목차

TCP

- 정의
- 특징
- 헤더 구성
- 장점
- 단점
- 사용
- HTTP/2

TCP vs UDP

QtNetwork 모듈

명령어

- 소캣
- 주소 관리
- 네트워크
- 보안
- 유틸리티
- 동작
- 장점

Qt 에서 UDP 통신

코드 구현 설명

- Cmake 설정
- 헤더파일 설정
- 소켓 생성
- 바인딩
- 송신
- 수신

통신 프로토콜

- 통신 프로토콜이란 네트워크 상에서 두 개체가 데이터를 교환할 때 지켜야 하는 규칙.

- 사람 간 대화의 언어·문법과 동일한 역할을 수행

- 데이터 손실, 중복, 지연을 최소화

- 기기·운영체제·네트워크 환경에서 호환성을 확보

- 웹 브라우저(HTTP), 메일(SMTP/IMAP)

주요기능

1. 주소 제어
2. 에러제어
3. 흐름 제어
4. 동기화

OSI

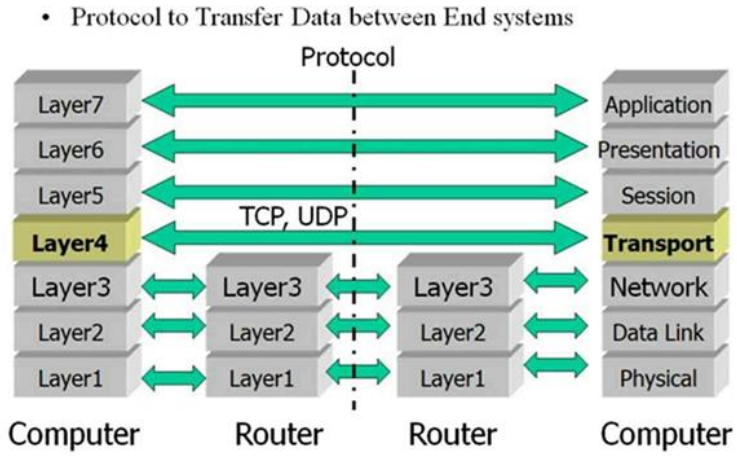
- 오픈 시스템 상호 연결로 7개의 계층으로 이루어져 있음

- 서로 다른 시스템 간의 네트워크 상호 운용을 가능하게 해줌

- 국제 표준화 기구에서 개발

구성

1. 물리 - 신호 전송 (케이블, wifi)
2. 데이터 링크 - MAC 주소 기반 프레임 전송
3. 네트워크 - IP 주소 기반 라우팅
4. 전송 - 종단 간 데이터 전달(TCP/UDP)
5. 세션 - 연결 관리
6. 표현 - 데이터 변환, 압축, 암호화
7. 응용 - 실제 서비스(HTTP, 영상 등)



UDP

- 네트워크에서 데이터를 효율적이고 빠르게 전송하는데 사용하는 통신 프로토콜

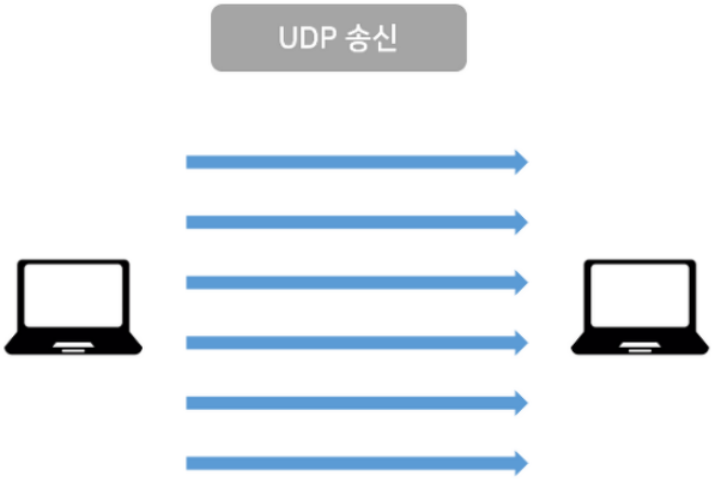
특징

- **비연결성** : 연결설정 없이 통신함

- **고속** : 데이터 검증이 적어 빠름

- **낮은 신뢰성**(커스텀 가능) : 검증이 적어 신뢰성이 낮음

- **헤더 단순** : 검증이 적어 보낼 내역인 헤더도 단순



헤더 구성

UDP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port														
4	32	Length																Checksum														

