

OSI

2023년 3월 28일 화요일 오후 4:18

0 네트워크
SSS 네트워크 → 서버넷
망스

IP 헤더 데이터

패킷

라우터

IPv4/v6 공인 IP, 사설 IP
32bit/128bit 10.102.102.108

전기
0, 1, 랜카드



HTTP/DNS/FTP/SMTP/POP3

이더넷, WIFI

전송망장치 (Port)

L7 스위치

MAC (48bit)



L2 스위치
MAC 기억

TCP/UDP → HTTP3

TCP 헤더 데이터

시그니처

SYN, ACK

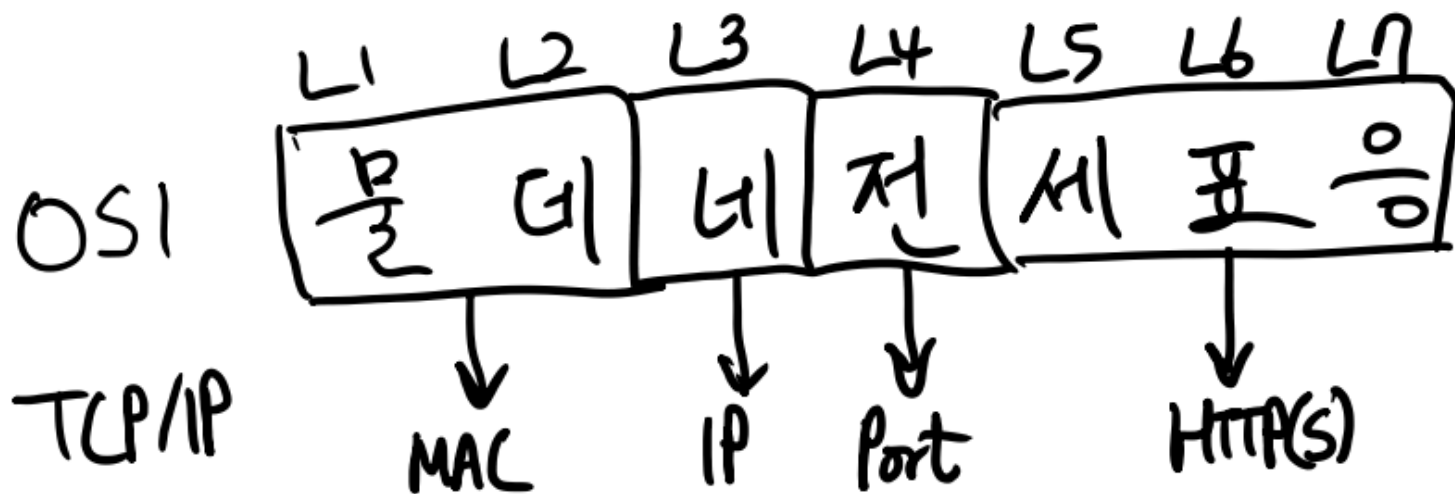
UDP 헤더 데이터그램

데이터그램

L4 스위치

클라이언트-서버 모델

2023년 4월 4일 화요일 오후 9:51

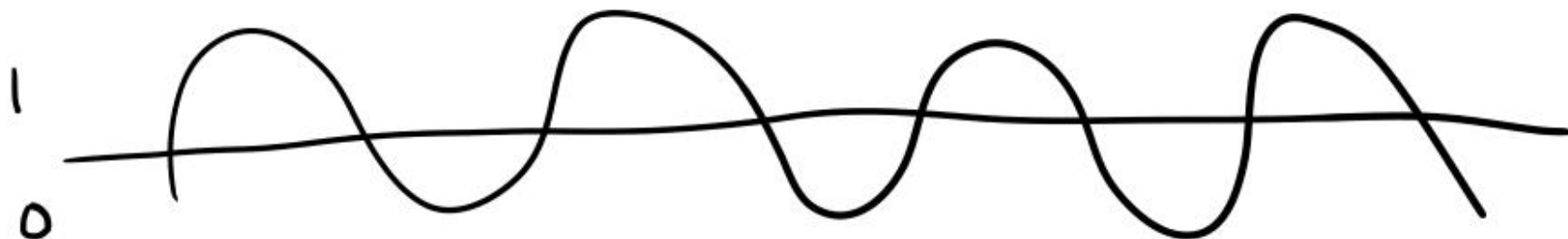


물리 계층 NI

2023년 4월 4일 화요일 오후 10:15



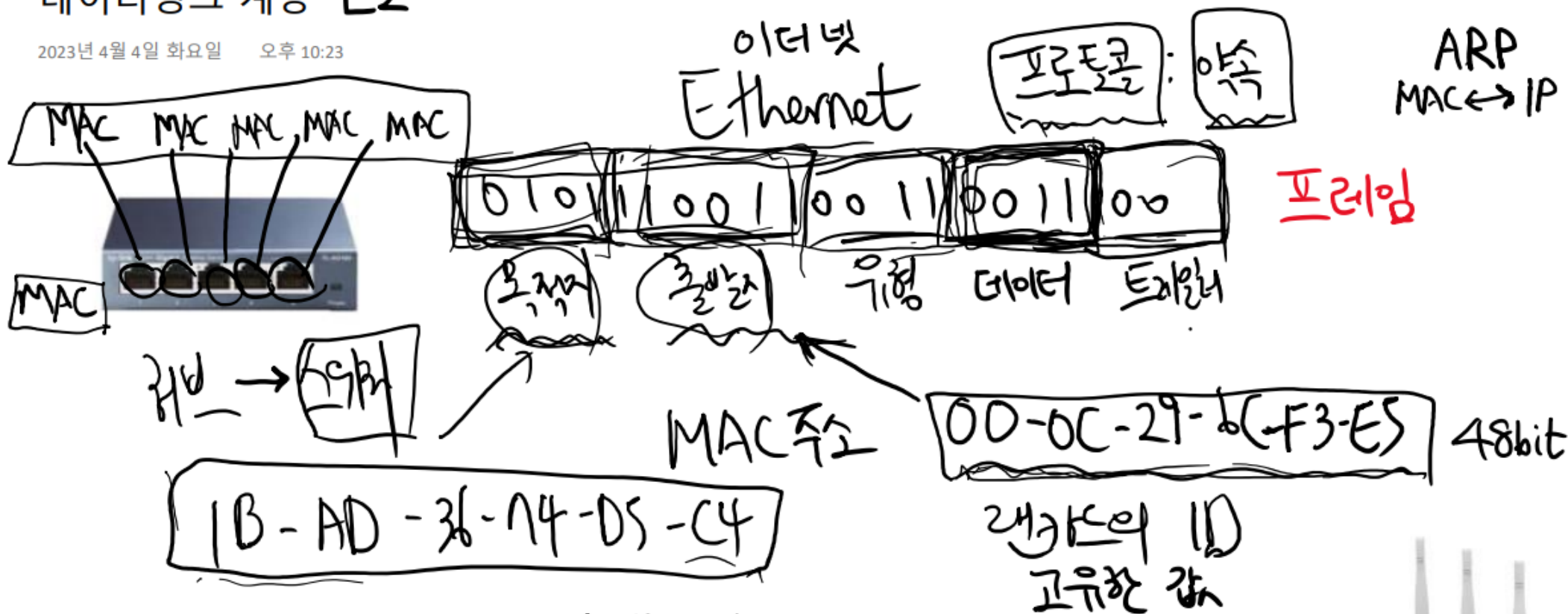
0 or 1 비트 bit 01110101 [8개 비트 byte
 옥텟 Octet



