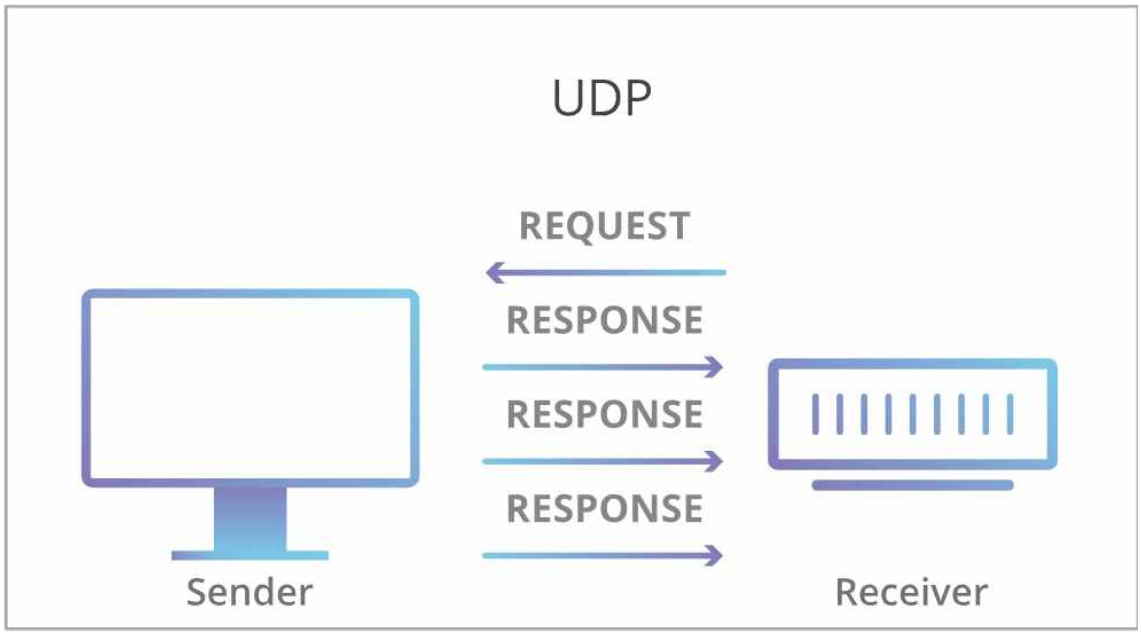
# C++&Qt day4 hw_2 UDP 조사 및 구현 보고서

로빛 20기 인턴 현창석

목차

# 1. UDP 란?




UDP 통신이란 인터넷 프로토콜 IP에서 동작하는 전송 계층 프로토콜 중 하나이다. 여기서 프로토콜이란 컴퓨터 사이의 데이터 교환 방식을 정의하는 규칙 체계라고 할 수 있다. 수많은 컴퓨터에서 막대한 양의 대이터를 형식 없이 통신을 하면 처리하는데 굉장한 비용이 소모될 것이기 때문이다. 우리가 택배를 보낼 때 수신자의 도로명 주소, 수신자 이름, 우편 번호 등을 기재하여 원활히 택배가 전달될 수 있도록 하는 것처럼 원활한 전송을 위해서 규칙을 정한 것을 프로토콜이라고 한다.

UDP 통신은 OSI 7계층 중 4계층 전송 층에서 사용된다.
OSI 7계층이란 무엇일까.

## -OSI 7계층

OSI 7계층이란? (Open System Interconnection)의 약어로 네트워크에서 통신이 일어나는 과정을 7단계로 구분하여 만든 표준규격이다.


계층으로 나눈 이유가 무엇일까
계층을 나눈 이유는 통신이 일어나는 과정이 단계별로 파악할 수 있기 때문이다.
데이터 송수신 흐름을 한눈에 알아보기 쉽고, 사람들이 쉽게 이해할 수 있으며 7단계 중 특정한 곳에 이상이 생기면 다른 단계의 장비 및 소프트웨어를 건들이지 않고도 이상

이 생긴 단계만 고칠 수 있기 때문이다.


1계층은 물리계층(Physical Layer)이다. 이 계층에서는 주로 전기적, 기계적, 기능적인 특성을 이용해서 통신 케이블로 데이터를 전송하게 된다.
이 계층에서 사용되는 통신 단위는 비트이다. 이것은 1과 0으로 이진수로 나타낸다, 전기적으로 On, Off 상태이다.

이 계층에서는 단지 데이터만 전달하며 전송 또는 받으려는 데이터가 무엇인지, 어떤 에러가 있는지 등에는 고려하지 않고 진행된다.