- **Source port**: 데이터를 생성한 애플리케이션이 사용하는 포트 번호

- **Destination port** : 목적지 애플리케이션이 사용하는 포트 번호

- **Checksum**: 중복 검사, 오류 검사(옵션)

장점

- 실시간성
- 번거로운 확인 절차 없음
- 매우 효율적
- 단순한 헤더

단점

- 낮은 신뢰도
- 전송 대역폭 점유
- 순서 보장 안됨
- 네트워크 부하 발생 가능

사용

스트리밍 분야 : 실사간성을 보장해야 하기에 손실이 발생하더라도 다음 프레임으로 대체가 가능하여 사용함

온라인 게임 : 실시간성이 유저에게 더욱 중요하기에 사용, 단 무결성이 중요한 데이터는 TCP 와 같은 통신을 사용

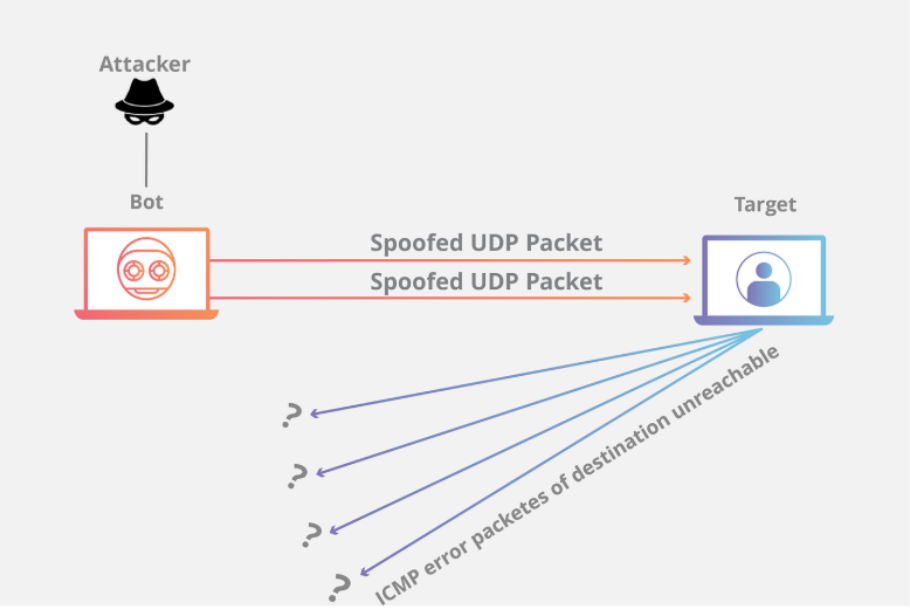
실시간 모니터링 : 지연 최소화가 우선

DDOS

- UDP 는 핸드셰이크(TCP 같이 연결 확인)이 필요 없어 공격할 서버에 트래픽을 폭주시킬 수 있음

- 서버를 보호하는 자원마저 고갈시켜버려 정상적인 접근의 트래픽도 처리 못하게 만들어 서버를 마비시킴

- UDP 폭주를 막기 위해 해당 트래픽이 처리할 수 없다는 알림을 보내는 ICMP 패킷의 응답 속도를 제한시킬 수 있음(단 정상적인 트래픽도 필터링 됨)



UDP 커스터마이징

- UDP의 신뢰성이 없는게 아닌 검증 기능을 탑재를 안했을 뿐

- 커스터마이징하여 신뢰성을 높일 수 있음

- 직접 코드를 작성해서 TCP와 유사하게 개조 가능

(예시 QUIC)

