C++ & QT day4 보고서

로빛 20기 인턴 김다은

1. UDP에 개요와 배경

UDP(User Datagram Protocol)는 1980년 데이빗 리드(David P. Reed)에 의해 설계되어, RFC 768을 통해 표준으로 정의된 전송 계층 프로토콜이다. 인터넷 상의 여러 장치와 애플리케이션 사이에서 "데이터그램"이라는 독립적인 단위로 데이터를 주고받는 방식이다. 각 데이터그램은 별도의 연결 설정 없이 개별적으로 전송되며, 최소한의 헤더(8바이트)만 가지고 빠르게 전달된다. UDP는 소스/목적지 포트, 길이, 체크섬만을 포함하며, IP 프로토콜 상단에서 동작한다.

UDP는 "비연결성(Connectionless)", "최소 오버헤드", "빠른 전송"이라는 특징을 가진다. TCP와 달리 데이터의 순서, 중복, 손실을 확인/보장하지 않는다. 대신 오류 검출(체크섬) 기능만 제공하고, 신뢰성 보장/흐름 제어는 전혀 없다. 그렇기에 애플리케이션에서 자체적으로 중요 정보를 재전송하거나, 순서를 재정렬해야 한다.

또한 전송 계층(Transport Layer)에서 동작하는 대표적인 프로토콜로, 1980년대 초 Jon Postel이 설계하고 RFC 768로 표준화되었다. 네트워크에서 빠르고 단순한 데이터 송수신을 목적으로 개발되었으며, IP 프로토콜 위에서 동작한다. 비연결성(Connectionless) 통신 방식을 제공하며, 실시간성과 함께 저지연, 소규모 데이터 통신이 필요한 환경에서 널리 활용되고 있다

그럼 UDP란 무엇인가?
UDP는 데이터그램 기반의 전송 프로토콜로, 각 패킷(데이터그램)이 서로 독립적으로 전송된다. 송신자는 연결 설정 없이 바로 데이터를 보낼 수 있으며, 수신 여부, 데이터 순서, 오류에 대한 보장 없이 빠름을 우선시한다. 데이터그램에는 송수신 포트번호, 데이터 길이, 체크섬 정보가 붙으며, 헤더는 8바이트로 매우 간결하다. UDP의 데이터 전송은 "최대한 단순화된 전송"을 구현하기 위한 방식으로, 신뢰성보다 빠른 전송이 우선인 환경에 잘 어울린다

- **역사적 배경**

인터넷이 성장하던 초기에는 모든 데이터 통신에서 신뢰성과 무결성을 중요하게 생각했고, TCP가 이 역할을 담당했다. 그러나 1970~80년대 ARPANET

및 초기 인터넷 실험에서 볼 수 있듯, 네트워크가 진화하며 "빠르고 실시간의 데이터 전송"이 반드시 필요한 분야(음성, 영상, 실시간 조사 데이터 등)가 급부상했다. 이런 분야는 일정량의 데이터 손실은 허용하는 대신, 느려지는 것에 민감했다.

이에 David P. Reed는 1980년에 UDP 프로토콜을 설계했다. 주요 목표는 "단순함"과 "빠른 처리"였다. 데이터그램이 독립적으로 전달되고, 각 패킷의 도착 여부·순서·중복 체크 없이 전송한다. TCP/IP 구조에서 TCP는 신뢰성과 제어, UDP는 빠르고 단순한 송수신이라는 '역할 분담 구조'가 확정된 순간이었다.

UDP는 IP 네트워크의 보급 및 인터넷 멀티미디어/음성/실시간 통신의 급성장과 함께 핵심 역할을 하며 확산됐다. 특히 VoIP, 실시간 스트리밍, 온라인 게임, DNS, DHCP등 다양한 응용에 채택됐고, 멀티캐스트·브로드캐스트 지원이 필요한 환경에서 필수 프로토콜로 자리 잡았다.

- **OSI 7계층 모델**

1. 물리 계층 (Physical Layer)
전기적 신호, 광신호 등 실제 물리 매체를 통해 비트를 전송하는 계층이다. 케이블, 무선 주파수, 허브 등이 물리 계층 장치에 속하며, 전송 매체와 신호의 전기적, 기계적 특성을 다룬다.

2. 데이터 링크 계층 (Data Link Layer)
물리 계층을 통해 전달되는 신호를 프레임이라는 단위로 묶어 전송하며, 오류 제어와 흐름 제어를 담당한다. MAC 주소를 기반으로 같은 네트워크

내 장치들을 식별하며, 스위치와 브리지 등이 이 계층에서 동작한다.

3. 네트워크 계층 (Network Layer)
서로 다른 네트워크 간 데이터 전송과 라우팅을 담당한다. IP 주소를 사용해 목적지까지 최적 경로를 찾아 패킷을 전달하며, 라우터가 주로 네트워크 계층 장비다.

4. 전송 계층 (Transport Layer)
종단 간 신뢰성 있는 데이터 전송을 담당한다. TCP, UDP 프로토콜이 여기에 속하며, 데이터의 분할/재조립, 오류 제어, 흐름 제어 등의 기능을 수행한다.

5. 세션 계층 (Session Layer)
통신 세션을 설정, 관리, 종료하는 계층이다. 양쪽 호스트 간의 연결을 유지하고 동기화를 제공하여 데이터 교환을 원활히 한다.

6. 표현 계층 (Presentation Layer)
데이터 표현 방식 변환, 압축, 암호화 기능을 담당해, 서로 다른 시스템 간 데이터 형식을 호환할 수 있도록 한다.

