C++&Qt day4 hw2 UDP 보고서

2025407012/로봇학부/송연우

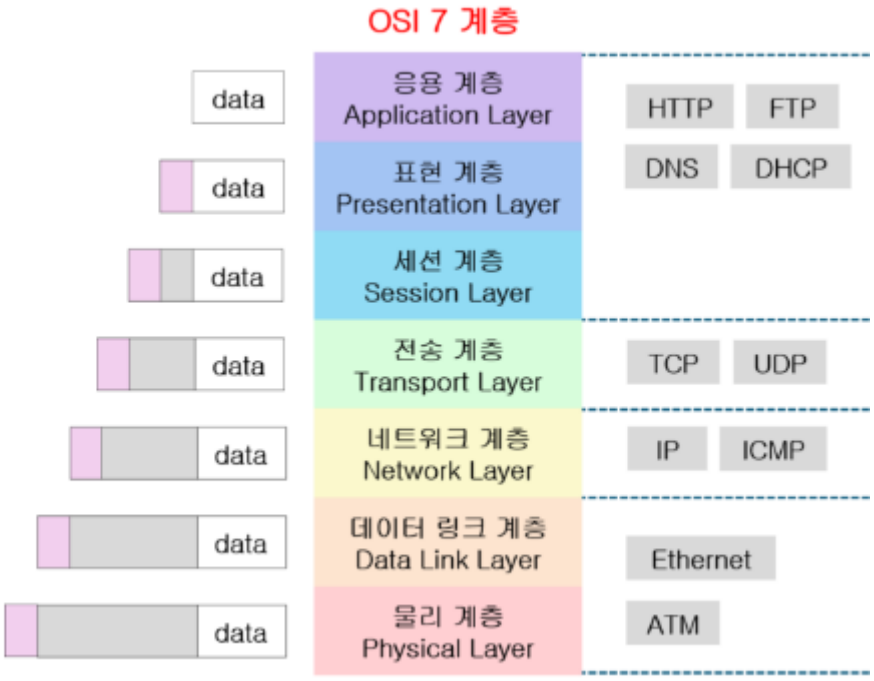
1. UDP 개요

1) UDP 통신은 데이터의 신뢰성보다 속도가 중요한 환경에서 사용되는 전송 계층 프로토콜입니다. 같은 4계층의 TCP 와 다르게 패킷 확인 과정이 없어 데이터를 즉시 전송하고, 데이터 순서나 도착 여부를 보장하지 않아 실시간 스트리밍, 온라인 게임, DNS 통신 등에서 주로 활용됩니다. 통신 속도가 빠르지만 신뢰성이 낮아 DDoS 공격에 악용될 수 있습니다.

(여기서 DDoS 공격은 UDP Flooding 으로 정상적인 사용자 및 요청이 시스템, 서버, 대역폭 또는 시스템을 사용할 수 없도록 설계된 DoS 공격의 한 종류를 말합니다. UDP 의 효율적인 전송 속도를 통해 대상 서버에 대한 UDP 패킷을 서버에서 처리할 수 없을 정도의 양으로 보내 시스템을 마비시킵니다.)

2) OSI 모형:

OSI 7계층은 국제표준화기구(ISO)에서 제안한 네트워크 통신 모델로, 데이터 송수신 과정을 7개의 계층으로 나눈 것을 의미합니다. 가장 하위 계층부터 물리 계층, 데이터 링크 계층, 네트워크 계층, 전송 계층, 세션 계층, 표현 계층, 응용 계층 순서로 구성되어 각 계층은 고유한 기능을 수행하고 상호 협력해 데이터를 주고받습니다.



- 1. 물리 계층: 0과 1 같은 비트 데이터를 물리적인 전기 신호/광 신호로 변환해 전송하는 계층입니다.

- 2. 데이터 링크 계층: 물리 계층에서 전달된 비트 데이터를 프레임 단위로 묶어 오류를 검출하고, 물리적인 장치를 식별해 데이터를 신뢰성 있게 전송합니다.

