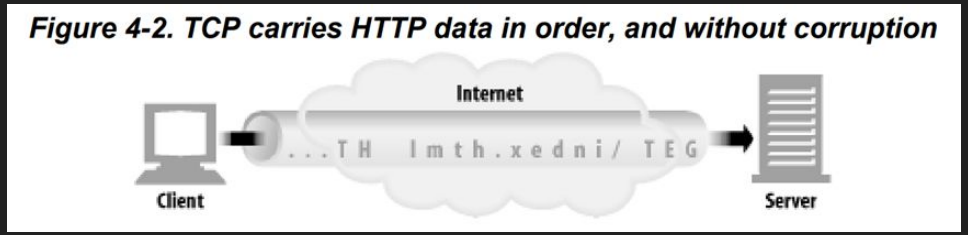
「TCP/IP는 신뢰성이 있다」는 것의 의미

클라이언트와 서버 사이에 TCP/IP 커넥션이 맺어지면
주고받은 메시지들은 손실 혹은 손상되거나 순서가 바뀌지 않는다.

- **메시지가 손실/손상되지 않는다** : TCP는 데이터 스트림을 segment단위로 나누고, 그 segment를 IP패킷에 담아서 (손실/손상없이) 전달한다.



- **메시지의 순서가 바뀌지 않는다** : TCP커넥션 한쪽에 있는 바이트들이 반대쪽으로 순서에 맞게 정확히 전달된다.



TCP 커넥션 유지하기

컴퓨터는 항상 TCP커넥션을 여러 개 가지고 있다.

이 네가지 값으로 유일한 커넥션을 생성한다.

<발신지 IP 주소, 발신지 포트, 수신지 IP 주소, 수신지 포트>

커넥션 생성 : TCP 소켓 프로그래밍

- Windows, Linux 등의 운영체제는 TCP 커넥션의 생성 관련 기능을 제공한다.

- TCP 커넥션을 위한 소켓 인터페이스(=소켓 API)들이 [표 4-2]에 소개되어 있다 (소켓 생성, 소켓에 로컬 포트번호를 할당 등).
  - 소켓 API(Application Program Interface)
  - 소켓 : 프로그램이 네트워크에서 데이터를 주고받을 수 있도록 통신 양끝단에 만들어진 연결부

- 소켓 API를 사용하면 TCP 종단 데이터 구조를 생성하고, 원격 서버의 TCP 종단에 그 구조를 연결하여 데이터 스트림을 읽고 쓸 수 있다.

TCP 커넥션의 성능 문제 🤔

HTTP 트랜잭션 성능이 좋아지려면? → TCP 성능이 좋아야 한다.

TCP 성능이 좋아지려면? → TCP 커넥션 성능이 좋아져야 한다.

- 애플리케이션 계층 : **HTTP**, FTP, DNS, SMTP …
- 전송 계층 : **TCP**, UDP, DCCP …
- 네트워크 계층 : IP …
- 데이터 링크 계층 :
- 물리 계층 : 이더넷, wi-fi …

어떤 성능문제가 있을까??

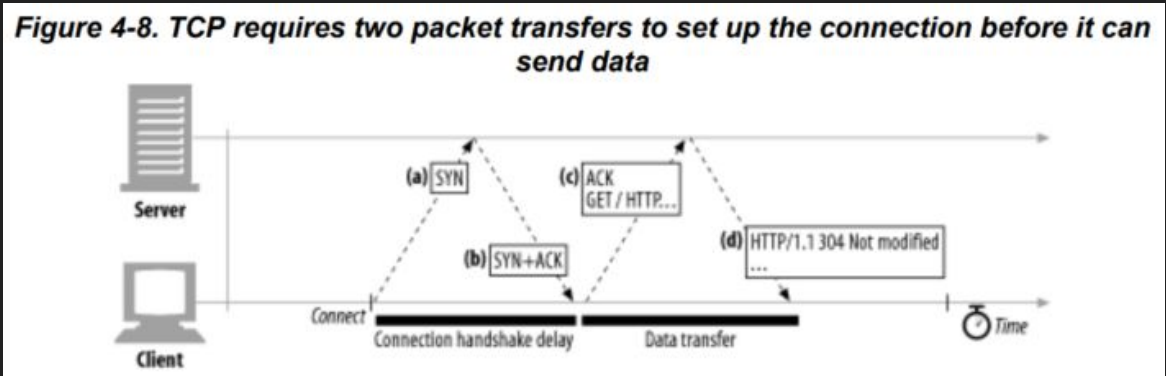
1. 처음 방문하거나 최근에 방문한 적 없는 URI에 접속하는 데 시간이 걸린다.