QUIC

- UDP기반 HTTP3.0에 사용중인 프로토콜

- 기존 TCP+TLS+HTTP 기능을 모두 구현함

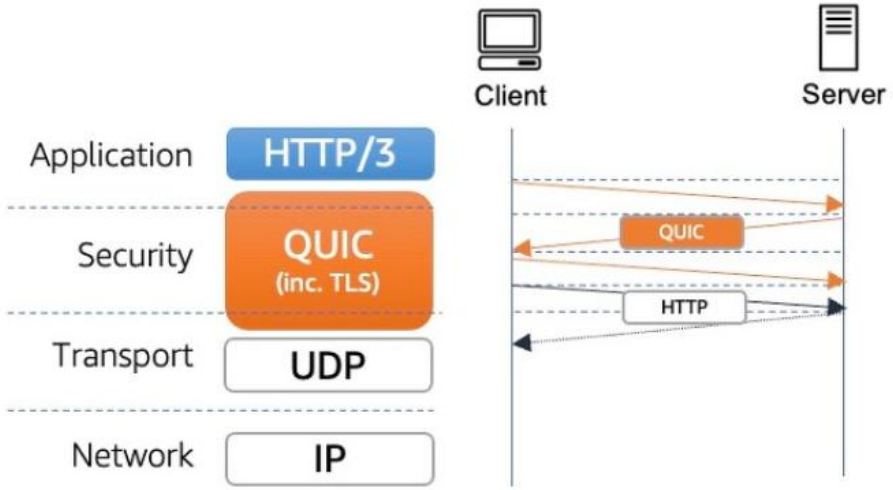
- 통신을 시작할 때 3 way handshake과정 안 거침

- 연결 설정에 필요한 정보와 함께 데이터를 보냄

- 성공시 서버가 해당 설정을 캐싱함

- 다음 연결시 캐싱을 기반으로 연결하여 바로 통신

- 다음 연결시 캐싱을 기반으로 연결하여 바로 통신

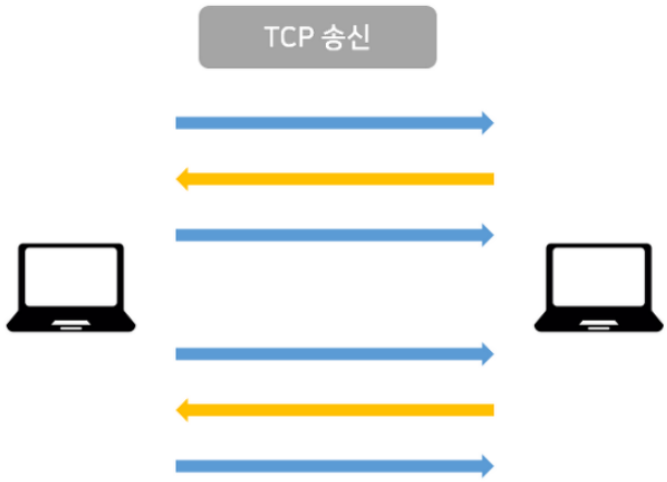


TCP

IP 프로토콜만으로 통신하기에 부족하거나 불안정하여 IP 의 단점을 보완하여 패킷 전송을 제어해 신뢰성을 보증한 프로토콜

특징

- **연결지향적** : 반드시 연결 설정 후 데이터 전송
- **신뢰성 보장** : 데이터 손실 시 재전송, 순서 제어
- **흐름 제어** : 수신 측의 처리 능력을 고려해 전송 속도 조절
- **혼잡 제어** : 네트워크 상태에 따라 전송 속도 조절
- **순차 전송** : 데이터 순서 보장



헤더 구성

TCP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port														
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)														
20	160	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.)																															
...	...	...																															