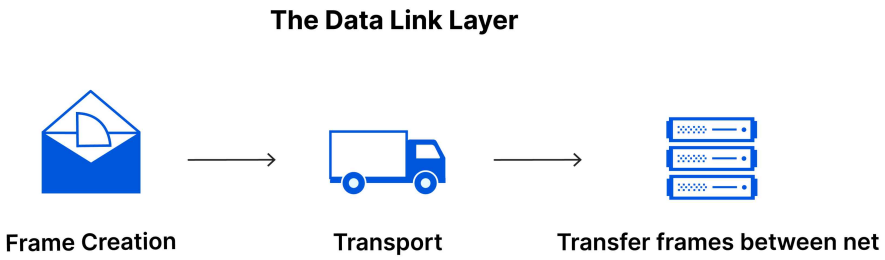
쉽게 데이터 전기적인 신호로 변환해서 주고받는 기능만 할 뿐이다. 이 계층에 속하는 대표적인 장비는 통신 케이블, 리피터, 허브 등이 있다.

리피터

허브

2계층은 데이터 링크계층(DataLink Layer)이다.

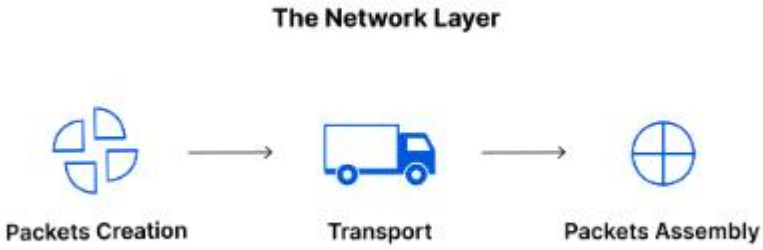
데이터 링크계층이 하는 일은 물리계층을 통해 송수신되는 정보의 오류와 흐름을 관리하여 안전한 정보의 전달을 수




행할 수 있도록 도와주는 것이다.
결과적으로 통신에서의 오류를 찾고 또한 재전송도 하는 기능을 제공한다.
이 계층에서는 맥 주소를 가지고 통신하게 된다.
이 계층에서 전송되는 단위를 프레임이라고 하고, MAC주소를 사용하는 대표적인 장비로 브리지, 스위치 등이 있다.

스위치

데이터 링크 계층(Data link layer)은 포인트 투 포인트간 신뢰성 있는 전송을 보장하기 위한 계층인데
CRC 기반의 오류 제어와 흐름 제어가 필요하다는 특징이 있다. 네트워크 위의 개체들 간 데이터를 전달하고,
물리 계층에서 발생할 수 있는 오류를 발견하며 수정하는 데 필요한 기능적 및 절차적 수단을 제공하는 역할을 한다. 주소 값은 물리적으로 할당 받는다. 이는 네트워크 카드가 만들어질 때부터 맥 주소(MAC address)가 정해져 있기 때문이다.
주소 체계는 계층이 없는 단일 구조이며 이러한 예 중 하나가 바로 이더넷이다..
이 외에도 HDLC나 ADCCP 같은 포인트 투 포인트(point-to-point) 프로토콜이나
패킷 스위칭 네트워크나 LLC, ALOHA 같은 근거리 네트워크용 프로토콜이 있다.
네트워크 브릿지나 스위치 등이 이 계층에서 동작하며, 직접 이어진 곳에만 연결할 수 있다.

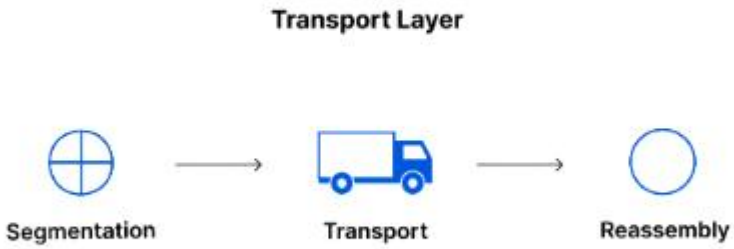



3계층은 네트워크 계층이다.
이 계층에서 가장 중요한 기능은 데이터를 목적지까지 가장 안전하고 빠르게 전달하는 기능이며 이 기능은 라우팅이라고도 불린다.
여기에 사용되는 프로토콜의 종류와 라우팅하는 기술은 매우 다양하다.
이 계층은 경로와 주소를 정하고 경로에 따라 패킷을 해당 주소를 향해 전달해주는 것이 이 계층의 역할이라고 볼 수 있다. 이 계층의 대표적인 장비는 '라우터'이며, 요즘은 2계층의 장비 중 스위치라는 장비에 라우팅 기능을 장착한 Layer 3 스위치도 나오고 있다고 한다.

다시 정리해 네트워크 계층(Network layer)은 여러 개의 노드를 거칠 때마다 경로를 찾는 기능을 하는 계층으로
다양한 길이의 데이터를 네트워크들을 통해 전달하고, 그 과정에서 전송 계층이 요구하는 서비스 품질(QoS)을 제공하기 위한 수단들을 제공한다.