→ 현재는 인터넷 인프라의 발전으로 대부분 밀리초 단위로 DNS 이름분석이 끝나므로 중요한 문제는 아니게 됐다.

2. **커넥션이 맺어지는 데 시간이 걸린다.**
   - a. TCP 커넥션 핸드셰이크 지연
   - b. 확인응답 지연
3. **커넥션이 맺어진 후 메시지가 처리되는 데 시간이 걸린다.**
   - c. TCP 느린 시작(slow start)
   - d. 네이글 알고리즘
4. **커넥션이 끊어진 후 문제**
   - e. TIME_WAIT의 누적과 포트 고갈

a. TCP 커넥션의 핸드셰이크 지연

- 새로운 TCP 커넥션을 열 때 TCP 소프트웨어는 연속으로 IP패킷을 교환한다.
- IP패킷은 HTTP 요청메시지 전체를 전달할 수 있을 만큼 사이즈가 크다.
- 작은 크기의 데이터 전송에 커넥션이 사용된다면 패킷교환은 HTTP 성능을 크게 저하시킬 수 있다.
- 크기가 작은 HTTP트랜잭션은 50% 이상의 시간을 TCP를 구성하는데 쓰게 된다. => 이미 존재하는 TCP 커넥션을 재활용하는 방법으로 해결!



b. 확인응답 지연

- 패킷전송이 성공했다는 것을 보장하기 위해 TCP는 자체적인 확인 체계를 갖는다(각 TCP 세그먼트는 순번과 데이터 무결성 체크섬을 갖는다).
- 각 세그먼트의 수신자는 세그먼트를 온전히 받으면 작은 확인응답 패킷을 송신자에게 반환한다.
- TCP는 같은 방향으로 송출되는 데이터 패킷에 확인응답을 편승시킨다.
- 막상 편승할 패킷이 많지 않기 때문에 지연이 발생하게 된다.
- '확인응답 지연' 알고리즘
  - 송출할 확인응답을 특정 시간동안 버퍼에 저장해두고, 편승시킬 패킷을 찾는다.
  - 만약 일정 시간 안에 편승시킬 패킷을 찾지 못하면 확인응답은 별도의 패킷을 만들어 전송된다.

c. TCP 느린 시작(slow start)

- 인터넷의 급작스러운 부하와 혼잡을 방지하기 위해 TCP 느린 시작이 쓰인다.
- TCP 느린 시작은 TCP가 한번에 전송할 수 있는 패킷의 수를 제한한다.
- 이러한 혼잡제어 기능 때문에 새로운 커넥션은 이미 어느정도 데이터를 주고받은 '튜닝'된 커넥션보다 느리다.

=> 이미 존재하는 TCP 커넥션을 재활용하는 방법으로 해결!

d. 네이글 알고리즘

- TCP가 작은 크기의 데이터를 포함한 많은 수의 패킷을 전송한다면 네트워크의 성능은 크게 떨어진다.
- 이를 개선하기 위해네이글 알고리즘은 패킷을 전송하기 전에 많은 양의 TCP데이터를 한 개의 덩어리로 합친다.
- 네이글 알고리즘은 segment가 최대 크기가 되지 않으면 전송을 하지 않는다.