아날로그 ← 랜카드 → 디지털

데이터링크 계층 L2

2023년 4월 4일 화요일 오후 10:23



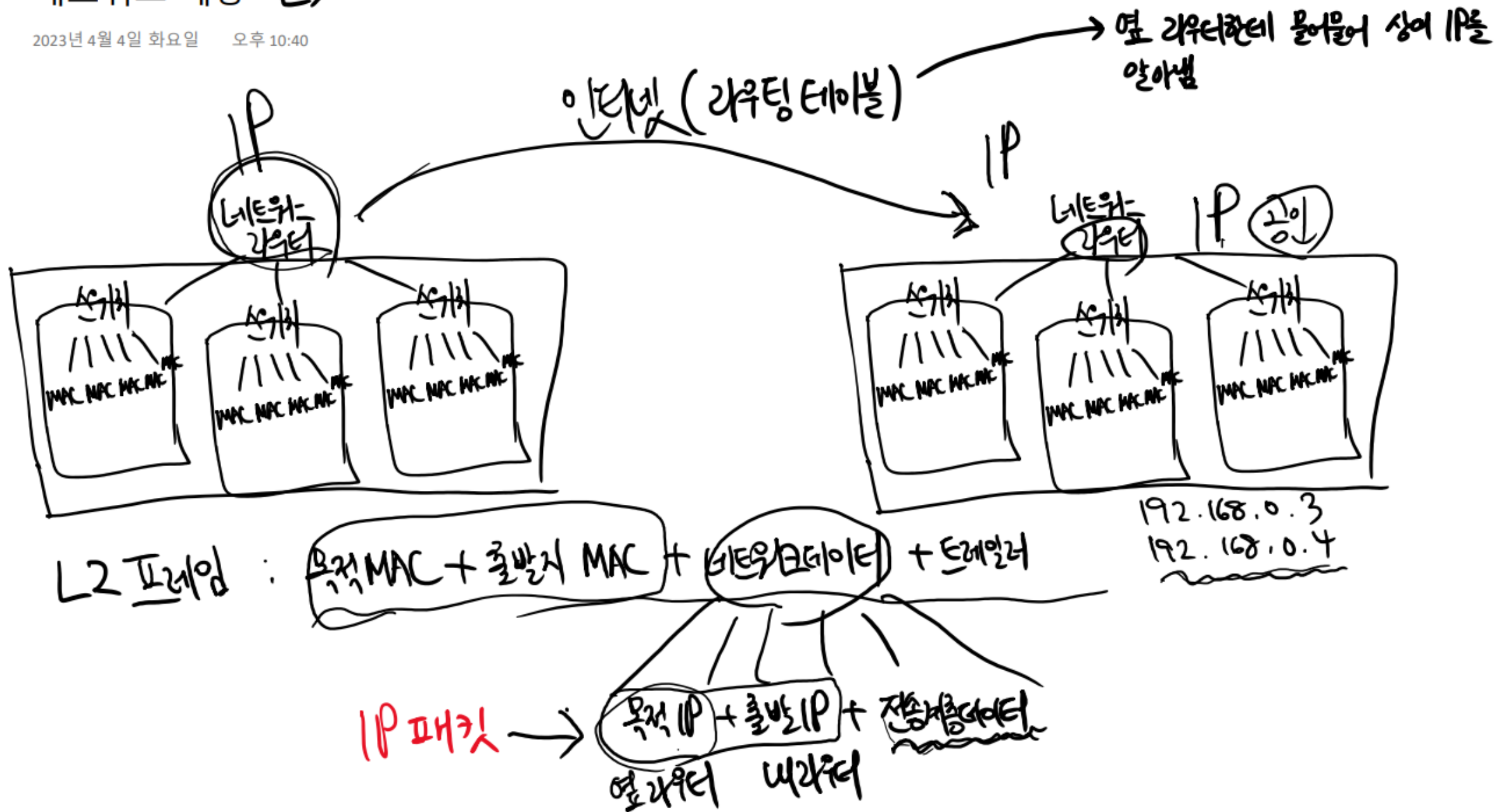
0001 → 1
0010 → 3
1010 → A
1111 → F

0101로 나타내면 너무 길어서
4자리씩 0~F 16진법으로
48bit → 16진법 12자리



네트워크 계층 L3

2023년 4월 4일 화요일 오후 10:40



공인IP vs 사설 IP

2023년 4월 22일 토요일 오후 2:03

네트워크에는 대표 주소인 IP가 있음 but 계속 한정으로 내부 네트워크는 사설IP쓰

ex) 공인IP 123.45.67.89 ————— 192.168 공유기같은 것에서 뿜을 수?

↑
내가 요청보낼 때는 이 IP로

사설
IP
192.16 ~ 192.31
10.

ex) 공유기 192.168.0.1
공용망 192.168.0.23
에어컨 192.168.0.34

특수 IP (Loopback)

127.0.0.1 = localhost 내 컴퓨터를 가리킴

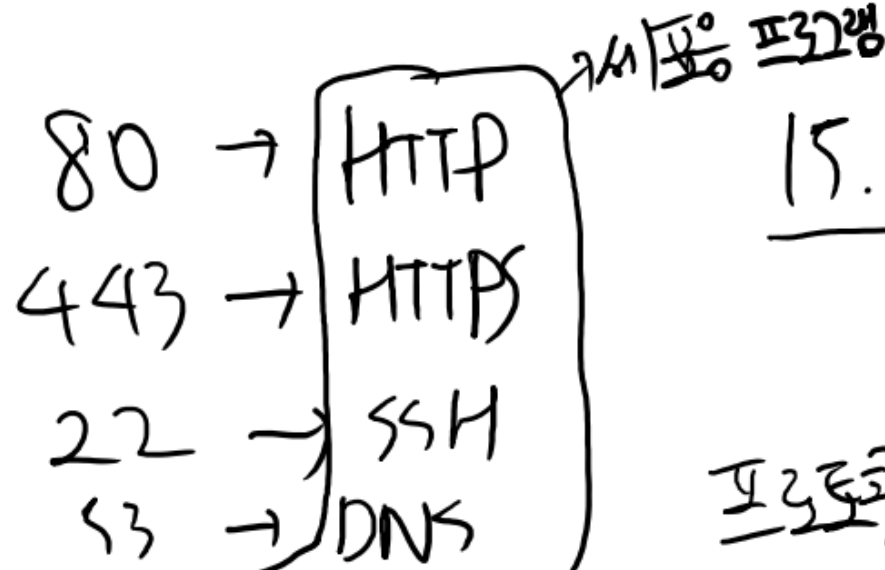
포트폴리오 배포URL

<http://localhost:9090/>

← 이런 실수 하지 맙시다!
진짜 없어보임 ㅋㅋ

전송 계층

2023년 4월 4일 화요일 오후 10:57



15.163.24.8 : 443
IP HTTPS 서버

naver.com : 443 (HTTPS 생략됨)

HTTP 경위 80. 생략가능

0~1023 Well Known Port
역할 정해져있음 (Unix에서는 sudo)

TCP
세그먼트



SMTP
POP3

HTTP 1.1/2

프로토콜
TCP / UDP

동영상, 사진

연결 맺어놓기 / 확인
데이터를 받았는지 확인
데이터 전송 확인

3 way handshake

SYN, ACK ← TCP헤더에 있음

UDP헤더 + 데이터
데이터그램

데이터그램

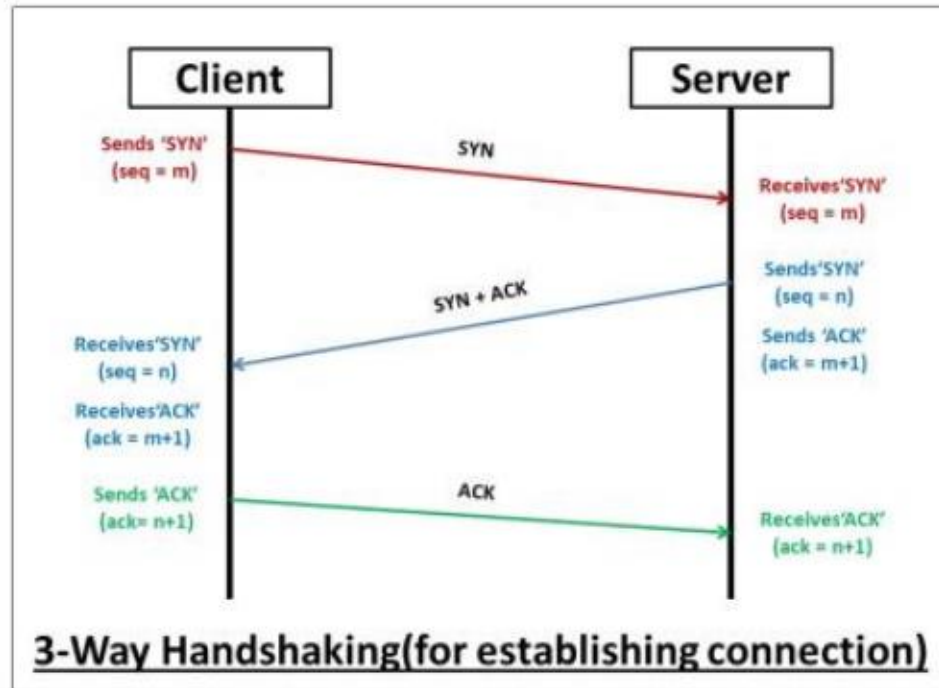
3way handshake

2023년 4월 22일 토요일 오후 2:16

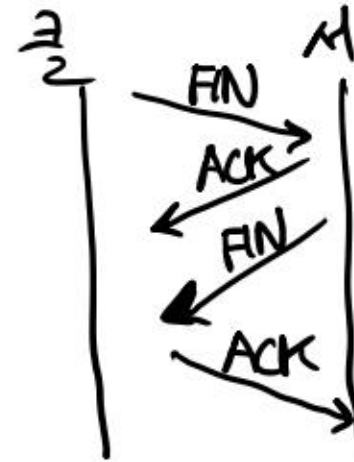
TCP 연결 시 항상 발생

↳ 비로졸적이라 개선 시도 대상

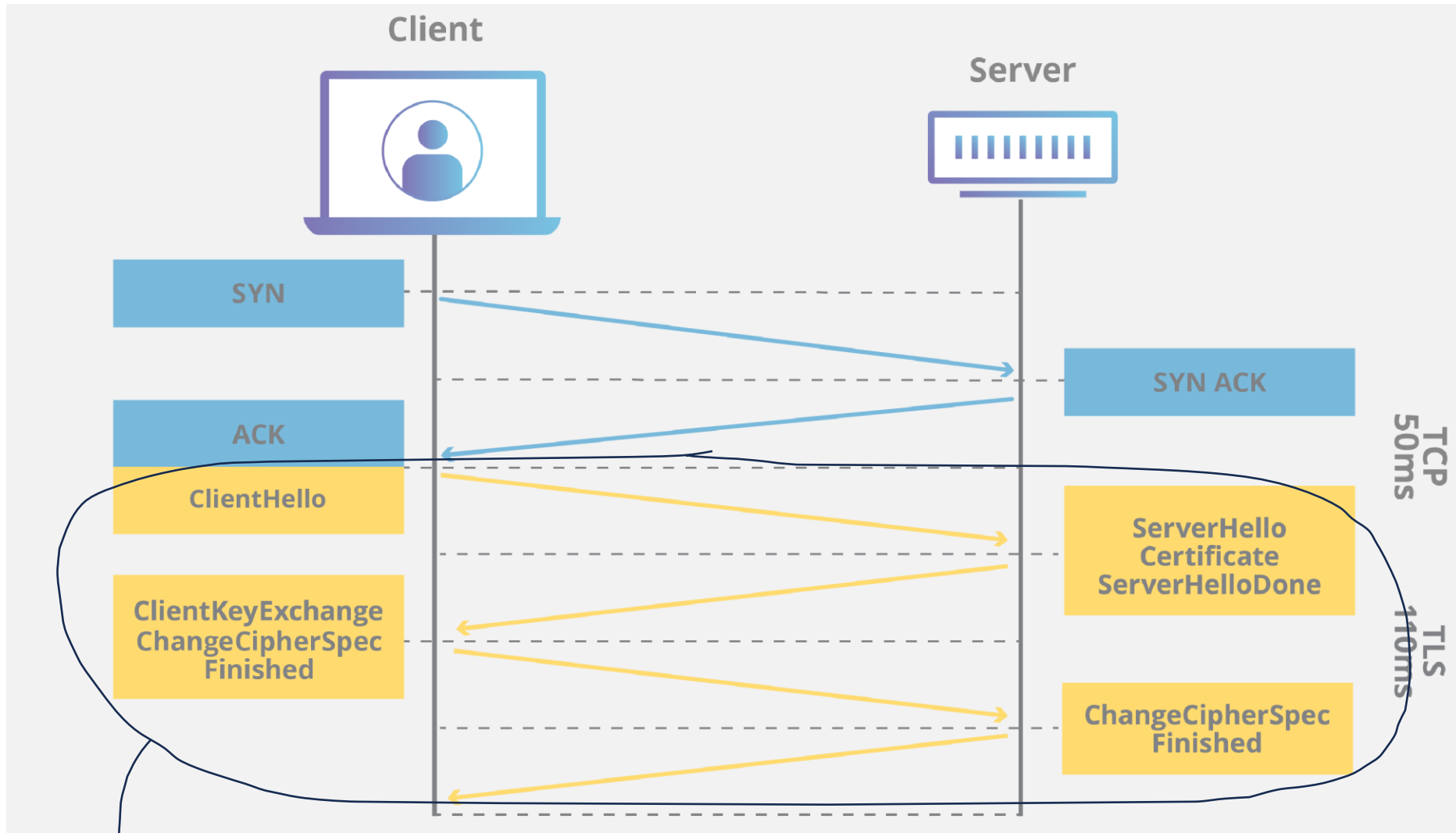
Connection: Keep-Alive



참고 4way handshake (연결 종료시)