축약하여 네트워크 계층은 라우팅, 흐름 제어, 세그멘테이

션(segmentation/desegmentation), 오류 제어, 인터네트워킹(Internetworking) 등을 수행한다고 할 수 있다.

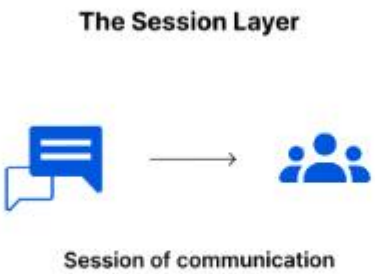



4계층에 해당하는 전송 계층(Transport Layer)이 우리가 구현하는 부분이다.
이 계층은 통신을 활성화하기 위한 계층이다. 보통 TCP프로토콜 또는 UDP를 이용하며, 포트를 열어서 응용프로그램들이 전송을 할 수 있도록 한다.
만약 데이터가 왔다면 4계층에서 해당 데이터를 하나로 합쳐 후 다음 5계층에 해당하는 세션 계층에 던져 준다.
단대단 오류제어 및 흐름제어 이 계층 까지는 물리적인 계층에 속한다. 전송 계층(Transport layer)은 양 끝단(End to end)의 사용자들이 신뢰성 있는 데이터를 주고 받을 수 있도록 해 주는 역할을 한다.
상위 계층들이 데이터 전달의 유효성이나 효율성을 생각하지 않도록 해준다.
시퀀스 넘버 기반의 오류 제어 방식을 사용한다.

전송 계층은 특정 연결의 유효성을 제어하고, 일부 프로토콜은 상태 개념이 있으며 연결을 기반으로 한다.

이는 전송 계층이 패킷들의 전송이 유효한지 확인한 후 전송에 만약 실패하였다면 그 전송 실패한 패킷들을 다시 전송한다는 것을 뜻한다.
종단간(end-to-end) 통신을 다루는 최하위 계층에 해당하며 종단간 신뢰성 있고 효율적인 데이터를 전송한다,
오류검출 및 복구와 흐름제어, 중복검사 등의 기능을 수행할 수 있다.

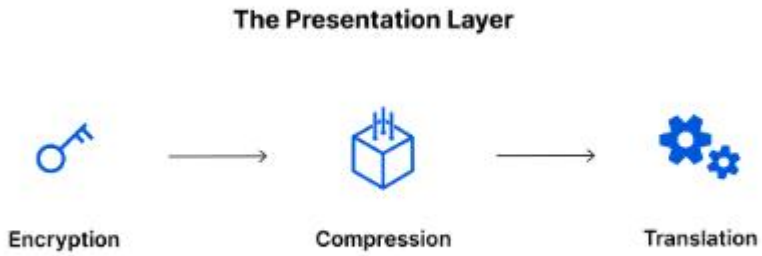
다음  5계층에 해당하는 세션 계층(Session Layer) 이다.
데이터가 통신하기 위한 논리적인 연결을 말한다. 통신을 하기위한 대문의 역할이다.




하지만 4계층에서도 연결 후 종료할 수 있기 때문에 우리가 어느 계층에서 통신이 끊어 졌나 판단할 수 없다는 점에서 한계가 있다.
그러므로 세션 계층은 4 계층과 무관하게 응용 프로그램

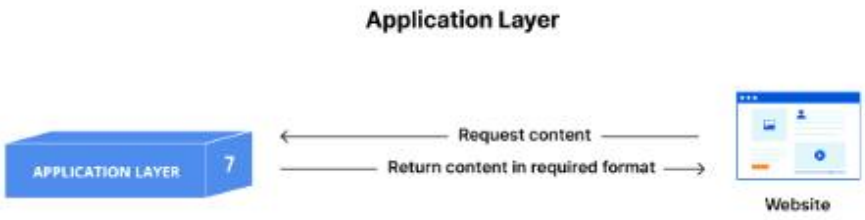
관점에서 봐라 보아야할 필요성이 있다.
세션 계층은 세션 설정, 유지, 종료, 전송 중단 시 복구 등의 기능을 사용할 수 있다.
세션 계층은 양 끝단의 응용 프로세스가 통신을 관리하기 위한 방법을 제공하며
동시 송수신 방식(duplex), 반이중 방식(half-duplex), 전이중 방식(Full Duplex)의 통신과 함께, 체크 포인팅과 유휴, 종료, 다시 시작 과정 등을 수행한다.

즉 한 줄로 설명하자면 이 계층은 TCP/IP 세션을 만들고 없애는 책임을 진다.




6계층에 해당하는 계층인 표현 계층(Presentation Layer)이다.
데이터 표현이 상이한 응용 프로세스의 독립성을 제공하고, 암호화하는 역할을 담당한다.