- 3. 네트워크 계층: 여러 네트워크를 통해 데이터를 목적지까지 전달하기 위한 최적의 경로를 결정하고, IP 주소를 사용해 목적지 컴퓨터를 찾아 데이터를 전송합니다.

- 4. 전송 계층: 데이터 신뢰성을 보장하기 위해 데이터를 여러 개 패킷으로 나누고, 수신 측에서 재전송 여부를 확인해 데이터 손실을 막습니다. 대표적인 프로토콜로 TCP 와 UDP 가 있고 포트 정보를 통해 통신할 어플리케이션을 결정합니다.

- 5. 세션 계층: 양쪽 컴퓨터 간 통신 세션을 담당합니다. 세션을 설정/유지/관리하는 역할을 하며, 통신 연결을 제어하고 오류를 복구합니다.

- 6. 표현 계층: 응용 계층에서 사용되는 다양한 종류의 데이터를 네트워크 상에서 호환되는 형식으로 변환하고, 암호화 및 압축하는 등 기능을 수행합니다.

- 7. 응용 계층: 사용자가 직접적으로 사용하는 응용 프로그램과 네트워크 간 인터페이스 역할을 합니다. HTTP, FTP, SMTP 등 다양한 응용 프로토콜이 이 계층에서 동작합니다.

UDP 는 OSI 7계층 모델에서 4계층에 위치하며, 데이터를 전송합니다.

3) UDP 구조:

UDP 의 전송 단위는 데이터그램이며, 8바이트로 통신하는 프로토콜입니다. 아래는 UDP 의 패킷 구조를 나타낸 그림으로, 헤더가 8바이트를 차지해 헤더가 20바이트를 차지하는 TCP 에 비해 매우 간단하게 나타납니다.

밑의 그림은 UDP 헤더의 구조를 나타낸 것이며, 각각의 항목이 2바이트씩 차지함을 알 수 있습니다.

출발지 포트(16Bit)	목적지 포트(16Bit)	길이(16Bit)	오류 검사(16Bit)

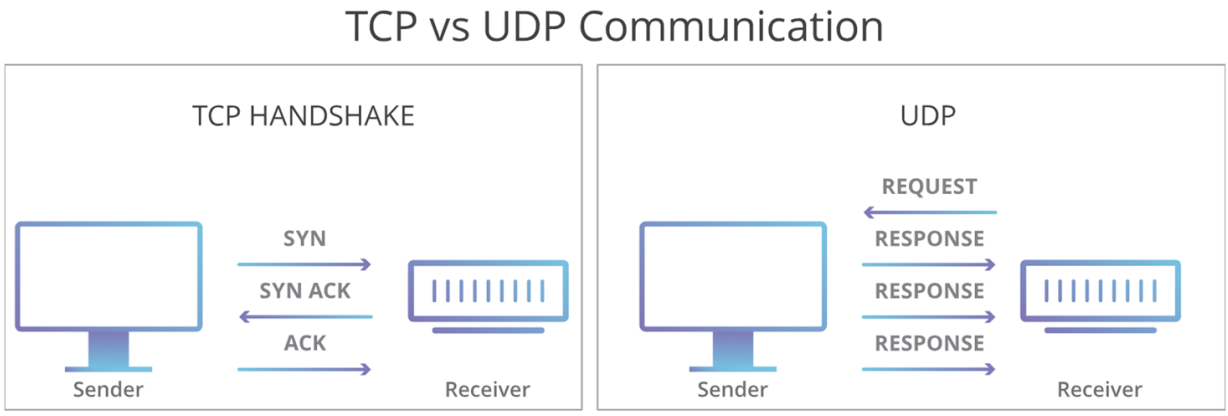
발신/수신하는 포트의 번호는 16비트의 포트 번호를 사용하며 길이는 최소 8바이트, 최대 65,535바이트까지 늘어날 수 있습니다. 단 최대 데이터의 크기는 IP (헤더가 20 바이트 이므로)수용 제한에 따라 약 65,507바이트까지입니다.
데이터 전송에서 오류가 발생하지 않았는지 체크하는 체크섬은 선택 항목으로 값이 0이면 수신측은 따로 체크섬 계산을 하지 않습니다.
UDP 에서의 체크섬 계산 대상은 다음 그림과 같이 가상 헤더+실제 헤더+데이터+Padding 입니다.