FIN도 TCP 헤더에 있음

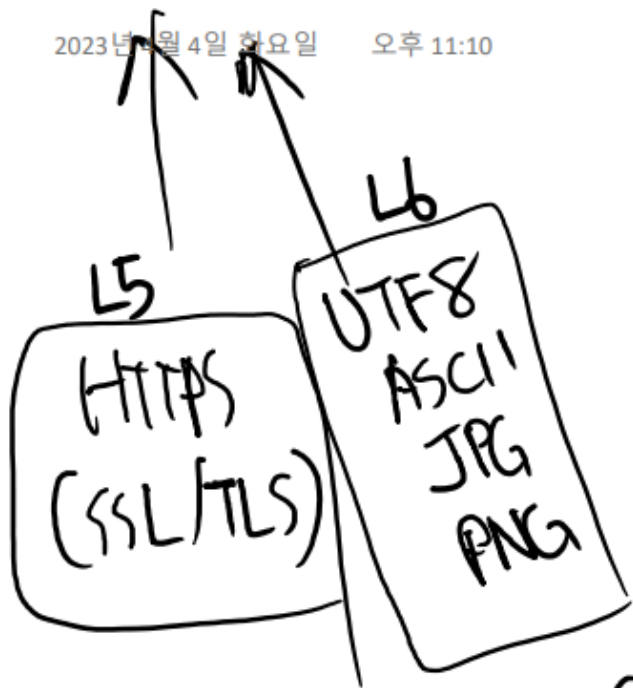


클라이언트와 서버간의 인증 및 암호화 키 교환을 위한 과정

SSL은 웹사이트와 브라우저 사이 (또는 두 서버 사이)에 전송되는 데이터를 암호화하여 인터넷 연결을 보호하기 위한 표준 기술
SSL은 TLS (Transport Layer Security)로 발전

세표응(응용 계층, HTTP)

2023년 4월 4일 화요일 오후 11:10



Header
HTTP 헤더 + HTTP 본문 (Body)

요청 / 응답
Header Header

Header [GET /index.html
Host: Zerocho.com
(헤더 쓰기)
(Body)

HTTPS는 무슨 계층?
L6이면 LS는 암호화 안돼?

세표응 구분은 의미X
응용프로그램 마당대로

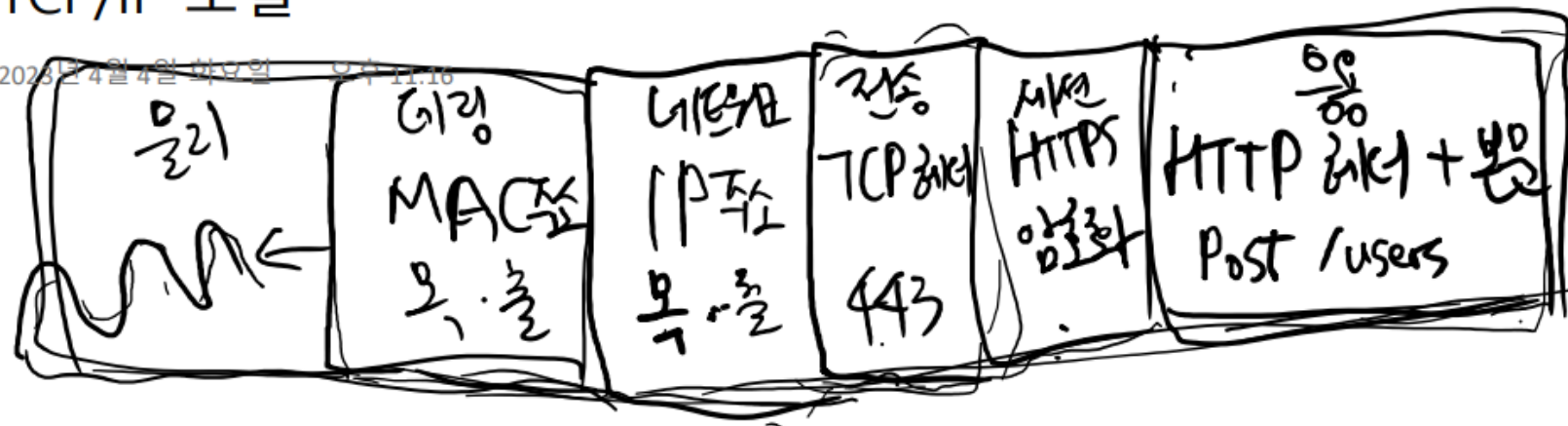
소켓 (Socket)
다른 비트코인/프로그램과
통신하기 위해 쓰는 창구

양방향 가능 (HTTP는 단방향)

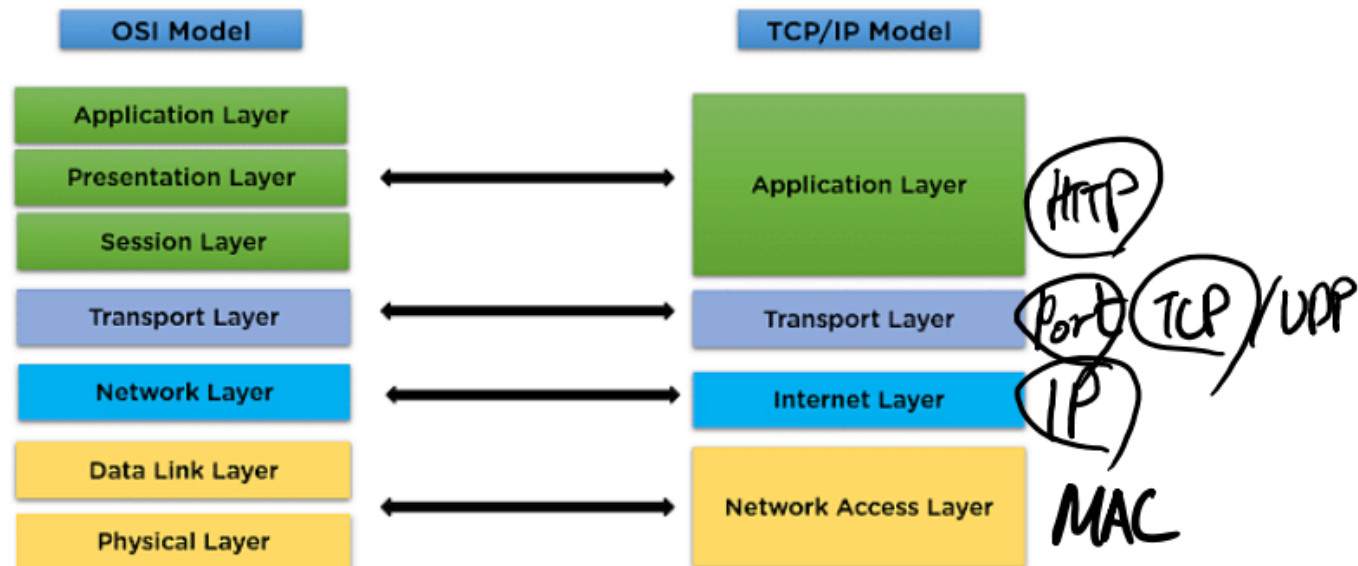
클 서
소켓 ← → 소켓
IP:Port IP:Port
응용 계층에서 TCP/UDP 계층 컨트롤

