1주차 - 디자인패턴, 네트워크

예상질문

1.1 디자인패턴

- 하나의 '규약' 같은 것
- 프로그램을 설계할 때 발생했던 문제점들을 객체 간의 상호 관계 등을 이용하여 해결할 수 있도록 함

1.1.1 싱글톤 패턴(singleton pattern)

- 하나의 클래스에 오직 하나의 인스턴스만 가지는 패턴
- 보통 데이터베이스 연결 모듈에 많이 사용
- 하나의 인스턴스를 만들어 놓고 다른 모듈이 공유하며 사용

  → 장점 : 인스턴스 생성시 드는 비용이 줄어듦

  → 단점 : 의존성이 높아짐

싱글톤 패턴의 단점

- TDD(Test Driven Development)시 걸림돌
- TDD시 단위테스트를 주로 하는데, 단위 테스트는 테스트가 서로 독립적이어야 하며 테스트를 어떤 순서로든 실행할 수 있어야 함
- 그러나 싱글톤 패턴은 미리 생성된 하나의 인스턴스를 기반으로 구현하는 패턴이므로

  각 테스트마다 '독립적인' 인스턴스를 만들기가 어려움

의존성 주입

- 싱글톤 패턴은 사용하기 쉽고 굉장히 실용적임
- 하지만 모듈 간의 결합을 강하게 만들 수 있음

- 해결방법 : 의존성 주입(DI : Dependency Injection)

> 💡 **의존성(종속성)**
>
> - A가 B에 의존성이 있다 = B가 변하면 A도 변해야 한다

의존성 주입의 장점

- 모듈들을 쉽게 교체할 수 있는 구조가 됨

  → 테스팅 쉬움, 마이그레이션 수월함
- 구현시 추상화 레이어를 넣고 이를 기반으로 구현체를 넣어줌

  → 애플리케이션 의존성 방향이 일관적

  → 애플리케이션 쉽게 추론 가능

  → 모듈간 관계가 조금 더 명확해짐

의존성 주입의 단점

- 모듈들이 더욱더 분리되므로 클래스 수가 늘어나 복잡성이 증가될 수 있음
- 약간의 런타임 페널티가 생길 수 있음

의존성 주입 원칙

- “상위 모듈은 하위 모듈에서 어떤 것도 가져오지 말아야 합니다. 또한 둘 다 추상화에 의존해야 하며, 이 때 추상화는 세부 사항에 의존하지 말아야 합니다.”

1.1.2 팩토리 패턴(factory pattern)

- 객체를 사용하는 코드에서 객체 생성 부분을 떼어내 추상화한 패턴
- 또한 상속 관계에 있는 두 클래스에서 상위 클래스가 주용한 뼈대를 결정,

  → 하위 클래스에서 객체 생성에 관한 구체적인 내용을 결정하는 패턴

- 상위 클래스와 하위 클래스가 분리
  - 느슨한 결합
  - 더 많은 유연성 (상위 클래스에서는 인스턴스 생성 방식을 알 필요가 없기 때문)

- 객체 생성 로직이 따로 떼어져 있음
  - 유지보수성 증가 (한 곳만 고칠 수 있어서)

- 예시 : 바리스타 공장의 생산 공정

1.1.3. 전략 패턴(stragegy pattern)

- 전략(strategy) 혹은 정책(policy) 패턴
- 객체의 행위를 바꾸고 싶은 경우
  - 전략이라고 부르는 '캡슐화된 알고리즘'을 컨텍스트 안에서 바꿔주면서 상호 교체가 가능하게 만드는 패턴

- 예시 : 결제시 네이버페이, 카카오페이 등 결제 방식을 바꿈

1.1.4 옵저버 패턴(observer pattern)

- 주체가 어떤 객체의 상태 변화를 관찰하다가 상태 변화가 있을 때마다 메서드 등을 통해 옵저버 목록에 있는 옵저버들에게 변화를 알려주는 패턴

- 주체 : 객체의 상태 변화를 보고 있는 관찰자
- 옵저버 : 객체의 상태 변화에 따라 전달되는 메서드 등을 기반으로 '추가 변화 사항'이 생기는 객체들

- 주체와 객체를 따로 주지 않고 상태가 변경되는 객체를 기반으로 구축하기도 함

- 예시 : 트위터

- 옵저버 패턴은 주로 이벤트 기반 시스템에 사용됨
- MVC 패턴에도 사용됨
    - 모델(주체)에서 변경사항 발생 → update() 메서드로 뷰(옵저버)에게 알려줌 → 컨트롤러가 작동

상속과 구현

- 상속(extends) : 자식 클래스가 부모 클래스의 메서드 등을 상속받아 자식 클래스에서 추가 및 확장
    - 재사용성, 중복성 최소화
    - 일반 클래스, 추상 클래스
- 구현(implements) : 부모 인터페이스를 자식 클래스에서 재정의하여 구현
    - 반드시 부모 클래스의 메서드를 재정의하여 구현해야 함
    - 인터페이스

- 자바스크립트에서의 옵저버 패턴은 프록시 객체를 통해 구현할 수도 있음

프록시 객체

- 어떤 대상의 기본적인 동작? 작업을 가로챌 수 있는 객체
    - 동작은 속성 접근, 할당, 순회, 열거, 함수 호출 등이 있다
- 자바스크립트에서는 두 개의 매개 변수를 가짐
    - target : 프록시 할 대상
    - handler : target 동작을 가로채고 어떤 동작을 할 것인지 설정되어 있는 함수

1.1.5 프록시 패턴과 프록시 서버 (proxy pattern)

- 프록시 객체는 사실.. 프록시 패턴이 녹아들어 있는 객체!
- 프록시 패턴은 대상 객체에 접근하기 전, 그 접근에 대한 흐름을 가로챔
    - 해당 접근을 필터링하거나 수정하는 등의 계층이 있는 디자인 패턴
- 이를 통해 객체의 속성, 변환 등을 보완
- 보안, 데이터 검증, 캐싱 등에 사용함

프록시 서버에서의 캐싱

- 멀리 원격 서버에 캐시 정보를 요청하지 않고 캐시 안에 있는 데이터를 활용
- 트래픽을 줄일 수 있음 (멀리 요청하지 않으니까)

프록시 서버

- 서버와 클라이언트 사이에서 클라이언트가 자신을 통해 다른 네트워크 서비스에 간접적으로 접속할 수 있게 해주는 컴퓨터 시스템 혹은 응용 프로그램

프록시 서버로 쓰는 nginx

- nginx : 비동기 이벤트 기반의 구조와 다수의 연결을 효과적으로 처리할 수 있는 웹서버
    - 주로 Node.js 서버 앞단의 프록시 서버로 활용됨

💡 **버퍼 오버플로우**

- 버퍼 : 데이터가 저장되는 메모리 공간
- 버퍼 오버플로우 : 데이터가 메모리 공간을 벗어나는 경우
  - 이때 사용되지 않아야 할 영역에 데이터가 덮어씌워져 주소, 값을 바꾸는 공격이 일어나기도 함

💡 **gzip 압축**

- LZ77과 Huffman 코딩의 조합인 DEFLATE 알고리즘을 기반으로 한 압축 기술
- 데이터 사용량을 줄일 수 있음
- 그러나 압축 해제 시 CPU 오버헤드를 생각해서 gzip 기술 사용 여부를 결정해야 함

프록시 서버로 두는 cloudflare

- cloudflare : 전세계적으로 분산된 서버가 있고, 이를 통해 어떤 시스템의 콘텐츠 전달을 빠르게 할 수 있는 CDN 서비스
- 웹 서버 앞단에 프록시 서버로 두어 DDOS 공격 방어나 HTTPS 구축에 쓰임
- 의심스러운 트래픽인지판단해서 CAPTCHA 등을 기반으로 일정 부분 막아주기도 함

- cloudflare는 시스템을 통해 오는 트래픽을 자동 차단하여 DDOS 공격으로부터 보호
- cloudflare가 거대한 네트워크 용량과 캐싱 전략을 가짐
  - 소규모 DDOS 공격 막아낼 수 있음
  - 방화벽 대시보드 제공

💡 **DDOS 공격 방어**

- DDOS : 짧은 기간 동안 네트워크에 많은 요청을 보내 네트워크를 마비시켜 웹사이트의 가용성을 방해하는 사이버 공격

- HTTPS 구축시 cloudflare를 사용하면 별도의 인증서 설치 없이 손쉽게 HTTPS를 구축할 수 있음

💡 **CDN(Content Delivery Network)**

- 사용자들이 인터넷에 접속하는 곳과 가까운 곳에 콘텐츠를 캐싱 또는 배포하는 서버 네트워크
- 웹서버에서 콘텐츠 다운로드하는 시간을 줄일 수 있음

