Network Systems 내용 정리

# Index	4
CreatedAt	
Person	A Ally Hyeseong Kim
Status	In progress
Tags	Computer Science Network Systems
UpdatedAt	

References

[정보통신기술용어해설]

http://www.ktword.co.kr/index.php

Interview_Question_for_Beginner/Network at master · JaeYeopHan/Interview_Question_for_Beginner

:boy: :girl: Technical-Interview guidelines written for those who started studying programming. I wish you all the best. :space_invader: - Interview_Question_for_Beginner/Network at master · JaeYeo...

https://github.com/JaeYeopHan/Interview_Question_for_Beginner/tree/master/Network

Tech

tech_interview.zip/Network.md at main · 4z7l/tech_interview.zip

✅ 취준하면서 모았던 면접 질문 모음집 ✅. Contribute to 4z7l/tech_interview.zip development by creating an account on GitHub.

https://github.com/4z7l/tech_interview.zip/blob/main/직무/Network.md

4z7l/
tech_interview.zip

✅ 취준하면서 모았던 면접 질문 모음집 ✅

1 Contributor 1 Issue 1k Stars 132 Forks

https://github.com/AllyHyeseongKim/backend-interview-question

---

References
1. 전산 기본
   1.1. OSI 7계층
   1.2. TCP/IP 4계층
   1.3. DNS
   1.4. www.naver.com을 접속할 때 일어나는 일
   1.5. 도메인 이름으로 실제 IP를 찾는 흐름
   1.6. 유니캐스트, 멀티캐스트, 브로드캐스트
2. TCP/UDP
   2.1. TCP vs UDP
   2.2. TCP 헤더
   2.3. MTU
   2.4. 3-way handshake, 4-way handshake 흐름
3. HTTP
   3.1. HTTP 프로토콜
   3.2. HTTP vs HTTPS
   3.3. HTTPS 동작 방식
   3.4. HTTP Method
   3.5. HTTP 1.0 vs 1.1
   3.6. HTTP2
   3.7. HTTP 헤더
   3.8. keep-alive 헤더
   3.9. HTTP GET vs POST
   3.10. 쿠키 vs 세션
4. 웹

4.1. www.naver.com에 접속할 때
4.2. 웹브라우저에서 서버로 요청했을 때 흐름
4.3. CORS
4.4. 웹 서버 vs 웹 애플리케이션 서버(WAS)
4.5. REST API
4.6. RESTful
5. 기타
5.1. 네트워크 계층 라우팅 알고리즘
5.2. 로드밸런서, 로드밸런싱
5.3. 멀티플렉싱
5.4. ARP, RARP
5.5. TCP HOL 블로킹
5.6. polling, long polling, websocket

1. 전산 기본

▼ OSI 7계층: 컴퓨터 네트워크에서 통신이 이루어지는 과정을 7단계로 나눈 모델

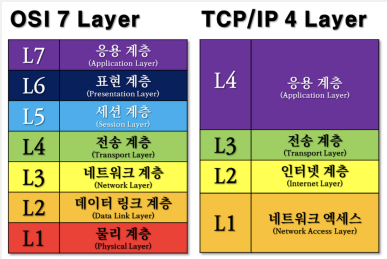
OSI 7 계층이란?, OSI 7 계층을 나눈 이유

1. OSI 7 계층이란? OSI 7 계층은 네트워크에서 통신이 일어나는 과정을 7단계로 나눈 것을 말한다. 1.1 OSI 7 계층을 나눈이유는? 계층을 나눈 이유는 통신이 일어나는 과정이 단계별로 파악할 수 있기 때문이다. 흐름을 한눈에 알아보기 쉽고, 사람들이 이해하기 쉽고,7단계 중 특정한 곳에 이상이 생기면

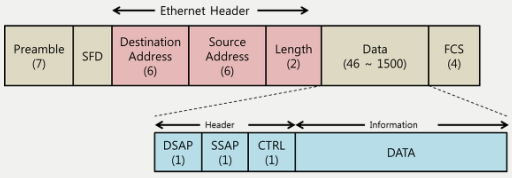
https://shlee0882.tistory.com/110

> *각 계층의 기능을 명확히 분리해 네트워크 구성 요소를 평준화시킴 → 특정한 계층에 이상이 생기면 특정 계층만 고치면 해결 가능*

> *패킷 전달, 라우팅, IP 주소 사용*



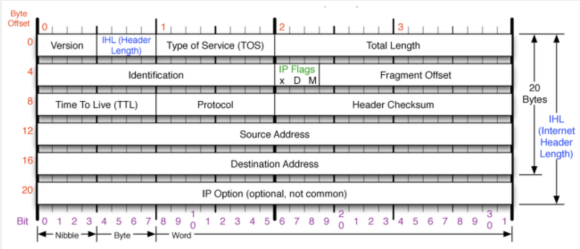
- 1계층(물리계층): 데이터를 전기신호로 변환해 전송
  - 물리적 케이블, **허브**(브로드캐스팅), 리피터(신호 증폭) 등
  - 전송 단위: 비트(0, 1)
  - 이더넷(전선), SONET(광섬유), Wi-Fi(무선)
- 2계층(데이터링크계층): 물리계층으로 송수신되는 데이터의 오류 및 흐름 제어
  - **L2 스위치**(VLAN, LACP), 브리지 → MAC Address Table (IP - MAC 주소) 사용
    - learning, flooding, forwarding, filtering, aging
  - 전송 단위: **프레임**



IEEE 802.3 Frame

    - Header: Destination, Source MAC Address

        - FCS: Frame **Check Sum** → CRC 알고리즘
    - 프로토콜: BMA (Local Area), NBMA (Wide Area → 브로드캐스팅 제한), Point-to-Point (중간에 다른 장비 없이 케이블이 end-to-end로 직접 연결 → 물리적 주소 필요X)
        - BMA 프로토콜: **이더넷**(MAC 주소를 물리적 주소로 사용), 토큰링, FDDI
            - MAC 주소(48비트) = 제조사(24비트) + 고유 식별자(24비트)
        - Point-to-Point 프로토콜: PPP, HDLC
    - MAC(매체 접근 제어): CSMA/CD (유선, 랜덤 시간 대기), CSMA/CA (무선, 일정시간 대기 후 전송 포기)
        - 현재는 스위치를 사용해 이더넷 환경에서도 충돌X
            - **주소 학습** → 특정 포트끼리 통신하므로 다른 포트에 영향X
            - **Buffer로 순차처리** → 먼저 전송되는 프레임이 목적지 포트로 모두 전송되는 동안 이후 프레임은 Buffer(임시저장소)에 저장해 순차처리
- 3계층(네트워크계층): 논리적 주소로 데이터를 목적지로 전송하기 위한 경로를 결정(**라우팅**)
    - L3 스위치(), **라우터**()
    - 전송 단위: **패킷**



IPv4 헤더

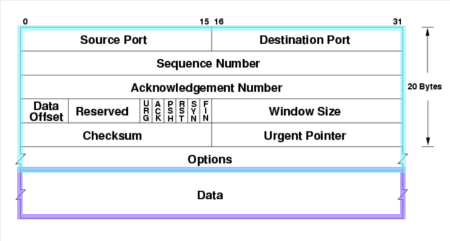
    - 논리적 주소 프로토콜: **IP 프로토콜**
        - IPv4(32비트): **Subnet Mask**로 Network ID, Host ID 구분
            - Subnet Mask: Net ID는 1로 표시, Host ID는 0으로 표시해서 10진수로 변환 → AND 연산으로 Net ID를 알 수 있음. Prefix로 1의 개수 표시

              ex. A Class: 11111111.00000000.00000000.00000000 → 255.0.0.0/8
            - 네트워크 대표주소(네트워크 안에서 첫 주소), 브로드캐스트 주소(네트워크 안에서 마지막 주소)
            - 공인 IP(라우터의 WAN 포트에 할당), 사설 IP: NAT(Network Address Translation)로 트래픽 변환(출발지 주소 → 공인 IP 주소로 변환)
            - IPv4 Class: A(Net ID 8비트), B(Net ID 16비트), C(Net ID 24비트), D(Multicast), E(연구용)

Class	IP 주소의 첫번째 옥텟	첫째 옥텟의 최소값 (2진수)	첫째 옥텟의 최대값 (2진수)	첫째 옥텟의 값의 범위 (10진수)	이론적 IP 주소 범위
A Class	0xxx xxxx	0000 0000	0111 1111	0 ~ 127	0.0.0.0 ~ 127.255.255.255
B Class	10xx xxxx	1000 0000	1011 1111	128 ~ 191	128.0.0.0 ~ 191.255.255.255
C Class	110x xxxx	1100 0000	1101 1111	192 ~ 223	192.0.0.0 ~ 223.255.255.255
D Class	1110 xxxx	1110 0000	1110 1111	224 ~ 239	224.0.0.0 ~ 239.255.255.255
E Class	1111 xxxx	1111 0000	1111 1111	240 ~ 255	240.0.0.0 ~ 255.255.255.255

    - IP - MAC 주소 변환 프로토콜: ARP, RARP
    - 라우팅 테이블 업데이트 프로토콜: RIP, OSPF, BGP
        - Local Area: RIP (30초마다 hop count 기준 벨만-포드 알고리즘), OSPF (link 변화 시 다익스트라 알고리즘을 사용하여 flooding)
        - Wide Area (Area vs Area): BGF

    - ICMP 프로토콜
- 4계층(전송계층): 통신을 활성화
    - L4 스위치
    - 전송 단위: 세그먼트



TCP 세그먼트

    - 프로토콜: TCP(신뢰성 → 웹, 이메일), UDP(효율성 → 스트리밍 서비스, 게임 서버)
- 5계층(세션계층): 데이터가 통신하기 위한 논리적 연결
- 6계층(표현계층): 데이터 표현에 대한 독립성을 제공, 암호화
- 7계층(애플리케이션계층): 최종 목적지, 응용프로그램과 연관해 서비스
    - L7 스위치

💡 **OSI 7계층**: 통신망, 프로토콜의 개발 등을 효율적으로 수행하기 위해 도입된 개념적 계층화 → 독립성(독립적 개발), 이해 용이(설계 용이)