TCP/IP 모델

2023년 4월 4일 화요일 오후 11:16



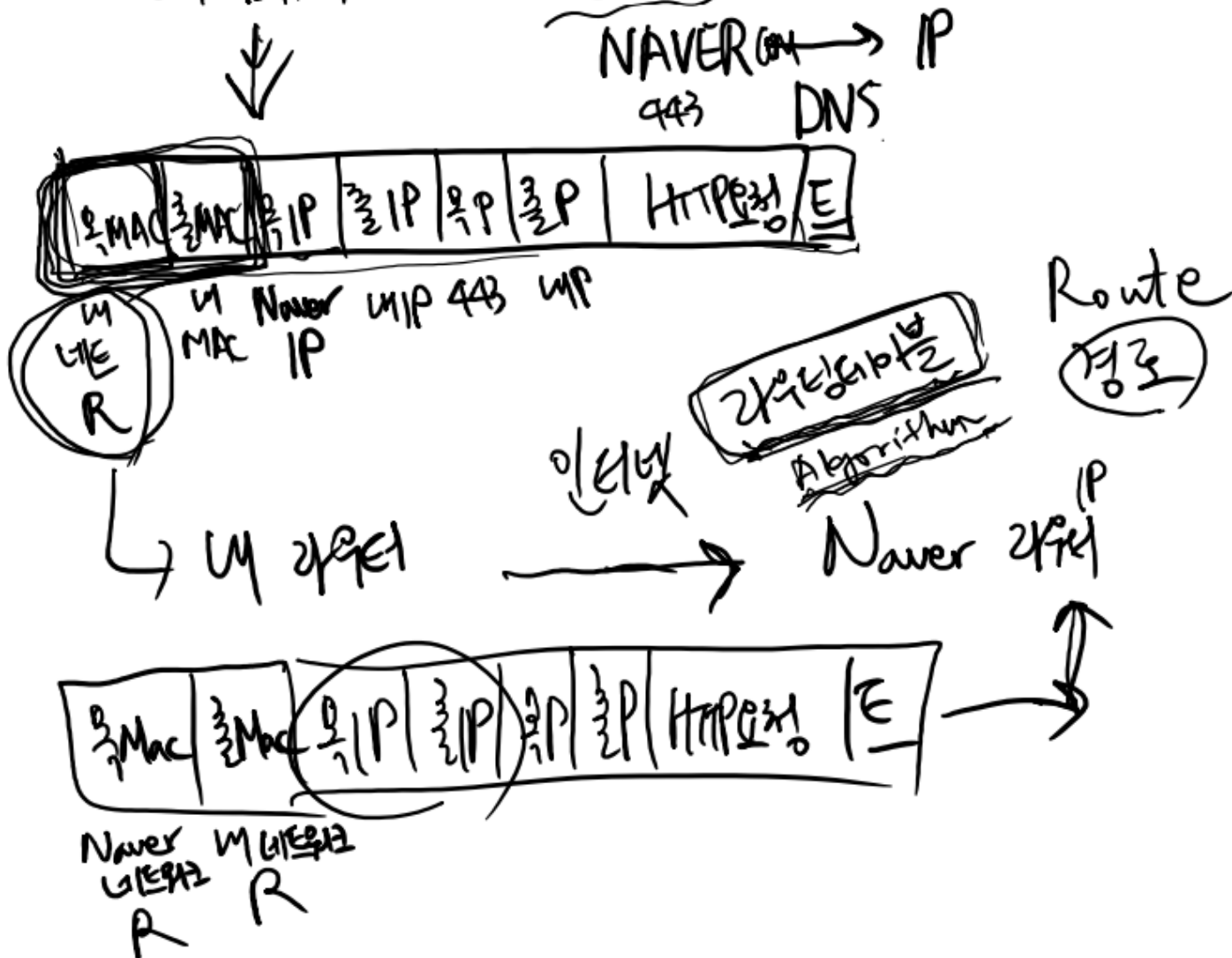
반응 때는 역순으로 헤더 제거



데이터 흐름 총정리

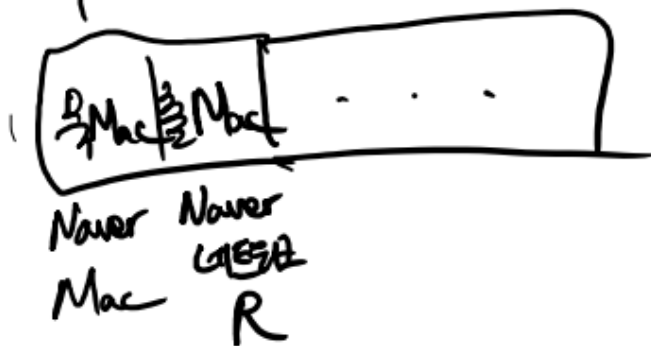
2023년 4월 4일 화요일 오후 11:38

내 컴퓨터 → 다른 컴퓨터 데이터 전송 전체 과정

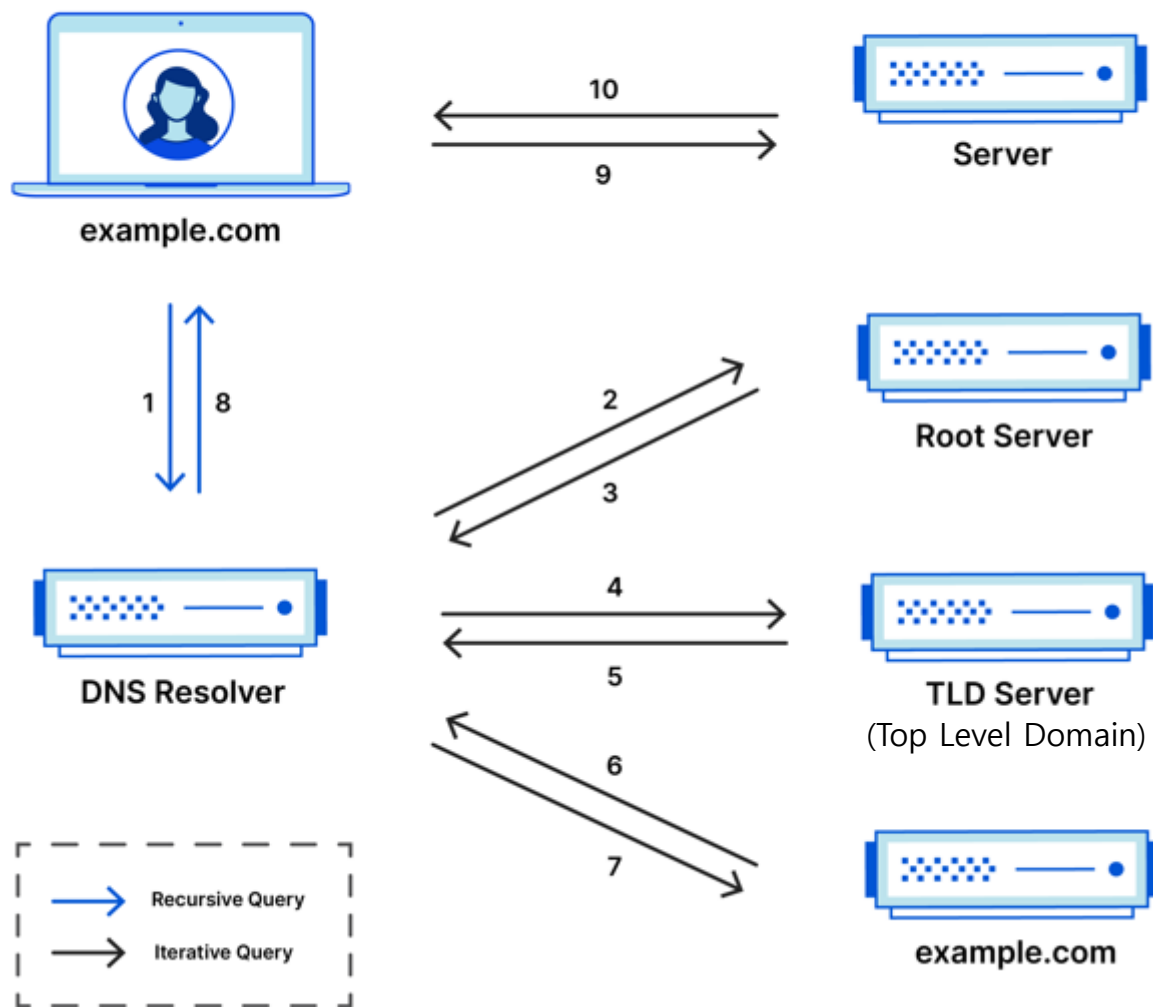


Naver 서버 컴퓨터

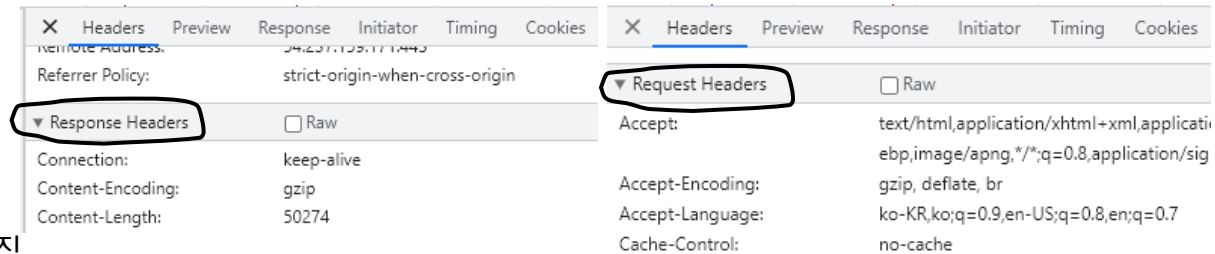
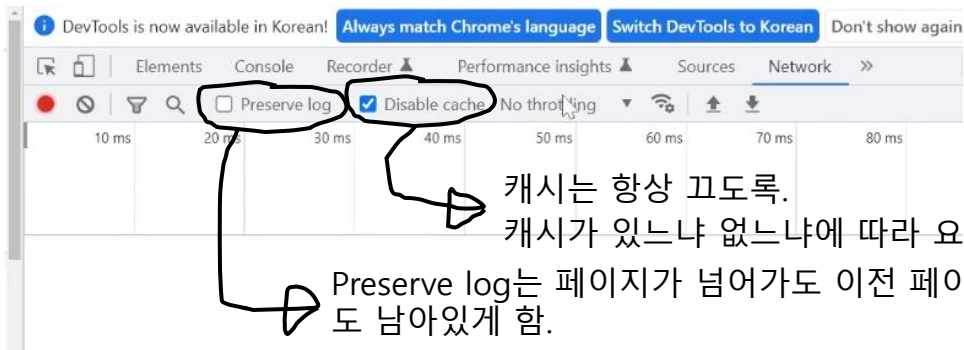
HTTP 요청
이고 응답 HTTP 메시지 작성
즉 완료



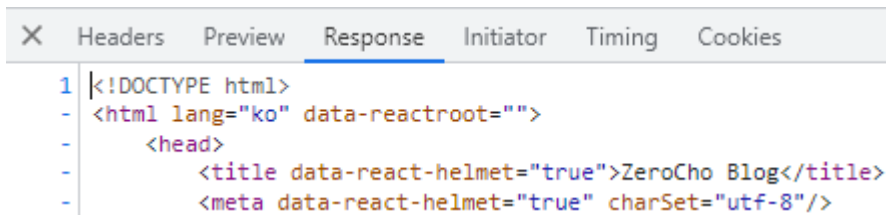
Complete DNS Lookup and Webpage Query



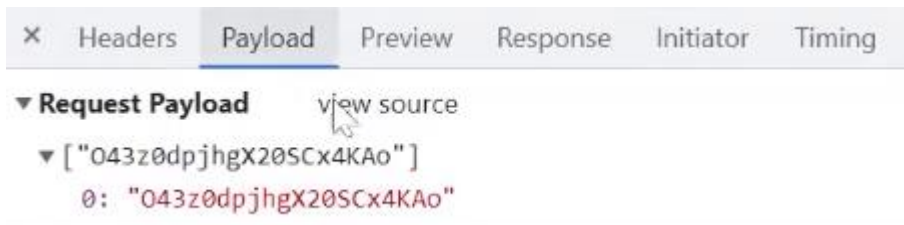
1. 사용자가 웹 브라우저에 'example.com'을 입력하면, 쿼리가 인터넷으로 이동하고 DNS Resolver이 수신합니다.
2. 이어서 확인자가 DNS 루트 이름 서버(.)를 쿼리합니다.
3. 다음으로, 루트 서버가, 도메인에 대한 정보를 저장하는 최상위 도메인(TLD) DNS 서버(예: .com 또는 .net)의 주소로 확인자에 응답합니다. example.com을 검색할 경우의 요청은 .com TLD를 가리킵니다.
4. 이제, 확인자가 .com TLD에 요청합니다.
5. 이어서, TLD 서버가 도메인 이름 서버(example.com)의 IP 주소로 응답합니다.
6. 마지막으로, 재귀 확인자가 도메인의 이름 서버로 쿼리를 보냅니다.
7. 이제, example.com의 IP 주소가 이름 서버에서 확인자에게 반환됩니다.
8. 이어서, DNS 확인자가, 처음 요청한 도메인의 IP 주소로 웹 브라우저에 응답합니다.
DNS 조회의 8단계를 거쳐 example.com의 IP 주소가 반환되면, 이제 브라우저가 웹 페이지를 요청할 수 있습니다
9. 브라우저가 IP 주소로 [HTTP](http://example.com) 요청을 보냅니다.
10. 해당 IP의 서버가 브라우저에서 렌더링할 웹 페이지를 반환합니다 (10단계).