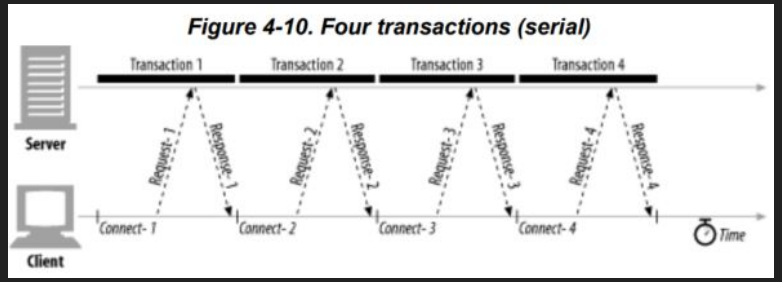
  => 문제점 : 기다리느라 시간이 걸리고, 확인응답 지연과 함께 쓰일 경우 형편없이 작동한다.

- HTTP 애플리케이션은 TCP_NODELAY 파라미터 값을 설정하여 네이글 알고리즘을 비활성화하기도 한다.

e. TIME_WAIT의 누적과 포트 고갈

- 실제 상황에서는 문제가 나타나지 않고, 성능 시험 시 나타나는 문제이다.
- TCP커넥션의 종단에서 TCP 커넥션을 끊으면 종단에서는 커넥션의 IP 주소와 포트번호를 control block(메모리의 제어영역)에 기록해 놓는다.
- 이 정보는 이전과 동일한 IP와 포트번호를 가지는 TCP커넥션이 일정 시간 동안(보통 2MSL=2분)에는 생성되지 않게 하기 위한 것이다.
- 동일한 IP와 포트번호를 가지는 TCP커넥션이 곧바로 생기지 못하게 함으로써, 패킷이 중복되고 TCP 데이터가 충돌하는 것을 방지해준다.
- 포트고갈 문제는 거의 일어나지 않는다.

HTTP 커넥션 성능을 향상시키려면? 🤔

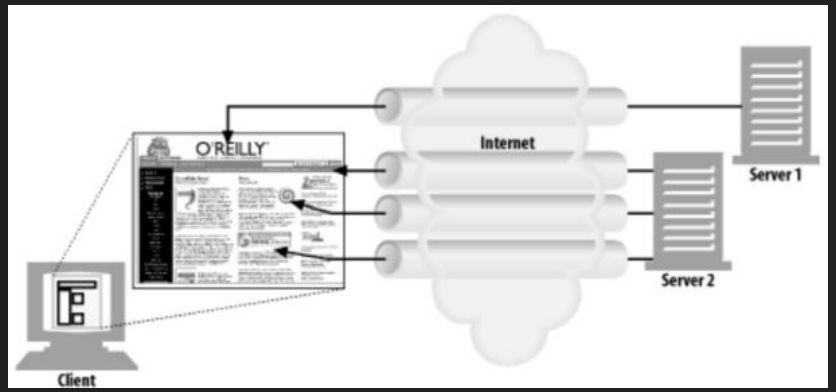


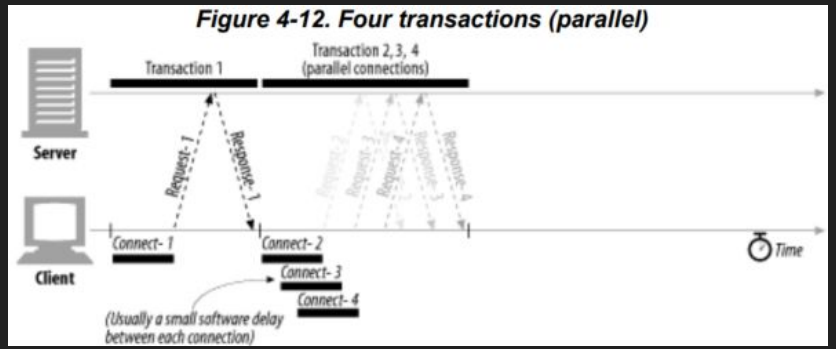
네 개의 트랜잭션이 순차적으로...
각 트랜잭션에 새로운 커넥션이 필요하면
시간이 엄청 많이 걸릴 것이다.