CORS와 프론트엔드의 프록시 서버

💡 **CORS(Cross-Origin Resource Sharing)**

- 서버가 웹 브라우저에서 리소스를 로드할 때 다른 오리진을 통해 로드하지 못하게 하는 HTTP 헤더 기반 매커니즘

- 프론트엔드 개발시 프론트 엔드 서버와 백엔드 서버가 통신할 때 CORS 에러가 생길 수 있는데 이를 해결하기 위해 프론트엔드 서버에서 프록시 서버를 만들기도 함

💡 **오리진**

- 프로토콜과 호스트 이름, 포트의 조합을 말함
- https://kundol.com:12010/test
  - 오리진은 https://kundol.com:12010임

- 프록시 서버를 둘 경우 포트번호가 다를 때 오류가 날 수 있을 때 오리진을 바꿔주어 CORS 에러를 해결하고 다양한 API 서버와의 통신도 매끄럽게 할 수 있음

1.1.6 이터레이터 패턴(iterator pattern)

- 이터레이터를 사용하여 컬렉션의 요소들에 접근하는 패턴

  → 자료형의 구조와 상관없이 이터레이터라는 하나의 인터페이스로 순회 가능

1.1.7 노출모듈 패턴(revealing module pattern)

- 즉시 실행 함수를 통해 private, public 같은 접근 제어자를 만드는 패턴
- 자바스크립트의 경우 private, public 같은 접근 제어자가 존재하지 않고 전역 범위에서 스크립트가 실행됨

  → 따라서 노출모듈 패턴을 통해 private과 public 접근 제어자를 구현하기도 함

💡 **즉시 실행 함수**

- 함수를 정의하자마자 바로 호출하는 함수
- 초기화 코드, 라이브러리 내 전역 변수의 충돌 방지 등에 사용

1.1.8 MVC 패턴

- 모델, 뷰, 컨트롤러로 이루어짐
- 애플리케이션 구성 요소를 세 가지 역할로 구분
- 개별 프로세스에서 각각의 구성 요소에만 집중하여 개발할 수 있음
- 장점 : 재사용성과 확장이 용이함
- 단점 : 애플리케이션이 복잡해질수록 모델과 뷰의 관계가 복잡해짐

모델

- 애플리케이션의 데이터인 데이터베이스, 상수, 변수 등을 뜻함
- 뷰에서 데이터를 생성하거나 수정 → 컨트롤러를 통해 모델을 생성하거나 갱신

뷰

- inputbox, checkbox, textarea 등 사용자 인터페이스 요소를 뜻함
    - 즉, 모델을 기반으로 사용자가 볼 수 있는 화면을 뜻함
- 모델이 가지고 있는 정보를 따로 저장하지 않아야 함
- 단순히 사각형 모양 등 화면에 표시하는 정보만 가지고 있어야 함

컨트롤러

- 하나 이상의 모델과 하나 이상의 뷰를 잇는 다리 역할
- 이벤트 등 메인 로직 담당
- 모델과 뷰의 생명주기 관리
- 모델이나 뷰의 변경 통지를 받으면 이를 해석하여 각각의 구성 요소에 해당 내용을 알려줌

스프링

- MVC 패턴을 이용한 프레임 워크
- 웹서비스 구축에 편리한 기능 제공
    - 애너테이션 기반으로 요청값을 쉽게 분석
    - 유효한 요청인지 쉽게 거를 수 있음
- 재사용 가능한 코드, 테스트, 쉽게 리디렉션할 수 있게 함

1.1.9 MVP 패턴

- MVC 패턴에서 파생
- C인 컨트롤러가 프레젠터(presenter)로 교체
- 뷰와 프레젠터는 일대일 관계 → MVC 패턴보다 더 강한 결합을 지님

1.1.10 MVVM 패턴

- C인 컨트롤러가 뷰모델(view model)로 바뀜
- 뷰모델 : 뷰를 더 추상화한 계층
- MVC 패턴과 다르게 커맨드와 데이터 바인딩을 가짐
- 뷰와 뷰모델 사이의 양방향 데이터 바인딩 지원
- UI를 별도의 코드 수정 없이 재사용할 수 있음
- 단위 테스팅이 쉬움

- MVVM 패턴을 가진 대표적인 프레임워크 : Vue.js
- Vue.js : 반응형이 특징인 프론트엔드 프레임워크
    - watch나 computed 등으로 쉽게 반응형 값을 구축 가능

💡 **커맨드**

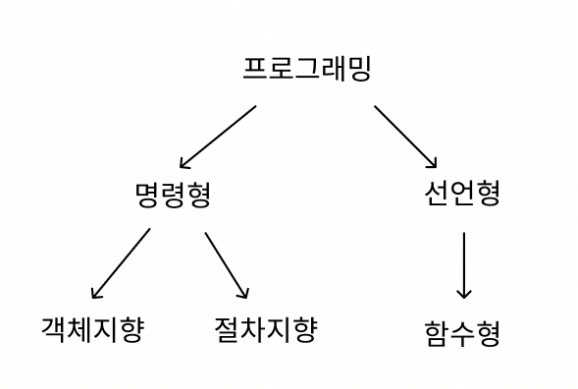
- 여러 가지 요소에 대한 처리를 하나의 액션으로 처리할 수 있게 해주는 기법

데이터 바인딩

- 화면에 보이는 데이터와 웹 브라우저의 메모리 데이터를 일치시킴
- 뷰모델을 변경하면 뷰가 변경된다

1.2 프로그래밍 패러다임

- 프로그래밍 패러다임은 프로그래머에게 프로그래밍의 관점을 갖게 해주는 개발방법론
- 예시
  - 객체지향 프로그래밍 : 프로그램을 상호 작용하는 객체들의 집합으로 볼 수 있게 해줌
  - 함수형 프로그래밍 : 상태 값을 지니지 않는 함수값들의 연속으로 생각하게 함



1.2.1 선언형과 함수형 프로그래밍

선언형 프로그래밍(declarative programming)

- '무엇을' 풀어내는가에 집중하는 패러다임
- "프로그램은 함수로 이루어진 것이다"

함수형 프로그래밍(functional programming)

- 선언형 패러다임의 일종
- 작은 '순수 함수'들을 블록처럼 쌓아 로직을 구현,
  '고차 함수'를 통해 재사용성을 높인 프로그래밍 패러다임
- 자바스크립트는 단순 유연, 함수가 일급객체이므로 함수형 프로그래밍 방식이 선호됨

순수함수

- 출력이 입력에만 의존
- 들어오는 매개변수 a,b에만 영향을 받음
- 만약 다른 전역변수 c가 이 출력에 영향을 주면 순수함수가 아님

고차함수

- 함수를 값처럼 매개변수로 받아 로직을 생성할 수 있는 것

일급 객체

- 고차 함수를 쓰기 위해서는 해당 언어가 일급 객체여야 하며 특징이 다음과 같다
  - 변수나 메서드에 함수를 할당할 수 있음
  - 함수 안에 함수를 매개변수로 담을 수 있음
  - 함수가 함수를 반환할 수 있음

1.2.2 객체지향 프로그래밍(OOP, object-oriented programming)

- 객체들의 집합으로 프로그래밍의 상호작용을 표현
- 데이터를 객체로 취급
- 객체 내부에 선언된 메서드를 활용하는 방식

- 설계에 많은 시간 소요
- 처리 속도가 다른 프로그래밍 패러다임에 비해 상대적으로 느림

객체지향 프로그래밍의 특징 - 추상화, 캡슐화, 상속성, 다형성

추상화(abstraction)

- 복잡한 시스템으로부터 핵심적인 개념, 또는 기능을 간추려내는 것

캡슐화(encapsulation)

- 객체의 속성과 메서드를 하나로 묶고 일부를 외부에 감추어 은닉

상속성(inheritance)

- 상위 클래스의 특성을 하위 클래스가 이어받아서 재사용하거나 추가, 확장
- 코드의 재사용 측면, 계층적인 관계 생성, 유지 보수성 측면에서 중요

다형성(polymorphism)

- 하나의 메서드나 클래스가 다양한 방법으로 동작
- 대표적으로 오버로딩, 오버라이딩

오버로딩(overloading)

- 같은 이름을 가진 메서드를 여러 개 두는 것
- 메서드 타입, 매개변수의 유형, 개수 등으로 여러 개
- 컴파일 중 발생하는 *'정적'* 다형성

오버라이딩(overriding)

- 주로 메서드 오버라이딩을 말함
- 상위 클래스로부터 상속받은 메서드를 하위 클래스가 재정의하는 것을 말함
- 런타임 중 발생하는 *'동적'* 다형성

설계원칙

- 객체지향 프로그래밍 설계시 SOLID 원칙을 지킬 것

단일 책임 원칙(SRP, Single Responsibility Principle)

- 모든 클래스는 각각 하나의 책임만 갖는다

개방 폐쇄 원칙(OCP, Open Closed Principle)

- 유지 보수 사항이 생긴다면 코드를 쉽게 확장할 수 있도록 하고, 수정할 때는 닫혀 있어야 함

  → 즉, 기존의 코드는 잘 변경하지 않으면서도 확장은 쉽게 할 수 있어야 함

리스코프 치환 원칙(LSP, Liskov Substitution Principle)