▼ **1계층(물리)**: 기계적, 물리적 통신망 접면 정의

- 단위: 비트
- 프로토콜: Coax, Fiber, Wireless

▼ **2계층(데이터링크)**: 데이터 링크 제어(프레임화, 데이터 투명성, 오류 제어 등)

- 단위: 프레임
- 프로토콜: 이더넷, SLIP, PPP, FDDI

▼ **3계층(네트워크)**: 경로 배정, 주소 설정 등(라우팅)

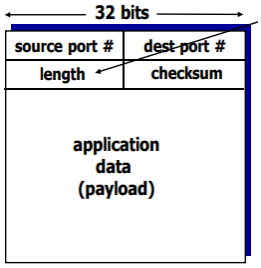
- 단위: 패킷
- 프로토콜: IP, IPSec, ICMP, IGMP

▼ **4계층(전송)**: 종단(프로세스) 간의 메세지 전송(호스트에 도착한 패킷을 프로세스로)

- 단위: 세그먼트(TCP), 데이터그램(UDP)
- **포트**(프로세스의 번호)를 사용해 프로세스 판단

PORT 번호	프로토콜
20, 21	FTP(20: 데이터, 21: 제어)
22	SSH
25	SMTP
53	DNS
80	HTTP
443	HTTPS over SSL

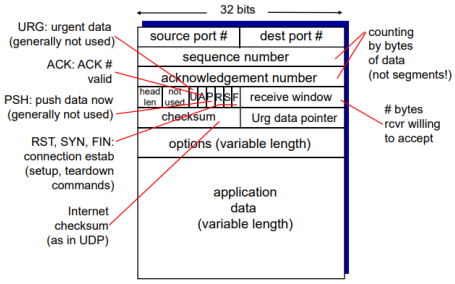
- **다중화**(애플리케이션 계층 메세지를 세그먼트 단위로 만들어 전송 계층 헤더 붙임), **역다중화**(네트워크 계층 데이터를 받아 세그먼트를 추출해 알맞은 **소켓**에 전송)
  - 소켓: IP 주소, 포트 번호를 가지는 엔드포인트
- 프로토콜: **TCP, UDP**
  - UDP: 비연결형 프로토콜, 신뢰성이 낮으나 속도가 빨라 스트리밍 서비스에 유리함(DNS, 음성통화, 스트리밍 서비스 등)



UDP segment format

UDP 세그먼트: 헤더(8byte, 포트번호+길이+checksum) + 데이터

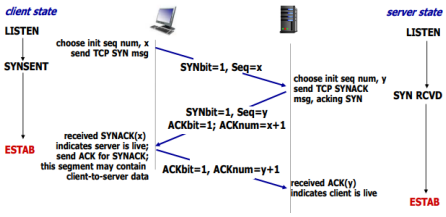
  - TCP: 연결형 프로토콜, 신뢰성이 높음(HTTP, SMTP, FTP 등), 순서 보장, 흐름제어 및 혼잡제어



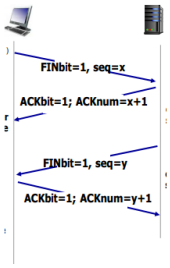
TCP 세그먼트: 헤더(20byte) + 데이터

헤더 필드	크기(bits)	역할
출발, 도착 포트	16	포트 번호
sequence number	32	송신 데이터 순서 번호
ACK number	32	데이터 전송 보장. 데이터 번호 응답 값
Header len	4	TCP 헤더 길이
not used	3~6	미래 통신을 위해 남김
URG	1	Urgent point 값의 유효성
ACK	1	ACK 값의 유효성
PSH	1	버퍼링된 데이터를 상위계층에 즉시 전달 요청
PST	1	연결 초기화
SYN	1	동기화. 양측 최초 패킷에만 부여
FIN	1	전송 끝
window size	16	송수신 흐름제어. 수신자가 수용 가능한 버퍼 크기
checksum	16	오류체크(가상헤더+실제헤더+데이터+패딩)
Urgent Pointer	16	URG 값이 있을 때 요청 부분까지 긴급한 데이터 처리를 위한 일련번호
Options	0~320	다양한 TCP 옵션

- **3-way**(연결), **4-way**(종료) **handshake** 통한 연결



3-way handshake



4-way handshake

- **흐름제어**(속도): 수신버퍼에 저장된 데이터 양을 체크 → **흐름 제어 신호** → 수신 측의 버퍼 오버플로우, 데이터 유실방지
  - **Stop-and-Wait**: 송신 측 데이터 전송 후 수신 측으로부터 ACK(응답) 신호를 기다리면 대기 → ACK 수신 후 다음 데이터 전송

        - 전송 지연이 발생할 수 있음(비효율적)
    - **Sliding Window**: 송신 측 일정한 크기의 window size(수신 측의 버퍼 크기)의 여러 개의 패킷 전송 → 수신 측 패킷을 받으면 ACK(TCP 헤더 window size) 송신 측에 보냄 → ACK만큼 window 이동 후 다음 패킷 전송
        - 구현 방법: Go-Back-N(손상된 패킷부터 재전송 → 데이터가 손실될 확률 낮은 통신망), Selective Repeat(손상된 패킷만 재전송 → 데이터 손실될 확률 높은 무선 통신 환경)
        - TCP는 혼합형(selective ACK) 구현: 패킷 전송(순차적으로 일련번호 부여) → 패킷 손상시 selective ACK 패킷(손상된 데이터의 시퀀스 번호, checksum) 전송
  - **혼잡제어**: 송신측에서 보내는 데이터가 라우터가 처리할 수 있는 양을 초과하지 않도록 제어함
    - **AIMD(합 증가/곱 감소)**: 처음에 패킷을 하나씩 보내고 문제 없이 도착하면 윈도우 크기를 1씩 증가시켜 전송 → 전송 실패 시 윈도우 크기 반으로 줄임
        - 윈도우 크기를 조금씩 늘리므로(선형적) 모든 대역을 활용하여 통신하기까지 시간이 오래 걸릴 수 있음
    - **Slow Start(느린 시작)**: 윈도우 크기를 지수적으로 증가시키다가 혼잡이 감지되면 윈도우 크기를 1로 줄임
    - **Fast Retransmit(빠른 재전송)**: 수신측에서 순서대로 잘 도착한 마지막 패킷의 다음 번호를 TCP 헤더(ACK)에 실어서 보내는데, 중복 ACK가 3번 발생하면 재전송함
        - 설정한 타임 아웃 시간이 지나지 않아도 바로 재전송하므로 빠른 재전송률 유지
    - **Fast Recovery(빠른 회복)**: 혼잡 상태 시 윈도우 크기를 반으로 줄이고 선형 증가시킴(혼잡 후 **AIMD** 방식)
    - 정책: **TCP Tahoe**(Slow Start → AIMD, 중복 ACK 혹은 타임아웃 시 Slow Start 임계점 수정 및 윈도우 크기 1), **TCP Reno**(Slow Start → AIMD, 중복 ACK 시 Fast Recovery 및 윈도우 크기로 임계점 수정, 타임아웃 시 동일)

▼ **5계층(세션)**: 통신하기 위한 논리적 연결

- 단위: 데이터
- **세션**(일정 시간 동안 같은 사용자로부터 들어오는 요청들을 하나의 상태로 유지함)을 관리함
    - 세션 설정, 유지, 종료, 복구
- 프로토콜: **SSH**
    - SSH: 다른 컴퓨터에 원격 로그인(클라이언트/서버 모델)
        - 클라이언트가 서버의 TCP 22번 포트로 SSH 접속 요청 → 서버가 클라이언트에게 지원하는 프로토콜 버전 응답 → 클라이언트와 일치하는 것이 있다면 연결 지속
        - 서버가 공개키를 클라이언트에 전송 → 클라이언트가 서버에게 받은 공개키를 로컬에 저장 (`home/.ssh/knwon_hosts`)
        - 올바른 서버인지 확인() → 세션키(대칭키) 생성

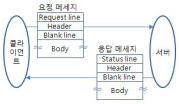
▼ **6계층(표현)**: 데이터 표현 방식 변환 및 암호화

- 단위: 메세지
- 프로토콜: JPEG, MPEG, ASCII, **SSL**

▼ **7계층(응용)**: 응용 계층 내 프로세스 간의 통신(파일 전송・접근・관리, 문서・메세지 교환 등)

- 단위: 메세지
- 하위 표현 계층이 제공하는 서비스를 이용해 응용 계층 프로세스들 사이의 통신
- 통신 구조: **클라이언트/서버 구조**(웹, 이메일 등), **P2P 구조**(블루투스 등)
- 프로토콜: **HTTP**, FTP, IRC, SSH, DNS, SMTP, CDN
    - HTTP: 웹 상에서 웹 서버, 웹 브라우저 간의 데이터 전송

- **클라이언트/서버 모델**로 동작



- **HTTP 메세지** 교환 형태의 프로토콜 → SMTP 프로토콜과 유사

💡 **TCP/IP 4계층**: 캡슐화 - 역캡슐화

- TCP/IP: 두 가지 프로토콜(전송 조절, 인터넷) 방식을 조합해 인터넷 통신

▼ **1계층(네트워크 연결)**: 물리 + 데이터링크 계층

- 데이터 단위: 프레임
- 물리적으로 데이터가 어떻게 전송되는지 정의(물리적 주소로 MAC 주소 사용)
- 프로토콜: 이더넷, Token Ring
- ex. LAN, 패킷망 등

▼ **2계층(인터넷)**: 네트워크 계층

- 데이터 단위: 패킷
- 네트워크 최종 목적지까지 정확히 연결되도록 라우팅(논리적 주소로 IP 사용)
- 프로토콜: IP, ARP, RARP

▼ **3계층(전송)**: 전송 계층

- 데이터 단위: 세그먼트
- 통신 노드 간의 연결을 제어하고, 신뢰성있는 데이터 전송을 담당함
- 프로토콜: TCP, UDP

▼ **4계층(응용)**: 세션 + 표현 + 응용 계층

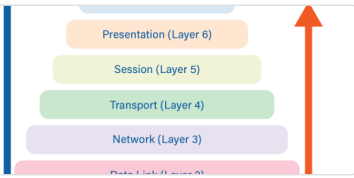
- 데이터 단위: 데이터
- 응용 프로그램 간의 통신
- 프로토콜: FTP, HTTP, SSH, DNS 등