(성능을 향상시킬 수 있는 최신 기술)

1. 병렬 커넥션 : 여러 개의 TCP 커넥션을 통한 동시 HTTP 요청
2. 지속 커넥션 : 맺고 끊는데서 발생하는 지연을 제거, TCP 커넥션 재활용
3. 파이프라인 커넥션 : 공유 TCP 커넥션을 통한 병렬 HTTP 요청

병렬 커넥션 여러 개의 TCP 커넥션을 통한 동시 HTTP 요청





- HTML 페이지를 먼저 내려받고 남은 3개의 트랜잭션이 각각 별도의 커넥션에서 동시에 처리되는 모습
- 다수의 커넥션은 메모리를 많이 소모하기 때문에 병렬 커넥션이 항상 더 빠르지는 않지만, 더 빠르게 느껴지기는 한다.
- 실제로 브라우저는 병렬 커넥션을 사용하기는 하지만 적은 수(보통 6~8개)의 병렬 커넥션만을 허용한다.

지속 커넥션 맺고 끊는데서 발생하는 지연을 제거하기 위해 TCP 커넥션 재활용

- 사이트 지역성(site locality) : 서버에 HTTP 요청을 하기 시작한 애플리케이션은 같은 서버에 또 요청하는 경우가 많다.
- HTTP/1.1을 지원하는 기기는 처리가 완료된 후에도 TCP 커넥션을 유지하여 앞으로 있을 HTTP 요청에 재사용할 수 있다.
- 이처럼 처리가 완료된 후에도 계속 연결된 상태로 있는 TCP 커넥션을 지속 커넥션이라 한다.
- 지속 커넥션은 병렬 커넥션과 함께 사용될 때에 가장 효과적이다.

병렬 커넥션 단점

1. 새로운 커넥션을 맺고 끊는 데에 시간과 대역폭이 소요된다.
2. 새로운 커넥션은 TCP 느린시작때문에 성능이 떨어진다.
3. 실제로 연결할 수 있는 병렬 커넥션의 수에 제한이 있다.

지속 커넥션 장점

1. 커넥션을 맺기위한 사전 작업과 지연을 줄여준다.
2. 튜닝된 커넥션을 유지한다.
3. 커넥션의 수를 줄여준다.

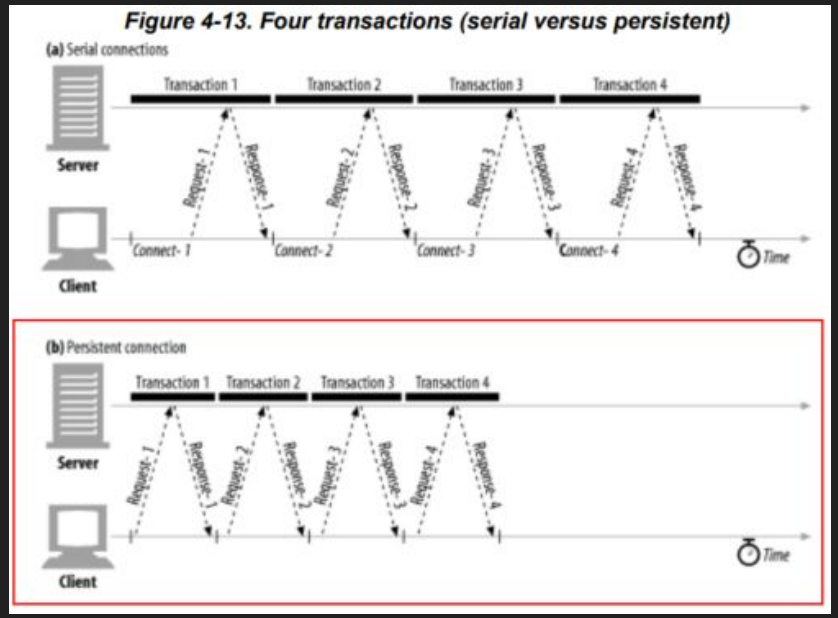
지속 커넥션 단점

1. 수많은 커넥션이 쌓일 수 있다…

첫번째 지속 커넥션 타입, keep-alive 커넥션

많은 HTTP/1.0 브라우저와 서버들은 1996년부터 keep-alive 커넥션을 지원하기 위해 확장되어 왔다.