- 프로그램의 객체는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 하는 것
- 클래스는 상속이 되기 마련이고 부모, 자식이라는 계층 관계가 만들어짐
  - 이 때 부모 객체에 자식 객체를 넣어도 시스템이 문제 없이 돌아가게 만드는 것을 말함

인터페이스 분리 원칙(ISP, Interface Segregation Principle)

- 하나의 일반적인 인터페이스보다 구체적인 여러 개의 인터페이스를 만들어야 하는 원칙

의존 역전 원칙(DIP, Dependency Inversion Principle)

- 자신보다 변하기 쉬운 것에 의존하던 것을 추상화된 인터페이스나 상위 클래스를 두어 변하기 쉬운 것의 변화에 영향받지 않게 하는 원칙

  → 즉, 상위 계층은 하위 계층의 변화에 대한 구현으로부터 독립해야 함

1.2.3 절차형 프로그래밍

- 로직이 수행되어야 할 연속적인 계산 과정으로 이루어짐
- 일이 진행되는 방식으로 그저 코드를 구현하기만 하면 됨
  - 코드의 가독성이 좋음
  - 실행속도가 빠름
- 그래서 계산이 많은 작업에 쓰임
- 단점
  - 모듈화하기 어려움
  - 유지보수성 떨어짐

1.2.4 패러다임의 혼합

- 상황과 맥락에 따라 조합하세요

chapter 2 네트워크

2.1 네트워크의 기초

- 네트워크 : 노드와 링크가 서로 연결되어 있으며 리소스를 공유하는 집합
- 노드 : 서버, 라우터, 스위치 등 네트워크 장비
- 링크 : 유선 또는 무선

2.1.1 처리량과 지연시간

- 좋은 네트워크 ?
  - 많은 처리량을 처리할 수 있고
  - 지연시간이 짧고
  - 장애 빈도가 적으며
  - 좋은 보안을 갖춘 네트워크

처리량(throughput)

- 링크 내에서 성공적으로 전달된 데이터의 양
- 보통 얼만큼 트래픽을 처리했는지를 나타냄
- '많은 트래픽을 처리한다 = 많은 처리량을 가진다'
- 단위 : bps(bits per second) - 초당 전송 또는 수신되는 비트 수
- 처리량에 영향을 주는 요인들
  - 사용자들이 많이 접속할 때마다 커지는 트래픽
  - 네트워크 장치 간의 대역폭
  - 네트워크 중간에 발생하는 에러
  - 장치의 하드웨어 스펙
- 트래픽 : 특정 시점에 링크 내에 '흐르는' 데이터의 양
  - 예 : 서버에 저장된 파일을 클라이언트가 다운로드할 때 발생되는 데이터의 누적량

- 트래픽이 많아졌다 = 흐르는 데이터가 많아졌다
- 처리량이 많아졌다 = 처리하는 트래픽이 많아졌다

대역폭

- 주어진 시간 동안 네트워크 연결을 통해 흐를 수 있는 최대 비트 수

지연시간(latency)

- 요청이 처리되는 시간
  - 어떤 메시지가 두 장치 사이를 왕복하는데 걸린 시간
- 지연시간에 영향을 주는 요인
  - 매체 타입(무선, 유선)
  - 패킷 크기
  - 라우터의 패킷 처리 시간

2.1.2 네트워크 토폴로지와 병목 현상

네트워크 토폴로지(network topology)

- 노드와 링크가 어떻게 배치되어 있는지에 대한 방식이자 연결 형태를 의미함

트리 토폴로지(tree topology) = 계층형 토폴로지

- 노드의 추가, 삭제가 쉬움
- 특정 노드에 트래픽이 집중될 때 하위 노드에 영향을 끼칠 수 있음

버스 토폴로지(bus topology)

- 중앙 통신 회선 하나에 여러 개의 노드가 연결되어 공유하는 네트워크 구성
- 근거리 통신망(LAN)에 사용

스푸핑

- 스위칭 기능을 마비시키거나 속여서 특정 노드에 해당 패킷이 오도록 처리하는 것
- 스위칭 기능? LAN 상에서 송신부의 패킷을 송신과 관련 없는 다른 호스트에 가지 않도록 하는 기능
- 스푸핑을 적용하면 올바르게 수신부로 가야 할 패킷이 악의적인 노드에 전달됨

스타 토폴로지(star topology)

- 중앙에 있는 노드에 모두 연결된 네트워크 구성

장점

- 노드를 추가하거나 에러를 탐지하기 쉬움
- 패킷의 충돌 발생 가능성이 적음
- 어떤 노드에 장애가 발생해도 쉽게 에러를 발견할 수 있음
- 장애 노드가 중앙 노드가 아닐 경우 다른 노드에 영향을 끼치는게 적음

단점

- 중앙 노드에 장애가 발생하면 전체 네트워크를 사용할 수 없음
- 설치 비용이 고가

링형토폴로지(ring topology)

- 각각의 노드가 양 옆의 두 노드와 연결
- 고리처럼 하나의 연속된 길을 통해 통신
- 데이터는 노드에서 노드로 이동, 각각의 노드는 고리 모양의 길을 통해 네트워크 처리

장점

- 노드 수가 증가되어도 네트워크상의 손실이 거의 없음
- 충돌이 발생되는 가능성이 적음
- 노드의 고장 발견을 쉽게 찾을 수 있음

단점

- 네트워크 구성 변경이 어려움
- 회선에 장애가 발생하면 전체 네트워크에 영향을 크게 끼침

메시 토폴로지(mesh topology)

- 망형 토폴로지, 그물망처럼 연결됨

장점

- 한 단말 장치에 장애가 발생해도 여러 개의 경로가 존재
- 따라서 네트워크를 계속 사용할 수 있음
- 트래픽도 분산 처리가 가능함

단점

- 노드의 추가가 어려움
- 구축 비용과 운용 비용이 고가임

병목현상

- 토폴로지가 중요한 이유?
  - 병목 현상을 찾을 때 중요한 기준이 되기 때문
- 병목현상시 토폴로지 형태를 확인하고 어떤 경로인지 알면 병목 현상을 잘 해결할 수 있음

병목 현상(bottleneck)

- 전체 시스템의 성능이나 용량이 하나의 구성 요소로 인해 제한을 받는 현상
- 서비스에서 이벤트를 열었을 때 트래픽이 많이 생기고, 그 트래픽을 잘 관리하지 못하면 병목 현상이 생겨 사용자는 웹 사이트로 들어가지 못하게 됨
- 병목 현상의 주된 원인
  - 네트워크 대역폭
  - 네트워크 토폴로지
  - 서버 CPU, 메모리 사용량
  - 비효율적인 네트워크 구성

2.1.3 네트워크 분류

- 규모 기반으로 분류

LAN(Local Area Network)

- 근거리 통신망
- 같은 건물이나 캠퍼스 등 좁은 공간에서 운영
- 전송 속도가 빠르고 혼잡하지 않다

MAN(Metropolitan Area Network)

- 대도시 지역 네트워크
- 도시 같은 넓은 지역에서 운영
- 전송 속도는 평균, LAN보다 혼잡

WAN(Wide Area Network)

- 광역 네트워크
- 국가 또는 대륙 같은 더 넓은 지역에서 운영
- 전송 속도 낮음, MAN보다 혼잡

2.1.4 네트워크 성능 분석 명령어

ping(Packet INternet Groper)

- 네트워크 상태를 확인하려는 대상 노드를 향해 일정 크기의 패킷을 전송하는 명령어
- 이를 통해 해당 노드의 패킷 수신 상태와 도달하기까지의 시간 등을 알 수 있음
- 해당 노드까지 네트워크가 잘 연결되어 있는지 확인 가능
- TCP/IP 프로토콜 중 ICMP 프로토콜을 통해 동작
  - ICMP 프로토콜을 지원하지 않는 기기를 대상으로는 실행 못함
  - 네트워크 정책상 ICMP나 traceroute를 차단하면 ping 테스팅 불가능

netstat

- 접속되어 있는 서비스들의 네트워크 상태를 표시
- 네트워크 접속, 라우팅 테이블, 네트워크 프로토콜 등 리스트를 보여줌
- 주로 서비스의 포트가 열려있는지 확인할 때 씀

nslookup

- DNS에 관련된 내용을 확인하기 위해 씀
- 특정 도메인에 매핑된 IP주소를 확인할 때 씀

tracert

- 윈도우 : tracert
- 리눅스 : traceroute
- 목적지 노드까지 네트워크 경로를 확인할 때 사용
- 목적지 노드까지 구간들 중 어느 구간에서 응답 시간이 느려지는지 등을 확인