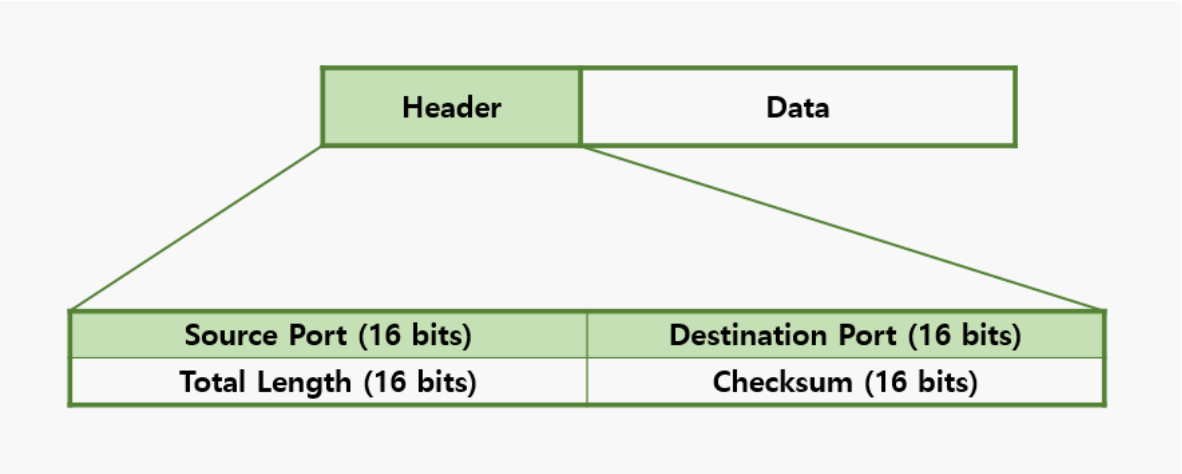
7. 응용 계층 (Application Layer)
사용자 응용프로그램과 직접 연결되어 이메일, 웹, 파일 전송 등 다양한 네트워크 서비스를 제공한다.

2. UDP 프로토콜 구조

UDP는 전송 계층에서 동작하는 비연결형(connectionless) 프로토콜로, 데이터를 독립적인 단위인 데이터그램(datagram) 형태로 전송한다. UDP의 가장 큰 특징은 연결 설정과 연결 유지 과정이 없으며, 신뢰성 보장(재전송, 순서 보장 등) 기능이 없어서 전송 오버헤드가 매우 적고 속도가 빠르다는 점이다.

UDP는 IP 프로토콜 내에서 프로토콜 번호 17로 구분되며, 전송될 데이터는 IP 패킷 내부에 UDP 데이터그램 형태로 삽입되어 전달된다. UDP 데이터그램은

크게 8바이트 고정 크기의 헤더(Header) 부분과 그 뒤에 이어지는 데이터(페이로드)로 구성된다.



- Source Port (16 bits) : 출발지(송신) 포트 번호
- Destination Port (16 bits) : 목적지(수신) 포트 번호
- Total Length (16 bits) : 헤더와 데이터부를 포함한 전체 길이
- Checksum (16 bits) : 전체 데이터그램에 대한 오류를 검사하기 위한 필드

- UDP 헤더 구성 상세 설명

UDP 헤더는 총 4개의 16비트(2바이트) 필드로 이루어져, 총 8바이트 길이를 갖는다. 각 필드는 다음과 같은 기능과 의미를 가진다.

출발지 포트 (Source Port, 16 bits)

송신 측 애플리케이션이 사용하는 포트 번호를 나타낸다. 이 값은 응답을 요구하는 양방향 통신에서 송신자 주소 역할을 하며, 보통 설정된다. 그러나 일부 단방향 통신에서는 0으로 설정할 수 있다.

목적지 포트 (Destination Port, 16 bits)

UDP 데이터그램을 받을 대상 애플리케이션이나 서비스의 포트 번호이다. 이

값은 필수적으로 지정해야 하며, 네트워크 상에서 데이터가 정확한 프로세스에 전달되도록 한다.

길이 (Length, 16 bits)
UDP 데이터그램 전체의 길이를 바이트 단위로 나타낸다. 여기에는 8바이트 헤더와 그 뒤에 이어지는 데이터(페이로드)의 크기가 모두 포함된다. 최소 값은 8(헤더만), 최대 값은 65,535이다.

체크섬 (Checksum, 16 bits)
UDP 데이터그램 전체의 무결성을 검증하는 필드이다. 체크섬은 UDP 헤더, 데이터, 그리고 가상 헤더(출발지 IP, 목적지 IP, 프로토콜 번호, UDP 길이)를 포함하여 계산된다. 이 체크섬은 데이터 전송 과정에서 발생하는 오류를 검출하며, IPv6에서는 필수지만, IPv4에서는 선택 사항이다. 체크섬 값이 0일 경우 계산을 하지 않았다는 의미다.

- **UDP 데이터그램 구조**

UDP 데이터그램은 다음과 같이 구성되어 있다.

IP 헤더 (최소 20바이트)
IP 프로토콜 계층에서 패킷을 목적지까지 전달하는 데 필요한 라우팅 및 주소 정보를 담는다. UDP 데이터그램은 이 IP 패킷 내부 페이로드로 포함된다.

UDP 헤더 (8바이트)
앞서 설명한 4개의 필드를 포함하며, UDP 통신을 위한 최소한의 제어 정보를 제공한다.

데이터(페이로드)
실제 응용 프로그램이 전송하는 데이터 부분으로, 최대 크기는 IP 패킷 최대 크기(65,535 바이트)에서 IP 헤더와 UDP 헤더 크기를 뺀 값이다.

- UDP 데이터그램 전송 방식

UDP는 송신자가 수신자와 연결을 맺지 않고 독립적인 데이터그램을 네트워크

에 전송하는 비연결성 프로토콜이다. 송신자는 UDP 헤더를 생성하고 페이로드 데이터를 붙여 IP 계층에 넘기며, IP 계층은 이를 목적지 IP 주소를 기반으로 라우터를 통해 전달한다.