1.1. OSI 7계층

[네트워크/기본] 프로토콜(Protocol)이란? - OSI 7 계층 별 프로토콜 - 하나몬

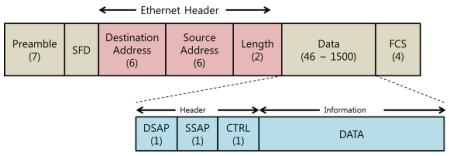
❗프로토콜(Protocol)이란? 프로토콜은 데이터를 송수신하기 위한 규칙을 말한다. 손님이 주문을 받는 사람에게 대뜸 찾아가, 외계어로 주문을 할 수 없다. 주문을 하기 위해서는 꼭 지켜야 하는 약속이 몇가지 존재한다. 통신하기 위한 다양한 방법이 존재할 수 있다. 직원에게 주문, 앱에서 주문, 키오스크 주문

https://hanamon.kr/네트워크-기본-프로토콜이란-osi-7-계층-별-프로토콜/



- 1계층(물리계층): 데이터를 전기 신호로 전송
  - 물리적 케이블, 허브(브로드캐스팅), 리피터(신호 증폭)
  - 전송 단위: 비트(0, 1)
- 2계층(데이터링크계층): 물리 계층으로 송수신되는 데이터의 오류 및 흐름제어
  - 스위치(MAC 주소 테이블에 목적지 있으면 해당 포트로 전송, 없으면 모든 포트로 전송, 300초 동안 사용하지 않으면 삭제)

- 전송단위: 프레임



  - 헤더: 목적지 MAC주소, 출발지 MAC주소, 길이
  - FCS: CRC 알고리즘 checksum
- 프로토콜: **이더넷, MAC**(매체 접근 제어)
  - 이더넷 → 현재는 스위치를 사용하므로 이더넷 환경에서 충돌이 발생하지 않음
    - 주소 학습 → 특정 포트끼리 통신하므로 다른 포트에 영향을 주지 않음
    - Buffer로 순차처리 → 먼저 전송되는 프레임이 목적지 포트로 모두 전송되는 동안 이후 프레임은 버퍼에 저장해 순차처리
  - 유선 LAN의 MAC: CSMA/CD(랜덤 시간 대기), Token Ring
  - 무선 LAN의 MAC: CSMA/CA(일정 시간 대기 후 전송 포기)

- 3계층(네트워크계층): 데이터를 목적지로 전송하기 위한 경로 결정(라우팅)
  - 라우터(라우팅)
  - 전송 단위: 패킷

  - 프로토콜
    - 논리 주소: IP 프로토콜
    - IP-MAC 주소 변환: ARP(IP → MAC 주소), RARP(MAC → IP 주소)
    - 라우팅 테이블 업데이트: RIP, OSPF
- 4계층(전송계층): 종단(프로세스) 간의 메세지 전송(호스트에 도착한 패킷을 프로세스로 전송)
- 5계층(세션계층): 통신을 위한 논리적 연결
- 6계층(표현계층): 데이터 표현 방식 변환 및 암호화
- 7계층(애플리케이션계층): 최종 목적지, 응용 프로그램과 연관해 서비스

1.2. TCP/IP 4계층

- 1계층(네트워크 연결 계층): 물리 + 데이터 링크 계층
  - 단위: 프레임
  - 물리적으로 데이터가 어떻게 전송되는지 정의
  - 물리주소로 MAC 주소 사용
  - 프로토콜: 이더넷, Token Ring
- 2계층(인터넷 계층): 네트워크 계층
  - 단위: 패킷
  - 네트워크 최종 목적지까지 정확히 연결되도록 라우팅
  - 논리주소로 IP 주소 사용

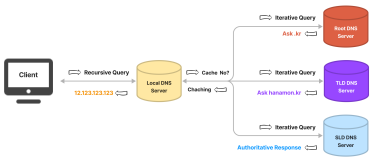
    - 프로토콜: IP, ARP, RARP
- 3계층(전송 계층): 전송 계층
    - 단위: 세그먼트
    - 통신 노드 간의 연결을 제어하고, 신뢰성있는 데이터 전송을 담당
    - 프로토콜: TCP, IP
- 4계층(응용 계층): 응용 계층
    - 단위: 데이터
    - 응용프로그램 간의 통신
    - 프로토콜: HTTP, SSH, FTP, DNS 등

1.3. DNS

DNS란? (도메인 네임 시스템 개념부터 작동 방식까지) - 하나몬

이 게시물의 중요 포인트 DNS(도메인 네임 시스템)이 사람이 읽을 수 있는 도메인 이름 (www.hanamon.kr)을 IP 주소로 변환하는 시스템이라는 것은 쉽게 알 수 있습니다. 이번 글에서는 이렇게 도메인 주소를 IP 주소로 변환할 수 있게 하는 인프라가 어떻게 구성 되어있고 실제로 어떤 기계들

https://hanamon.kr/dns란-도메인-네임-시스템-개념부터-작동-방식까지/

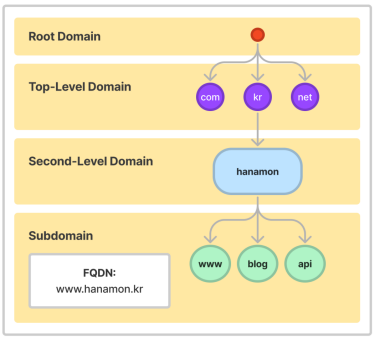


가비아에서 구매한 도메인을 AWS Route53에서 호스팅하기 – Yunseok's Dev Blog

가비아에서 구매한 도메인을 AWS Route53에서 호스팅하기

https://hannut91.github.io/blogs/route53/gabia

- **Domain Name System**: 도메인 주소(www.naver.com)를 IP 주소로 변환하는데 사용되는 시스템
- 구성 요소
    - **Domain Name Space**: DNS가 저장하고 관리하는 계층적 구조(폴더 구조와 비슷함)

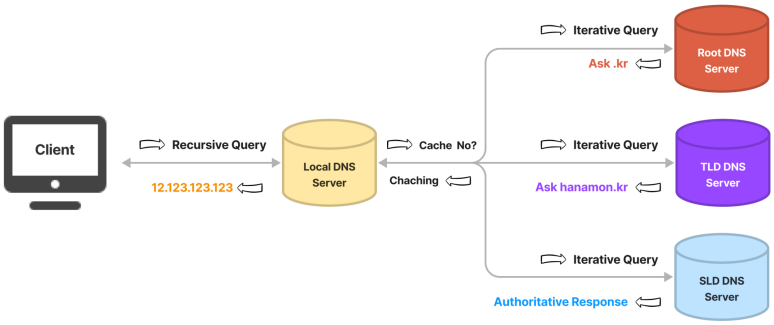


        - **계층적 도메인 레벨**: 도메인 네임 스페이스의 트리구조는 최상위 레벨부터 순차적으로 계층적 소속 관계를 나타냄

          ex. `naver.com` 도메인은 `com` 도메인을 관리하는 네임 서버에 등록

          ex. `www.naver.com` 도메인은 `naver.com` 도메인을 관리하는 네임 서버에 등록(`www`는 `naver.com` 도메인을 관리하는 네임 서버의 **레코드**)
            - 각 하위 기관의 관리 책임자에게 위임되는 구조 → 호스트 증가에 대한 관리가 효율적으로 이루어짐
            - DNS 레코드 종류: SOA, A(IPv4 주소), AAAA(IPv6 주소), CNAME(도메인), MX, SRV, NS(도메인 서버)
        - FQDN(Fully Qualified Domain Name, 전체 도메인 이름)
            - 도메인 이름: `naver.com`
            - 호스트 이름: `www`
            - FQDN: `www.naver.com`
    - **Name Server**(권한 있는 DNS 서버): Domain Name Space의 트리구조에 대한 정보를 가진 서버

        - Root DNS 서버: 전 세계에 13개가 구축되어 있음. Top-Level Domain DNS 서버의 IP 주소 저장/안내
        - Top-Level Domain(TLD) DNS 서버: 도메인 등록 기관(.com 등). Second-Level 서버의 주소 저장/안내
        - Second-Level Domain(SLD) DNS 서버: 실제 도메인 - IP 주소의 관계가 기록되는 서버
            - ex. 가비아 SLD - Route 53 Hosted zone (SLD) 매핑
    - **Resolver**(권한 없는 DNS 서버): 클라이언트(브라우저)의 요청을 네임서버로 전달하고 정보(도메인 주소, IP 주소)를 받아 클라이언트에게 제공
- **동작 과정**



Local DNS Server: ISP

1. 브라우저에 www.naver.com 입력
2. 캐시 확인
    a. 브라우저 캐시
    b. OS 캐시
    c. 라우터 캐시
    d. ISP(Local DNS Server) 캐시
3. ISP에서 DNS Iterative하게 쿼리를 날림: Root DNS → TLD DNS → SLD DNS
4. SLD DNS 서버에서 IP 주소를 응답받음
5. ISP는 IP 주소를 캐시하고 브라우저에 응답

1.4. www.naver.com을 접속할 때 일어나는 일

1. 브라우저에 도메인 주소 www.naver.com을 입력
2. URL 주소 중 도메인 주소의 IP 주소 탐색(DNS Lookup)
    a. 캐시 확인(브라우저 → OS → 라우터 → ISP)
    b. ISP에서 Iterative Query(Root → TLD → SLD DNS 서버) 날림
    c. ISP에 IP 주소 캐시하고 반환
3. HTTP 프로토콜을 사용해 HTTP request 메세지 생성
4. 프로토콜 스택에 의해 패킷에 요청 메세지를 담고
5. TCP 프로토콜을 사용해 해당 IP 주소의 서버로 메세지 전송
6. 서버에서 응답 데이터를 HTTP 프로토콜을 사용해 HTTP response 메세지 생성
7. TCP 프로토콜을 사용해 클라이언트로 메세지 전송
8. 클라이언트에서 웹 페이지 데이터로 변환해 브라우저에 출력