표현 계층(Presentation layer)은 코드 간의 번역을 담당하여 사용자 시스템에서 데이터의 형식상 차이를 다루는 부담을 응용 계층으로부터 덜어 준다. 예로 들어 MIME 인코딩이나 암호화 (,EBCDIC로 인코딩된 문서 파일을 ASCII로 인코딩된 파일로 바꿔 주기), 등의 동작을 이 계층에서 수행한다.
해당 데이터가 TEXT인지, 그림인지, GIF인지 JPG인지의 구분 등이 표현 계층이 해야할 과제이다.




마지막 7계층에 해당하는 응용 계층(Application Layer)이다.

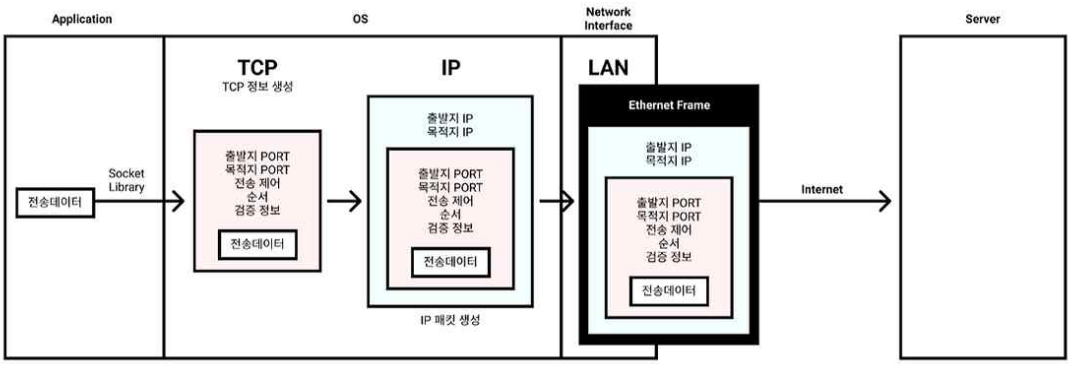
최종 목적지로서 해당하는 프로토콜로 HTTP, FTP, SMTP, POP3, IMAP, Telnet 등이 있다.
해당 통신 패킷들은 바로 위 문장에서 나열한 프로토콜에 의해 모두 처리되며 우리가 사용하는 브라우저나, 메일 프로그램은 프로토콜을 보다 쉽게 사용하게 해주는 응용프로그램에 해당한다. 즉, 모든 통신의 양 끝단은 HTTP와 같은 프로토콜이지 응용프로그램이 아니다.

응용 계층(Application layer)은 응용 프로세스와 직접 관련되어 있어서 일반적인 응용 서비스를 수행하는 기능을 한다.
일반적인 응용 서비스는 관련된 응용 프로세스들 사이의 전환을 제공한다고 한다.

다시 UDP설명으로 돌아와서 위에서 언급했듯 프로토콜의 종류에는 HTTP. TCO, UDP 등등 다양하게 존재한다.
UDP는 TCP와 다르게 흐름 제어를 하지 않는다는 특징이 있다.
UDP와 TCP의 차이점을 여기서 알아볼 필요가 있다.
먼저 TCP란 무엇일까?