수신 측은 도달한 UDP 데이터그램의 목적지 포트를 참고하여 해당 데이터를 응용 프로그램에 전달한다. UDP는 데이터그램의 순서 보장, 재전송, 흐름 제어, 혼잡 제어 등을 지원하지 않으므로 네트워크의 상태나 오류에 따른 데이터 손실, 중복, 순서 변경 가능성이 존재한다.

이 구조 덕분에 UDP는 오버헤드가 적고 처리 속도가 매우 빠르며, 실시간 통신이나 간단한 요청-응답 기반 통신에 적합하다. 대표적인 UDP 응용 서비스에는 DNS 질의, 실시간 음성 및 영상 스트리밍, 온라인 게임 및 VoIP 등이 있다.

3. UDP 통신의 특징

- **무연결성 (Connectionless)**

UDP는 무연결성 통신 프로토콜로서, 데이터를 주고받기 전에 송신자와 수신자 간에 논리적인 연결을 설정하거나 유지하지 않는다. TCP에서 사용하는 3-way handshake 같은 연결 설정 과정이 없으며, 데이터 전송이 시작되면 별도의 통신 경로 유지 없이 독립적으로 데이터그램을 전송한다.

이 특징 덕분에 UDP는 통신 지연이 매우 적으며, 연결 설정 및 종료에 필요한 시간을 아낄 수 있다. 하지만, 이로 인해 송수신자는 서로의 상태를 확인하지 않기 때문에 데이터가 제대로 전달되었는지 확인할 수 없으며, 데이터 중복, 손실, 순서 변경 가능성도 존재한다. 이 때문에 UDP는 빠른 전송이 우선시되는 실시간 음성통화, 게임, 스트리밍 등에 주로 사용된다.

UDP는 서버와 클라이언트에게 연결 기반 소켓 구분이 없으며, 단일 소켓을 사용해 여러 통신 상대와 독립적으로 데이터를 주고받을 수 있다. 이것은 멀티캐스트나 브로드캐스트 서비스에 유리하며, 대규모 통신 환경에 적합하다.

- **비신뢰성 (Best-effort delivery)**

UDP는 비신뢰성 전송 방식을 채택하여 최선을 다해(Best-effort) 데이터를 전달하지만, 데이터의 송수신 성공을 보장하지 않는다. 연결 유지, 재전송, 흐름 제어, 순서 보장 등과 같은 신뢰성 확보 기능이 내장되어 있지 않다.

따라서 패킷 손실, 중복, 지연, 순서 변경 등이 네트워크 상황에 따라 발생할 수 있으며, UDP 자체는 이러한 문제를 감지하거나 자동 복구하지 않는다. 이러한 비신뢰성은 네트워크 부하와 처리 지연을 줄여주지만, 데이터 정확성이 중요한 응용에는 부적합하다.

신뢰성이 필요한 데이터 전송은 보통 TCP를 이용하며, UDP는 성능이 중요하고 일부 데이터 손실을 허용할 수 있는 환경에서 더 적합하다. 예를 들어, 실시간 스트리밍에서는 약간의 데이터 손실보다 지연 최소화가 우선시된다.

- **UDP 체크섬 사용 및 선택적 필드 설명**

UDP 프로토콜에서 체크섬(Checksum)은 데이터 전송 중 발생할 수 있는 오류를 검출하기 위해 사용되는 중요한 무결성 검사 수단이다. UDP 데이터그램의 무결성 확인을 위해 송신자는 UDP 헤더와 데이터(페이로드), 그리고 IP 가상 헤더(Pseudo Header)를 포함해 체크섬을 계산한 후, 체크섬 필드에 값을 넣어 전송한다.

체크섬 계산 대상

체크섬은 단순히 UDP 헤더와 페이로드 데이터뿐만 아니라, IP 프로토콜의 일부 정보를 포함한 가상 헤더까지 포함해서 계산한다. 가상 헤더는 송신지 IP 주소(32비트), 목적지 IP 주소(32비트), 프로토콜 번호(8비트, UDP는 17), UDP 길이(16비트) 등으로 구성된다. 이렇게 함으로써 데이터그램이 네트워크에서 올바른 주소와 프로토콜에 의해 전달되는지 검사할 수 있다.

체크섬 계산 방법

1. UDP 헤더, 페이로드, 그리고 가상 헤더에 포함된 데이터를 모두 16비트 단위의 워드로 쪼갠다.
2. 이 16비트 워드들을 모두 더한다.
3. 덧셈 과정에서 만약 오버플로우(carry)가 발생하면, 발생한 값은 다시 결과

에 더한다.

4. 마지막으로 그 합의 1의 보수(비트 반전)를 취한 값이 체크섬이 된다.

예를 들어, 합산 결과가 0x533d0이라면, carry를 더해서 0x33d5가 되고, 이것의 1의 보수를 취해 최종 체크섬 값을 얻는다.

송신 측은 계산된 체크섬을 UDP 헤더의 체크섬 필드에 넣어 보내고, 수신 측은 받은 데이터그램에 대해 동일한 계산을 수행해 송신 측이 보낸 체크섬 값과 비교한다. 만약 두 값이 일치하지 않으면 데이터가 전송 도중 손상된 것으로 판단하여 해당 데이터그램을 폐기한다.

그러나 체크섬은 오류 검출만 가능하며, 오류 복구나 재전송 기능은 포함하지 않는다. 즉, 해당 데이터그램이 손상되어도 재전송은 하지 않고 폐기만 하므로, UDP 통신에선 데이터 손실 가능성이 그대로 존재한다.