- HTTP 버전에 따라 TCP 프로토콜 연결 주기 다름

1.4.1. OSI 7계층과 연관

웹 통신의 큰 흐름

📍웹 통신의 큰 흐름 브라우저가 URL에 적힌 값을 파싱해서 HTTP request Message를 만들고, OS가 TCP 프로토콜을 사용하여 인터넷을 거쳐 해당 IP 주소의 웹 서버로 전송하도록 한다. 이 때, DNS 서버에서 해당 도메인 네임에 해당하는 IP 주소를 찾는 DNS Lookup을 수행합니다. DNS 룩업 과정은 크롬의

https://carnival.tistory.com/53

▼ Spring MVC

1. naver.com → DNS lookup(브라우저 → OS → 라우터 → ISP 캐시 → Iterative Query → 캐싱) → IP 가져옴
2. 리소스 GET Method를 가져와야 함 → Request message를 위한 header 구성, contents-type, version
3. 암호화 헤더
4. 웹서버 TCP 80번 포트 설정
5. IP로 패킷 헤더 구성
6. 라우터의 MAC 주소, 컴퓨터 MAC + CRC checksum
7. 전선 물리적 신호
8. 라우팅 테이블 → 라우팅
9. naver.com 서버 도착 → 1 ~ 7 계층
10. 3계층에서 IP에 따라
11. 4계층 80번 포트
12. 5-6 암호 해독
13. 7계층에서 HTTP

1. 브라우저를 실행하고 주소창에 www.naver.com 을 입력
   a. 7계층(응용) 사용자의 요청(www.naver.com)이 생성됨
2. 브라우저는 DNS Lookup을 통해 도메인의 IP 주소를 찾음

1.5. 도메인 이름으로 실제 IP를 찾는 흐름

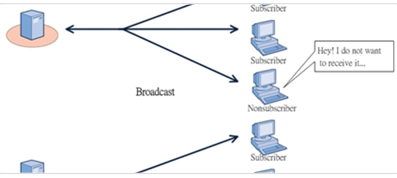
- **DNS Lookup**
  1. 캐시 확인
     a. 브라우저
     b. OS
     c. 라우터
     d. ISP
  2. Local DNS 서버(ISP)에서 Iterative Query로 IP 주소를 찾음
     a. Root DNS 서버 → .com 서버(TLD Domain DNS) 주소
     b. TLD Domain DNS 서버 → naver.com(SLD Domain DNS) 주소
     c. SLD Domain DNS 서버 → www.naver.com의 IP 주소
  3. Local DNS 서버에 캐싱 → 브라우저로 도메인 주소와 IP 주소 반환

1.6. 유니캐스트, 멀티캐스트, 브로드캐스트

유니캐스트, 멀티캐스트, 브로드캐스트

- 유니캐스트(Unicast) 유니캐스트는 정보를 전송하기 위한 프레임에 자신의 MAC 주소와 목적지의 M...

https://m.blog.naver.com/wnrjsxo/221250742423



- **유니캐스트**(1:1): 프레임에 Source MAC 주소, Destination MAC 주소를 붙여 전송하는 방식
  - 같은 네트워크에 있는 시스템은 Destination MAC 주소와 자신의 MAC 주소를 비교해 같을 경우 처리함

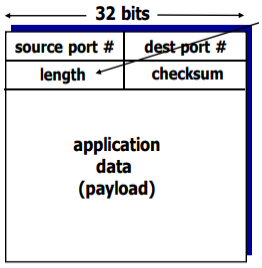
- **브로드캐스트**(1:불특정 다수): 로컬 네트워크에 연결되어 있는 모든 시스템에 프레임을 전송하는 방식 (네트워크 안에서 마지막 주소)
  - 브로드캐스트 주소가 오면 CPU가 패킷을 처리함
  - 통신하고자 하는 시스템의 MAC 주소를 모를 경우, 네트워크에 있는 모든 시스템에 통신하는 경우, 라우터끼리 정보를 교환하거나 새로운 라우터를 찾는 경우
- **멀티캐스트**(1:특정집단):

---

2. TCP/UDP

2.1. TCP vs UDP

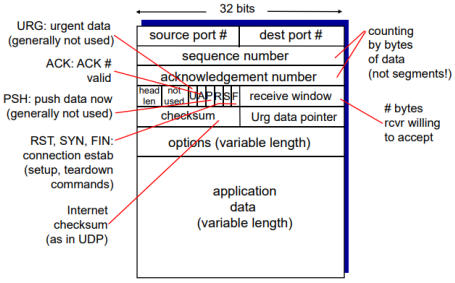
- OSI 7계층(**전송계층**)에서 사용하는 프로토콜, **포트**를 기반으로 **프로세스** 사이를 연결하는 프로토콜
- **UDP**: 비연결형 프로토콜, 신뢰성이 낮으나 속도가 빨라 스트리밍 서비스에 유리함(DNS, 음성통화, 스트리밍 서비스 등)



UDP segment format

UDP 세그먼트: 헤더(8byte, 포트번호+길이+checksum) + 데이터

- **TCP**: 연결형 프로토콜, 신뢰성이 높음(HTTP 등), 순서 보장, 흐름제어 및 혼잡제어

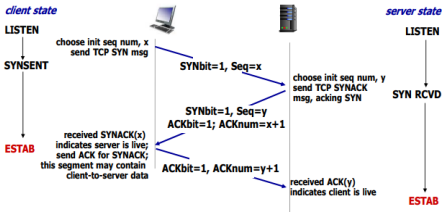


TCP 세그먼트: 헤더(20byte) + 데이터

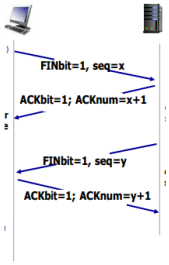
헤더 필드	크기(bits)	역할
출발, 도착 포트	16	포트 번호
sequence number	32	송신 데이터 순서 번호
ACK number	32	데이터 전송 보장. 데이터 번호 응답 값
Header len	4	TCP 헤더 길이
not used	3~6	미래 통신을 위해 남김
URG	1	Urgent point 값의 유효성
ACK	1	ACK 값의 유효성
PSH	1	버퍼링된 데이터를 상위계층에 즉시 전달 요청
PST	1	연결 초기화
SYN	1	동기화. 양측 최초 패킷에만 부여
FIN	1	전송 끝
window size	16	송수신 흐름제어. 수신자가 수용 가능한 버퍼 크기
checksum	16	오류체크(가상헤더+실제헤더+데이터+패딩)

Urgent Pointer	16	URG 값이 있을 때 요청 부분까지 긴급한 데이터 처리를 위한 일련번호
Options	0~320	다양한 TCP 옵션

- **3-way**(연결), **4-way**(종료) **handshake** 통한 연결



3-way handshake

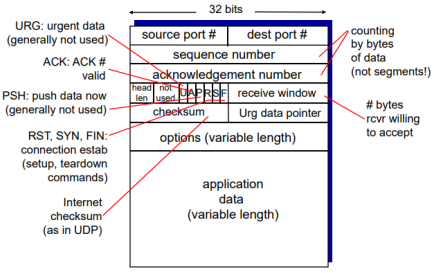


4-way handshake

- **흐름제어**(속도): 수신버퍼에 저장된 데이터 양을 체크 → **흐름 제어 신호** → 수신 측의 버퍼 오버플로우, 데이터 유실방지
  - **Stop-and-Wait**: 송신 측 데이터 전송 후 수신 측으로부터 ACK(응답) 신호를 기다리면 대기 → ACK 수신 후 다음 데이터 전송
    - 전송 지연이 발생할 수 있음(비효율적)
  - **Sliding Window**: 송신 측 일정한 크기의 window size(수신 측의 버퍼 크기)의 여러 개의 패킷 전송 → 수신 측 패킷을 받으면 ACK(TCP 헤더 window size) 송신 측에 보냄 → ACK만큼 window 이동 후 다음 패킷 전송
    - 구현 방법: Go-Back-N(손상된 패킷부터 재전송 → 데이터가 손실될 확률 낮은 통신망), Selective Repeat(손상된 패킷만 재전송 → 데이터 손실될 확률 높은 무선 통신 환경)
    - TCP는 혼합형(selective ACK) 구현: 패킷 전송(순차적으로 일련번호 부여) → 패킷 손상시 selective ACK 패킷(손상된 데이터의 시퀀스 번호, checksum) 전송
- **혼잡제어**: 송신측에서 보내는 데이터가 라우터가 처리할 수 있는 양을 초과하지 않도록 제어함
  - **AIMD(합 증가/곱 감소)**: 처음에 패킷을 하나씩 보내고 문제 없이 도착하면 윈도우 크기를 1씩 증가시켜 전송 → 전송 실패 시 윈도우 크기 반으로 줄임
    - 윈도우 크기를 조금씩 늘리므로(선형적) 모든 대역을 활용하여 통신하기까지 시간이 오래 걸릴 수 있음
  - **Slow Start(느린 시작)**: 윈도우 크기를 지수적으로 증가시키다가 혼잡이 감지되면 윈도우 크기를 1로 줄임
  - **Fast Retransmit(빠른 재전송)**: 수신측에서 순서대로 잘 도착한 마지막 패킷의 다음 번호를 TCP 헤더(ACK)에 실어서 보내는데, 중복 ACK가 3번 발생하면 재전송함
    - 설정한 타임 아웃 시간이 지나지 않아도 바로 재전송하므로 빠른 재전송률 유지
  - **Fast Recovery(빠른 회복)**: 혼잡 상태 시 윈도우 크기를 반으로 줄이고 선형 증가시킴(혼잡 후 **AIMD** 방식)
  - 정책: **TCP Tahoe**(Slow Start → AIMD, 중복 ACK 혹은 타임아웃 시 Slow Start 임계점 수정 및 윈도우 크기 1), **TCP Reno**(Slow Start → AIMD, 중복 ACK 시 Fast Recovery 및 윈도우 크기로 임계점 수정, 타임아웃 시 동일)

2.2. TCP 헤더

헤더 필드	크기(bits)	역할



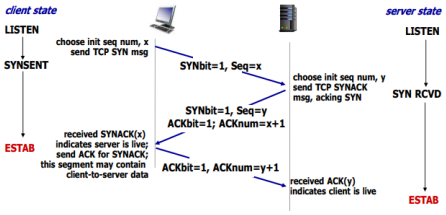
TCP 세그먼트: 헤더(20byte) + 데이터

출발, 도착 포트	16	포트 번호
sequence number	32	송신 데이터 순서 번호
ACK number	32	데이터 전송 보장. 데이터 번호 응답 값
Header len	4	TCP 헤더 길이
not used	3~6	미래 통신을 위해 남김
URG	1	Urgent point 값의 유효성
ACK	1	ACK 값의 유효성
PSH	1	버퍼링된 데이터를 상위계층에 즉시 전달 요청
PST	1	연결 초기화
SYN	1	동기화. 양측 최초 패킷에만 부여
FIN	1	전송 끝
window size	16	송수신 흐름제어. 수신자가 수용 가능한 버퍼 크기
checksum	16	오류체크(가상헤더+실제헤더+데이터+패딩)
Urgent Pointer	16	URG 값이 있을 때 요청 부분까지 긴급한 데이터 처리를 위한 일련번호
Options	0~320	다양한 TCP 옵션