2. 특징,/다른 통신과의 비교

1) 비연결성: 데이터를 보내기 전에 송신자와 수신자 간의 연결을 수립하는 과정이 없습니다.

2) 빠른 속도: 연결 설정과 확인 절차가 없어 TCP 보다 훨씬 빠르게 데이터를 전송할 수 있습니다.
3) 신뢰성 부족: 데이터의 도착 여부, 순서, 오류 등을 보장하지 않습니다.
4) 낮은 오버헤드: TCP 에 비해 헤더의 크기가 작아 (8바이트) 데이터 전송 효율이 높습니다.

5) TCP 와의 비교: 상대방의 수신을 확인하는 과정이 있는 TCP 는 신뢰성이 높고 속도는 느린 반면, 확인과정이 없는 UDP 의 경우 신뢰성이 낮고 속도가 빠르다는 특징이 있습니다.



특징	UDP	TCP
연결성	비연결형	연결형
신뢰성	신뢰성 낮음(데이터 보장 안 함)	신뢰성 높음(데이터 보장)
속도	빠름	느림
오버헤드	낮음(8바이트 헤더)	높은(20바이트 헤더)
주요 용도	스트리밍, 온라인 게임, DNS	웹 브라우징(HTTP), 이메일(SMTP)

3. 작동 방식

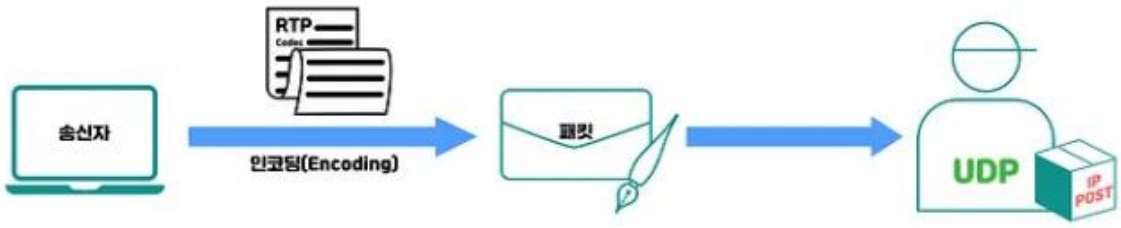
다른 네트워킹 프로토콜과 같이 UDP 는 네트워크의 두 컴퓨터 간에 데이터를 전송하기 위한 표준화된 방법입니다. UDP 는 먼저 연결을 설정하거나, 해당 패킷의 순서를 표시하거나, 의도한 대로 도착했는지 여부를 확인하지 않고 패킷(데이터 전송 단위)을 대상 컴퓨터로 직접 보내는 간단한 방식으로 프로세스를 수행합니다. 여기서 UDP 패킷을 '데이터그램'이라고 부릅니다.

4. 주요 사용 사례

1) 실시간 스트리밍: 동영상 스트리밍이나 음성 통화처럼 데이터 손실이 일부 허용되는 실시간 서비스에 적합합니다.

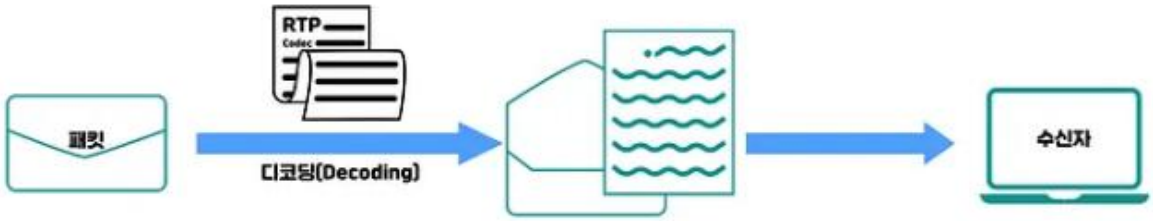
- 실시간 스트리밍: 실시간 스트리밍과 같은 실시간 비디오, 오디오 및 데이터 등을 전송하는 프로그램에서는 UDP 를 활용한 RTP 가 쓰입니다. UDP 가 데이터를 데이터그램 단위로 쪼개어 다중화된 망을 통해 가장 효율적이고 빠르게 전송하는 방법에 관한 프로토콜이었다면, RTP 는 이 UDP 위에서 패킷들을 해석하는 방법에 대한 약속입니다.