그런데 만약 체크섬 필드가 '0'일 경우라면?
체크섬 필드 값이 모두 0이면, 송신자가 체크섬 계산을 생략했다는 의미이며, 이 경우 수신자는 체크섬 검사를 수행하지 않고 데이터그램을 그대로 처리한다. 하지만 이는 위험할 수 있으므로 대부분 환경에서는 체크섬을 계산하여 포함하는 것이 권장된다.

4. UDP의 장점과 단점

장점

1. 빠른 전송 속도

UDP는 TCP와 달리 연결 설정 과정(3-way handshake)이나 연결 유지, 종료 과정이 없으므로 데이터를 즉시 전송할 수 있다. 이로 인해 통신 지연이 획기적으로 줄어들어 실시간성 요구가 높은 응용 프로그램에 적합하다.
그 결과 인터넷 전화(VoIP), 온라인 게임, 실시간 스트리밍과 같이 지연 시간이 중요한 서비스에서 UDP가 널리 사용된다.

2. 낮은 오버헤드

UDP 헤더 크기는 8바이트로 매우 작고 단순하여, TCP 헤더(최소 20바이트)보

다 훨씬 적은 네트워크 자원을 요구한다.
이 작은 헤더는 네트워크 대역폭을 효율적으로 사용하도록 돕고, 처리 속도 및 메모리 사용 측면에서도 장점을 제공한다.

3. 간단한 구조로 인한 높은 처리 효율성
UDP는 연결 설정, 흐름 제어, 혼잡 제어, 재전송 등의 복잡한 절차가 없기 때문에 구현이 간단하고 처리 부담이 적다.
그래서 시스템 리소스가 제한적인 임베디드 시스템이나 고속 데이터 처리에 효과적이다.

4. 실시간 전송에 적합
UDP는 데이터 일부 손실이 발생하여도 전송 지연이 상대적으로 적어야 하는 실시간 서비스에 이상적이다.
실시간 음성 통화, 화상 회의, 스트리밍 서비스는 약간의 데이터 손실을 허용하면서도 낮은 지연을 필요로 하므로 UDP를 선호한다.

5. 멀티캐스트 및 브로드캐스트 전송 지원
TCP가 1:1 전송만 지원하는 반면, UDP는 한번에 여러 수신자에게 동일 데이터를 전송하는 멀티캐스트 및 브로드캐스트가 가능하다.
그래서 IPTV, 실시간 공지사항, 일부 게임 서버 등 다중 대상 네트워크 서비스에 매우 유용하다.

6. 비연결성으로 인한 유연성
UDP는 서로 연결 상태를 유지하지 않으므로, 서버가 다수 클라이언트와 독립적으로 빠르게 통신할 수 있다.
그래서 상태 저장이 필요 없는 간단한 요청-응답 응용 프로토콜에 적합하며, 확장성도 뛰어나다.

단점
1. 데이터 손실 가능성
UDP는 데이터 전송 시 신뢰성 없는 전송 방식을 사용하므로, 네트워크 혼잡이나 전송 오류로 인해 데이터그램이 손실될 수 있다.

애플리케이션 수준에서 손실 복구 기능을 구현하지 않으면, 전송한 데이터가 최종 수신자에게 도달하지 않는 경우가 빈번히 발생한다.

2. 순서 보장 없음
데이터그램은 네트워크 내에서 서로 다른 경로로 전송될 수 있으며, 수신 측에서 도착 순서가 바뀔 수 있다.
UDP 자체는 순서 보장 및 재정렬 기능이 없어, 순서가 중요한 데이터 전송에는 부적합하다.

3. 오류 검출 부족 및 복구 미흡
UDP 헤더에는 체크섬 필드가 있으나, 오류 검출 기능이 비교적 단순하며 오류 복구를 위한 재전송 메커니즘이 없다.
데이터 손상 시 검출은 가능하지만, 자동으로 재전송하거나 복구하지 않기 때문에 신뢰성 확보는 응용 계층에 전적으로 맡겨진다.

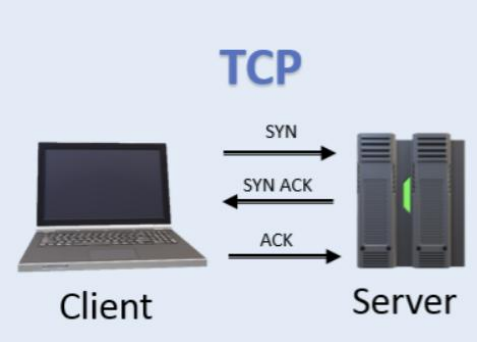
4. 흐름 제어 및 혼잡 제어 부재
UDP는 송신 속도를 조절하거나 네트워크 혼잡 상황을 감지하여 데이터를 조절하는 기능이 없다.
그래서 네트워크 혼잡 시 성능 저하나 패킷 손실이 심화될 위험이 있으며, TCP에 비해 네트워크 안정성이 떨어진다.

5. 응용 계층 책임 증가
신뢰성, 오류 제어, 순서 보장, 흐름 제어 등 많은 기능을 제공하지 않아, 이러한 기능을 필요로 하는 애플리케이션은 별도의 메커니즘을 직접 구현해야 한다. 개발 복잡도를 높이고, 잘못된 구현 시 통신 품질 저하를 초래할 수 있다.

UDP는 이처럼 빠르고 단순하며 실시간성이 중요한 환경에 적합한 프로토콜이라는 장점이 있지만, 신뢰성이 요구되는 데이터 전송에는 적합하지 않다.
따라서 UDP를 사용할 때는 애플리케이션 요구 사항과 네트워크 환경을 신중히 고려하여야 한다

5. UDP와 TCP의 비교

TCP는 전송 계층(Transport Layer)에 속하는 대표적인 프로토콜로, 네트워크 상에서 두 시스템 간에 신뢰성 있는 데이터 전송을 보장하기 위해 설계되었다. TCP는 인터넷의 핵심 프로토콜 중 하나로, IP 프로토콜 위에서 동작하며, 데이터를 안정적이고 순서대로 전달하는 일을 담당한다.