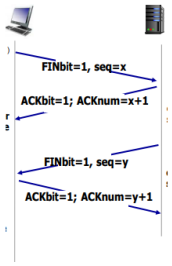
2.3. MTU

- Maximum Transmission Unit: 패킷, 프레임의 최대 크기

2.4. 3-way handshake, 4-way handshake 흐름



3-way handshake



4-way handshake

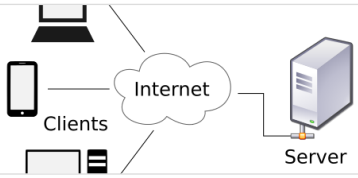
3. HTTP

3.1. HTTP 프로토콜

[네트워크/HTTP] HTTP와 클라이언트-서버 아키텍처 - 하나몬

클라이언트와 서버 간의 통신을 알아보려면, 먼저 프로토콜(Protocol)이라는 개념을 이해해야 한다.
❗웹 애플리케이션 프로토콜 : HTTP 웹 애플리케이션 아키텍처에서는 클라이언트와 서버가 서로 HTTP 라는 프로토콜을 이용해 소통한다. HTTP를 이용해 주고받는 메시지를 HTTP 메시지라고 부른다. ❗

https://hanamon.kr/네트워크-http와-클라이언트-서버-아키텍처/



[네트워크/HTTP] HTTP란? - 특징(무상태와 비 연결성) - 하나몬

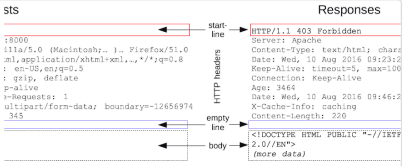
⚡ HTTP 개념, 역사, 특징 간략히 알아보기 ❗HTTP란? HTTP는 하이퍼텍스트 전송 프로토콜(HyperText Transfer Protocol)이다. HTTP는 1989년부터 1991년에 발명되었다. HTTP는 월드 와이드 웹(WWW)에 내재된 프로토콜이다. HTTP는 인터넷에서 데이터를 주고 받을 수 있는 프로토콜이다.

https://hanamon.kr/네트워크-http-http란-특징-무상태-비연결성/

[네트워크/HTTP] HTTP 메세지(message) - 요청과 응답 구조 - 하나몬

목차 HTTP란? HTTP messages란? HTTP 요청(Requests)과 응답(Responses) messages의 구조 HTTP 요청(Requests) messages HTTP 응답(Responses) messages ❗HTTP란? HTTP는 HyperText Transfer Protocol(하이퍼텍스트 트렌스퍼 프로토콜)의 줄임말이다. HTTP는 HTML과 같

https://hanamon.kr/네트워크-http-메세지-message-요청과-응답-구조/



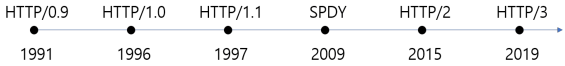
HTTP/1 to HTTP/2 to HTTP/3

Subscribe to our weekly system design newsletter: https://bit.ly/3tfAlYD

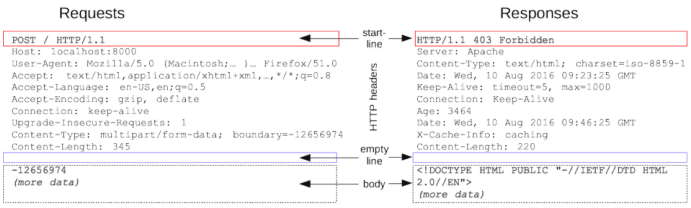
Checkout our bestselling System Design Interview books:

https://youtube.com/watch?v=a-sBfyiXysI&feature=share

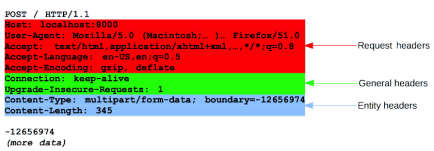
- OSI 7계층 중 **7계층(응용계층)**의 프로토콜, 웹 애플리케이션에서 클라이언트와 서버가 HTTP 프로토콜(하이퍼텍스트 전송 프로토콜)을 이용해 HTTP 메세지로 소통
- **클라이언트/서버 모델**(클라이언트가 요청, 서버가 응답)로 동작
- 클라이언트는 HTTP URL로 서버에게 자원을 명시하는데 이 URL에 대한 디자인이 필요함 → **REST API**



- **HTTP/0.9**(GET 메서드만 지원, 헤더 없음) → **HTTP/1.0**(메서드, 헤더 추가 → HTML 외 다른 파일 전송 가능) → **HTTP/1.1**(현재 주로 사용) → **HTTP/2** → **HTTP/3**(UDP)
- 특징
  - **클라이언트/서버 구조**
  - **stateless**(서버가 클라이언트의 상태를 보존하지 않음 → 서버가 바뀌거나 증설 가능)
    - 상태가 필요한 경우: 로그인
  - **connectionless**(HTTP/1.0)
    - 비연결성 → TCP 연결을 새로 맺기 위한 3-way handshake 시간 추가
    - 많은 자원이 필요한데 그때마다 재연결은 비효율적 → **Persistent Connection(HTTP/1.1)**으로 문제해결
- **HTTP 메세지**

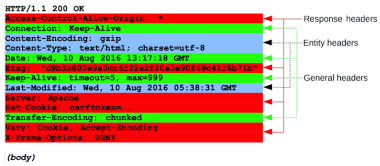


- request 메세지
  - **start line**: HTTP method, 요청 대상(URL), HTTP 버전
  - **header**: request, general, entity header



  - **body**: POST, PUT같은 요청에 body를 사용함

- response 메세지
  - **status line**: HTTP 버전, 상태코드, 상태 텍스트
  - **header**: response, general, entity



  - **body**

3.2. HTTP vs HTTPS

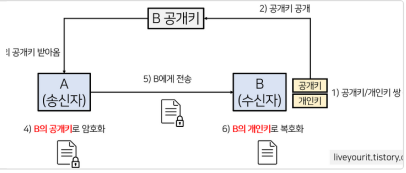
HTTPS와 SSL 인증서 - 생활코딩

https://www.opentutorials.org/course/228/4894

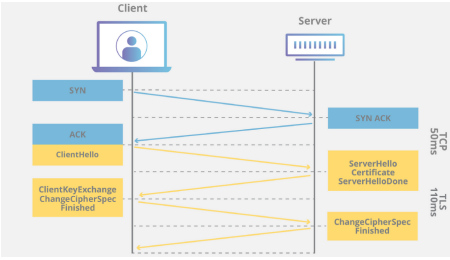
[네트워크] TLS & SSL Handshake

cs-study에서 스터디를 진행하고 있습니다. SSL Handshake란? Handshake는 악수를 의미하는데, 통신을 하는 브라우저와 웹 서버가 서로 암호화 통신을 시작할 수 있도록 신분을 확인하고 필요한 정보를 클라이언트와 서버가 주거니 받거니 하는 과정이 악수와 비슷하여 붙여진 이름이다. 대칭 키와 비대칭 키 대

https://steady-coding.tistory.com/512



- **HTTP**: 일반 텍스트로 자원을 주고 받음(암호화되지 않은 데이터) → 네트워크를 가로채면 내용이 유출될 수 있음
- **HTTPS**: SSL 프로토콜로 암호화함(**대칭키**(같은 키), **공개키**)
  - **SSL handshake**



1. 클라이언트 → 서버, Client Hello: 클라이언트에서 생성한 랜덤 데이터, 클라이언트가 지원하는 암호화 방식, 세션ID(이미 SSL handshake를 한 경우 기존 세션 재활용), SSL 프로토콜 버전
2. 서버 → 클라이언트, Server Hello: 서버에서 생성한 랜덤 데이터, 서버가 선택한 암호화 방식, SSL 프로토콜 버전
3. 서버 → 클라이언트, Certificate: 인증서(CA, 서버 공개키 등)
   a. 클라이언트가 CA 리스트(클라이언트 내장) 확인(없으면 사용자에게 경고 메세지) → CA의 공개키(클라이언트 내장)를 이용해 인증서 복호화 → CA로 인증된 인증서 확인
4. 서버 → 클라이언트, Server Hello Done
5. 클라이언트 → 서버, Client Key Exchange: 클라이언트가 서버의 랜덤 데이터와 클라이언트의 랜덤 데이터를 조합해 **대칭키**(세션키) 생성 → 서버 공개키로 대칭키 값을 암호화해서 서버에 전송
   a. 클라이언트가 전송한 암호화된 대칭키 값을 서버가 서버의 비공개키로 복호화 → 세션키
6. Change Cipher Spec, Finished

- 데이터 전송
- 세션종료: 세션키 폐기

3.3. HTTPS 동작 방식

HTTP와 HTTPS - 전송 프로토콜 비교 - AWS

HTTPS와 HTTP의 차이점은 무엇인가요? AWS에서 HTTPS와 HTTP를 사용하는 방법

https://aws.amazon.com/ko/compare/the-difference-between-https-and-http/#:~:text=HTTP%20메시지는%20일반%20텍스트,을%20확신할%20수%20있습니다

1. 사용자가 브라우저에 https://{URL}을 입력
2. 브라우저는 서버에 SSL 인증서(인증서 발급한 CA, 서비스 도메인 등 서비스 정보, 서버 공개키)를 요청해 사이트의 신뢰성 검증 시도
3. 서버는 퍼블릭 키가 포함된 SSL 인증서 응답