실시간으로 영상/음성 데이터를 보내고자 하는 사람과 받고자 하는 사람이 있다고 할 때, 데이터를 보내는 쪽에서 데이트럴 패킷으로 잘게 쪼개고, 각각의 패킷에 대한 정보를 편지지에 적습니다. 이때 RTP 에는 이 패킷을 어떤 코덱으로 인코딩을 할 지에 대한 명세가 담깁니다.
이렇게 작성된 패킷 편지는 UDP 라는 택배회사를 통해서 IP 네트워크를 경유해 목적지까지 도착합니다.



인코딩 과정의 요약도. 송신자 측의 패킷은 RTP에 담긴 코덱에 맞춰 인코딩되고, 이 패킷은 UDP를 통해 수신자에게 전달된다.

각각의 패킷이 목적지에 도달하게 되면 데이터를 받고자 하는 사람은 도착한 패킷 편지를 하나하나 읽어봅니다. 이때 편지에 적힌 내용이 코덱에 맞춰 인코딩되어 있기 때문에 다시 이에 맞춰 읽을 수 있는 형태로 해석해야 합니다. 이 과정이 디코딩입니다.



디코딩 과정의 요약도. 목적지에 도착한 패킷은 다시 코덱에 따라 디코딩되고 수신자는 이를 전달받아 데이터를 들여다본다.

이때 실시간으로 영상 데이터를 전송하는 경우, 크기가 매우 커 한번에 모든 데이터를 다 보내지 못합니다. 그렇기에 RTP 에서 사용되는 코덱들은 빠르지만 효고적인 방법을 제시합니다. 처음에 한번만 비디오의 전체 프레임을 보내고, 그 다음부터는 바뀐 부분만을 전송합니다. 비슷한 프레임이 연속적으로 이어지는 영상 데이터의 특성상 가능한 일입니다. RTP 는 UDP 를 통해 통신하고 문제가 발생했을 때 이를 해결하는 메커니즘 또한 가지고 있기도 합니다.

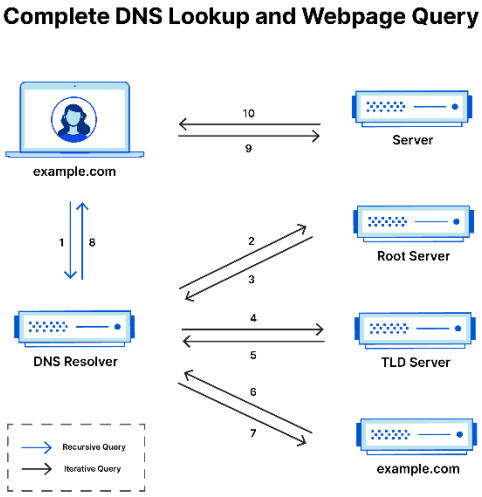
2) 온라인 게임: 빠르고 지연 없는 통신이 중요한 게임 서버와 클라이언트 간의 통신에 사용됩니다.

- 온라인 게임: 온라인 게임에서는 TCP 와 UDP 통신방식 모두 사용할 수 있지만 온라인 격투게임과 같이 통신 속도에 민감한 장르의 게임들은 UDP 통신을 사용합니다. TCP 에서는 통신의 전달을 보증하기 때문에 송신한 데이터가 유실되었다고 인식하면 다시 재전송합니다. 세그먼트를 송신한 후 재전송 시간까지 상대방에게서 ACK 신호를 기다리고 신호가 오지 않으면 재전송을 하게 되는데, 이때 지연시간이 발생하여 게임이 멈추는 일이 일어날 수도 있습니다. 따라서 신뢰성이 낮더라도 통신 속도를 높이고 재전송을 없애기 위해 UDP 를 사용합니다.