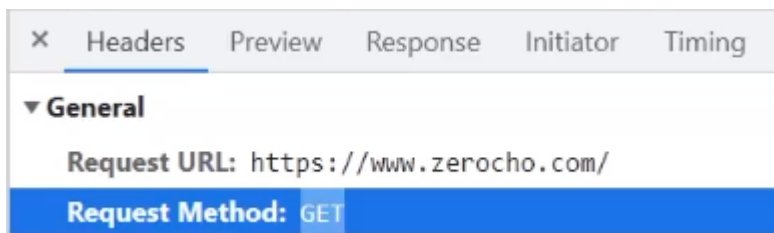
네트워크 탭의 헤더에는 요청, 응답이 같이 기록되어있다.



Response는 응답의 http body이다



Payload는 요청의 http body이다



GET이나 DELETE로 되어있으면 일반적으로 요청의 body를 안보낸다

▼ Request Headers

:authority: jnn-pa.googleapis.com
:method: POST
:path: /\$rpc/google.internal.waa.v1.Waa/Create

▶ 양옆으로 콜론이 찍혀있는 게 있으면 http2이거나 http3이다.

Name	Method	Status	Protocol	Domain	Type
5500cc1944...	GET	200	h2	s.gravatar.com	png
e209a68cee...	GET	200	h2	s.gravatar.com	png
437012aaa4...	GET	200	h2	s.gravatar.com	png

Name	Method	Status	Protocol	Domain	Type
embed.js	GET	200	h3	www.youtube.c...	script
Create	POST + Pre...	200	h3	jnn-pa.googlea...	xhr

▼ General

Request URL: https://www.zerocho.com/app.js

Request Method: GET

Status Code: 🟢 200 OK

Remote Address: 3.226.182.14:443

Referrer Policy: no-referrer

▼ Response Headers

[View source](#)

Accept-Ranges: bytes

Cache-Control: public, max-age=0

Connection: keep-alive

Content-Encoding: gzip

Content-Length: 233196

Content-Type: application/javascript; charset=UTF-8

Cross-Origin-Opener-Policy: same-origin

Date: Thu, 06 Apr 2023 14:04:43 GMT

Etag: W/"38eec-1874fe2dee8"

Expect-Ct: max-age=0

Last-Modified: Wed, 05 Apr 2023 05:28:01 GMT

Origin-Agent-Cluster: ?1

여기 있는 용어들이 궁금하면
RFC7231(최신) 또는 RFC2616

▼ Request Headers ☒ Raw

```

GET /book/2 HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Cache-Control: no-cache
Connection: keep-alive
Cookie: heroku-session-affinity=ACyDaANoA24IAeWIwP////8HYgAELKViAAEnumEBbAAAAAftAAAABXdlYi4xahl46RsvsuidEToYhdDhcRU+za/k; sessionId=s%3AS2t6waXHU0eqRCNdIgsfhp5cdobd-Zjz.U3f3mrXPqW%2FW8yFGOQQedImWsCI6pJHBlic%2Bm9a%2FQAs; _ga=GA1.2.1651219147.1690273576; _gid=GA1.2.90713021.1690273576; _gat=1; _ga_WHKFYRKE39=GS1.2.1690282143.2.0.1690282143.60.0.0; __gads=ID=4d1db48b2020c97b-2214c59af3e20077:T=1690273575:RT=1690282143:S=ALNI_MYedYZFVtupYIHP0D6yIKNxDC90Hg; __gpi=UID=00000d12c84efcd1:T=1690273575:RT=1690282143:S=ALNI_MaNdk0ESLLrXnGYVd3Vqsh05LCVwQ
Host: www.zerocho.com
Pragma: no-cache
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0 Safari/537.36
sec-ch-ua: "Not/A)Brand";v="99", "Google Chrome";v="115", "Chromium";v="115"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"

```

서버쪽에 어떤 데이터를 원한다는 걸 요청

쿠키의 문자열을 서버쪽에서 찾아서 누구의 http요청인지 식별한다

Host가 첫번째 행으로부터 분리되어 있는 이유는 http0.9 시절에는 Host가 없었고 다른 네트워크에서 정보를 가져올 생각을 못 했다. 옛날에 잘못된 설계가 지금까지 영향을 끼친 것

사용자가 어떤 환경인지 알려주는 역할이지만 의미가 없다

200은 상태코드 ←
서버가 Cowboy라는 걸 알림 ←
(숨기는 경우 많음)

```
▼ Response Headers ☒ Raw
HTTP/1.1 200 OK
Server: Cowboy
Connection: keep-alive
Cross-Origin-Opener-Policy: same-origin
X-Dns-Prefetch-Control: off
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-Download-Options: noopen
X-Content-Type-Options: nosniff
Origin-Agent-Cluster: ?1
X-Permitted-Cross-Domain-Policies: none
Referrer-Policy: no-referrer
X-Xss-Protection: 0
Content-Type: text/html; charset=utf-8
Content-Encoding: gzip
Content-Length: 50274
Etag: W/"c462-iPLmPsCyoMNHZ/q70yBo7QaVHVQ"
Date: Tue, 25 Jul 2023 08:27:36 GMT
Via: 1.1 vegur
```

요청 헤더의 Accept랑 매칭되는 부분 ←

서버시간 ←
어디 거쳐서 왔는지 알림 ←

URL 구조

href									
protocol			auth		host		path		hash
					hostname	port	pathname	search	
								query	
" https:	//		user : pass	@	sub.example.com	: 8080	/p/a/t/h	? query=string	#hash "
					hostname	port			
protocol			username	password	host				
origin					origin		pathname	search	hash
href									

http 메소드를 서버에서 미리 정해둔다. (백엔드 개발자와 프론트엔드 개발자끼리 약속을 하며 이를 HTTP API를 만든다라고 말함)

POST /user // 유저 등록 + 최근 스펙 body 보내도 되는 걸로
POST /post // 게시글 등록
POST /comment // 댓글 등록
GET /users // 유저 전체 데이터 가져오기
GET /users/1 // 1번 유저 데이터 가져오기
HEAD /users 200 // 이 라우터가 제대로 동작하는지 체크 (GET과 동일하지만 응답 body만 안보내준다)
DELETE /user // 유저 데이터 삭제
PUT /user // 유저 데이터 전체 교체
PATCH /user // 유저 데이터 부분 교체

REST API -> HTTP API를 아름답게 설계하는 규칙 (그러나 정확한 규칙 같은 게 없고 정확하게 쓰이지 않는다)

GET /registerUser // 유저 등록 + 최근 스펙 body 보내도 되는 걸로
GET /user // 유저 데이터 가져오기
GET /deleteUser // 유저 데이터 삭제
GET /updateUser // 유저 데이터 수정

엄격하지 않기 때문에 이렇게 해도 되긴 하다.

Create	POST	Preflight	200	h3	jnn-pa.googleapis.com	xhr
embed.js	GET		200	h3	www.youtube.com	script
embed.js	GET		200	h3	www.youtube.com	fetch
data:image/png;base64...	GET		200	data		png
AL5GRJV0cbY5RKA...	GET		200	h3	yt3.ggpht.com	jpeg
KFOmCnqEu92Fr1M...	GET		200	h3	fonts.gstatic.com	font
Create	OPTIONS		200	h3	jnn-pa.googleapis.com	preflight
remote.js	GET		200	h3	www.youtube.com	script
2ordrZu4NrapatEoB...	GET		200	h3	www.google.com	script
hqdefault.jpg?sqp=...	GET		200	h3	i.ytimg.com	jpeg
Create	POST	Preflight	200	h3	jnn-pa.googleapis.com	xhr

추가로 OPTIONS 메소드도 있다.

내 서버가 다른 도메인으로 요청을 보내고 받고 하는 걸 허용하려면 OPTIONS 메소드를 만들어놔야한다.

Preflight가 OPTIONS랑 연결된다.

안전한 메소드: 서버의 상태를 바꾸지 않는 메소드

- GET
- HEAD
- OPTIONS
- TRACE

멥등성 메소드: 같은 거를 여러 번 호출해도 최종적으로 서버에서
는 한번만 바뀐 거나 다름이 없는 메소드 (안전한 메소드 포함)

- PUT (ex A를 B로 한번 바꾸나 여러 번 바꾸나 서버에서는 한번만 바뀜)
- DELETE (ex 1번 유저를 한번 제거하나 여러 번 제거하나 서버에서 한번만 제거됨)

캐시가 가능한 메소드 : GET, HEAD, POST (대부분의 캐시 구현은 GET, HEAD만 지원)

상태코드

1. 100 – 199 정보
2. 200 – 299 성공
3. 300 – 399 리다이렉션
4. 400 – 499 클라이언트 에러
5. 500 – 599 서버 에러

상태코드는 메소드와 마찬가지로 엄격하지 않다.
POST를 성공했을때는 200이 아닌 201 Created를 보내는 경우도 많다.
비어있는 번호도 있는데 그 번호들은 자유롭게 쓰면 된다.