4. 브라우저에서 인증(CA 리스트 확인 → CA의 공개키로 인증서 복호화)되면, 브라우저가 퍼블릭 키를 사용해 secret 세션 키가 포함된 메세지를 암호화하고 전송
5. 서버는 프라이빗 키를 사용해 메세지를 해독하고 세션 키를 검색 → 세션 키를 암호화하고 브라우저에 승인 메세지 전송
6. 브라우저와 서버 모두 동일한 세션 키를 사용해 메세지를 안전하게 교환하도록 전환

3.4. HTTP Method

3.5. HTTP 1.0 vs 1.1

HTTP (1) - version 별 특징 (0.9 / 1.0 / 2.0 / 3.0)
[HTTP 0.9] [HTTP 1.0] [HTTP 2.0] [HTTP 3.0]

https://velog.io/@neity16/HTTP-HTTP-버전-별-특징

- **HTTP/1.0**: HTTP 헤더 추가, 버전 정보, HTTP 메서드, 상태코드, Content-Type(HTML 외의 문서 전송)
  - 커넥션 하나 당 요청 하나와 응답 하나

```
/* 요청 */
GET /mypage.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

/* 응답 */
200 OK
Date: Tue, 15 Nov 1994 08:12:31 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/html
(content)
```

- **HTTP/1.1**: **Persistent Connection**(지정한 timeout 동안 커넥션을 닫지 않음), **Pipelineing**(하나의 커넥션에 여러 개의 요청, multiplexing X)
  - Head of Line Blocking(HOL): 앞 요청의 응답이 너무 오래걸리면 뒤 요청은 Blocking
  - Header 구조 중복: 연속 요청된 헤더의 많은 중복

```
/* 요청 */
GET /en-US/docs/Glossary/Simple_header HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/en-US/docs/Glossary/Simple_header

/* 응답 */
200 OK
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 20 Jul 2016 10:55:30 GMT
Etag: "547fa7e369ef56031dd3bff2ace9fc0832eb251a"
Keep-Alive: timeout=5, max=1000
Last-Modified: Tue, 19 Jul 2016 00:59:33 GMT
Server: Apache
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
(content)
```

3.6. HTTP2

- **HTTP/2**: HTTP/1.1의 성능향상 → HTTP/1.1(표준), HTTP/2(확장)
  - **HTTP 메세지 전송 방식**
    - (기존) 일반 텍스트 형식
    - (개선) Binary Framing 계층 추가 → 메세지를 프레임 단위로 분할해 바이너리로 인코딩 → 파싱속도, 전송속도 빠름

- **Multiplexed Streams**
  - (기존) Pipelining(하나의 커넥션에 여러 요청) → 동시에 여러 요청을 처리해 응답하지 못함
  - (개선) 하나의 커넥션 안에 여러개의 Stream(바이트의 양방향 흐름) → multiplexing 가능
- **Stream Prioritization**: 리소스 간 우선순위 설정 → Stream에 우선순위 부여
- **Server Push**: 서버에서 클라이언트에 필요한 추가적인 리소스를 push
- **Header Compression**
  - (기존) 연속된 요청의 경우 많은 중복된 헤더의 전송 → 오버헤드
  - (개선) 요청과 응답의 헤더를 압축 → 전송되는 헤더 필드(static dynamic table)를 서버에 유지, 이전 헤더를 제외한 필드를 **허프만 인코딩**(데이터 압축)
- 각 요청마다 Stream으로 구분해서 병렬처리하지만, TCP 고유의 HOL Blocking 있음
  - 서로 다른 Stream이 전송될 때, 하나의 Stream에서 문제 발생 시 다른 Stream도 해당 문제가 해결될 때까지 지연됨

- **HTTP/3**: QUIC 기반
  - **QUIC**: Google, UDP 기반의 전송 프로토콜(TCP의 구조적 문제로 성능 향상이 어렵다고 판단)
    - TCP의 3-way handshake 과정을 최적화: 각 Stream 당 독립된 Stream chain을 구성 → HOL Blocking 해결

3.7. HTTP 헤더

[서버] REST API 디자인 가이드 - 하나몬

REST API는 공식적으로 정해진 뚜렷한 규격이 없다. 그렇다 보니, 개발자에 따라 REST API 특징(원칙)에 맞춰 조금씩 다른 모습을 하고 있다. 그러나 REST API 모범 사례는 여전히 논의되고 통합되고 있기 때문에, 수많은 디자인 중에 분명히 보기 좋은 REST API 디자인은 있다. 다음의 API 작성 가이드를

https://hanamon.kr/rest-api-디자인-가이드/

[Network] HTTP 헤더의 종류 및 항목 - Heee's Development Blog

Step by step goes a long way.

https://gmlwjd9405.github.io/2019/01/28/http-header-types.html

웹 서비스 캐시 똑똑하게 다루기

웹 성능을 위해 꼭 필요한 캐시, 제대로 설정하기 쉽지 않습니다. 토스 프론트엔드 챕터에서 올바르게 캐시를 설정하기 위한 노하우를 공유합니다.

https://toss.tech/article/smart-web-service-cache

- **HTTP 헤더**: 요청/응답에 대한 정보
  - general: date(HTTP 메세지 생성 날짜, 시간), **connection**(클라이언트/서버 연결 옵션)
    - `connection: close`: 현재 HTTP 메세지 직후 TCP 연결 끊음
    - `connection: keep-alive`: 현재 TCP 커넥션 유지
  - request: host(요청하는 호스트명, 포트번호), user-agent(클라이언트 소프트웨어 정보), from(클라이언트 사용자 메일), cookie(서버 set-cookie 설정된 정보), referer(직전에 머물렀던 웹 링크 주소), if-modified-since(제시한 일시 이후로 변경된 리소스만 요청), authorization(인증 토큰), origin(POST 요청이 시작된 주소 → **CORS**), accept, accept-charset, accept-encoding, accept-language
  - response: server(서버 소프트웨어 정보), set-cookie(세션 쿠키 정보 설정), expires(캐시 만료 날짜 시간), cache-control(max-age로 캐시 유효 시간 설정), age(max-age 시간 내 얼마나 흘렀는지), ETag(contents가 바뀌었는지 검사), allow(지원가능 메소드 리스트), access-control-allow-origin(요청 보내는 프론트 주소와 받는 주소가 다르면(프로토콜, 서브도메인, 도메인, 포트) → **CORS**)
  - entity: 리소스에 대한 정보

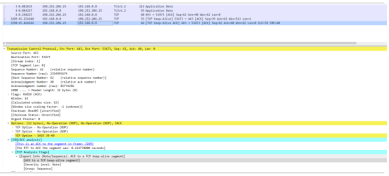
    - 공통: content-type(MIME 유형 컨텐츠 타입, 인코딩 방식), content-language(자연어), content-encoding(데이터 압축 방식), content-length, content-location
    - 응답: content-disposition(응답을 브라우저가 표시할 방식), content-security-policy(차단할 소스와 불러올 소스), location(201, 3xx 응답 시 이동할 페이지 경로), last-modified(리소스를 마지막으로 갱신한 일시), transfer-encoding(길이를 모르는 파일, chuncked)
- **caching** 관련 헤더
  - cache-control: no-store, no-cache, must-revalidate, public, private, max-age, s-maxage(중간서버)
  - expires: 컨텐츠 만료 일시, cache-control에 max-age가 있는 경우 무시
  - age: max-age 시간 내 얼마나 흘렀는지
  - ETag: HTTP 컨텐츠가 바뀌었는지 확인할 수 있는 태그
  - if-none-match: 서버에 ETag가 달라졌는지 검사 요청 → 같다면 304 Not Modified
- **cookie**(상태 정보를 브라우저에 저장) 관련 헤더
  - cookie: 서버에서 set-cookie로 설정된 쿠키 정보
  - set-cookie: 서버에서 클라이언트에 세션 쿠키 정보 설정 → 서버에서 검증 로직 필요

3.8. keep-alive 헤더

[Network 04] Persistent Connection을 위한 기술 01 - Keep Alive (TCP, HTTP)

0. 들어가며 최근, http의 persistent connection을 유지하는 기술들에 대해서 공부를 시작하면서 keep-ali...

https://blog.naver.com/whdgml1996/222153047879



[HTTP] keep alive란? (persistent connection에 대하여)

HTTP 관련 서비스를 구현하다보면 persistent connection을 빼놓을 수 없다. 그렇다고 필자가 완벽히 이걸 다 이해한건 아니고 공부중이다 하핫. 본 글에서는 persistent connection의 기본적인 개념을 소개하고 HTTP/1.0의 keep-alive에 대해서 알아본다. Persistent Connection HTTP에서

https://etloveguitar.tistory.com/137

- timeout, max(최대 요청 수)
- blind relay 문제: proxy가 keep-alive를 지원하지 않는 경우

3.9. HTTP GET vs POST

- **GET**: 리소스를 가져올 때(SELECT) 사용. 요청하는 리소스를 header의 url에서 query param에 담아 전송, 캐싱(로컬 캐시)을 통해 이전에 요청한 데이터를 빠르게 받을 수 있음
  - url 공간에 담아야 하므로 리소스의 크기가 제한적, 비밀번호 같은 보안이 필요한 데이터는 url에 노출
    - 브라우저 혹은 웹서버에서 url 길이 제한 정의. 일반적으로 브라우저에서 (ASCII 문자 기준) 2048자 이하로 정의됨
  - 캐싱된 리소스 변경 시 즉각적으로 갱신되지 않을 수 있음 → **HTTP 헤더**로 캐시 유효기간 설정가능
    - 서버에서 응답 헤더에 cache-control 값의 max-age 설정
    - 클라이언트에서 요청 헤더에 if-modified-since(ETag)와 if-none-match(ETag 변경 확인)로 재검증 할 수 있음 → 캐시 유효시 304 Not Modified
- **POST**: 리소스를 등록할 때(CREATE) 사용. 요청하는 리소스를 body에 담아 전송
  - body 길이 제한은 웹서버에서 정의할 수 있음 → Spring Boot 보통 10MB default, application.yml에서 max-request-size 값 설정으로 변경 가능