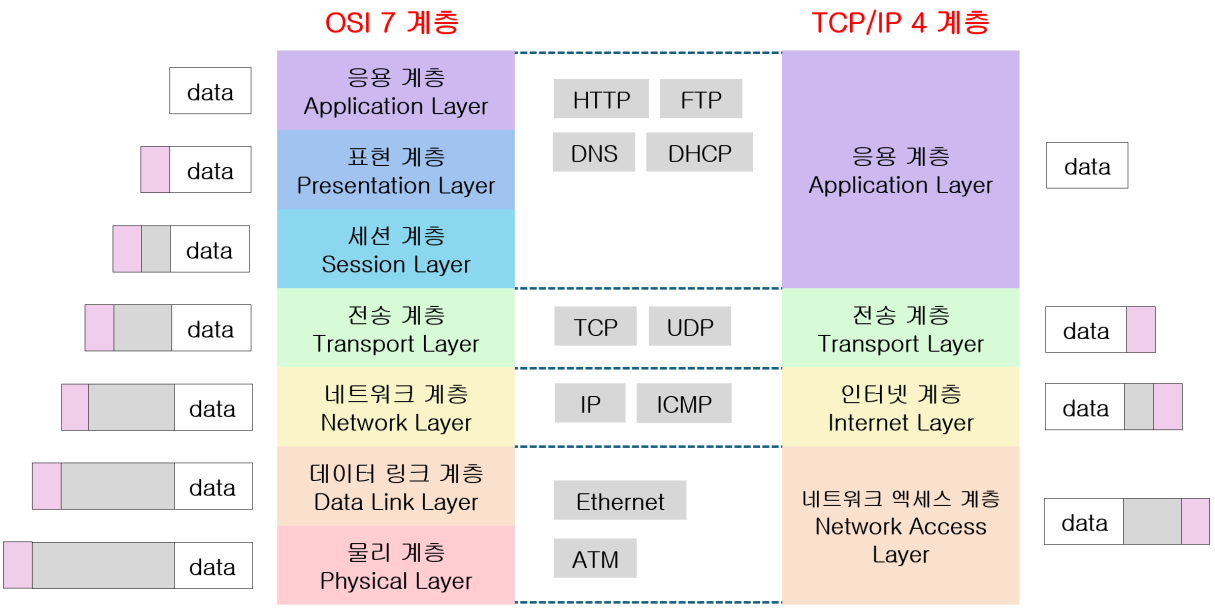
2.1.5 네트워크 프로토콜 표준화

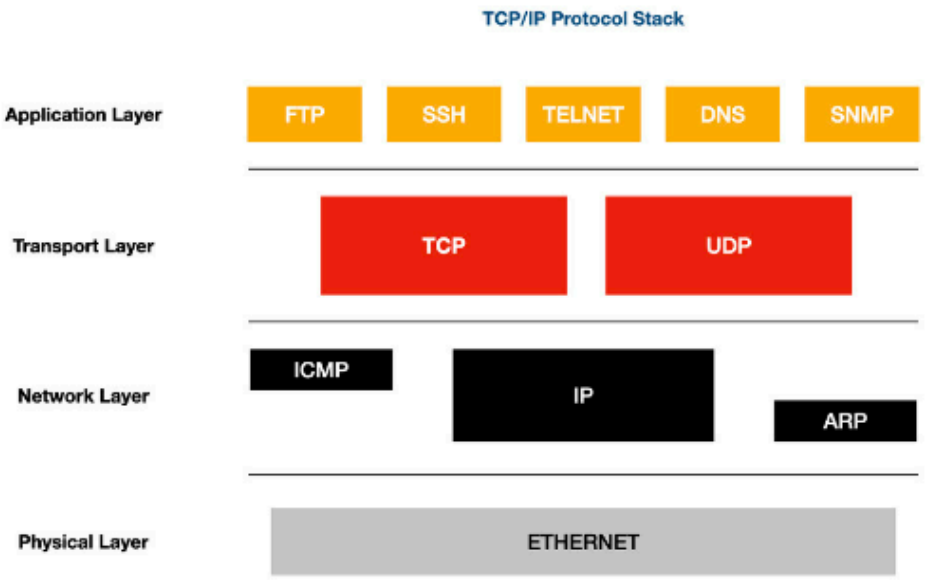
- 네트워크 프로토콜 : 다른 장치들끼리 데이터를 주고 받기 위해 설정된 공통된 인터페이스
- IEEE 또는 IETF라는 표준화 단체가 정함
- IEEE802.3 : 유선 LAN 프로토콜

2.2 TCP/IP 4계층 모델

- 인터넷 프로토콜 스위트(internet protocol suite) : 인터넷에서 컴퓨터들이 서로 정보를 주고받는 데 쓰이는 프로토콜의 집합
  - TCP/IP 4계층 모델로 설명하거나 OSI 7계층 모델로 설명하기도 함
- 특정 계층이 변경되었을 때 다른 계층이 영향을 받지 않도록 설계됨

2.2.1 계층 구조





애플리케이션 계층(application layer)

- FTP, HTTP, SSH, SMTP, DNS
- 응용 프로그램이 사용되는 프로토콜 계층
- 웹서비스, 이메일 등 서비스를 실질적으로 사람들에게 제공하는 층

💡 **FTP**

- 장치와 장치 간의 파일을 전송하는 데 사용되는 표준 통신 프로토콜

SSH

- 보안되지 않는 네트워크에서 네트워크 서비스를 안정하게 운영하기 위한 암호화 네트워크 프로토콜

HTTP

- World Wide Web을 위한 데이터 통신의 기초이자 웹사이트를 이용하는 데 쓰는 프로토콜

SMTP

- 전자 메일 전송을 위한 인터넷 표준 통신 프로토콜

DNS

- 도메인과 IP 주소를 매핑해주는 서버

전송계층(transport layer)

- 송신자와 수신자를 연결하는 통신 서비스 제공
- 연결 지향 데이터 스트림 지원, 신뢰성, 흐름 제어를 제공할 수 있음
- 애플리케이션과 인터넷 계층 사이의 데이터가 전달될 때 중계 역할
- TCP, UDP

가상회선 패킷 교환 방식

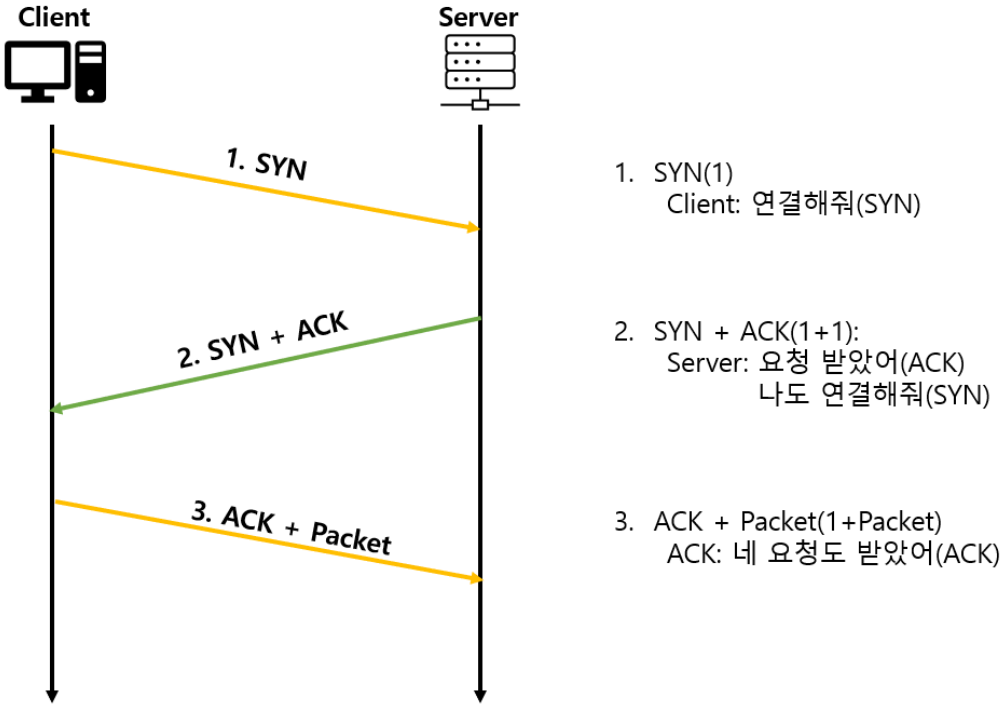
- 각 패킷에 가상회선 식별자가 포함
- 모든 패킷을 전송하면 가상회선이 해제되고 패킷들은 전송된 '순서대로' 도착하는 방식

데이터그램 패킷 교환 방식

- 패킷이 독립적으로 이동, 최적의 경로를 선택하여 감
- 하나의 메시지에서 여러 패킷을 분할하여 서로 다른 경로로 전송될 수 있음
- 도착한 '순서가 다를 수 있음'

TCP 연결 성립 과정

- TCP는 신뢰성 확보시 '3-웨이 핸드셰이크'를 진행함 (3-way handshake)