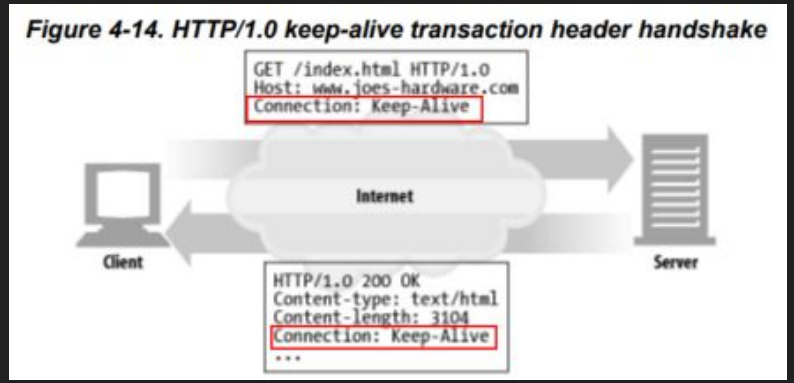
아직 많은 클라이언트와 서버가 사용하고 있고 설계상의 문제는 HTTP/1.1에서 수정되었다.



keep-alive는 사용하지 않기로 결정되어 HTTP/1.1 명세에서 빠졌지만, 아직도 브라우저와 서버 간에 keep-alive 핸드셰이크가 널리 사용되고 있다.

← 커넥션을 맺고 끊는 데 필요한 작업이 없어서 시간이 단축되는 것을 확인할 수 있다.

keep-alive 동작과 옵션



- HTTP/1.0 keep-alive 커넥션을 구현한 클라이언트는 커넥션을 유지하기 위해서 요청에 Connection:Keep-Alive 헤더를 포함시킨다.
- 이 요청을 받은 서버는 그 다음 요청도 이 커넥션을 통해 받고자 한다면, 응답 메시지에 같은 헤더를 포함시켜 응답한다.

- 위 헤더는 커넥션을 유지하기를 바라는 요청일 뿐이고, 당사자들은 언제든지 현재의 keep-alive 커넥션을 끊을 수 있으며, keep-alive 커넥션에서 처리되는 트랜잭션 수도 제한할 수 있다.
- 예시)

```
Connection: Keep-Alive
Keep-Alive: max=5, timeout=120
```

서버가 약 5개의 추가 트랜잭션이 처리될 동안 커넥션을 유지하거나, 2분동안 커넥션을 유지해라.

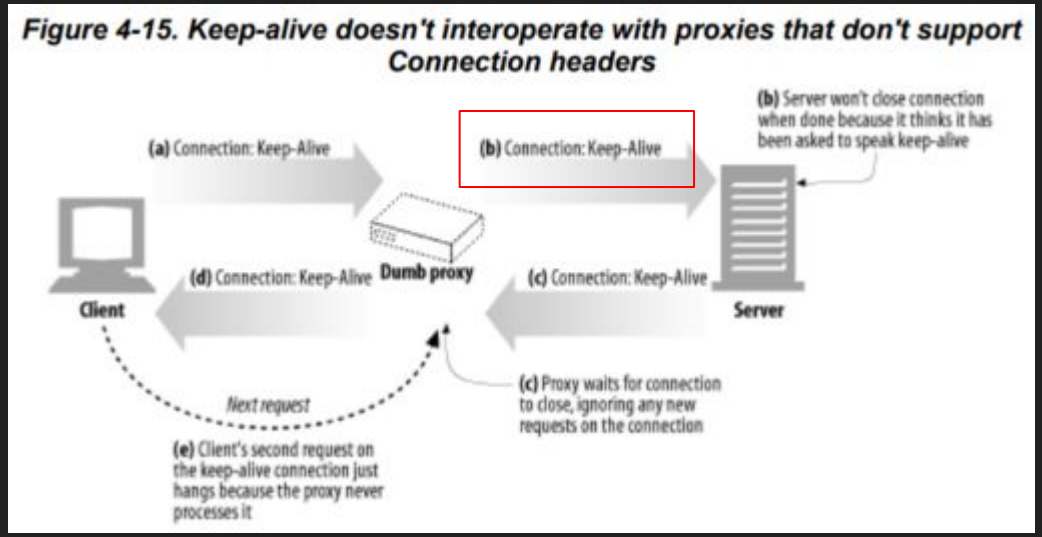
keep-alive 커넥션 상세 몇가지만!