3.10. 쿠키 vs 세션

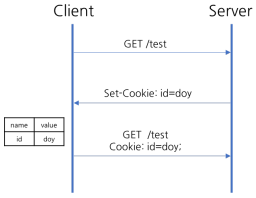
14주차 암호화 세션 / 쿠키

암호화는 일련의 정보를 임의의 방식을 사용하여 다른 형태로 변환하여 해당 방식에 대한 정보를 소유한 사람을 제외하고 이해할 수 없도록 알고리즘을 이용해 정보를 전달(관리)하는 과정을 의미한다.어떠한 문자열에 임의의 연산을 적용하여 다른 문자열로 변환하는 것민감한 정보를

https://velog.io/@ellie12/14주차-인증-보안-세션-쿠키

velog

- HTTP 프로토콜에서 상태 정보가 필요할 때 사용
- **쿠키**: 클라이언트 로컬에 저장(key=value) → 만료시간에 따라 브라우저를 종료해도 계속 남아있음



1. 클라이언트가 서버에 요청
2. 서버가 응답 시 유지하고 싶은 값을 HTTP 헤더 set-cookie에 담아 전송

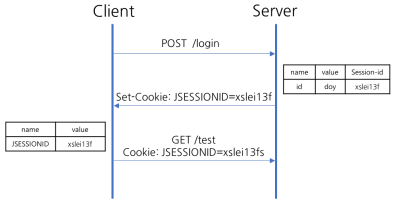
```
Set-Cookie: name=value
```

3. 클라이언트에서 쿠키를 관리하다가 다음 요청 때 쿠키를 HTTP 헤더 cookie에 담아 전송

```
Cookie: name=value
```

4. 서버에서 쿠키 정보를 읽어 이전 상태 확인 후 응답

- **세션**: 서버에 저장 → 브라우저가 종료되면 클라이언트에서 삭제됨
  - 세션 ID가 저장된 서버에서만 인증가능 → 공용 세션 저장소, sticky session(사용자의 요청이 session ID를 발급받은 서버로 가도록)



1. 클라이언트가 서버에 요청
2. 서버가 클라이언트에 유일한 세션 ID 부여해 HTTP 헤더 set-cookie에 담아 전송

```
Set-Cookie: JSESSIONID=abcds
```

3. 클라이언트는 브라우저를 닫기까지 다음 요청에 부여된 세션 ID를 HTTP 헤더 cookie에 담아 전송

```
Cookie: JSESSIONID=abcds
```

4. 서버에서 세션 ID를 확인하고, 해당 세션에 관련된 정보 확인 후 응답

4. 웹

4.1. www.naver.com에 접속할 때

1. 브라우저에 www.naver.com 입력

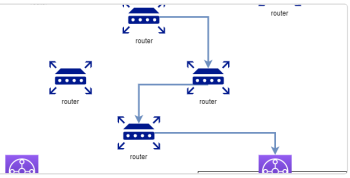
2. DNS lookup → IP 주소 찾음
3. 서버와 TCP 연결(3-way handshake)
4. 서버에 HTTP 요청(GET)
5. 서버가 요청처리 후 HTTP 응답
6. 브라우저가 HTML 출력

4.2. 웹브라우저에서 서버로 요청했을 때 흐름

브라우저에서 서버까지 응답 흐름

인터넷 브라우저(IE, Chrome, Safari 등)에서 URL 입력이 들어왔을 때 서버까지의 흐름에 대해서 알아본다. 1. 사용자의 URL 요청 사용자가 브라우저 검색창에 www.google.com과 같은 주소를 요청하는 단계이다. 2. 브라우저의 URL 분석 URL은 인터넷에서 자원의 위치(Uniform Resource Locator)를

https://doqtqu.tistory.com/313



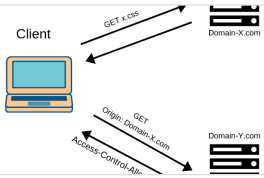
1. 사용자가 URL 요청
2. 브라우저에서 URL 분석: 프로토콜 + 도메인 주소 + 포트 + 경로 + 파라미터
3. HSTS(HTTP Strict Transport Security) 목록 조회: HSTS 목록에 URL이 존재하면 명시적으로 HTTP를 통해 요청해도 브라우저가 HTTPS로 요청함
    - HTTP로 요청이 올 경우 HTTP 응답 헤더에 Strict Transport Security 필드를 포함해 응답 → 브라우저는 해당 서버에 응답할 때 HTTPS만을 통해 통신, 자신의 HSTS 캐시에 해당 URL 저장(HSTS 목록)
4. URL을 IP주소로 변환: DNS lookup(브라우저 → OS → 라우터 → ISP 캐시 → DNS 서버)
5. 라우터를 통해 해당 서버의 게이트웨이로 전송: 라우터에서 라우팅 테이블을 통해 해당 요청이 어떤 경로로 갈지 라우팅
6. **ARP**로 IP주소 → MAC주소 변환
7. 대상 서버와 TCP 소켓 연결(3-way handshake + **TLS/SSL handshake**)
8. 서버에 HTTP 메세지 요청 → 서버에서 요청을 수락할 수 있는지 검사 → 응답 메세지 생성 → 브라우저로 전송
9. 브라우저에서 응답 해석 → 사용자에게 출력

4.3. CORS

[네트워크/HTTP] OPTIONS 메소드를 쓰는 이유와 CORS란? - 하나몬

⚡ HTTP OPTIONS 메소드를 쓰는 이유와 CORS란? ❗CORS란? 웹 브라우저에서 외부 도메인 서버와 통신하기 위한 방식을 표준화한 HTTP 헤더 기반 메커니즘이다. 클라이언트와 서버가 서로 다른 도메인일 경우(HTTP 헤더의 origin이 다른 경우)가 있으므로 CORS 기술이 도입 되었다. CORS는 웹 브라우저가

https://hanamon.kr/네트워크-http-options-메소드를-쓰는-이유와-cors란/



- **CORS**: 브라우저에서 외부 도메인 서버와 통신하기 위한 방식을 표준화. HTTP 헤더 기반
    - 클라이언트와 서버가 다른 도메인인 경우(헤더의 origin(프로토콜, 도메인, 포트번호)이 다른 경우)
    - preflight request: 실제 요청 전 OPTIONS 메서드로 권환 확인
        - `Access-Control-Allow-Origin` → '*' 와일드카드
        - `Access-Control-Allow-Credentials`: 쿠키를 통해 자격증명하는지 여부
        - `Access-Control-Max-Age`: preflight 결과 캐싱 기간

4.4. 웹 서버 vs 웹 애플리케이션 서버(WAS)

[Web] Web Server와 WAS의 차이와 웹 서비스 구조 - Heee's Development Blog

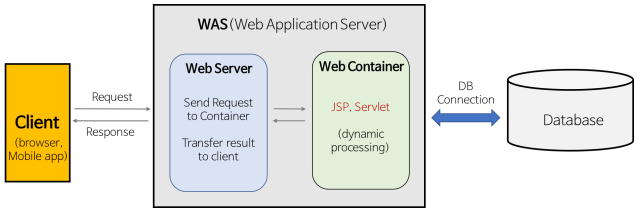
Step by step goes a long way.

https://gmlwjd9405.github.io/2018/10/27/webserver-vs-was.html

[Web] web.xml 설정 내용, 역할 및 간단한 예시 이해하기 - Heee's Development Blog

Step by step goes a long way.

https://gmlwjd9405.github.io/2018/10/29/web-application-structure.html



- 웹 서버: 클라이언트로부터 HTTP 요청을 받아 서비스

  ex. Apache Server, Nginx 등

  - 정적 컨텐츠 제공
  - 동적 컨텐츠 제공을 위해 WAS에 요청 전달 및 결과 반환
- WAS: 로직 처리를 요구하는 동적인 컨텐츠를 제공하기 위해 만들어진 애플리케이션 서버(웹 서버 + 웹 컨테이너)

  ex. Apache Tomcat: JSP, Servlet 구동 환경 제공

4.5. REST API

[서버] REST API란? - 하나몬

REST의 개념을 이해한다. REST의 특징을 이해한다. REST API의 개념을 이해한다. REST API의 설계 규칙을 이해한다. RESTful의 개념을 이해한다.

https://hanamon.kr/rest-api/

- **REST**: 자원의 표현과 상태(정보)를 전달(일반적으로 JSON, XML로 전달)하는 것. 자원 기반 구조(ROA)

  ex. 자원의 표현: 학생(관리하는 자원) → students

  - HTTP URL로 자원을 명시하고, **HTTP Method**(POST, GET, PUT, DELETE)로 자원에 대한 CRUD operation을 적용함
  - **서버/클라이언트 구조**: 서버(API 제공, 비즈니스 로직 처리), 클라이언트(API 호출, 응답 처리) → 독립성, 느슨한 결합
  - **stateless**: HTTP 프로토콜과 마찬가지로 무상태 → 클라이언트의 정보를 서버에 저장하지 않음
  - **cacheable**: HTTP의 캐싱 기능을 적용할 수 있음. Last-Modified, E-Tag 이용
- **REST API**: REST를 기반으로 만든 서비스 API(클라이언트가 리소스를 요청할 수 있도록 서버에서 만들 인터페이스)
  - 응답 상태 코드: 1XX(전송 프로토콜 수준의 정보 교환), 2XX(성공), 3XX(클라이언트 추가 행동 요청), 4XX(클라이언트 잘못 요청), 5XX(서버 오류)

4.6. RESTful

- **RESTful**: **REST**를 따르는 것

5. 기타

5.1. 네트워크 계층 라우팅 알고리즘

- 중앙 집중형 라우팅 알고리즘: 네트워크 전체에 대한 완전한 정보를 가지고 출발지와 목적지 사이의 최소 비용 경로 계산
  - 모든 라우터가 연결 상태와 링크 비용을 알고 있음. 이벤트 기반 갱신 → Link State 알고리즘(주로 **다익스트라 알고리즘** 사용)
- 분산 라우팅 알고리즘: 최소 비용 경로 계산이 라우터들에 의해 반복적이고 분산된 방식으로 수행
  - 각 라우터는 연결된 인접 노드에 대한 링크 비용 정보를 알고 있음. 주기적으로 갱신 → Distance Vector 알고리즘(주로 **벨만 포드 알고리즘** 사용)

5.2. 로드밸런서, 로드밸런싱

로드밸런서(Load Balancer)의 개념과 특징

[BY 가비아] 현대의 모든 정보는 인터넷을 통해 연결되어있습니다. 인터넷의 발달은 데이터 통신을 보다...

https://m.post.naver.com/viewer/postView.naver?volumeNo=27046347&memberNo=2521903