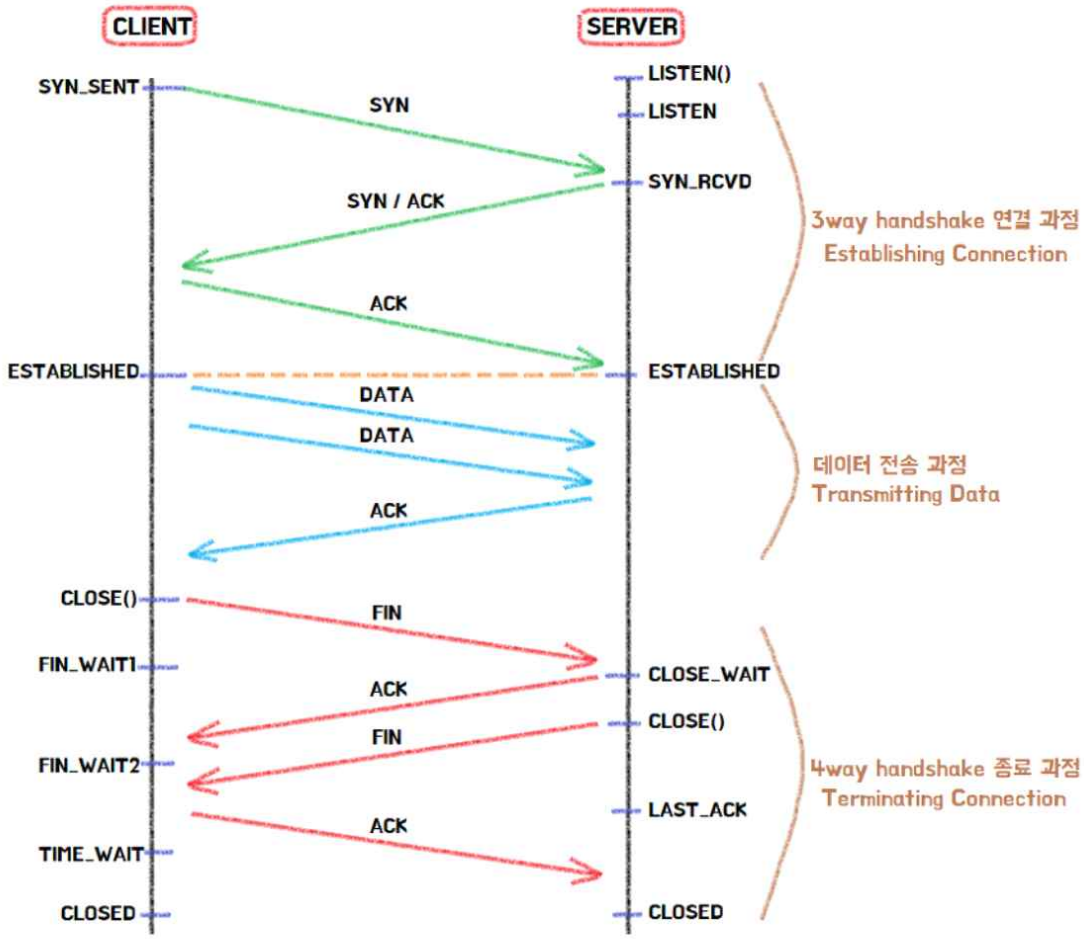
-TCP

아까 택배를 예로 다시 들어서  IP는 배달 주소지로 가정을 하면 TCP는 이 배달지로 문제없이 전송되도록 택배 스티커와 같은 여러 부가 정보들을 추가 한 것이라고 보면




된다. 단순히 목적지 뿐만 아니라 순서, 검증, 전송 제어 정보가 들어있어 IP 주소지로만 물품을 배달하기엔 불안정한 부분들을 확실히 보완하여  배달품이 목적지까지 안전하게 도착하도록 보증한다. 이를 전송 데이터로 생각하여 전송 과정을 표현해보면 우선 첫 번째로 전송데이터를 TCP 포장한다. 두 번째로 포장한 전송데이터를 IP 포장한다. 세 번째로 포장한 전송데이터를 이더넷 포장한다.  인터넷을 통해 상대 서버에 도달하면 포장을 풀며 전송데이터를 받는다.

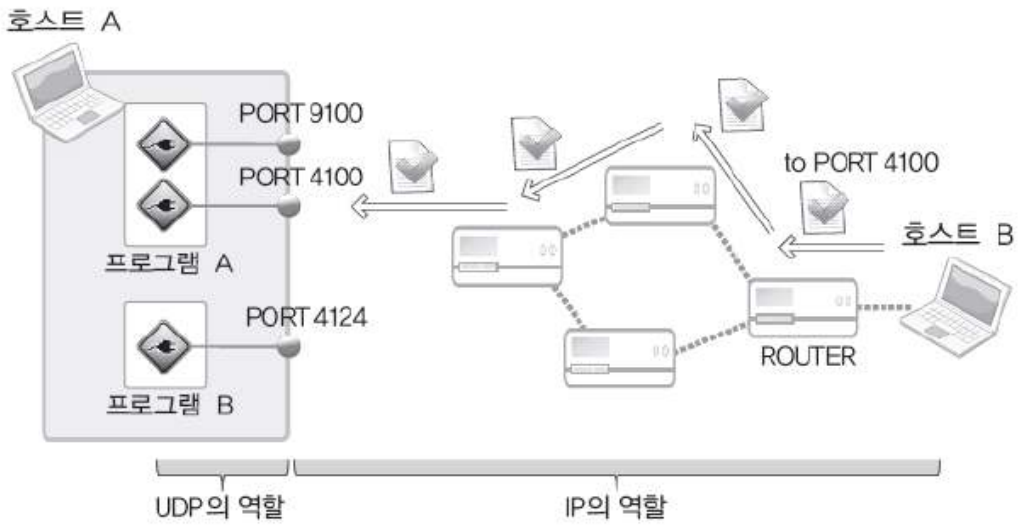
TCP는 통신을 시작하거나 종료할 때 송수신 준비 여부를 확인하고 패킷을 전송할 순서를 정하고 본격적으로
통신한다 . 이렇게 TCP는 뛰어난 신뢰성을 보이는 프로토콜로 은행 업무나 메일과 같이 상대방이 무조건 정보를 받아야하는 신뢰성 있는 통신이 필요할 때 사용한다. 이러한 과정을 3 Way Handshake 와 4 Way Handshake 과정이라고 부른다.