- **Source port** : 송신 포트 번호

- **Destination port** : 수신 포트 번호

- **Sequence number** : 전송되는 데이터의 순서 식별 번호

- **Acknowledgment number** : 수신자가 다음에 기대하는 바이트 번호

- **Flags (제어 비트)** :
    - SYN : 접속 요청시 보내는 패킷

    - ACK : SYN 을 잘 받았다고 알려주는 패킷

    - FIN : 접속 종료

    등 연결 상태 제어

- **Window size** : 수신자가 허용 가능한 데이터 양

- **Checksum** : 데이터 무결성 검사

장점

- **높은 신뢰성** : 오류 발생 시 재전송 가능

- **순서 보장** : 전송 순서대로 데이터 도착

- **안정성 확보** : 흐름 제어, 혼잡 제어로 안정적인 전송

단점

- **속도 느림** : 연결 설정 과정과 오류 처리로 지연 발생

- **자원 소모** : 신뢰성을 위해 추가적인 메모리와 CPU 사용

사용

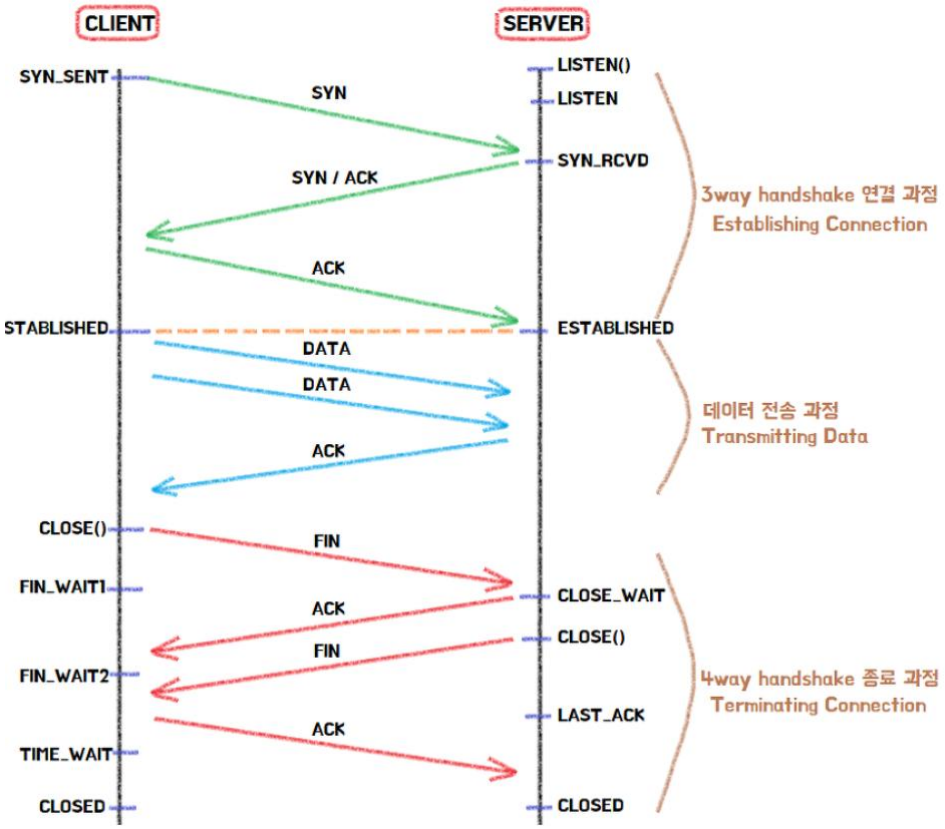
- **웹 서비스 (HTTP, HTTPS)** : 정확한 데이터 송수신 필요

- **파일 전송 (FTP, SFTP)** : 데이터 손실이 허용되지 않음

- **이메일 (SMTP, IMAP, POP3)** : 데이터 무결성 필요

- **원격 접속 (SSH, Telnet)** : 명령과 응답이 정확히 전달되어야 함

HTTP/2



TCP 연결과정 (3-Way Handshake)

- 사용자 : SYN 전송 (연결 요청, 초기 시퀀스 번호 x)
- 서버 : SYN + ACK 전송 (요청 수락, 시퀀스 번호 y, Ack = x+1)
- 사용자 : ACK 전송 (서버 번호 확인, Ack = y+1)
- 연결 상태 : ESTABLISHED (데이터 송수신 가능)

RTT

- 패킷 송신지 → 수신지 → 송신지로 돌아오는 데 걸린 시간
- TCP 3-way handshake 에서 **최초 RTT 측정**이 이루어짐
- SYN 보낸 시점부터 SYN+ACK 응답을 받는 시점까지 → RTT 초기값
- 이후 RTT 값은 **재전송 타이머(RTO)** 및 혼잡 제어에 활용

HTTP/2 연결 시 RTT 누적

- TCP : 1 RTT 소모
- TLS : 1~2 RTT 추가 소모 (TLS 1.2 는 2 RTT, TLS 1.3 은 1 RTT)
- 최종적으로 HTTP/2 요청이 서버에 도달하기까지 2~3 RTT 필요

TCP vs UDP

	TCP	UDP
연결 방식	연결형 서비스	비연결형 서비스
패킷 교환	가상 회선 방식	데이터그램 방식
전송 순서 보장	보장함	보장하지 않음
신뢰성	높음	낮음
전송 속도	느림	빠름

요약: 정보가 매우 중요하지 않는 이상 UDP 가 좋음

QtNetwork 모듈

- Qt 에서 네트워크 관련 기능을 제공하는 모듈
- TCP, UDP, HTTP, FTP 등 다양한 프로토콜 지원
- 이벤트 기반(시그널/슬롯) 구조로 비동기 네트워크 프로그래밍 가능

명령어

소캣

- QTcpSocket : TCP 클라이언트 통신 지원
- QTcpServer : TCP 서버 구축 지원
- QUdpSocket : UDP 송수신 지원

주소 관리

- QHostAddress : IPv4, IPv6 주소 표현
- QHostInfo : DNS 조회, 호스트 이름 확인

네트워크

- QNetworkAccessManager : HTTP, HTTPS 요청 처리

- QNetworkReply / QNetworkRequest : REST API 통신, 파일 다운로드 등에 활용

보안

- SSL/TLS 암호화 지원 (QSslSocket)

- HTTPS, 보안 연결 제공

유틸리티

- 프록시 설정 (QNetworkProxy)

- 네트워크 설정 조회 (QNetworkConfigurationManager)