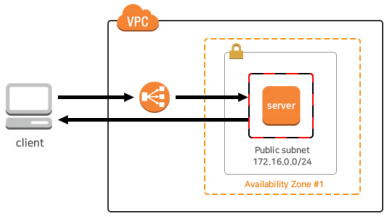
로드 밸런싱(Load balancing) 종류와 알고리즘 | DevelopersIO

로드 밸런싱이라고 하면「트래픽을 분산시켜주는 컴퓨터 네트워크의 기법」이라고만 생각했지만, 로드 밸런스를 구현해보고 사용해보면서 다양한 종류와 알고리즘이 있다는 것을 알았습니다. 이번 블로그에서는 간단하게 로드 밸런싱의 종류와 알고리즘에 대해서 알아보고자 합니다.

https://dev.classmethod.jp/articles/load-balancing-types-and-algorithm/

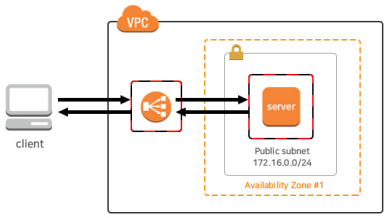
- **로드밸런서**: 서버에 가해지는 부하를 분산(scale-out 방식에서 필요)
  - 로드밸런싱 알고리즘
    - **라운드로빈**: 순서대로 배정 → 서버가 동일한 스펙, 연결이 오래 지속되지 않는 경우
    - **가중 라운드로빈**: 서버마다 가중치를 매기고 가중치 높은 서버에 우선 배정 → 서버의 트래픽 처리 능력이 다른 경우

      ex. A 서버(가중치 5), B 서버(가중치 2) → 라운드로빈으로 A 서버 5개, B 서버 2개 요청
    - **IP 해시**: 클라이언트의 IP 주소를 해싱해 특정 서버로 매핑 → 클라이언트가 항상 동일한 서버로 연결
    - **최소 연결**: 요청이 들어온 시점에 가장 적은 연결 상태를 갖는 서버에 우선 배정 → 자주 세션이 길어지거나 서버에 분배된 트래픽이 일정하지 않은 경우
    - **최소 응답 시간**: 서버의 현재 연결 상태, 응답시간을 고려해 배정. 가장 적은 연결 상태와 짧은 응답 시간을 보이는 서버에 우선 배정
- 로드밸런서 종류
  - **L4 로드밸런서(NLB)**: 3계층 네트워크 계층의 IP주소, 4계층 전송 계층의 포트, TCP, UDP 정보를 바탕으로 부하를 분산



    - 서버로 들어오는 트래픽만 로드밸런서를 통함
    - NLB에는 보안그룹 적용되지 않음 → 서버에 적용된 보안그룹으로 보안
  - **L7 로드밸런서(ALB)**: 7계층 애플리케이션 계층의 HTTP 헤더 정보를 바탕(패킷의 내용 확인 가능)으로 부하를 분산

    ex. URL에 따른 분산, HTTP 헤더 쿠키 값에 따른 분산



    - 서버에 들어오고 나가는 트래픽 모두 로드밸런서를 거침
    - 보안그룹 적용 가능

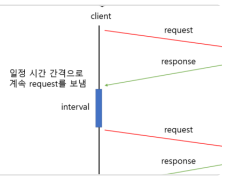
    - IP 주소가 변동되므로 클라이언트에서 ALB의 도메인주소로 요청해야 함(Route53에서 CNAME 레코드 등록 필)

5.3. 멀티플렉싱

[Network 05] Persistent Connection을 위한 기술 02 - poling, long poling (feat 양방향 통신)

0. 들어가며 지난 시간에는 persistent connection을 위한 기술로 keep-alive에 대해서 소개해 드렸습니다....

https://blog.naver.com/whdgml1996/222153682983



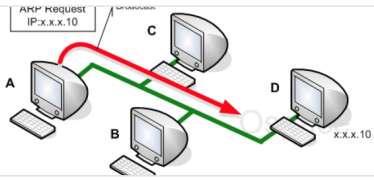
- 멀티플렉서: 2^n → 1, 디멀티플렉서: 1 → 2^n
- 4계층(전송계층): end-point의 host끼리의 안전한 송수신을 보장하기 위한 계층.
    - 특정 포트 번호를 가진 TCP/UDP 패킷들을 포트 번호대로 분류 → 포트와 연결된 소켓으로 데이터 전달(하나의 전송계층이 여러 소켓에게 데이터 분배. DEMUX)
    - 여러 소켓이 하나의 전송계층으로 요청 → 하나의 네트워크계층으로 전달(MUX)
- 서버 유형: 웹서버를 생성하는 방식(Single Process/thread, multi-thread, multi-process, **multiplexing**)
    - **multiplexing**: 단 하나의 서버 프로세스(서버 소켓 1개)로 여러 클라이언트 소켓들의 요청을 처리함 → 여러개의 소켓을 하나의 소켓으로 연결. MUX
        - **polling**(서버 프로세스가 file descriptor 지속적으로 감시 → 입력/출력 flag set → 해당 descriptor에 서버 소켓 연결해 작업 처리 후 해당 클라이언트 소켓 연결 끊음 → 다른 작업) 사용
- **HTTP/2.0**: 여러 데이터를 동시에 전송할 수 있는 **multiplexing** 도입
    - 여러 데이터가 하나의 TCP 연결 위에서 stream 생성 → stream으로 데이터 전송 → 여러 데이터 stream을 하나의 커넥션으로 연결. MUX

5.4. ARP, RARP

ARP(Address Resolution Protocol)란? - 개념과 동작원리

ARP 란? 주소 결정 프로토콜(Address Resolution Protocol) 해당 IP를 그 IP주소에 맞는 물리적인 주소 즉, MAC주소를 가지고 오는 프로토콜입니다. RARP : 역순 주소 결정 프로토콜 (Reverse Address Resolution Protocol) ARP와 반대로 해당 MAC주소에 맞는 IP값을 알아오는 프로토콜 입

https://musclebear.tistory.com/12



- **ARP**: IP → MAC 주소 변환
    1. 목적지 IP 주소를 지정해 패킷 전송
    2. IP 프로토콜 → ARP 프로토콜에 ARP 요청 메세지(출발 MAC 주소, 출발 IP 주소, 00-00-00-00-00-00, 목적 IP 주소) 생성 요청
    3. 2계층에서 이더넷 프레임으로 캡슐화(출발 MAC 주소, 브로드캐스트 주소)
    4. 모든 호스트와 라우터에서 프레임 수신(**브로드캐스트**) → 자신의 ARP 프로토콜에 전달
    5. 목적지 IP 주소와 일치하는 시스템 → 출발 host에 자신의 물리주소를 포함하는 ARP 응답 메세지 전송(**유니캐스트**)
    6. IP 주소에 대응하는 MAC 주소 획득
- **RARP**: MAC → IP 주소 변환

5.5. TCP HOL 블로킹

- TCP는 cumulative ACK 사용 → 하나의 패킷이 유실될 경우 이후 모든 패킷에 대한 ACK가 누적 ACK가 됨
    - 유실된 패킷이 처리되기 전에 다음 패킷을 받지 못함 → 전체 작업 지연(HOL 블로킹)

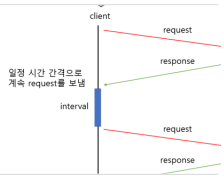
5.6. polling, long polling, websocket

- 양방향 통신 방법

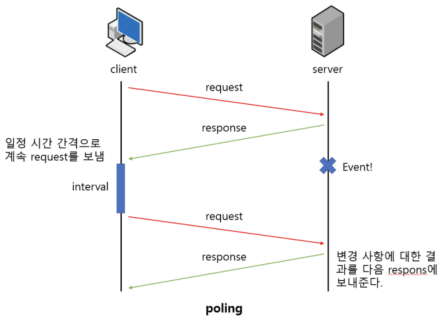
[Network 05] Persistent Connection을 위한 기술 02 - poling, long poling (feat 양방향 통신)

0. 들어가며 지난 시간에는 persistent connection을 위한 기술로 keep-alive에 대해서 소개해 드렸습니다....

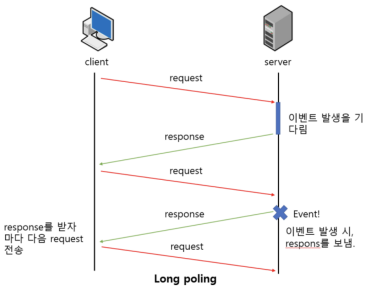
https://blog.naver.com/whdgml1996/222153682983



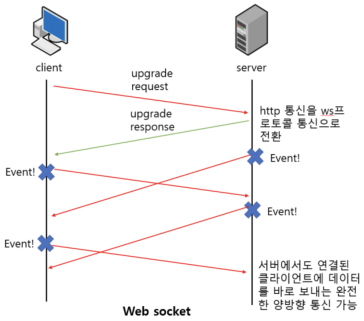
- **HTTP AJAX poling**: 일정 시간 간격으로 request 전송

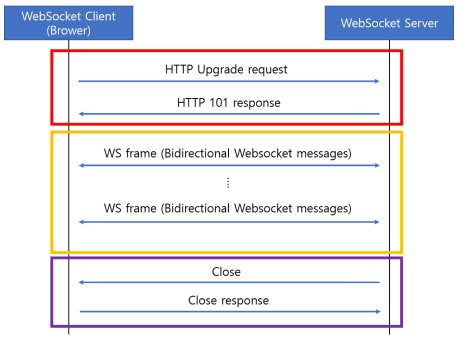


- **long polling**: 이벤트 발생 시까지 서버에서 대기 후 response



- **websocket**: 양방향 실시간 연결. TCP/IP 기반 응용 계층 프로토콜. 이벤트 발생 시 서버에서 바로 클라이언트에 전송 가능





1. TCP 연결(3-way handshake)
2. open handshake
   a. HTTP upgrade로 handshake 요청

```
GET /chat HTTP/1.1
 Host: localhost:8080
 Upgrade: websocket
 Connection: Upgrade
 Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
 Sec-WebSocket-Protocol: chat, superchat
 Sec-WebSocket-Version: 13
 Origin: http://localhost:9000
```

    b. HTTP 응답: 101 프로토콜 전환 승인

3. 데이터 전송

4. close handshake

5. TCP 연결 종료(4-way handshake)