Information responses

101 Switching Protocols 프로토콜을 전환하라는 응답

This code is sent in response to an Upgrade request header from a client. The code indicates the protocol the server is switching to.

Successful responses

200 OK 성공

The request succeeded. The result meaning of "success"

201 Created POST가 성공일 때 201 보내는 경우도 있음

The request succeeded, and a new resource was created as a result. This is typical sent after POST requests, or some PUT requests.

204 No Content 응답 Body가 없을 때

There is no content to send for this request, but the headers may be useful. The user agent may update its cached headers for this resource with the new values.

206 Partial Content 데이터가 일부만 왔을 때

This response code is used when the Range header is sent from a client and the server supports Range requests. It indicates that only part of the resource is being returned.

Redirection messages

300 Multiple Choices 요청을 애매하게 보낼 때

The request has more than one possible response. The client must choose one of them. (There is no standardized way to choose, but HTML links to the possibilities are provided.)

302 Found 다른 페이지로 옮겨주게 할 때 (일시적으로)

This response code means that the URI of requested resource has changed temporarily. Further changes in the URI might be made in the future. The same URI should be used by the client in future request.

301 Moved Permanently

다른 페이지로 옮겨주게 할 때 (영구적으로)
(Location: /user<- 헤더에 이런식으로 알려줘야 함)
The URL of the requested resource has been changed permanently. The new URL is given in the response.

307 Temporary Redirect 302랑 똑같으나 HTTP 메소드를 바꾸면 안됨

The server sends this response to direct the client to get the resource at another URI with the same method that was used in the previous request. It has the same semantics as the 302 Found HTTP response code, with the exception that the user agent *must not* change the HTTP method used: if a `POST` was used in the first request, a `POST` must be used in the second request.

Client error responses

400 Bad Request 범용적인 클라이언트 에러 상태코드

The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

403 Forbidden 로그인을 해도 못들어가는 페이지일 때 (ex 등급에 따라 못들어가는 페이지)

The client does not have access rights to the content; that is the server is refusing to give the requested resource. Unlike client's identity is known to the server.

405 Method Not Allowed 사용할 수 없는 메소드일 때 (Allow: 메소드이름 <- 헤더에 이런식으로 알려줘야 함)

The request method is known by the server but is not supported for the resource. For example, an API may not allow calling `DELETE` to remove

401 Unauthorized 로그인 해야만 들어갈 수 있는 페이지일 때

Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.

404 Not Found 주소가 없는 페이지일 때

The server cannot find the requested resource. In the browser, this status code is not recognized. In an API, this can also mean that the endpoint resource itself does not exist. Servers may also send this response code to hide the existence of a resource from an unauthorized client. This response code is probably the most well known due to its frequent use on the web.

408 Request Timeout 요청을 했다가 너무 오래 걸렸을 때

This response is sent on an idle connection by the server after some time without a previous request by the client. It means that the connection is now unused. This response is used much less frequently than the others. Chrome, Firefox 27+, or IE9, use HTTP pre-connection

Server error responses

500 Internal Server Error 범용적인 서버 에러 상태코드

The server has encountered a situation it does not know how to handle.

의도적으로 서버를 내렸을 때나 서버에 에러가 생겼을 때

503 Service Unavailable

(Retry-After: 3600 <- 헤더에 이런 식으로 알려줘야 함)

The server is not ready to handle the request. Common causes include server maintenance or that it is overloaded. Note that together with this response, a `Retry-After` HTTP header should, if possible, contain the estimated time before the service may be available.

502 Bad Gateway

서버 앞단에 있는 프록시, 게이트웨이, 방화벽쪽에 에러가 있을 때

This error response means that the server, while working as a gateway to get a response needed to handle the request, got an invalid response.

504 Gateway Timeout

서버 앞단에 있는 프록시, 게이트웨이, 방화벽쪽에서 시간 초과일 때

This error response is given when the server is acting as a gateway or proxy and did not receive a timely response from the upstream server.

컨텐츠협상

2023년 4월 22일 토요일 오후 2:40

HTTP(Hyper Text Transfer Protocol)
Header, Body(본문, payload, content)로 구성

헤더는 Key: Value 꼴이며 한글 안됨.

ex) GET /users HTTP/1.1

Host: zerocho.com

Set-Cookie: key=value; domain=zerocho.com

Accept = text/plain, text/html
Accept-Encoding = gzip, br
Accept-Language = ko-KR; q=0.9
Accept-Charset = utf-8

Content-Type = text/html; charset=utf-8
Content-Encoding = br
Content-Language = ko-KR

Value의 한글은 인코딩으로 변환해야 함. js기준 인코딩 방법 =>

```
> encodeURIComponent('조현영')  
< '%EC%A1%B0%ED%98%84%EC%98%81'
```