3) DNS: IP 주소로 도메인 이름을 변환하는 DNS 통신에도 UDP 가 사용됩니다.

- DNS(도메인 이름 시스템): 도메인 이름을 IP 주소로 변환하며, 해당 주소는 브라우저가 인터넷 페이지를 로드하는 데 사용합니다. 인터넷에 연결된 모든 장치에는 자체 IP 주소가 있으며, 다른 장치에서 해당 장치를 찾기 위해 이 주소를 사용합니다. DNS 서버를 사용하면 모든 웹사이트의 IP 주소를 추적할 필요 없이 .com 으로 끝나는 인터넷 주소와 같이 일반적인 단어를 브라우저에 입력할 수 있습니다.

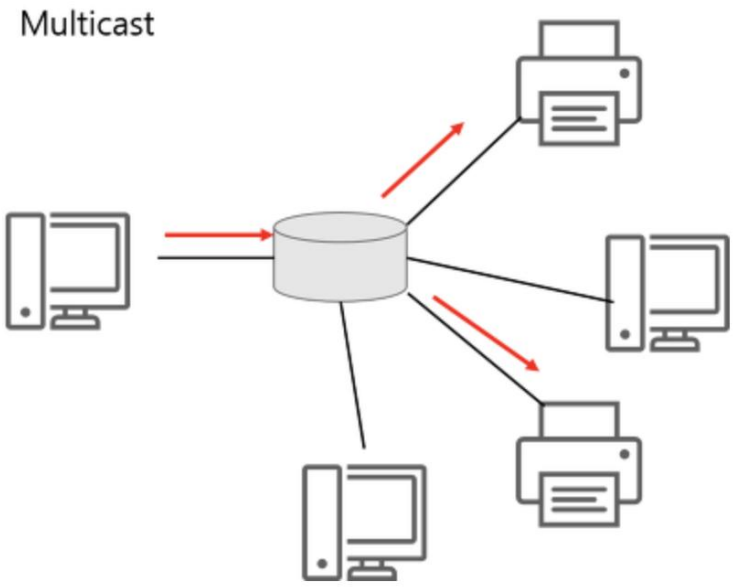
DNS 서버가 올바른 IP 주소를 찾으면 브라우저가 해당 주소를 가져와서 이를 사용해 콘텐츠 전송 네트워크(CDN)에지 서버 또는 원본 서버로 데이터를 전송합니다. 이 작업이 완료되면 사용자가 웹사이트의 정보에 엑세스할 수 있습니다. DNS 서버는 웹사이트의 URL 에 해당하는 IP 주소를 찾고, 프로세스를 시작합니다.



이와 같이 빠른 조회 속도가 중요한 에플리케이션을 위해 UDP 가 사용될 수 있습니다.

4) 멀티캐스트: 다수의 수신자에게 동시에 데이터를 전송할 때 유용합니다.

- 멀티캐스트는 특정 그룹에 속한 여러 장치에게 동시에 데이터를 전송하는 방식을 의미합니다. 이는 개별적으로 데이터를 보내는 유니캐스트(Unicast)와 달리, 같은 데이터를 한 번만 전송하여 여러 수신자가 받을 수 있도록 하는 방식입니다.



한 서버에서 다수 기기로 데이터를 전송하는 멀티캐스트에서도 UDP 가 사용됩니다.

5. qtcreator 에서의 UDP 통신

qtcreator 에서 UDP 통신을 하려면 Qudpsocket 클래스를 사용해야 합니다. Qudpsocket 클래스를 사용하기 위해서는 CMakeLists.txt 파일, 헤더파일에 설정을 추가해야 합니다.