위 사진처럼 클라이언트가 처음 서버와 통신을 하기 위해 TCP 연결을 생성할 때 SYN와 ACK이라는 패킷을 주고 받고 통신을 종료하는 과정에서는 FIN이라는 패킷을 주고 받는다. FLAG들의 기능을 정리하여 설명하면  SYN은 서버에 접속요청을 할 때 보내는 패킷을 의미하여 TCP 접속시 가장 먼저 보내는 패킷이다. ACK는 상대로부터 패킷을 받은 뒤 정상적으로 수신이 되었다는 것을 알려주는 패킷이다. PSH 는 데이터를 즉시 목적지로 보내라는 플래그이며 FIN은 젒속 종료를 위한 플래그로 이 패킷을 보내는 곳이 현재 접속하고 있는 곳과 접속을 끊고자 할 때 사용한다.

## - TCP와의 차이점

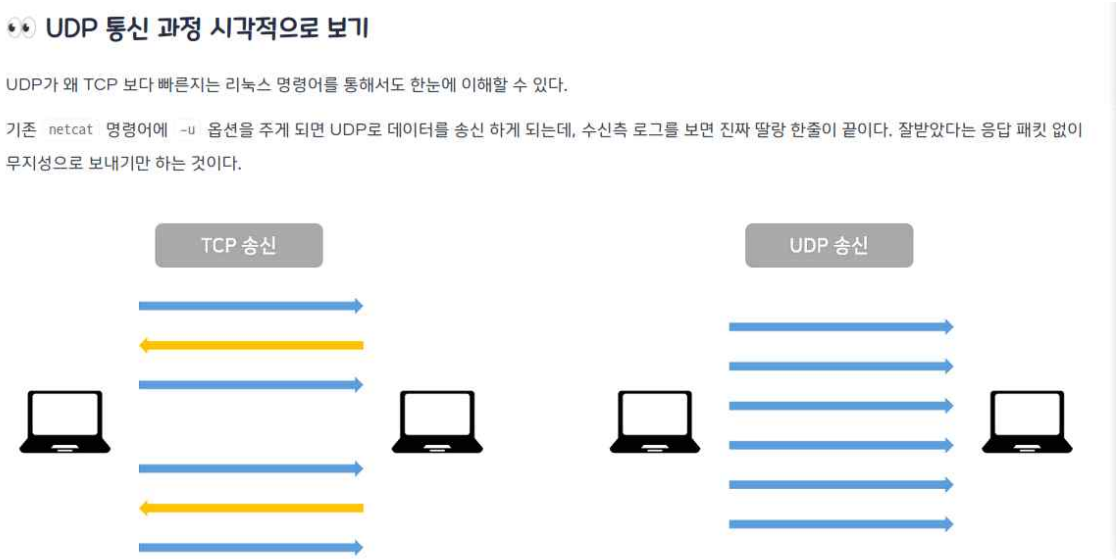
클라이언트와 서버가 서로 신뢰성 있는 통신을 할 수 있도록 TCP는 핸드쉐이크 과정을 거쳐 통신을 주고받기 때문에 결과적으로 속도 한계점이 명확히 존재했다. 비해 UDP 통신은 굉장히 빠른 전송 속도를 보여준다.  그 이유는 패킷의 교환 방식에 있다.
UDP(즉 User Datagram Protocol)는 단어에서도 알 수 있듯이 데이터그램 방식을 사용하는 프로토콜이기 때문에 애초에 각각의 패킷 간의 순서가 존재하지 않는 독립적인 패킷을 사용한다.
이 데이터 그램 방식은 패킷의 목적지만 설정해주면 핸드쉐이크 과정을 거치지 않고 통신할 수 있다.




▶ 그림 06-1 : 패킷 전송에 있어서의 UDP와 IP의 역할

위에서 설명을 했듯 UDP는 TCP와 달리 흐름제어를 하지 않는다. 수신자에서 UDP 패킷이 수신자에게 전달되도록

하는 것은 IP의 역할이다.
이렇게 전달된 UDP 패킷을 UDP 소켓 중 하나에 최종적으로 전달하는 것은 UDP의 역할이다.




기존 TCP가 신뢰성을 확보하기 위해서 거치던 핸드쉐이크 과정을 거치지 않기 때문에 통신 속도가 매우 빠르지만 신뢰성이 없기 때문에 손실, 중복, 순서 꼬임 등이 발생할 수 있다.
이러한 특징으로 실시간성이 중요한 온라인 게임이나 화상 회의, 스트리밍 등에 주로 이용된다.