1. SYN 단계
    a. 클라이언트는 서버에 클라이언트의 ISN을 담아 SYN을 보냄
    b. ISN : 새로운 TCP 연결의 첫 번째 패킷에 할당된 임의의 시퀀스 번호 (장치마다 다를 수 있음)
2. SYN + ACK단계
    a. 서버는 클라이언트의 SYN을 수신
    b. 서버의 ISN과 승인번호를 보냄
    c. 서버의 승인번호는 클라이언트의 ISN + 1
3. ACK 단계
    a. 클라이언트는 승인번호를 담아 ACK를 서버로 보냄

b. 클라이언트의 승인 번호는 서버의 ISN + 1

- 3-웨이 핸드셰이크 과정 이후 신뢰성이 구축되고 데이터 전송을 시작함
- TCP는 이 과정 있음 ⇒ 신뢰성이 있는 계층
- UDP는 이 과정이 없음 ⇒ 신뢰성 없는 계층

SYN

- SYNchronization, 연결 요청 플래그

ACK

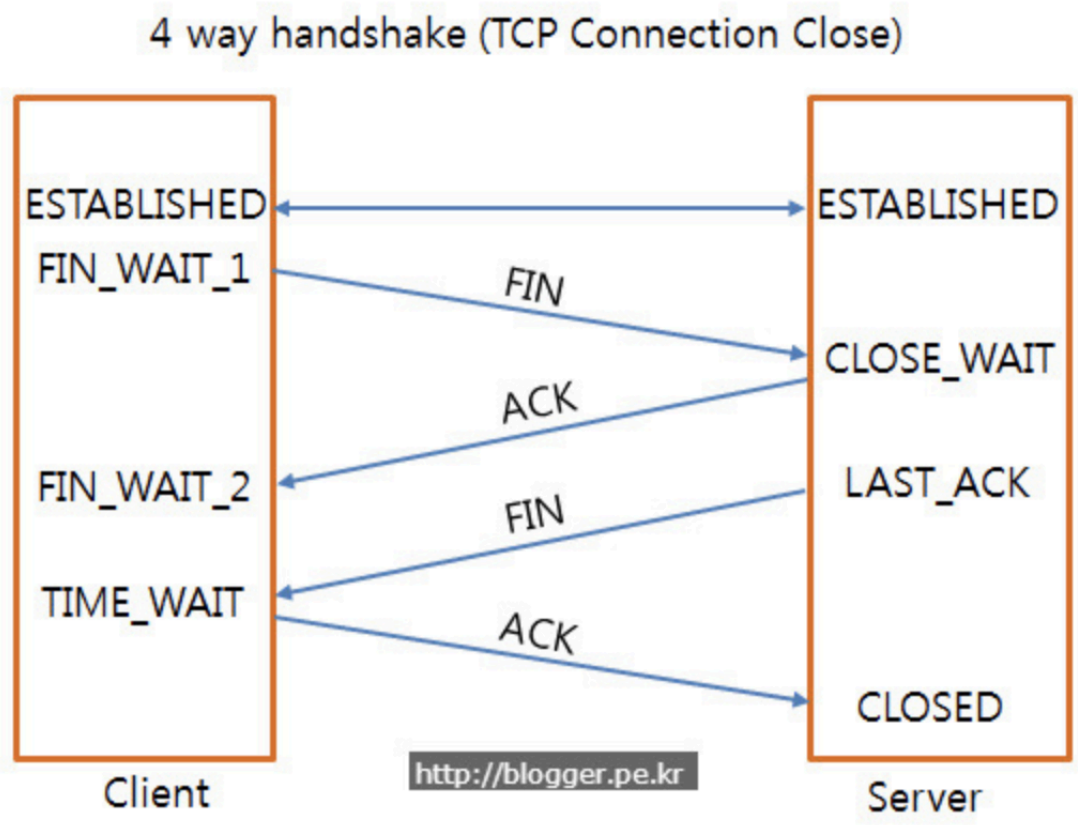
- ACKnowledgement, 응답 플래그

ISN

- Initial Sequence Numbers, 초기 네트워크 연결시 할당된 32비트 고유 시퀀스 번호

TCP 연결 해제 과정

- '4-웨이 핸드셰이크' 발생



1. FIN_WAIT_1
    a. 클라이언트가 연결 닫을 때 FIN으로 설정된 세그먼트를 보냄
    b. 클라이언트는 FIN_WAIT_1 들어가고 서버의 응답을 기다림
2. CLOSE_WAIT
    a. 서버는 클라이언트에 ACK라는 승인 세그먼트를 보냄
    b. 그리고 CLOSE_WAIT 상태에 들어감
    c. 클라이언트가 세그먼트를 받으면 FIN_WAIT_2 상태로 들어감
3. LAST ACK
    a. 서버는 ACK를 보내고 일정시간 이후 클라이언트에 FIN을 보냄
4. TIME_WAIT
    a. 클라이언트는 TIME_WAIT 상태가 됨
    b. 다시 서버로 ACK를 보냄
    c. 서버는 CLOSED 상태가 됨
    d. 클라이언트는 일정 시간 대기 후 연결이 닫힘

e. 클라이언트와 서버의 모든 자원의 연결이 해제됨

- TIME_WAIT
  - 왜 일정시간 뒤에 닫을까?
    1. 지연 패킷이 발생할 경우를 대비(데이터 무결성 문제 방지)
    2. 두 장치가 연결이 닫혔는지 확인하기 위함(LASK_ACK 상태시 접속 오류)

TIME_WAIT

- 소켓이 바로 소멸되지 않고 일정 시간 유지하는 형태
- 지연 패킷 등의 문제점을 해결하는데 쓰임
- CentOS6, 우분투 - 60초, 윈도우 - 4분

데이터 무결성

- 데이터의 정확성과 일관성을 유지하고 보증하는 것

인터넷 계층(internet layer)

- 장치에서 받은 네트워크 패킷을 IP 주소로 된 목적지로 전송하기 위해 사용하는 계층
- IP, ARP, ICMP 등
- 패킷을 수신해야 할 상대의 주소를 지정하여 데이터 전달
- 비연결형적인 특징 : 상대방이 제대로 받았는지에 대해 보장하지 않음

링크계층(link layer) = 네트워크 접근 계층

- 전선, 광섬유, 무선 등으로 실질적으로 데이터 전달
- 장치 간에 신호를 주고받는 '규칙'을 정하는 계층
- 링크 계층을 물리 계층과 데이터 링크 계층으로 나누기도 함
  - 물리 계층 : 무선 LAN과 유선 LAN을 통해 0과 1로 이루어진 데이터를 보내는 계층

    - 데이터 링크 계층 : '이더넷 프레임'을 통해 에러 확인, 흐름 제어, 접근 제어를 담당하는 계층

유선LAN(IEEE802.3)

- 유선LAN을 이루는 이더넷은 IEEE802.3 프로토콜을 따름
- 전이중화 통신을 사용

전이중화 통신?(full duplex)

- 양쪽 장치가 동시에 송수신할 수 있는 방식
- 송신로와 수신로를 나눠서 데이터를 주고 받음
- 현대의 고속 이더넷

CSMA/CD?

- 예전 유선 LAN 방식
- 데이터를 '보낸 후' 충돌이 발생하면 일정 시간 이후 재전송하는 방식
    - 송신로와 수신로가 분리되지 않음
    - 한 경로 기반으로 데이터를 보냄

유선LAN을 이루는 케이블

- TP(트위스트 페어) 케이블과 광섬유 케이블이 대표적

무선LAN(IEEE802.11)

- 무선 LAN 장치는 수신과 송신에 같은 채널을 사용
- 반이중화 통신 사용

반이중화 통신? (half duplex)

- 양쪽 장치는 서로 통신할 수 있지만 동시에 통신할 수는 없음
- 한 번에 한 방향만 통신할 수 있는 방식

- 일반적으로 장치가 신호를 수신하기 시작하면 전송이 완료될 때까지 기다려야 함
- 둘 이상의 장치가 동시에 전송되면 충돌이 발생 → 충돌 방지 시스템 필요

CSMA/CD의 과정

1. 사용 중인 채널이 있다면 다른 채널을 감지하다 유후 상태인 채널 발견
2. IFS 시간만큼 기다림
    a. IFS(InterFrame Space) : 프레임 간 공간 시간
    b. IFS는 프레임의 우선 순위를 정의할 때도 사용
    c. IFS가 낮으면 우선순위가 높다
3. 프레임을 보내기 전 0 ~ 2^k -1 사이에서 결정된 랜덤 상수를 기반으로 결정된 시간만큼 기다린 후 프레임을 보냄
    a. 프레임이 제대로 송신되고 ACK를 받으면 마침
    b. 받지 못하면 k = k+1하여 과정 반복
    c. k가 정해진 Kmax보다 커지면 해당 프레임 전송을 버림(abort)