- **Tcp 주요 특징**

1. 연결 지향형 프로토콜 (Connection-oriented)

TCP는 송신자와 수신자가 통신하기 전에 먼저 논리적인 연결을 확립한다. 이 연결 설정은 '3-way handshake'라는 과정을 통해 이루어진다. 연결이 성립되면 두 시스템은 신뢰성 있는 통신을 보장하기 위한 상태 정보를 서로 유지하며 데이터 교환을 시작한다.

2. 신뢰성 있는 데이터 전송

TCP는 데이터가 손실되거나 손상될 경우 이를 감지하고, 실패한 데이터를 재전송한다. 또한 도착한 데이터가 순서가 맞는지 확인해 순서대로 애플리케이션에 전달하며, 수신자가 데이터를 제대로 받았는지 확인하는 응답(ACK)을 통해 송신자는 다음 데이터를 보낼 준비를 한다.

3. 스트림 기반 전송 방식

TCP는 데이터를 바이트 스트림으로 취급한다. 즉, 데이터 단위를 고정된 크기의 패킷 대신 임의 길이의 바이트 연속체로 관리하며, 이를 여러 TCP 세그먼트로 나누어 전송한다. 수신 측에서는 온전한 바이트 스트림으로 재조립해 애플리케이션에 전달한다.

4. 흐름 제어 (Flow Control)
송신자는 수신자의 처리 능력과 수신 버퍼의 상태를 고려하여 한 번에 보낼 수 있는 데이터 양을 조절한다. 이는 수신자가 처리하지 못하는 데이터를 과도하게 보내는 혼잡을 막아 통신 효율을 높인다. 흐름 제어는 TCP 헤더 내 '윈도우 크기(Window Size)' 필드를 통해 구현된다.

5. 혼잡 제어 (Congestion Control)
네트워크 혼잡이 발생했을 때 TCP는 전송 속도를 줄여 네트워크 과부하를 완화한다. 혼잡 상황 인식은 패킷 손실이나 ACK 지연 등을 통해 이루어지며, 혼잡제어 알고리즘(예: AIMD, Slow Start 등)을 통해 동적으로 속도를 조절한다.

6. 양방향(full duplex) 통신
TCP는 송신자와 수신자가 동시에 데이터를 전송할 수 있는 양방향 통신을 지원한다.

7. 포인트 투 포인트 통신
한 쌍의 송신자와 수신자가 독립적으로 TCP 세션을 구성하여 통신한다.

- **Tcp 통신 과정**

  연결 설정 (3-way handshake)

- 클라이언트가 서버에 SYN 패킷을 보내 연결 요청
- 서버가 SYN+ACK 패킷으로 응답
- 클라이언트가 ACK 패킷을 보내 연결 확립 완료

데이터 전송

- 수진자의 데이터는 순서 번호(Sequence Number)를 붙여 세그먼트 단위로 분할 전송
- 수신자는 데이터 수신 후 ACK로 응답하며, 누락 시 재전송 요구
- 흐름 제어와 혼잡 제어를 수행해 네트워크 상태에 대응

연결 해제 (4-way handshake)

- 한 쪽이 연결 해제 요청(FIN 패킷)

- 상대방이 ACK 응답
- 반대 쪽도 연결 해제 요청 후 ACK 응답으로 통신 종료

- **UDP, TCP는 그럼 어떻게 차이가 날까?**

1. 전송 속도
   UDP: 연결 설정 및 유지 과정이 없고, 오류 검출·복구나 흐름 및 혼잡 제어가 없어서 전송속도가 매우 빠르다. 헤더가 8바이트로 작고, 데이터 전송에 필요한 처리량이 적어 실시간성이 요구되는 상황(예: 실시간 음성/영상, 게임)에 적합하다.
   TCP: 신뢰성 있는 데이터 전송을 위해 3-way handshake를 통한 연결 설정, ACK 기반 확인, 재전송과 흐름·혼잡 제어 등이 필요하다. 이로 인해 UDP보다 전송 지연이 크고 속도가 느려질 수 있지만 데이터의 정확성을 보장한다.

2. 신뢰성
   UDP: 비신뢰성 프로토콜이다. 데이터가 손실되어도 재전송하지 않고, 순서도 보장하지 않아 데이터그램은 도착하는 순서대로 처리된다. 수신 확인을 하지 않기 때문에 전송 성공 여부를 알 수 없다. 다만 체크섬을 이용해 데이터 무결성 검사는 한다.

   TCP: 데이터 전송의 신뢰성을 보장한다. 데이터가 손실되거나 오류가 발생한 경우 재전송을 통해 복구하며, 수신된 데이터의 순서를 보장해 애플리케이션에 올바른 순서대로 전달한다. 수신 확인(ACK)을 통해 전송 성공 여부를 체크하고, 수신자의 처리 능력에 맞춰 보내는 데이터 양을 조절하는 흐름 제어를 지원한다.

3. 연결 방식
   UDP: 비연결형(Connectionless) 프로토콜이다. 데이터는 독립적인 데이터그램 단위로 전송되고, 연결 설정 과정이 없으며, 상태를 유지하지 않는다. 임의로 데이터를 송신하며 빠른 통신을 지원한다.

TCP: 연결 지향형(Connection-oriented) 프로토콜로, 통신 시작 전에 명확한 연결 설정 과정이 있다. 3-way handshake를 통해 송신자와 수신자 간에 논리적 연결을 수립하며, 연결 유지 중 통신 상태를 추적한다. 연결 종료도 명확한 과정으로 이루어진다.

4. 헤더 크기 및 구조
   UDP: 고정 길이 8바이트 헤더로 단순하며, 출발지 포트, 목적지 포트, 길이, 체크섬 4가지 필드만 포함한다. 헤더가 작아 네트워크 자원 소모가 적다.