# 2. UDP 구현 코드 설명

```
7  MainWindow::MainWindow(QWidget *parent)
8      : QMainWindow(parent)
9  {
10     setWindowTitle("Kakaotalk");
11
12     QWidget *central = new QWidget(this);
13     setCentralWidget(central);
14
15     QVBoxLayout *layout = new QVBoxLayout(central);
16
17     chatDisplay = new QTextEdit(this);  //0 채팅을 출력하는 텍스트 창 생성
18     chatDisplay->setReadOnly(true);  //  읽기 전용임
19
20     messageInput = new QLineEdit(this);
21     sendButton = new QPushButton("Send", this);
22
23     //추가 UI
24     target_address = new QLineEdit(this);  // 상대의 IP를 입력하는 란
25     target_address->setPlaceholderText("Target IP");
26
27     rx_port = new QLineEdit(this);
28     rx_port->setPlaceholderText("Receive Port");    // 수신 포트 설정
29     rx_port->setValidator(new QIntValidator(1, 65535, this)); // 제한 범위 설정
30
31     tx_port = new QLineEdit(this);
32     tx_port->setPlaceholderText("Transmit Port");   // 송신 포트 설정
33     tx_port->setValidator(new QIntValidator(1, 65535, this));
34
35     nickname = new QLineEdit(this);
36     nickname->setPlaceholderText("Nickname");       // 내 이름을 입력하는 란
37
38     QHBoxLayout *inputLayout = new QHBoxLayout();
39     inputLayout->addWidget(messageInput);
40     inputLayout->addWidget(sendButton);
41     inputLayout->addWidget(target_address);
42     inputLayout->addWidget(rx_port);
43     inputLayout->addWidget(tx_port);
44     inputLayout->addWidget(nickname);
45                                                    // 위젯들 배치
46     layout->addWidget(new QLabel("Chat:"));
47     layout->addWidget(chatDisplay);
48     layout->addLayout(inputLayout);
49
```

아래 사진들은 mainwindow.cpp 파일의 코드를 모두 캡쳐한 사진들이다. 포인터 변수 central을 선언하여 mainwindow의 중앙 위젯으로 설정하였다. QVBoxLayout 포인터 변수 *layout 을 선언해 위젯을 수직으로 정렬되도록 하였다. chatDisplay = new QTextEdit(this); 줄은 채팅을 출력하는 텍스트 창을 생성

하며 setReadOnly를 활성화시켜주어 이를 읽기 전용으로 설정하였다.

QLineEdit으로 상대방의 주소를 입력하는 란을 만들어주었다. rx_port와 tx_port를 직접 입력할 수 있도록 Qlineedit으로 송수신 포트를 설정하였다. 숫자 범위를 1에서 65535로 제한하여 UDP 포트번호 내에서 입력받을 수 있도록 하였다. 이렇게 설정한 위젯들을 addwiget 으로 layout에 출력한다.

```
50
51      // UDP 통신용 소켓을 생성
52
53      udpSocket = new QUdpSocket(this);
54      udpSocket->bind(QHostAddress::Any, 8081, QUdpSocket::ShareAddress | QUdpSocket::ReuseAddressHint); // 포트 8081에 바인딩 한 후 수신 준비
55
56
57      connect(sendButton, &QPushButton::clicked, this, &MainWindow::sendMessage);
58      connect(messageInput, &QLineEdit::returnPressed, this, &MainWindow::sendMessage);  // 버튼 클릭 혹은 엔터키 입력하면 메세지 송신
59      connect(udpSocket, &QUdpSocket::readyRead, this, &MainWindow::receiveMessage); //UDP소켓에 데이터가 들어오면 메시지를 수신
60
61      //수신 포트 변경 시
62      connect(rx_port, &QLineEdit::editingFinished, this, [=](){
63          quint16 port = rx_port->text().toUShort();
64          if(port > 0) {
65              udpSocket->close();
66              udpSocket->bind(QHostAddress::Any, port, QUdpSocket::ShareAddress | QUdpSocket::ReuseAddressHint);
67              chatDisplay->append("Receive port changed to: " + QString::number(port));
68          }
69      });
70  }
```

UDP 통신을 하기위해서 udpsocket을 Qudpsocket의변수로 선언하여 생성해주었다. 그리고 우선 기본적으로 UDP 데이터를 받을 수 있도록 하기 위해서 임시로 포트 8081로 바인딩하여 초기화해주었다. connect를 이용해 버튼을 클릭하거나 또는 엔터키 를 치면 사용자가 입력한 메세지를 상대방에게 송신할 수 있도록 하였다. 또한 udp 소켓에 데이터가 들어오면 메세지를 수신하는 함수를 실행시키도록 한다. 그 아래 connect는 수신 포트 주소를 변경하였을 때 실행되는 코드이다.

```
74  void MainWindow::sendMessage()
75  {
76      QString text = messageInput->text();
77      if (text.isEmpty()) return;
78
79      QString ip = target_address->text();
80      quint16 port = tx_port->text().toUShort();//숫자용으로 변환              //IP, 포트, 닉네임 읽어 각 변수에 저장
81      QString name = nickname->text();
82      if(name.isEmpty()) name = "Me";
83
84      if(ip.isEmpty() || port == 0) {
85          chatDisplay->append("Error: Set target IP and port");
86          return;
87      }
88
89      // 닉네임과 메시지를 하나의 문자열로 합침
90      QString fullMessage = name + ": " + text;
91
92      QByteArray data = fullMessage.toUtf8();
93      udpSocket->writeDatagram(data, QHostAddress(ip), port);
94
95      // 내 화면에도 동일하게 표시
96      chatDisplay->append(fullMessage);
97      messageInput->clear();
98  }
99
```