MIME Type : 대분류/확장자

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
```

q는 우선순위 (0부터 1까지. 1이 가장 우선순위 높음)

/*은 모든 타입을 의미하는 와일드카드

서버 주도 협상: Accept로 원하는걸 알리면 서버가 원하는걸 하나를 정해서 보내는것

클라이언트 주도 협상: 서버가 여러 선택지를 주고 클라이언트가 골라서 요청을 한번 더 보내는 것 (두번 왔다갔다 하기 때문에 잘 안쓰임)

▼ Response Headers	<input type="checkbox"/> Raw
Connection:	keep-alive
Date:	Wed, 26 Jul 2023 02:12:17 GMT
Keep-Alive:	timeout=5
Transfer-Encoding:	chunked

Connect: keep-alive는 http1.0 옛날 버전을 위해 존재하는 것. 그러나 안넣어도 기본값으로 되어있다.

Keep-Alive: timeout=5 는 요청 보낼 때마다 3way handshake를 해야하니 5초안에 한번 맺으면 재사용하도록 하는 것.

Date는 서버에서 메시지가 생성된 시간 즉, 응답 Header, 응답 Body가 생성된 시간이며 서버시간을 알려준다.

Transfer-Encoding: chunked 는 데이터를 조각조각 전달됨을 의미. (또는 gzip)

▼ Request Headers	
:authority:	developer.mozilla.org
:method:	GET
:path:	/en-US/docs/Web/HTTP/Overview
:scheme:	https
Accept:	text/html,application/xhtml+xml,application/xml;q=0.9,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding:	gzip, deflate, br
Accept-Language:	ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Cache-Control:	no-cache
Cookie:	_ga=GA1.2.1552468797.1690445360; _gid=GA1.2.20233
Pragma:	no-cache
Referer:	https://developer.mozilla.org/en-US/docs/Web/HTTP

Referer은 이전페이지가 어딘지 알려주는 역할
(Referrer이 맞는 스펠링이지만 설계한 사람이 처음에 잘못 만들어서 지금까지 잘못된 철자로 이어져온 것)

▼ General	
Request URL:	https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
Request Method:	GET
Status Code:	● 200
Remote Address:	34.111.97.67:443
Referrer Policy:	strict-origin-when-cross-origin

개인정보 이슈 때문에 Referrer Policy가 정해져 있다.
strict-origin-when-cross-origin은 이전페이지의 origin과 현재페이지의 origin이
다르면 referer를 띄우지 않는다.

이외에도 헤더에는 커스텀헤더가 있다.
커스텀 헤더는 일반적으로 Key 앞에 X-를 붙이는게 관례이다.

쿠키 쿠키의 형식은 키=값; 옵션1; 옵션2;

2023년 4월 22일 토요일 오후 2:33

HTTP는 Stateless → 이전 요청을 기억 못함.

but 실제로는 State(상태)가 필요함.

그래서 쿠키를 매번 넣어 누구의 요청인지 알 수 있게! → 매번 새로 데이터 낭비 조심

Set-Cookie: 키=값; max-age=86400; HttpOnly; Secure; SameSite=None → HTTPS
↳ 하면 매 요청 헤더에 **Cookie: 키=값** 전송됨 document.cookie

Connection: keep-alive
Date: Thu, 27 Jul 2023 09:25:53 GMT
Keep-Alive: timeout=5
Set-Cookie: hello=world;
Transfer-Encoding: chunked

Domain: 쿠키를 전송 할 도메인 → 서브도메인까지 전송 가능.
Path: 특정 경로를 지정 → /api로 지정되어있으면 example.com/api까지 쿠키를 가져갈 수 있음. (하위 Path도 가능)
Expires/Max-Age: 쿠키 유효시간 → Session은 브라우저 종료하면 쿠키가 만료. Expires는 정확한 시간, Max-Age는 초. Max-Age를 더 쓰는 추세, 우선순위가 Max-Age가 더 높음.
HttpOnly: 자바스크립트로 쿠키 접근 허용 여부 → HttpOnly가 true면 자바스크립트에서 접근 불가 (js에서 document.cookie로 확인 가능)
Secure: https일 때만 쿠키 허용
SameSite: 다른 도메인으로 쿠키 도메인 전송 제어 → None일 때 다른 도메인으로 쿠키를 보내줄 수 있다 (secure일 때만), Strict일 때는 같은 도메인에서만 가능, Lax는 링크같은 것만 예외적으로 외부도메인과의 쿠키를 허용해준다. (기본값)

응답 헤더에 Set-Cookie를 쓰면 브라우저에 쿠키가 저장
Request Headers를 보면 매번 서버로 쿠키가 전달된다 (만료되기 전까지)

Request Cookies									
<input type="checkbox"/> show filtered out request cookies									
Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	
hello	world	localhost	/	Session	10	✓			

Cookies 탭이 생기며 쿠키가 저장된 걸 볼 수 있다

Application									
Filter									
Application									
Manifest									
Service Workers									
Storage									
Storage									
Local Storage									
Session Storage									
IndexedDB									
Web SQL									
Cookies									
http://localhost:8080									
Name	Value	Domain	Path	Expires / Max-Age	Size	Http			
hello	world	localhost	/	Session	10				

*Application탭에서도 쿠키를 확인할 수 있다

요청: GET /users -> Cache -> 서버
응답: 서버 -> Cache -> HTTP/1.1 200 ok

위와 같이 실제로 요청, 응답을 할 때 캐시를 거치며 캐시저장소에 저장을 하고 간다.

저장소(DB, 메모리, 파일)에 원하는 데이터를 찾고 미적중이면 서버로 간다.

Cache Hit: 적중 (원하는 대상을 찾았을 때)

Cache Miss: 미적중 (원하는 대상을 못찾았을 때)


캐시의 장점: 데이터 비용을 아낄 수 있다.

캐시의 단점: 오래된 데이터를 갖고 있을 수 있다.


▼ Response Headers	<input type="checkbox"/> Raw
Cache-Control:	max-age=6, must-revalidate
Connection:	keep-alive
Date:	Thu, 27 Jul 2023 12:43:37 GMT
Keep-Alive:	timeout=5
Transfer Encoding:	chunked


max-age=6
캐시 유효시간을 정할 수 있다 (6초)
must-revalidate
유효기간 지나면 신선도 검사



▼ General	
Request URL:	http://localhost:8080/favicon.ico
Request Method:	GET
Status Code:	 200 OK
Remote Address:	[::1]:8080
Referrer Policy:	strict-origin-when-cross-origin

처음 응답은 서버로부터 받음

▼ General	
Request URL:	http://localhost:8080/favicon.ico
Request Method:	GET
Status Code:	 200 OK (from disk cache)
Remote Address:	[::1]:8080
Referrer Policy:	strict-origin-when-cross-origin

▼ General	
Request URL:	http://localhost:8080/favicon.ico
Request Method:	GET
Status Code:	 200 OK
Remote Address:	[::1]:8080