   TCP: 최소 20바이트 이상의 가변 길이를 가진 헤더를 이용한다. 시퀀스 번호(Sequence Number), ACK 번호, 윈도우 크기(Window Size), 플래그 등 다양한 필드를 포함해 신뢰성 및 흐름 제어를 위한 많은 정보를 담는다.

5. 오류 처리 및 제어
   UDP: 최소한의 오류 검출(체크섬)만 제공하며, 오류 복구, 재전송, 흐름 제어나 혼잡 제어 기능이 전무하다. 오직 Best-effort 전송을 하며 속도와 처리량이 중요시되는 환경에서 그 강점이 발휘된다.

   TCP: 네트워크 혼잡 감지, 흐름 제어, 오류 복구 등을 적극적으로 수행한다. 누락된 패킷은 재전송 처리하며, 전송 속도를 동적으로 조절하여 네트워크 안정성을 유지한다.

위에서도 말 했듯이 UDP는 데이터 정확성보다 빠른 전송과 낮은 지연이 중요한 응용에 적합하며, 네트워크 자원 소모가 적고 실시간 처리에 유리하고 TCP는 연결 설정과 데이터 전송 중 신뢰성을 중시하는 애플리케이션에 적합하다. 다만 처리 오버헤드와 속도 저하가 불가피하다.

6. UDP의 응용 분야 중 가장 많이 활용 되는 분야들
   - 실시간 스트리밍 (음성, 영상)
     실시간 동영상 또는 음성 스트리밍 서비스는 전송 지연이 최소화되어야

하며, 일부 데이터 패킷 손실은 크게 문제되지 않는다.
UDP는 연결 설정 과정이 없고, 지연 시간이 짧으며, 작은 헤더 크기로 네트워크 오버헤드가 적어 실시간 데이터 전송에 최적화되어 있다.
예를 들어, 유튜브 라이브, 넷플릭스 라이브 스트리밍, 화상 회의 솔루션 등에서 UDP를 주로 사용한다.

- 온라인 게임
  온라인 멀티플레이어 게임에서는 낮은 지연 시간과 빠른 데이터 전달이 게임 경험에 필수적이다.
  UDP는 빠르고 경량의 프로토콜로서 게임 상태 업데이트, 유저 입력 이벤트 등을 신속하게 주고받을 수 있다.
  일시적인 데이터 손실은 게임 진행에 큰 영향을 주지 않으며, 손실된 데이터는 다음 업데이트에서 보완 가능하다.

- VoIP (Voice over IP)
  음성 통화 서비스는 실시간 음성 데이터 전달이 매우 중요하다.
  UDP는 빠른 전송과 낮은 지연을 보장하여 통화 품질 유지에 필수적이다.
  그치만 일부 음성 패킷을 잃어도 통화 품질에 큰 영향을 미치지 않기 때문에 TCP보다 UDP가 더 적합하다.

- DNS 쿼리
  도메인 네임 시스템(DNS)은 빠른 질의-응답 기반 서비스로, 응답시간이 중요하다.
  UDP는 빠른 요청과 응답 전송에 적합하며, 응답이 없으면 클라이언트가 재요청하는 구조여서 신뢰성 문제를 완화한다.
  대부분 DNS 쿼리는 UDP를 이용하지만, 응답 데이터가 클 경우 TCP로 전환하기도 한다.

  이 외에도 SNMP(Simple Network Management Protocol) 같은 네트워크 장비 관리를 위한 경량 프로토콜이나 IoT 센서 데이터 전송, 실시간 모니터링에 사용하여 빠른 전송과 낮은 지연이 절대적으로 좀 필요한 환경에서 효율적이고, 데이터 손실을 일부 허용하는 특성을 갖는 어플리케이션

에 주료 쓰인다.

왜? U에를 쓰며 어플리케이션에 사용할까?

1. 빠른 데이터 전송과 짧은 지연 시간

연결 설정과 상태 유지 부하가 없어 즉각적인 데이터 전송이 가능해 실시간 서비스에 매우 적합하다.

2. 낮은 네트워크 오버헤드

작은 헤더와 단순한 처리 방식으로 네트워크 자원을 효율적으로 사용한다.

3. 신뢰성보다는 속도 우선

데이터 손실이 발생해도 문제되지 않는 환경에서 이상적이다. 예를 들어, 소리나 영상의 약간 손실은 사용자 경험에 큰 영향을 주지 않는다.

4. 멀티캐스트/브로드캐스트 가능

다중 수신자에게 데이터를 한 번에 전송할 수 있어 네트워크 효율성 향상에 기여한다.

7. UDP의 동작 과정

- 데이터 송수신 흐름

UDP는 비연결형 프로토콜로, 데이터 전송 전에 송신자와 수신자 간에 연결을 설정하지 않는다. 송신자는 IP 주소와 포트 번호만 알면, UDP 헤더를 붙여 데이터를 담은 데이터그램을 자유롭게 전송할 수 있다.

1. 데이터 생성 및 전송

애플리케이션이 보낼 데이터를 전송 계층에 넘기면, UDP는 송신자 포트 번호, 목적지 포트 번호, 데이터 길이 등의 정보를 포함해 UDP 헤더를 생성한다.
이 헤더와 사용자의 데이터를 합쳐 UDP 데이터그램을 만든다.

2. 네트워크 계층으로 전달

만든 UDP 데이터그램은 IP 프로토콜에 전달되어 IP 패킷으로 처리된다. IP는

최종 목적지의 IP 주소를 기반으로 라우팅하여 전송한다.

3. 데이터 수신
목적지 호스트에 도착한 IP 패킷에서 UDP 데이터그램을 추출한 후, UDP는 헤더에 적힌 목적지 포트 번호를 확인한다. 해당 포트 번호를 경유하는 애플리케이션 프로세스에 데이터를 전달한다.