Header:	`#include <QUdpSocket>`
CMake:	`find_package(Qt6 REQUIRED COMPONENTS Network)` `target_link_libraries(mytarget PRIVATE Qt6::Network)`
qmake:	`QT += network`
Inherits:	QAbstractSocket

Qtcreator 공식 사이트에 나온 QUdpSocket Class 사용 설명서의 일부분입니다. CMake 파일에 위와 같은 명령어를 추가하면 된다고 나와있었지만 그대로 추가하면 오류가 발생해 아래 두 코드로 추가했습니다.

```
find_package(Qt6 QUIET COMPONENTS Core Gui Widgets Network)
if(Qt6_FOUND)
    set(QT_VERSION_MAJOR 6)
    message(STATUS "Found Qt6: ${Qt6_VERSION}")
else()
    find_package(Qt5 REQUIRED COMPONENTS Core Gui Widgets Network)
    set(QT_VERSION_MAJOR 5)
    message(STATUS "Found Qt5: ${Qt5_VERSION}")
endif()
```

```
# Link the appropriate Qt Widgets library
if(QT_VERSION_MAJOR EQUAL 6)
    target_link_libraries(${PROJECT_NAME} PRIVATE Qt6::Core Qt6::Gui Qt6::Widgets Qt6::Network)
else()
    target_link_libraries(${PROJECT_NAME} PRIVATE Qt5::Core Qt5::Gui Qt5::Widgets Qt5::Network)
endif()
```

이후 mainwindow.h 파일에서 #include <QUdpSocket> 코드를 추가하고, 멤버 함수를 이용합니다.

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    void addBubble(const QString &message, bool isSender);
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

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

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

QMainWindow 클래스의 private 변수에 QUdpSocket 객체를 생성합니다.

```
8   MainWindow::MainWindow(QWidget *parent)
9       : QMainWindow(parent), ui(new Ui::MainWindow)
10  {
11      ui->setupUi(this);
12
13      udpSocket = new QUdpSocket(this);
14      udpSocket->bind(QHostAddress::Any, 9999);
```

다음으로 mainwindow.cpp 파일에서 생성자를 정의할 때 선언한 소켓 변수에 동적 할당합니다. 통신을 주고받을 포트를 설정하기 위해 bind() 멤버 함수를 사용합니다.

bind 함수는 QHostAddres 의 주소와 quint16형 변수인 port 값을 받아 특정 IP 주소와 포트 번호에 소켓을 연결합니다. QHostAddress::Any 로 설정하면 모든 네트워크 인터페이스의 IP 주소를 사용한다는 의미입니다. 뒤의 송수신 포트는 9999로 설정했습니다.

```
15      connect(udpSocket, &QUdpSocket::readyRead, this, &MainWindow::get_udp);
16
```

다음으로 커넥트 함수를 통해 시그널로 수신이 들어오면 그에 대한 슬롯으로 들어온 수신을 읽는 함수를 연결합니다. &QUdpSocket::readuRead 는 QUdpSocket 객체에서 발생되는 시그널이고, 슬롯으로 get_udp 함수가 연결됩니다. 다음은 get_udp 함수입니다.

```
30  void MainWindow::get_udp()
31  {
32      while (udpSocket->hasPendingDatagrams())
33      {
34          QByteArray datagram;
35          datagram.resize(int(udpSocket->pendingDatagramSize()));
36          QHostAddress sender;
37          quint16 senderPort;
38
39          udpSocket->readDatagram(datagram.data(), datagram.size(), &sender, &senderPort);
40
41          QString message = QString::fromUtf8(datagram);
42          addBubble(message, false);
43      }
44  }
```

여기서는 bool 형인 hasPendingDatagrams()함수가 udpSocket 객체에서 참인 동안(UDP 소켓에 읽지 않은 데이터그램이 남아있는 동안) 받아온 수신을 읽습니다. QUdpSocket 에서 송수신되는 데이터그램은 QByteArray 클래스 형태인데, 바이트 단위의 배열을 지원하기 위해 존재합니다. 따라서 값을 읽기 위해 QByteArray 객체 datagram 을 생성해 pendingDatagramSize 함수로 다음에 들어올 데이터그램의 크기만큼 resize 함수를 통해 재할당해줍니다.