Referrer Policy:	strict-origin-when-cross-origin

유효시간 6초가 지나면 다시 서버로부터 응답받게 된다.

Cache가 오래됐는데 서버에 물어보니 변경이 없는 경우 => Cache꺼 그냥 써, 304 Not Modified 바로 그 다음부터는 캐시로부터 불러옴

Cache-Control: no-cache // 캐시 저장해도 되는데 항상 신선도 검사한다 (유효기간 무시)
Cache-Control: no-store // 캐시 저장 x
Cache-Control: must-revalidate // 유효기간 지나면 신선도 검사
Cache-Control: private // 기본값
Cache-Control: public // 캐시 남들과 공유될 가능성이 생김 (Authorization이 있는 경우에는 private으로 전환됨)
Cache-Control: Stable-While-Revalidate // 일단 오래된거 주고, 뒤에서 서버한테 물어봐서 새거 있으면 몰래 업데이트
Cache-Control: Stable-if-error // 서버가 에러 났어도 일단 오래된거 줌 (다만 지정한 기간이 지나버리면 그 때는 에러를 표시)

캐시가 아닌 실제 데이터가 언제 최종적으로 바뀌었는지 정보도 제공하며 이를 통해 캐시랑 서버에 데이터가 같은지 확인
(Cache-Control: max-age는 캐시 자체를 얼마나 보관할지이므로 헛갈리면 안됨)

GET /count HTTP/1.1

HTTP/1.1 200 ok

Last-Modified: Wed, 21 Oct 2022 07:28:00 GMT

Etag: W/"5f2292ab0011b36bc72209aaa7a21f8c3"

일반적으로 캐시가 있는 경우에 이런식으로 헤더에 Last-Modified 또는 Etag가 존재한다.

다만 Etag:W/"값"이 아니라 Etag: "값" 형식이여야 한다.

W/는 약한 검사를 의미하며, 화면은 다르지만 핵심 내용이 같은 경우는 동일하게 취급

값에는 아무 단어나 넣어도 된다.

예를들어 보기 쉽게 Etag: "v1.1" (1.1버전일 때)

신선도 검사

If-Match: "<etag_value>" => 요청을 보낼 때 etag가 같으면 서버로부터 수신 (다르면 412)

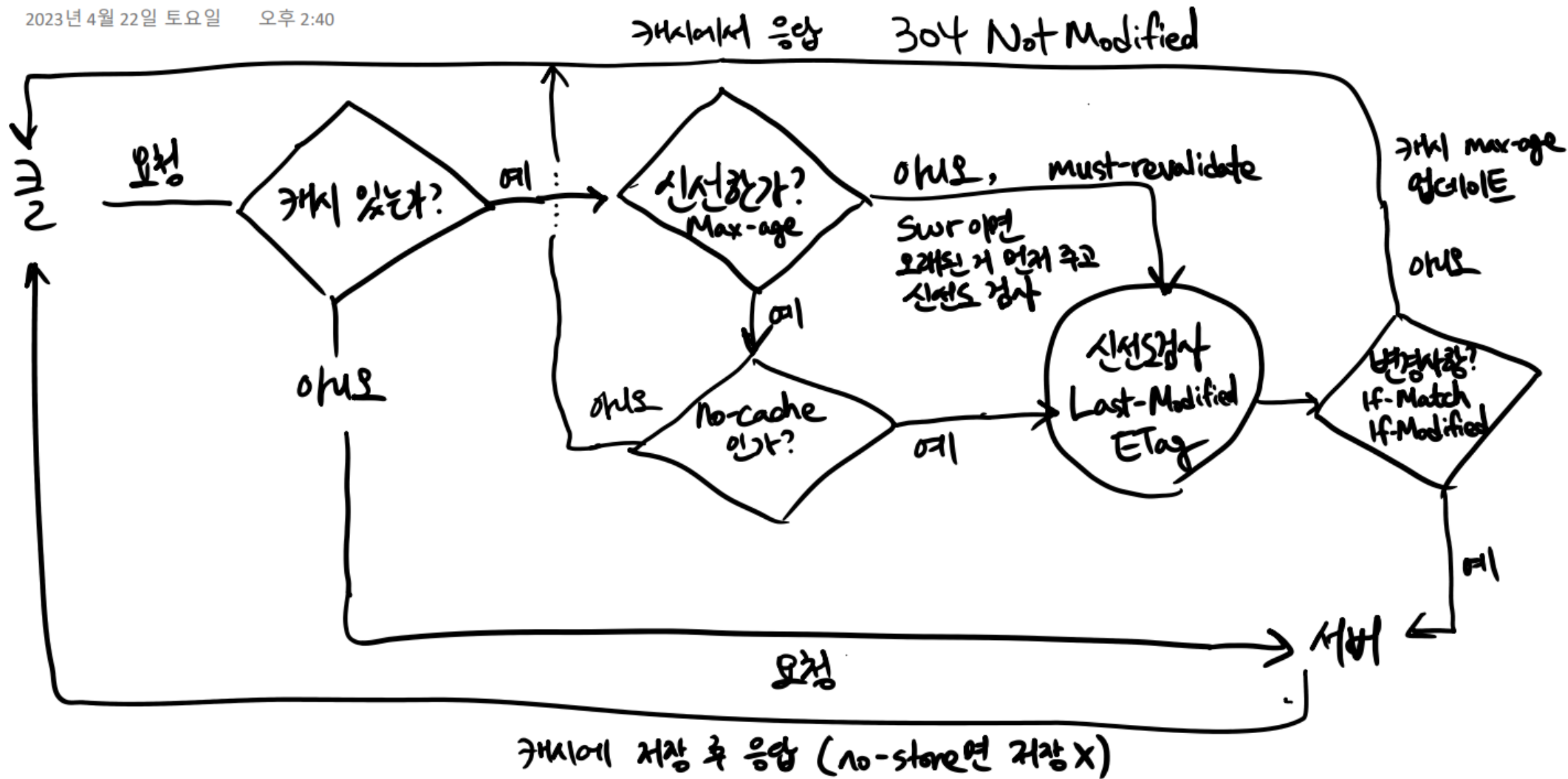
If-None-Match: "<etag_value>" => 요청을 보낼 때 etag가 다르면 서버로부터 수신 (같으면 304)

If-Modified-Since: "<Last-Modified-date>" => 요청을 보낼 때 date가 다르면 서버로부터 수신 (같으면 304)

If-Unmodified-Since: "<Last-Modified-date>" => 요청을 보낼 때 date가 같으면 서버로부터 수신 (다르면 412)

캐시

2023년 4월 22일 토요일 오후 2:40



서버 연결이 끊어진 경우 no-cache, must-revalidate는 504 응답

Origin(출처 또는 도메인)이 다르면 다른 Origin으로 요청을 보낼 수 없다. (CORS 에러 발생)
그러나 같은 도메인 또는 서브 도메인으로서는 요청 가능

Access-Control-Allow-Origin로 요청을 허용할 도메인을 지정할 수 있다. (이는 프론트가 아닌 서버가 작성하는 것이며 *이면 모두 허용)
Access-Control-Allow-Origin이 없어서 응답을 거부하는 것은 브라우저이며 CORS에러가 뜬다. (서버에서 서버로 요청을 보낼 때는 CORS에러가 뜨지 않는다)

브라우저 -> 다른origin서버 = CORS 에러 발생
브라우저 -> 같은origin서버 = CORS 에러 발생하지 않음
같은origin서버 -> 다른origin서버 = CORS 에러 발생하지 않음

브라우저에서 다른origin서버로 요청하기 위한 해결책은?



브라우저 -> 같은origin서버 -> 다른origin서버
만약 같은 origin서버가 없으면?
브라우저 -> proxy서버 -> 다른origin서버

Cross Origin Resource Sharing → 다른 오리진 사이트에 요청
(프론트, 프론트)

기본적으로 img, link, script 등의 태그는 다른 오리진 요청이 가능. fetch나 XMLHttpRequest 시 CORS 에러가 발생

Simple : GET, HEAD, POST 요청이면서, Content-Type이 x-www-form-urlencoded, multipart/form-data, text/plain 이고

Accept, Accept-Language, Content-Language, Range 와에는 헤더를 직접 설정하지 않은 경우
→ Access-Control-Allow-Origin : * 응답 헤더 필요

Preflighted : OPTIONS 요청이 먼저 감. 응답으로 가능한 메서드, 헤더, 오리진을 알릴

크레디탈도 보내고 싶다면 fetch에서는 credentials: include > fetch("https://facebook.com", { method: 'GET', credentials: 'include' })
응답에서는 Access-Control-Allow-Credentials: true (SameSite 옵션도 있음)

HTTPS

2023년 4월 18일 화요일 오후 9:46

HTTPS

클라이언트

브라우저

CA 공개키

공개키로 인증서 복호화
(CA가 미리 브라우저에게 공개키 제공)

인증서 안에 들어있던 것

서버 공개키

암호화

데이터 암호화용 키

암호화

HTTP 데이터

암호화된 데이터

HTTP 데이터

복호화

데이터 키

복호화

서버 비밀키

서버

Naver.com

CA

인증서 - CA 비밀키 암호화

Let's encrypt

CA 말고도
무료 인증기관 존재

암호화된 인증서

RSA

키 교환 키 만드는 쪽

DHE

HTTP3
TLS13

DHE는 특정 시간마다 키를 바꿔버리므로 해커로부터 방어가 가능
(RSA보다는 DHE 알고리즘이 추세)

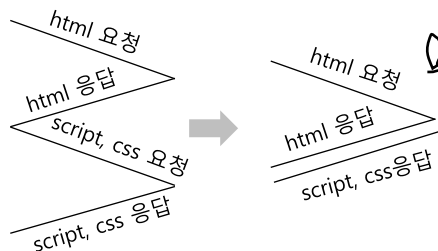
HTTP2, HTTP3

2023년 4월 22일 토요일 오후 3:55

HTTP/1.1은 Text 기반, 3way handshake 비효율, **Pipelining** 실패, 여러 커넥션 제한
HOL블로킹

HTTP/2 → **3개의 커넥션, 여러 스트림 (무선3위)** ⇒ 커넥션 한번만 맺어도 여러 개의 요청과 응답을 처리
헤더 압축, **서버 푸시** → 서버에서 코딩해줘야함

SPDY



HTTPS 필수 (암호화는 아니나 없이 사용 불가)

바이너리 (한 눈에 보기 어려움), 프레임

→ 텍스트 기반에서 하이퍼 미디어를 전송하기에 더 효율적인 바이너리 기반으로 바뀜

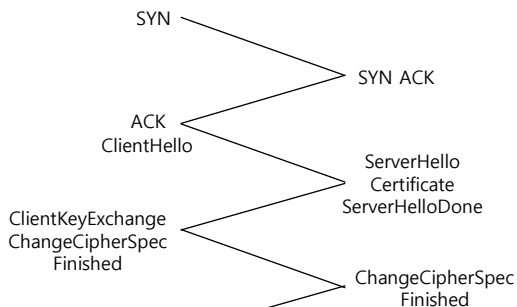
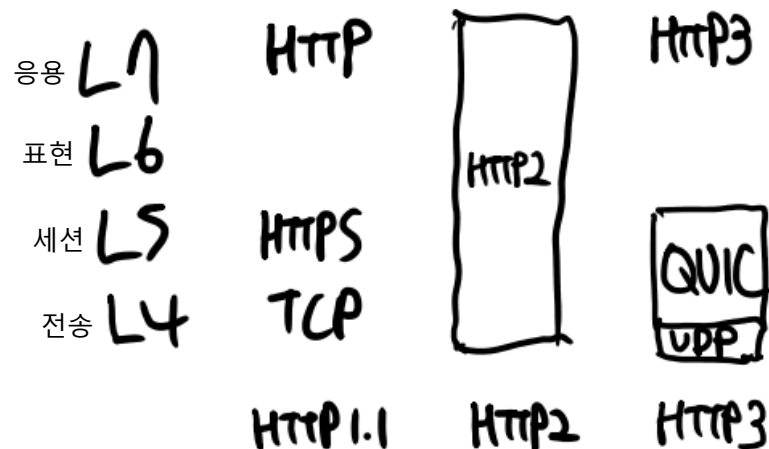
HTTP/3 → TCP의 비효율성을 UDP로 극복 (ex 패킷 손실 시 성능 저하)

QUIC (TLS1.3 따름)

동영상 서비스에서 속도가 바름

0 RTT (보안 위험 존재)

→ 리플레이 공격 기법에 취약하므로 1RTT



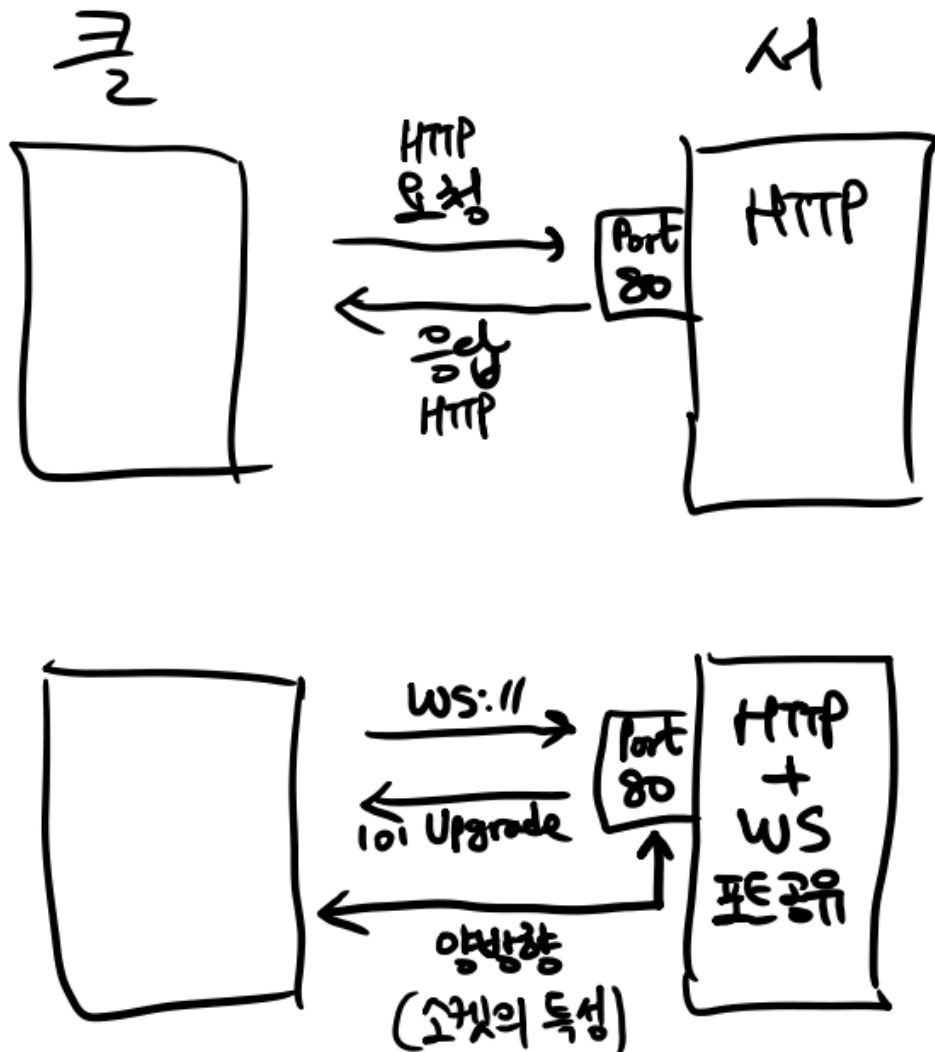
왕복이 3번이므로 3RTT

웹소켓

2023년 4월 22일 토요일 오후 3:56

웹소켓 프로토콜을 사용하는 이유

http에서는 항상 클라이언트에서 서버로 요청을 보낸 다음에 서버가 클라이언트로 응답을 보내야하는 구조였다.
그래서 서버에서 실시간으로 바뀌는 데이터를 클라이언트에게 보내려면 클라이언트는 매번 요청해서 물어봐야 하는(폴링방식) 불리한 점이 있기 때문에 소켓 프로토콜을 사용한다. 그러나 소켓 프로토콜도 처음 클라이언트가 서버랑 연결이 맺어져야 한다.



HTTPS 적용시
WSS:// 대신 사용
첫 요청시 쿠키 전송 가능

VPN

2023년 4월 20일 목요일 오후 9:09

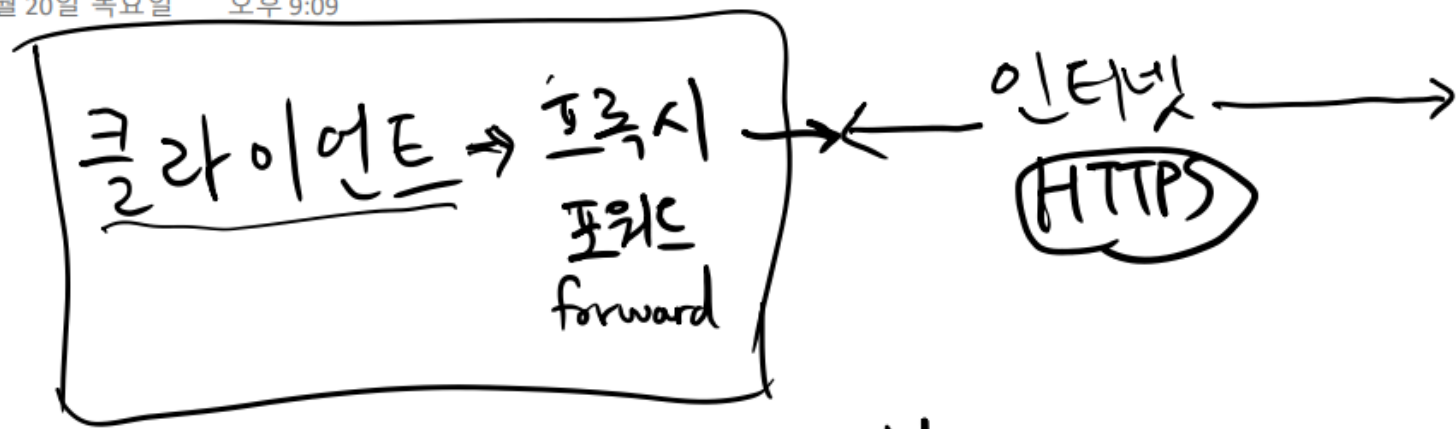
Virtual Private Network
가상

Private Network
사설망



Proxy, Gateway

2023년 4월 20일 목요일 오후 9:09



불필요한 처리 제거
필터링

→ 필터링
요청 분배 (로드밸런싱)

Apache, Nginx

로드밸런싱

요청 양에 따라서 진짜서버
개수를 조절 하는 것

Gateway 프로토콜 바꿈



API Gateway
프록시

CONNECT 메소드는 바꾸려 하지말고
서버한테 그대로 보내달라는 요청이다.
(그러나 프록시의 결정에 따라 달렸다)

HTTP Method

CONNECT
클 프 서버

정적 파일 서빙
Html, css, js
image, font

압축
gzip, br