무선LAN을 이루는 주파수

- 무선 신호 전달 방식을 이용하여 2대 이상의 장치를 연결하는 기술
- 주파수 대역은 2.4GHz 또는 5GHz
    - 2.4GHz : 장애물에 강함, 전파 간섭이 잘 일어남
    - 5GHz : 사용 가능한 채널 수 많음, 동시 사용 가능 ⇒ 상대적으로 깨끗한 전파 환경 구축

와이파이?(wifi)

- 전자기기들이 무선 LAN 신호에 연결할 수 있게 하는 기술
- 쓰려면 무선 접속 장치(AP, Access, Point) 필요 = 공유기
- 유선 LAN 신호를 무선 LAN 신호로 바꿔줌
- 무선 LAN 사용 신기술 : 와이파이, 지그비, 블루투스 등

BSS? (Basic Server Set)

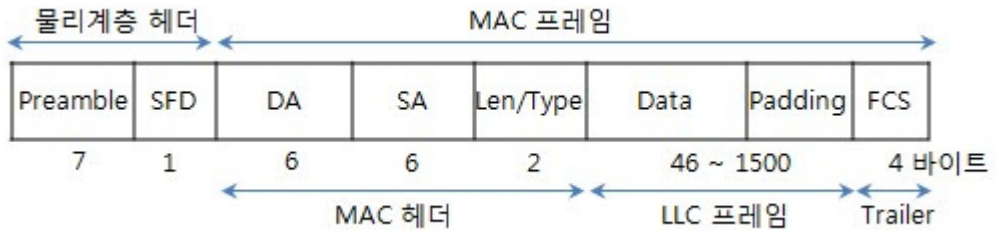
- 기본 서비스 집합
- 단순 공유기를 통해서 네트워크에 접속하는 게 아니라!
- 동일 BSS 내에 있는 AP들과 장치들이 서로 통신이 가능한 구조를 말함
- 근거리 무선통신 제공
- 하나의 AP만을 기반으로 구축됨
- 사용자가 자유롭게 이동하며 네트워크에 접속하는 것은 불가능

ESS? (Extended Server Set)

- 하나 이상의 BSS 그룹
- 장거리 무선 통신을 제공
- BSS보다 더 많은 가용성과 이동성 지원

이더넷 프레임

- 데이터 링크 계층은 이더넷 프레임을 통해 전달받은 데이터의 에러를 검출하고 캡슐화함
- 다음과 같은 구조

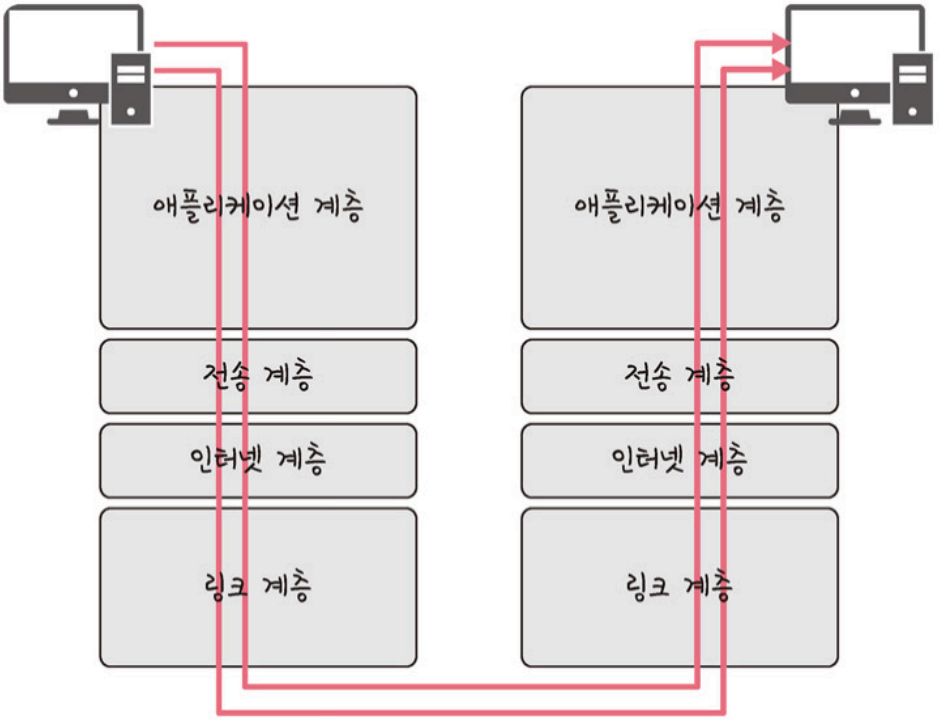


- Preamble : 이더넷 프레임이 시작임을 알림
- SFD(Start Frame Delimiter) : 다음 바이트부터 MAC 주소 필드가 시작임을 알려줌
- DMAC, SMAC : 수신, 송신 MAC 주소
- EtherType : 데이터 계층 위의 계층인 IP 프로토콜 정의(IPv4, IPv6)
- Payload : 전달받은 데이터

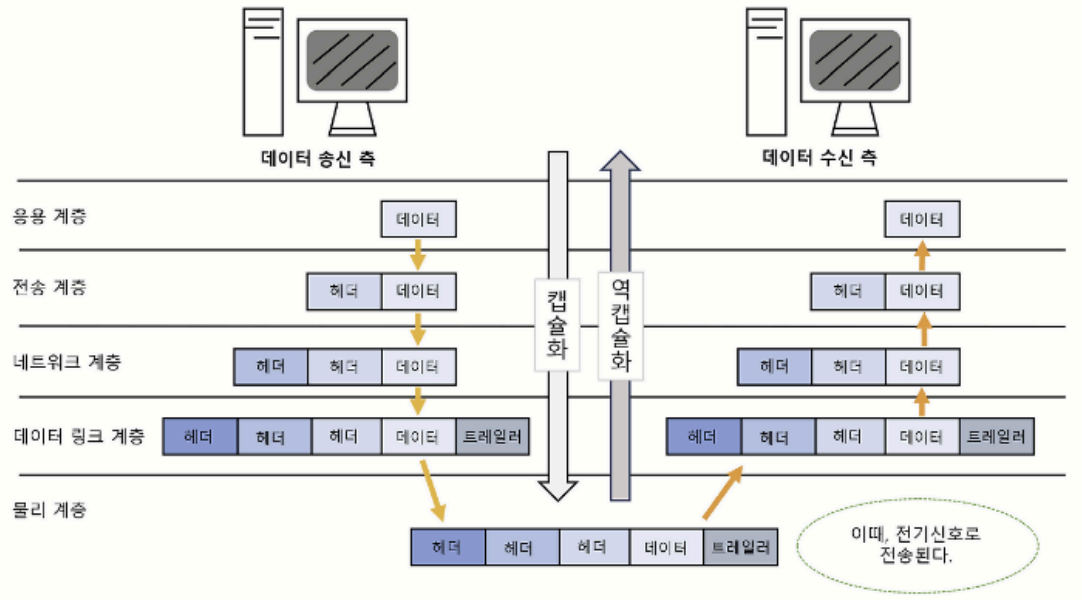
- CRC : 에러 확인 비트

> 💡 **MAC 주소**
>
> - LAN카드 구별하기 위한 식별번호
> - 6바이트(48비트)로 구성

계층간 데이터 송수신 과정



- 송신시 : 캡슐화
- 수신시 : 비캡슐화



캡슐화와 역캡슐화

캡슐화과정

- 상위 계층의 헤더와 데이터를 하위 계층의 데이터 부분에 포함시키고 해당 계층의 헤더를 삽입하는 과정
- '세그먼트' 또는 '데이터그램' → '패킷' → '프레임'

비캡슐화과정

- 하위 계층에서 상위 계층으로 감
- 각 계층의 헤더 부분을 제거
- '프레임' → '패킷' → '세그먼트' 또는 '데이터그램' → '메시지'
- 최종적으로 애플리케이션의 PDU인 메시지로 전달됨

2.2.2 PDU(Protocol Data Unit)

- 네트워크의 어떠한 계층에서 계층으로 데이터가 전달될 때 한 덩어리의 단위
- '헤더'와 '페이로드'로 구성되며 계층마다 부르는 명칭이 다름
  - 헤더 : 제어 관련 정보들을 포함
  - 페이로드 : 데이터

  → 애플리케이션 계층 : 메시지

전송 계층 : 세그먼트(TCP), 데이터그램(UDP)

인터넷 계층 : 패킷

링크 계층 : 프레임(데이터 링크 계층), 비트(물리 계층)

- PDU 중 아래 계층인 비트로 송수신하는 것이 모든 PDU 중 가장 빠르고 효율성이 높음
- 그러나 애플리케이션에서는 문자열을 기반으로 송수신함
  - 헤더에 authorization 값 등 다른 값들을 넣는 확장이 쉽기 때문

2.3 네트워크 기기

2.3.1 네트워크 기기의 처리 범위

- 애플리케이션 계층 : L7 스위치
- 인터넷 계층 : 라우터, L3 스위치
- 데이터 링크 계층 : L2 스위치, 브리지
- 물리 계층 : NIC, 리피터, AP