- 프락시와 게이트웨이는 Connection 헤더의 규칙을 철저히 지켜야 한다. 프락시와 게이트웨이는 메시지를 전달하거나 캐시에 넣기 전에 Connection 헤더에 명시된 모든 헤더필드와 Connection 헤더를 제거해야 한다.
- 원칙적으로, keep-alive 커넥션은 Connection 헤더를 인식하지 못하는 프락시 서버와는 맺어지면 안된다. <멍청한 프락시 관련>
- 기술적으로 HTTP/1.0을 따르는 기기로부터 받는 모든 Connection 헤더필드 (Connection: Keep-Alive 포함)는 무시해야 한다. 오래된 프락시 서버로부터 실수로 전달될 수 있기 때문이다.

멍청한 프락시 문제

Connection 헤더의 무조건 전달

프락시는 Connection 헤더를 이해하지 못해서 해당 헤더들을 삭제하지 않고 요청 그대로를 다음 서버에 전달한다.

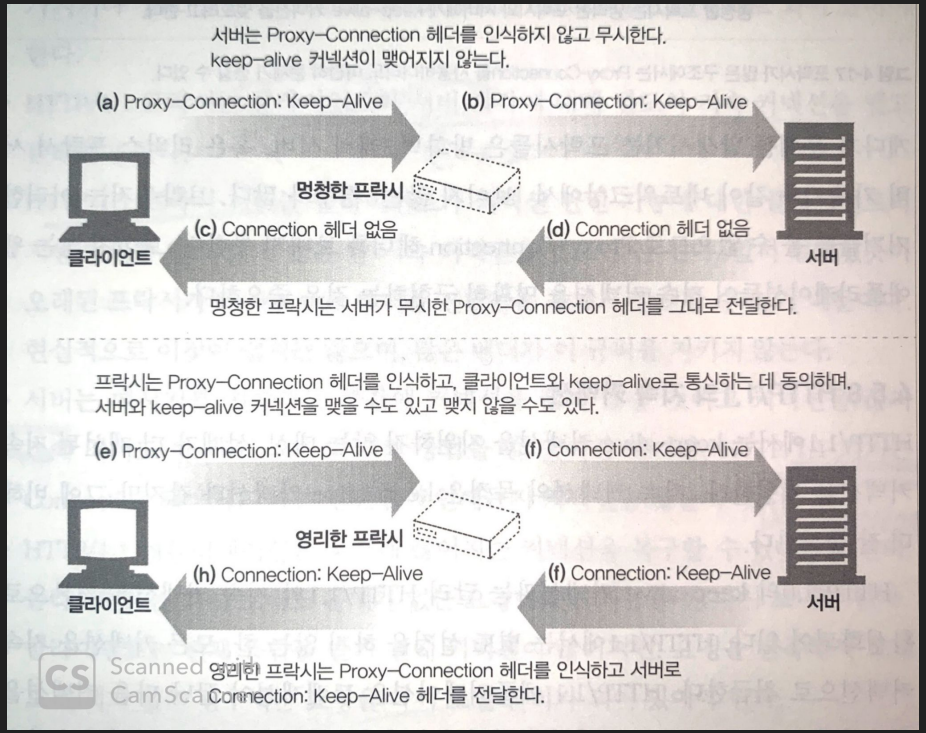


(문제점)
Client : 프락시가 커넥션을 유지하자는 것으로 알아 듣는다.
Proxy : 지속 커넥션(keep-alive)이 뭔지 모르는 상태이다. 그저 서버가 커넥션을 끊기를 기다린다.
Server : 프락시랑 커넥션을 유지 중이므로 커넥션을 끊지 않는다.

이런 상황에서 Client가 다음 요청을 보내게 되면, 프락시는 같은 커넥션으로 다른 요청이 오는 경우를 예상치 못하기 때문에 요청을 무시한다…

"클라이언트 입장 : keep-alive 한다며?? 왜 응답이 없니 멍청한 프락시야…"

멍청한 프락시 문제를 해결하려면 Proxy-Connection



비표준인 Proxy-connection 확장 헤더를 프락시에게 전달한다.