메세제를 수신하는 sendMessage() 함수에서는 통신을 하기 위해 필요한 text(내용) , ip, tx_port 주소를 변수에 저장하여 초기화 해준다.  네트워크 전송은 바이트 단위로 전송하기 때문에 메세지 내용, 즉 문자열을 UTF-8 인코딩된 바이트 배열로 변환하여 data 변수에 저장하였다. udpSocket->writeDatagram은 실제로 UDP패킷을 전송하는 함수이다.  보낼 데이터 (메세지 내용) 과 상대방의 ip 주소. 상대방의 송신 포트 번호를 매개변수로 하여 상대방에게 송신을 한다.

```
99
100
101 void MainWindow::receiveMessage()
102 {
103     while (udpSocket->hasPendingDatagrams()) { //대기 중인 패킷이 있으면 모두 처리
104         QByteArray datagram;
105         datagram.resize(int(udpSocket->pendingDatagramSize()));
106         QHostAddress sender;
107         quint16 senderPort;
108
109         // 실제로 UDP 패킷 데이터를 읽는 함수,  datagram.data()는 데이터를 담을 배열 포인터 .  size() → 배열 크기
110         //  &sender은 발신자의 IP 주소를 받을 변수의 주소,    &senderPort은 발신자의 포트를 받을 변수 주소
111         // 읽은 모든 패킷 내용은 datagram에 저장한다.
112         udpSocket->readDatagram(datagram.data(), datagram.size(), &sender, &senderPort);
113         QString msg = QString::fromUtf8(datagram); // 바이트 단위 데이터인 datagram의 데이터를 문자열로 변환, 글자로 사용이 가능
114         chatDisplay->append(msg);  // 새로운 메세지 출력
115     }
116 }
117
```

receiveMessage함수는 메세지를 수신하는 함수이다. 수신하여 대기 중인 패킷이 있으면 while문을 계속 돌며 int(udpSocket->pendingDatagramSize()) 로 대기 중인 다음 패킷의크기만큼 datagram의 공간을 확보한다(resize).

dpSocket->readDatagram함수에서  메세지 내용을 담은 배열 포인터와 베열의 크기, 송신자의 ip주소,  송신자의 포트 번호를 저장하는 변수를 이용해  읽은 모든 패킷 내용은 datagram에 저장한다. 위에서 설명했듯 UDP 통신은 단위로 이루어 지므로 글자로 사용하기 위해서 fromUtf8(datagram);로 바이트 단위 데이터를 문자열로 변환한 후 채팅화면에 출력한다.

UDP 구현 부분을 정리해보면 소켓 초기화 단계에선 우선적으로 QUdpSocket 객체를 생성한다.
그 후 bind() 함수를 사용해 특정 포트에 연결한다.

이때, 모든 IP 주소에서 오는 데이터를 받을 수 있도록 QHostAddress  Any를 사용한다.
ShareAddress와 ReuseAddressHint 옵션을 주면 소켓을 여러 프로그램에서 동시에 사용하거나, 소켓을 재사용할 수 있다.

데이터 송신 단계에선 사용자가 입력한 문자열 메시지를 UTF-8로 인코딩하여 바이트 배열로 변환한다.
writeDatagram()을 통해 특정 IP 주소와 포트로 UDP 패킷을 전송한다.
내가 보낸 메시지는 내 채팅창에도 출력된다..
메시지를 보낸 후 입력창을 비우고 다음 메시지를 준비한다.

데이터 수신 단계에서는
소켓에새로운데이터가 도착했는지hasPendingDatagrams()로 확인한다.
데이터 크기에 맞는 버퍼(QByteArray)를 만들어 공간을 확보한다.
readDatagram()으로 실제 데이터를 읽는데 이때 보낸 사람의 IP 주소와 포트도 함께 알 수 있다.
그 후 받은 데이터를 UTF-8 문자열로 변환하고 변환한 문자열을 채팅창에 출력한다.

유효성 검사 단계에서는 사용자가 입력하는 포트 번호가 잘못되지 않도록 QIntValidator를 이용해 1~65535 범위의

숫자만 입력 가능하게 한다.