동작(데이터가 도착했을 때**)**

- readyRead : 수신 데이터 도착 시

- bytesWritten : 송신 완료 시

- disconnected : 연결 종료 시

- errorOccurred : 에러 발생 시

- 슬롯 함수와 연결하여 비동기 처리 구현

장점

- 네이티브 소켓 API 대비 단순한 코드 구조

- Qt 이벤트 루프와 자연스럽게 통합 (GUI + 네트워크 동작 동시 처리)

- 크로스플랫폼 (Windows, Linux, macOS, 모바일) 환경에서 동일 코드로 동작

통신 절차

- 소켓 객체 생성

- QUdpSocket 생성

- 시그널 연결

- 소켓 바인딩

- 내 IP 입력

- 바인딩

- 데이터 송신

- QUdpSocket::writeDatagram() 사용

- 문자열 → QByteArray 변환

- 상대방 IP 와 포트 지정

- **데이터 수신**

    - 데이터가 도착하면 readyRead 시그널 발생

    - readDatagram() 호출해 데이터 추출

코딩

Cmake 설정

- 패키지 추가(Network)

- 링크 추가(Network)

```
cmake_minimum_required(VERSION 3.21)
project(hw_01 LANGUAGES CXX)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTOUIC ON)

find_package(Qt6 REQUIRED COMPONENTS Widgets Network)   # ← Network 추가

add_executable(${PROJECT_NAME}
  main.cpp
  mainwindow.cpp
  mainwindow.h
  mainwindow.ui
)

target_link_libraries(${PROJECT_NAME} PRIVATE Qt6::Widgets Qt6::Network)  # ← 링크
```

헤더 파일 설정

- QudpSocket 헤더 추가

- QudpSocket 변수 선언

```h
#pragma once
#include <QMainWindow>
#include <QUdpSocket>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow {
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent=nullptr);
    ~MainWindow();

private slots:
    void on_pushButton_clicked();
    void onReadyRead();

private:
    Ui::MainWindow *ui;
    QUdpSocket *sock;
};
```

소켓 생성

- **QUdpSocket *sock = new QUdpSocket(this);**

- 소켓은 QObject 상속 → 부모 지정 시 자동 메모리 관리

- 하나의 소켓으로 송신과 수신 모두 가능

```cpp
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
    , sock(new QUdpSocket(this))//소켓라이브러꺼
{
    ui->setupUi(this);
```

바인딩

- 자신의 IP 와 포트 번호에 소켓을 연결

- **sock->bind(QHostAddress("192.168.0.10"), 45454);**

```
QHostAddress myAddr("172.100.7.254");   // 본인 PC IP
quint16 myPort = 45455;                 // 수신용 포트
sock->bind(myAddr, myPort,
           QUdpSocket::ShareAddress | QUdpSocket::ReuseAddressHint);
```

송신

- QString 으로 메시지 입력

- 문자열 → QByteArray 변환

- writeDatagram() 호출

```
void MainWindow::on_pushButton_clicked() {
    QString msg = ui->send->toPlainText();
    QByteArray dat = msg.toUtf8();

    // 상대방 IP와 포트
    QHostAddress peer("172.100.4.199");
    sock->writeDatagram(dat, peer, 45454);
    ui->textEdit->append("나: "+msg);
    ui->send->clear();
}
```

- 매우 간단한 호출

- 송신 성공/실패 여부 확인은 별도 처리 필요

수신

- readyRead 시그널 입력시 슬롯 연결

- 슬롯에서 hasPendingDatagrams() 확인

- 버퍼 크기 확보 후 readDatagram() 실행

- 데이터, 송신자 IP 와 포트 정보 획득

```
void MainWindow::onReadyRead() {
    while (sock->hasPendingDatagrams()) {
        QByteArray buf;
        buf.resize(int(sock->pendingDatagramSize()));
        QHostAddress sender;
        quint16 senderPort;
        sock->readDatagram(buf.data(), buf.size(), &sender, &senderPort);
        ui->textEdit->append("상대 : " + QString::fromUtf8(buf));
    }
}
```

- 여러 개의 패킷이 동시에 도착할 수 있어 while 루프 사용

- 패킷 손실은 직접 보완 필요

참고 문헌:

- https://www.cloudflare.com/ko-kr/learning/ddos/glossary/user-datagram-protocol-udp/

- https://inpa.tistory.com/entry/NW-%F0%9F%8C%90-%EC%95%84%EC%A7%81%EB%8F%84-%EB%AA%A8%ED%98%B8%ED%95%9C-TCP-UDP-%EA%B0%9C%EB%85%90-%E2%9D%93-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EC%9E%90

죄송합니다