2.3.2 애플리케이션 계층을 처리하는 기기

L7 스위치

- 스위치 : 여러 장비를 연결하하고 데이터 통신을 중재하며 목적지가 연결된 포트로만 전기 신호를 보내 데이터를 전송하는 통신 네트워크 장비
- L7 스위치 = 로드밸런서
- 서버의 부하를 분산
- 클라이언트로부터 오는 요청들을 뒤쪽의 여러 서버로 나누는 역할
- 시스템이 처리할 수 있는 트래픽 증가를 목표로 함
- URL, 서버, 캐시, 쿠키들을 기반으로 트래픽을 분산
- 필터링 기능으로 바이러스, 불필요한 외부 데이터 등을 걸러냄
- 응용 프로그램 수준의 트래픽 모니터링도 가능
- 장애가 발생한 서버가 있다면 트래픽 분산 대상에서 제외해야 함

    - 정기적으로 헬스 체크를 이용하여 감시하면서 이루어짐

L4 스위치와 L7 스위치의 차이

- 로드밸런서 → L4스위치, L7스위치
- L4 스위치
    - 전송 계층을 처리하는 기기
    - 스트리밍 관련 서비스는 사용할 수 없음
    - 메시지 기반으로 인식하지 못함
    - IP와 포트를 기반으로(특히 포트) 트래픽 분산
- L7 스위치
    - IP, 포트, URL, HTTP헤더, 쿠키 등을 기반으로 트래픽 분산
- 클라우드 서비스(AWS 등)에서 L7 스위치를 이용한 로드 밸런싱은 ALB 컴포넌트로 함
- L4 스위치를 이용한 로드 밸런싱은 NLB 컴포넌트로 함

헬스 체크

- L4, L7 스위치 모두 헬스 체크를 통해 서버가 정상인지 비정상인지 판별
- 헬스체크 : 전송 주기, 재전송 회수 등을 설정해서 반복적으로 서버에 요청을 보내는 것
    - 서버가 부하되지 않을 만큼 요청 횟수가 적절해야 함

로드밸런서를 이용한 서버 이중화

- 로드밸런서의 대표적인 기능은 서버 이중화
- 서비스의 안정적인 운용을 위해서는 2대 이상의 서버가 필수적
- 로드밸런서는 2대 이상의 서버를 기반으로 가상 IP를 제공, 이를 기반으로 안정적인 서비스 제공

2.3.3 인터넷 계층을 처리하는 기기

라우터(router)

- 여러 개의 네트워크를 연결, 분할, 구분시켜주는 역할을 함

- “다른 네트워크에 존재하는 장치끼리 서로 데이터를 주고받을 때 패킷 소모를 최소화하고 경로를 최적화하여 최소 경로로 패킷을 포워딩” → 라우팅하는 장비

L3 스위치

- L2스위치의 기능과 라우팅 기능을 갖춘 장비
- L3스위치를 라우터라고 해도 됨
- 라우터 : 소프트웨어 기반의 라우팅과 하드웨어 기반의 라우팅으로 나눠짐
- L3 스위치는 하드웨어 기반의 라우팅을 담당

구분	L2 스위치	L3 스위치
참조 테이블	MAC 주소 테이블	라우팅 테이블
참조 PDU	이더넷 프레임	IP 패킷
참조 주소	MAC 주소	IP 주소

2.3.4 데이터링크 계층을 처리하는 기기

L2 스위치

- 장치들의 MAC 주소를 MAC 주소 테이블을 통해 관리
- 연결된 장치로부터 패킷이 왔을 때 패킷 전송을 담당함
- IP 주소를 기반으로 라우팅은 불가능
- 단순히 패킷의 MAC 주소를 읽어 스위칭 하는 역할을 함
- 목적지가 MAC 주소 테이블에 없다면 전체 포트로 전달하고 MAC 주소 테이블의 주소는 일정 시간 이후 삭제

브리지(bridge)

- 두 개의 근거리 통신망(LAN)을 상호 접속할 수 있도록 하는 통신망 연결 장치
- 포트와 포트 사이의 다리 역할
- 장치에서 받아온 MAC 주소를 MAC 주소 테이블로 관리함

- 통신망 범위를 확장하고 서로 다른 LAN 등으로 이루어진 '하나의' 통신망을 구축할 때 쓰임

2.3.5 물리 계층을 처리하는 기기

NIC(Network Interface Card)

- LAN 카드라고도 함
- 2대 이상의 컴퓨터 네트워크를 구성하는데 사용
- 네트워크와 빠른 속도로 데이터를 송수신할 수 있도록 컴퓨터 내에 설치하는 확장 카드
- 각 LAN 카드에는 주민등록번호같은 MAC 주소가 있음

리피터(repeater)

- 수신되는 신호를 증폭하여 다른 쪽으로 전달하는 장치
- 이를 통해 패킷이 멀리 갈 수 있음
- 근데 광케이블 보급으로 요즘 잘 안씀

AP(Access Point)

- 패킷을 복사하는 기기
- AP에 유선 LAN을 연결한 후 다른 장치에서 무선 LAN 기술(와이파이 등)을 사용하여 무선 네트워크 연결을 할 수 있음

2.4 IP 주소

- 컴퓨터와 컴퓨터 간 통신
    - IP 주소에서 ARP를 통해 MAC 주소를 찾아서 MAC 주소 기반으로 통신함

2.4.1 ARP(Address Resolution Protocol)

- IP 주소로부터 MAC 주소를 구함
- IP와 MAC 주소의 다리 역할을 하는 프로토콜

- ARP를 통해 가상 주소인 IP 주소를 실제 주소인 MAC 주소로 변환
- RARP를 통해서는 MAC 주소를 IP 주소로 변환

2.4.2 홉바이홉 통신

- IP 주소를 통해 통신하는 과정을 홉바이홉 통신이라고 함
- 통신 장치에 있는 '라우팅 테이블'의 IP를 통해 시작 주소로부터 시작하여 다음 IP로 계속해서 이동하는 '라우팅' 과정을 거쳐 패킷이 최종 목적지까지 도달하는 통신을 말함

라우팅 테이블(routing table)

- 송신지에서 수신지까지 도달하기 위해 사용됨
- 라우터에 들어가 있는 목적지 정보들과 그 목적지로 가기 위한 방법이 들어있는 리스트

게이트웨이(gateway)

- 서로 다른 통신망, 프로토콜을 사용하는 네트워크 간의 통신을 가능하게 하는 관문 역할을 하는 컴퓨터나 소프트웨어 등을 두루 일컫는 말

2.4.3 IP 주소 체계

- IPv4 : 32비트를 8비트 단위로 점을 찍어 표기함
- IPv6 : 64비트를 16비트 단위로 점을 찍어 표기함

클래스 기반 할당 방식(classful network addressing)

- IP 주소 체계는 처음에는 A,B,C,D,E 다섯 개의 클래스로 구분하는 이 방식을 사용했음
- 앞에 있는 부분이 네트워크 주소, 그 뒤에 있는 부분이 컴퓨터에 부여하는 주소인 호스트 주소
- A,B,C → 일대일 통신
- D → 멀티캐스트 통신
- E → 예비용

DHCP(Dynamic Host Configuration Protocol)

- IP 주소 및 기타 통신 매개변수를 자동으로 할당하기 위한 네트워크 관리 프로토콜

- 네트워크 장치의 IP 주소를 수동으로 설정할 필요 없이 인터넷에 접속할 때마다 자동으로 IP 주소를 할당
- 많은 라우터와 게이트웨이 장비에 DHCP 기능이 있으며 이를 통해 대부분의 가정용 네트워크에서 IP주소를 할당

NAT(Network Address Translation)

- 패킷이 라우팅 장치를 통해 전송되는 동안 패킷의 IP 주소 정보를 수정하여 IP 주소를 다른 주소로 매핑하는 방법
- IPv4 주소 체계만으로는 많은 주소를 감당하지 못함
  - 이를 해결하기 위해 NAT으로 공인 IP와 사설 IP를 나눠서 처리
- ICS, RRAS, Netfilter를 통해 NAT을 씀?