- **포트 번호 활용**

포트 번호는 UDP의 핵심 기능 중 하나이다. 송신자와 수신자 모두 포트 번호를 통해 네트워크상의 여러 애플리케이션을 구별을 하는데
출발지 포트(Source Port): 데이터 전송을 시작하는 애플리케이션의 포트 번호로, 응답이 필요한 경우 수신자가 이 포트로 응답을 돌려준다.

목적지 포트(Destination Port): 데이터를 받을 애플리케이션의 포트 번호로, UDP 스택은 이 번호를 기반으로 데이터를 올바른 프로세스에 전달한다.

8. 보안과 네트워크 성능 측면에서의 UDP

- UDP Flood DDoS 공격

UDP Flood 공격은 UDP 프로토콜의 비연결성 특성을 악용하여 대량의 UDP 패킷을 대상 서버의 무작위 포트로 보내는 서비스 거부(DoS) 또는 분산 서비스 거부(DDoS) 공격이다.

공격 대상 서버는 해당 포트에 수신 대기하는 애플리케이션을 확인하려고 하지만, 대상 포트가 닫혀있을 경우 ICMP "목적지 도달 불가(Destination Unreachable)" 메시지를 발송한다.

대량의 UDP 패킷과 이에 따른 ICMP 응답 메시지 처리로 서버 자원이 고갈되어 정상적인 서비스가 불가능해진다.

UDP는 연결 설정이 없어 공격자가 세션을 형성하지 않고도 대량의 요청을 빠르게 전송할 수 있어 공격이 상대적으로 쉽고 파괴력이 크다.

- 그럼 위 공격과 같은 것을 어떻게 방어하느냐

네트워크 트래픽 모니터링: 비정상적으로 증가한 UDP 트래픽을 감지하여 빠르게 대응한다.

포트 및 방화벽 설정: 사용하지 않는 UDP 포트를 차단해 공격 표면을 줄인다.

속도 제한: UDP 트래픽에 대해 대역폭 및 요청 빈도 제한을 설정한다.

DDoS 대응 서비스 이용: 클라우드 기반 DDoS 방어 솔루션을 활용해 대규모 공격을 완화한다.

실시간 트래픽 분석 및 자동 차단: IDS/IPS 시스템으로 비정상 패턴을 탐지하여 즉각 차단한다.

9. UDP를 사용하는 네트워크 최적화 전략

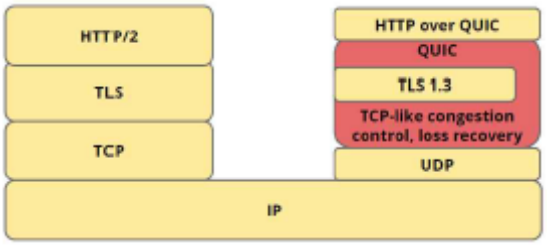
1. 소켓 버퍼 및 큐 크기 조정

UDP 소켓의 송수신 버퍼 크기를 조절하여 데이터 손실을 최소화한다. 너무 작으면 네트워크 지연이나 패킷 손실이 증가할 수 있다.
운영체제와 애플리케이션 수준에서 버퍼 크기를 최적화하여 높은 처리량을 유지한다.
2. 패킷 처리 성능 향상
멀티스레딩과 이벤트 기반 비동기 I/O를 이용해 네트워크 처리 효율을 증대시킨다.
패킷 캡처 및 처리 루프를 최적화하여 CPU 부하를 줄인다.
3. 네트워크 인프라 설계
네트워크 경로 최적화를 통해 지연 시간을 최소화한다.
QoS(Quality of Service)를 활용해 UDP 트래픽에 우선순위를 부여해 패킷 손실과 지연을 관리한다.
4. 프로토콜 레벨 최적화
애플리케이션에서 추가 오류 검출 및 수정 코드를 구현하여 UDP의 비신뢰성 문제를 보완한다.
필요 시 패킷 재전송 및 순서 재정렬 기능을 응용 계층에서 직접 구현한다.

10. 최신 UDP 관련 기술

QUIC(Quick UDP Internet Connections) 프로토콜은 Google이 2012년에 개발을 시작했고, 2021년 IETF RFC 9000으로 정식 표준화된 차세대 전송 계층 프로토콜이다. QUIC는 전통적인 TCP 대신 UDP를 사용하여 설계되었으며, UDP

의 빠른 전송 특성을 유지함과 동시에 TCP 이상의 신뢰성과 보안, 그리고 연결 효율성을 제공한다. 주요 특징으로 0-RTT 연결 설정을 지원하는데, 이 기능을 통해 이전에 연결한 서버에 바로 데이터를 전송할 수 있어 지연 시간을 크게 줄인다. 또 TLS 1.3이 통합되어 기본적으로 모든 QUIC 패킷을 암호화함으로써 중간자 공격 같은 보안 위협을 효과적으로 막는다. QUIC는 연결 마이그레이션 기능도 제공한다. 클라이언트가 네트워크 환경을 바꿔도 기존 연결을 유지해 재접속 없이 통신이 계속 가능하며, 모바일 환경에서 더욱 뛰어난 사용자 경험을 제공한다. 더불어 QUIC는 다중 스트림 멀티플렉싱을 지원하여 여러 데이터 스트림을 독립적으로 처리할 수 있다. 이를 통해 TCP에서 발생하는 헤드 오브 라인 블로킹 문제를 해결했다.
즉, 하나의 패킷이 손실되거나 지연돼도 다른 스트림에 영향을 미치지 않고 전체 통신 성능이 개선된다. 최근 HTTP/3가 QUIC를 기반으로 동작하면서 웹 서비스의 연결 속도와 응답성이 크게 향상됐으며, 실시간 스트리밍, 온라인 게임, 화상 회의 등 지연 시간이 중요한 서비스에서 QUIC 기반 UDP 사용이 점차 확대되고 있다. 이러한 UDP 기반 신기술은 기존 TCP의 한계를 극복했고, 현대 인터넷 서비스가 빠른 연결과 보안, 유연성을 동시에 만족시키는 데 중요한 역할을 한다.