readDatagram 에 송신자 IP 주소, 포트 번호 변수를 전달하고 UDP 데이터를 읽어옵니다. 여기서 매개변수 datagram.data()는 QByteArray 내부 실제 메모리 주소를, datagram.size()는 최대 읽을 크기, &sender 은 보낸 사람의 주소를 저장합니다. &senderPort 는 보낸 사람의 포트를 저장합니다.

```
46  void MainWindow::on_pushButton_clicked()
47  {
48      QString message = ui->textEdit->toPlainText();
49      if (message.isEmpty())
50          return;
51
52      udpSocket->writeDatagram(message.toUtf8(), QHostAddress("192.168.0.45"), 9999);
53      addBubble(message, true);
54      ui->textEdit->clear();
55  }
56
```

UDP 통신에서 송신을 하는 함수입니다. 보내고자 하는 값을 QString 객체와 toPlainText 함수로 읽어옵니다. 만약 보내고자 하는 값이 없다면 함수를 종료하고, 아니라면 writeDatagram()함수를 통해 보내고 싶은 값을 보낼 상대방의 IP 주소로 지정한 포트를 통해 송신합니다.

6. qtcreator 에서 카카오톡 구현

구현한 카카오톡 기능은 다음과 같습니다.

- 메시지를 입력하고 send 버튼을 누르면 UDP 로 상대방에게 송신
- 상대방이 송신한 메시지를 UDP 로 수신
- 수신/송신되는 메시지 로그를 화면에 말풍선 모양으로 표시

다음은 작성한 헤더파일입니다.

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QUdpSocket>

QT_BEGIN_NAMESPACE
namespace Ui
{
    class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    void addBubble(const QString &message, bool isSender);
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

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

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

클래스는 MainWindow 클래스를 그대로 사용해 QUdpSocket 객체를 선언하고 몇 가지 슬롯과 함수를 추가했습니다.

다음은 작성한 소스코드입니다.

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include <QUdpSocket>
#include <QHBoxLayout>
#include <QLabel>
#include <QScrollBar>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    udpSocket = new QUdpSocket(this);
    udpSocket->bind(QHostAddress::Any, 9999);
    connect(udpSocket, &QUdpSocket::readyRead, this, &MainWindow::get_udp);

    QWidget *container = new QWidget(this);
    container->setMinimumSize(1, 1);
    container->setFixedWidth(ui->scrollArea->width());
    ui->scrollArea->setWidget(container);
    ui->scrollArea->setWidgetResizable(false);
    ui->scrollArea->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

생성자와 소멸자 부분 코드입니다. 소멸자는 추가한 것이 없고 생성자에서만 UDP 설정과 카톡 UI 설정을 추가했습니다.

QWidget 객체를 선언하고 setMinimumSize, setFixedWidth 함수로 채팅창 크기를 설정했으며, scrollArea 에 container 를 붙여 스크롤 영역 안에 말풍선을 담을 공간을 만들었습니다.

```
30  void MainWindow::get_udp()
31  {
32      while (udpSocket->hasPendingDatagrams())
33      {
34          QByteArray datagram;
35          datagram.resize(int(udpSocket->pendingDatagramSize()));
36          QHostAddress sender;
37          quint16 senderPort;
38
39          udpSocket->readDatagram(datagram.data(), datagram.size(), &sender, &senderPort);
40
41          QString message = QString::fromUtf8(datagram);
42          addBubble(message, false);
43      }
44  }
45
46  void MainWindow::on_pushButton_clicked()
47  {
48      QString message = ui->textEdit->toPlainText();
49      if (message.isEmpty())
50          return;
51
52      udpSocket->writeDatagram(message.toUtf8(), QHostAddress("192.168.0.45"), 9999);
53      addBubble(message, true);
54      ui->textEdit->clear();
55  }
```

다음으로 UDP 통신/채팅 로그 표시 기능을 가진 송/수신 함수들입니다. 수신 함수인 get_udp 는 readDatagram 으로 메시지를 수신받아 QString 객체에 저장합니다. 다음으로 addBubble 함수로 채팅창에 메시지를 표시합니다.