프락시가 Proxy-connection 헤더를 무조건 전달하더라도 웹 서버는 그것을 무시하기 때문에 문제가 없다.

하지만 멍청한 프락시 양옆에 영리한 프락시가 있다면 다시 문제가 발생하게 된다...

두번째 지속 커넥션 타입, HTTP/1.1의 지속 커넥션

HTTP/1.1에서는 keep-alive 커넥션을 지원하지 않는 대신, 더욱 개선된 지속 커넥션을 지원한다.

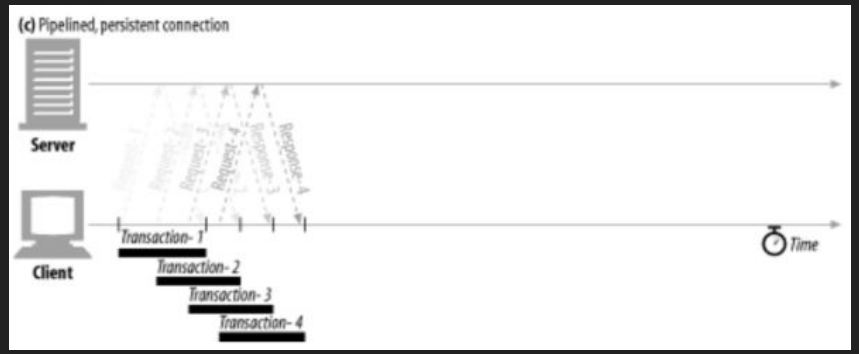
HTTP/1.0의 keep-alive 커넥션과 달리, HTTP/1.1의 지속 커넥션은 기본으로 활성화되어 있다. (모든 커넥션을 지속 커넥션으로 취급한다.)

다음 커넥션을 끊으려면 Connection: close 헤더를 명시해야 한다.

HTTP/1.1 지속 커넥션 상세 몇가지만!

커넥션에 있는 모든 메시지가 자신의 길이 정보를 정확히 가지고 있을 때에만 커넥션을 지속시킬 수 있다. 서버의 과부하 방지를 위해, 하나의 사용자 클라이언트는 넉넉잡아 2개의 지속 커넥션만을 유지해야 한다.

파이프라인 커넥션 공유 TCP 커넥션을 통한 병렬 HTTP 요청



- HTTP/1.1은 지속 커넥션을 통해 요청을 파이프라이닝할 수 있다.
- 여러 개의 요청은 응답이 도착하기 전까지 큐에 쌓인다.
- HTTP 클라이언트는 POST 요청(비멱등 요청)과 같이 반복해서 보낼 경우 문제가 생기는 요청은 파이프라인을 통해 보내면 안된다.

  ※ POST는 보통 회원가입 등에 사용하는 HTTP 요청 메서드이기 때문에 요청이 처리될 때마다 서버의 데이터에 변화가 생긴다. 이처럼 연산이 한번 일어날 때마다 결과가 변하는 요청을 비멱등 요청이라고 한다.

커넥션 끊기의 여러가지 모습

1. **마음대로 커넥션 끊기**

   커넥션 당사자는 언제든지 커넥션을 끊을 수 있다.

2. **Content-Length와 Truncation(끊김)**

   각 HTTP 응답은 본문의 정확한 크기 값을 가지는 Content-Length 헤더를 가지고 있어야 하는데 중간에 끊기면 헤더가 정확하지 않으므로 발신자에게 다시 물어봐야 한다.

3. **비멱등인 요청**(예: POST)을 다시 보내야 한다면, 이전 요청에 대한 응답을 받을 때까지 기다려야 한다.

4. **우아하게 커넥션 끊기**

   HTTP 명세에서는 예기치 않게 커넥션을 끊어야 한다면, 우아하게 커넥션을 끊어야 한다고 설명한다. 일반적으로 그 구현은 애플리케이션 자신의 출력 채널을 먼저 끊고 다른 쪽에 있는 기기의 출력 채널이 끊기는 것을 기다리는 것이다.