결론적으로 UDP는 간단하고 빠른 데이터 전송을 가능하게 하는 핵심 전송 계층 프로토콜이다. 그 단순성 덕분에 실시간 스트리밍, 음성 및 영상 통화, 온라인 게임, DNS 쿼리 등 속도가 중요한 네트워크 서비스에서 널리 쓰이고 있다. 앞으로도 초저지연 통신과 대용량 멀티미디어 전송이 증가하는 시대에 UDP는 매우 중요한 역할을 계속할 것이다. 특히, QUIC와 같은 UDP 기반 신기술들은 UDP의 속도와 유연성에 TCP 이상의 신뢰성과 보안을 결합하여 인터넷 서비스의 전송 효율과 품질을 크게 향상시키고 있다.

미래에는 UDP가 TCP, SCTP 등 다른 프로토콜과 협력하며 더욱 발전할 전망이다. 예를 들어, TCP의 신뢰성과 흐름 제어 기능을 응용 계층에서 보완하는 방식과, UDP 기반의 다중 스트림, 연결 마이그레이션, 강력한 암호화 기술이 결합된 프로토콜이 주류로 자리잡고 있다. 또한, IoT, 자율주행, 5G/6G 네트워크 등 다양한 신기술 분야에서 UDP의 경량성과 빠른 데이터 처리 특성이 중요하게 활용될 것이다.

11. UDP 통신 및 QUdpSocket 라이브러리

QUdpSocket은 Qt 프레임워크에서 UDP 소켓 통신을 위한 클래스이다. 비동기 방식으로 동작하여 시그널과 슬롯 메커니즘으로 데이터 도착을 알리고, 송수신 작업을 간단히 처리할 수 있다.

- IP 주소 및 포트 번호 지정

```
QHostAddress ROBOT_IP = QHostAddress("192.168.188.100");
QHostAddress OPERATOR_IP = QHostAddress("192.168.188.253");
uint16_t TEXT_PORT = 9999;
```

통신 대상인 송신자와 수신자의 IP 주소를 QHostAddress 객체로 선언하여 사용할 포트 번호는 0~65535 범위인데, 1024 이상의 포트를 사용하는 것을 권장하기에 9999로 하는 것이 좋다

(포트 번호
0~1023번 (Well-known Port) 시스템이 예약한 주요 서비스용 포트 (예: HTTP 80, FTP 21)

1024~49151번 (Registered Port) 사용자 및 애플리케이션에서 설계된 포트

49152~65535번 (Dynamic/Private Port) 클라이언트가 임시로 사용하는 포트
그래서 1024번 이상의 포트를 사용하는 것이 포트 끼리의 충돌을 방지할 수 있다.)

- **UDP 소켓 생성 및 바인딩**

```
text_socket = new QUdpSocket(this);

if(text_socket->bind(OPERATOR_IP, TEXT_PORT,
QUdpSocket::ShareAddress)) {
    connect(text_socket, SIGNAL(readyRead()), this, SLOT(udp_read()));
}
```

QUdpSocket 객체를 생성하고, bind() 함수로 소켓을 로컬 IP 및 포트에 연결을하고 ShareAddress 옵션으로 같은 포트 공유가 가능한데 (단 같은 와이파이를 ㅎ사용하여야 한다)

readyRead() 시그널과 슬롯 연결로 데이터 수신 처리를 설정한다.

- 데이터 수신 (udp_read 함수)

```
void MainWindow::udp_read()
{
    QByteArray buffer;
    buffer.resize(text_socket->pendingDatagramSize());

    QHostAddress sender_ip;
    quint16 sender_port;
    text_socket->readDatagram(buffer.data(), buffer.size(), &sender_ip, &se
nder_port);
```

수신 대기 중인 데이터그램 크기에 맞춰 버퍼를 지정하
고, readDatagram()으로 데이터를 읽는다. 근데 이때 그냥 데이터만 읽는 것이 아니라 송신자 ip와 포트 정보도 함께 받아와서 활용할 수 있다

- **데이터 송신 (udp_write 함수)**

void MainWindow::udp_write(QByteArray text, uint16_t port, QUdpSocket &socket)

{

QByteArray packet;

packet.push_back(text);

socket.writeDatagram(packet, ROBOT_IP, port);

}

전달 받은 텍스트 데이터를 QByteArray 형식으로 저장하고, 지정한 ip와 포트로 udp 패킷을 송신한다

그리고 qt에서 cmake 를 활용하면

find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS Widgets Network)

find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Widgets Network)

target_link_libraries(${PROJECT_NAME} PRIVATE Qt${QT_VERSION_MAJOR}::Widgets Qt${QT_VERSION_MAJOR}::Network)

이 코드를 추가해야 실행 시킬수 있는데

find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS Widgets Network) 구문은 먼저 Qt6부터 찾아보고 없으면 Qt5를 찾는다.

찾은 Qt 버전을 QT_VERSION_MAJOR 변수에 저장해 다음 find_package(Qt${QT_VERSION_MAJOR} REQUIRED ...) 호출에 전달한다.

이렇게 하면 Qt5인지 Qt6인지 상관없이 동일한 방식으로 라이브러리를 불러

올 수 있고, CMake가 자동으로 해당 버전에 맞는 모듈을 연결한다.

target_link_libraries에서도 Qt${QT_VERSION_MAJOR}::Widgets 같은 가변 변수를 사용하여 유연한 라이브러리 링크가 가능하다.