공유기와 NAT

- NAT을 쓰는 이유는 주로 여러 대의 호스트가 하나의 공인 IP 주소를 사용하여 인터넷에 접속하기 위함
- 예 : 인터넷 회선 하나를 개통하고 공유기를 달아서 여러 PC를 연결해서 사용할 수 있음
  - 이게 가능한 이유는 인터넷 공유기에 NAT이 달려있어서

NAT을 이용한 보안

- NAT을 이용하면 내부 네트워크에서 사용하는 IP 주소와 외부에 드러나는 IP 주소를 다르게 유지할 수 있음
- 이를 통해 내부 네트워크에 대한 어느 정도의 보안이 가능해짐

NAT의 단점

- 여러 명이 동시에 인터넷에 접속하므로 접속 속도가 느려질 수 있음

2.4.4 IP 주소를 이용한 위치 정보

- IP 주소로 위치를 알 수 있어용

2.5 HTTP

- HTTP는 애플리케이션 계층으로서 웹서비스 통신에 사용됨

2.5.1 HTTP/1.0

- 한 연결 당 하나의 요청 처리
- RTT 증가를 불러옴(서버에서 파일을 가져올 때마다 3 way handshake를 계속해서 열어야 하므로)

> 💡 **RTT**
>
> - 패킷이 목적지에 도달하고 나서 다시 출발지로 돌아오기까지 걸리는 시간
> - 패킷 왕복 시간

RTT의 증가를 해결하기 위한 방법

이미지 스플리팅

- 많은 이미지가 합쳐져 있는 하나의 이미지를 다운받음
- 이를 기반으로 background-image의 position을 이용하여 이미지를 표시

코드 압축

- 코드를 압축해서 개행문자, 빈칸을 없앰 → 코드 크기 최소화

이미지 Base64 인코딩

- 이미지 파일을 64진법으로 이루어진 문자열로 코딩
- 장점
  - 서버와의 연결을 열고 이미지에 대해 서버에 HTTP 요청할 필요 없음
- 단점
  - 인코딩 후 크기가 커짐

2.5.2 HTTP/1.1

- 매번 TCP 연결을 하는게 아니라 한 번 TCP 초기화한 후 keep-alive라는 옵션으로 여러 개의 파일을 송수신

- HTTP/1에도 keep-alive가 있지만 HTTP/1.1에서 표준화되어 기본 옵션으로 설정됨
- 장점
  - 한 번 TCP 3-웨이 핸드셰이크가 발생하면 그 다음부터는 발생하지 않음
- 단점
  - 문서 안에 있는 다수의 리소스를 처리하려면 요청할 리소스 개수에 비례해서 대기 시간이 길어짐

HOL Blocking(Head Of Line Blocking)

- 네트워크에서 같은 큐에 있는 패킷이 그 첫 번째 패킷에 의해 지연될 때 발생하는 성능 저하 현상

무거운 헤더 구조

- HTTP/1.1의 헤더에는 쿠키 등 많은 메타데이터가 들어있고 압축이 되지 않아 무거움

2.5.3 HTTP/2

- SPDY 프로토콜에서 파생된 HTTP/1.x보다
  - 지연시간을 줄이고
  - 응답 시간을 더 빨리할 수 있고
  - 멀티플렉싱, 서버 푸시, 헤더 압축, 요청의 우선순위 처리를 지원

멀티플렉싱

- 여러 개의 스트림을 사용하여 송수신하는 것
- 특정 스트림의 패킷이 손실되었더라도 해당 스트림에만 영향을 미치고 나머지 스트림은 멀쩡하게 동작
- 멀티플렉싱을 통해 단일 연결을 사용하여 병렬로 여러 요청을 받을 수 있고 응답을 줄 수 있음
- HTTP/1.x에서 발생하는 문제인 HOP blocking을 해결할 수 있음

💡 **스트림(stream)**

- 시간이 지남에 따라 사용할 수 있게 되는 일련의 데이터 요소를 가리키는 데이터 흐름

헤더 압축

- HTTP/1.x : 헤더의 크기가 너무 컸음
- HTTP/2 : 헤더 압축이 가능
  - 허프만 코딩 압축 알고리즘을 사용하는 HPACK 압축 형식을 가짐

허프만 코딩(Huffman Coding)

- 문자열을 문자 단위로 쪼개 빈도수를 셈
  - 빈도가 높은 정보는 적은 비트 수를 사용하여 표현
  - 빈도가 낮은 정보는 비트 수를 많이 사용
  - 전체 데이터의 표현에 필요한 비트 양을 줄임

서버푸시

- HTTP/1.x : 클라이언트가 서버에 요청해야 파일을 다운받을 수 있음
- HTTP/2 : 클라이언트 요청 없이 서버가 바로 리소스를 푸시할 수 있음

2.5.4 HTTPS

- HTTP/2는 HTTPS 위에서 동작
- HTTPS : 애플리케이션 계층과 전송 계층 사이에 신뢰계층인 SSL/TLS 계층을 넣음
  - 그래서 HTTPS는 신뢰할 수 있는 HTTP 요청을 말함
  - '통신을 암호화'

SSL/TLS

- SSL(Secure Socket Layer)는 버전이 올라가면서 TLS로 명칭이 변경됨
- 전송 계층에서 보안을 제공하는 프로토콜

- 클라이언트와 서버가 통신할 때 SSL/TLS를 통해 제3자가 메시지를 도청하거나 변조하지 못하게 함
- 네트워크 상 '인터셉터'를 방지 → 공격자가 서버인 척 하면서 사용자 정보를 가로채는 것
- SSL/TLS는 보안 세션을 기반으로 데이터를 암호화함
    - 보안 세션이 만들어질 때 인증 메커니즘, 키 교환 암호화 알고리즘, 해싱 알고리즘 사용

보안세션

- 보안이 시작되고 끝나는 동안 유지되는 세션
- SSL/TLS는 핸드셰이크를 통해 보안 세션을 생성하고 이를 기반으로 상태 정보 등을 공유함

세션

- 운영체제가 어떠한 사용자로부터 자신의 자산 이용을 허락하는 일정한 기간을 뜻함
- 사용자는 일정 시간 동안 응용 프로그램, 자원을 사용할 수 있음

인증 메커니즘

- 인증 메커니즘은 CA(Certificate Authorities)에서 발급한 인증서를 기반으로 이루어짐
- CA에서 발급한 인증서는 '공개키'를 클라이언트에게 제공, 사용자가 접속한 서버가 '신뢰할 수 있는 서버'임을 보장
- 서비스 정보, 공개키, 지문, 디지털 서명 등으로 이루어짐

암호화 알고리즘

- ECDHE 또는 모듈식 기반의 DHE를 사용 → 둘 다 디피-헬만 방식을 근간으로 만듦

디피-헬만 키 교환 암호화 알고리즘?

- 암호키를 교환하는 방법
  1. 처음에 공개 값을 공유
  2. 각자의 비밀 값과 혼합 후 혼합 값을 공유
  3. 각자의 비밀 값과 혼합
  4. 공통의 암호키인 PSK 생성

해싱 알고리즘

- 데이터를 추정하기 힘든 더 작고, 섞여 있는 조각으로 만드는 알고리즘
- SHA-256 알고리즘과 SHA-384 알고리즘 사용

SHA-256 알고리즘?

- 해시 함수 결괏값이 256비트인 알고리즘
- 해싱해야 할 메시지에 전처리하고 전처리된 메시지를 기반으로 해시를 반환

해시

- 다양한 길이를 가진 데이터를 고정된 길이를 가진 데이터로 매핑한 값

해싱

- 임의의 데이터를 해시로 바꿔주는 일
- 해시 함수가 함

해시함수

- 임의의 데이터를 입력받아 일정한 길이의 데이터로 바꿔주는 함수

SEO에도 도움이 되는 HTTPS

- 구글 : HTTPS 서비스를 하는 사이트가 그렇지 않은 사이트보다 SEO 순위가 높을 것이라고 함
- SEO(Search Engine Optimization) : 검색 엔진 최적화
  - 검색엔진으로 웹 사이트를 검색했을 때 그 결과를 페이지 상단에 노출시켜 많은 사람이 볼 수 있도록 최적화하는 방법
  - 이를 위해 케노니컬 설정, 메타 설정, 페이지 속도 개선, 사이트맵 관리 등을 함

케노니컬 설정

- 링크 태그에 canonical을 붙임

메타 설정

- html 파일의 가장 윗부분인 메타를 잘 설정함

페이지 속도 개선

- 사이트 속도를 빠르게 함

사이트맵 관리

- 사이트맵을 정기적으로 관리함

HTTPS 구축 방법

1. 직접 CA에서 구매한 인증키를 기반으로 HTTPS 서비스를 구축
2. 서버 앞단에 HTTPS를 제공하는 로드 밸런스를 둠
3. 서버 앞단에 HTTPS를 제공하는 CDN을 둠

2.5.5 HTTP/3

- HTTP/2 : TCP 위에서 돌아감
- HTTP/3 : QUIC 위에서 돌아감, UDP 위에서 돌아감

    - 멀티플렉싱 있음
    - 초기 연결 설정시 지연 시간이 감소됨

초기 연결 설정 시 지연 시간 감소

- QUIC은 TCP를 사용하지 않아서 3-웨이 핸드셰이크를 안함
- QUIC은 첫 연결 설정에 1-RTT만 소요됨
    - 클라이언트가 서버에 어떤 신호를 한 번 주고, 서버도 응답만 하면 바로 본 통신을 시작할 수 있음
- QUIC은 순방향 오류 수정 매커니즘이 적용되어있음
    - 전송된 패킷이 손실되었다면 수신 측에서 에러를 검출하고 수정하는 방식
    - 열악한 네트워크 환경에서도 낮은 패킷 손실률을 자랑함