송신 함수는 send pushButton 을 클릭했을 경우 발생하는 시그널에 따른 슬롯 함수로 지정했습니다. 버튼이 눌리면 채팅 화면의 문자열을 읽고, 문자가 존재할 때 상대방 PC 에 UDP 로 메시지를 송신합니다. 다음으로 다시 addBubble 함수를 사용해 메시지를 채팅창에 표시하고 사용자 입력 채팅 박스의 남은 문자를 지웁니다.

```
57  void MainWindow::addBubble(const QString &message, bool isSender)
58  {
59      QLabel *label = new QLabel(message);
60      label->setWordWrap(true);
61      label->setMaximumWidth(ui->scrollArea->width() - 60);
62      label->adjustSize();
63      label->setStyleSheet(isSender ? "background-color: #FFFFFF; border-radius:
64      label->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Preferred);
65
66      QHBoxLayout *layout = new QHBoxLayout;
67      layout->setContentsMargins(10, 5, 10, 5);
68
69      if (isSender)
70      {
71          layout->addStretch();
72          layout->addWidget(label);
73      }
74      else
75      {
76          layout->addWidget(label);
77          layout->addStretch();
78      }
```

채팅 화면에 메시지를 표시하는 addBubble 함수입니다. 매개변수로는 화면에 표시할 메시지와 bool 형 변수인 플래그를 받습니다. 플래그는 상대가 보낸 메시지인지 내가 보낸 메시지인지를 판별하기 위함입니다. 말풍선을 표현하기 위해 QLbel 을 사용했고, setWordWrap(true)로 자동 줄바꿈을 설정했습니다. setMaximumWidth()함수로는 최대 폭을 제한하고, adjustSize()함수로 텍스트 길이에 맞게 라벨 크기를 자동 조절하게 했습니다.
setStyleSheet()함수에서는 QSS 설정을 건드려서 말풍선의 색과 테두리 둥글기 등을 설정했습니다.

다음으로 QHBoxLayout 객체입니다. layout 에서는 가로 레이아웃으로 말풍선을 좌/우로 배치 가능하게 했고, 여백으로 좌우 10px, 위아래 5px 를 설정했습니다.
if 문에서는 받는 사람인지, 보낸 사람인지에 따라 label 의 정렬 방향을 조정했습니다. 내가 보낸 메시지이면 오른쪽 정렬을, 상대가 보낸 메시지이면 왼쪽 정렬을 사용했습니다.

```
80          QWidget *bubbleWidget = new QWidget;
81          bubbleWidget->setLayout(layout);
82
83          QWidget *container = ui->scrollArea->widget();
84          bubbleWidget->setParent(container);
85          bubbleWidget->show();
86
87          int yOffset = 0;
88          for (QObject *child : container->children())
89          {
90              QWidget *w = qobject_cast<QWidget *>(child);
91              if (w && w != bubbleWidget)
92              {
93                  yOffset += w->height() + 10;
94              }
95          }
96
```

위 사진에서 QWidget 으로 라벨과 레이아웃을 포함하는 독립된 위젯을 만들었는데, 이 위젯이 scrollArea 에 들어가는 말풍선 하나에 해당합니다. container 위젯을 가져와 bubbleWidget 을 붙임으로서 내용이 표시되게끔 합니다.

yOffset 변수로는 말풍선들의 좌표를 계산합니다. container 안에 있는 말풍선들의 높이를 모두 더한 뒤 새로운 말풍선의 y 좌표를 계산합니다. 10이 말풍선들의 간격입니다.

```
96
97          int containerWidth = container->width();
98          int bubbleWidth = bubbleWidget->sizeHint().width();
99          int xOffset = isSender ? containerWidth - bubbleWidth - 10 : 10;
100
101         bubbleWidget->move(xOffset, yOffset);
102         container->resize(container->width(), yOffset + bubbleWidget->height() + 20);
103         ui->scrollArea->verticalScrollBar()->setValue(ui->scrollArea->verticalScrollBar()->maximum());
104     }
```

addBubble 함수의 마지막 부분입니다. 이제 xOffset 변수로 가로 위치를 결정하고, 보낸 메시지는 오른쪽 끝으로, 받은 메시지는 왼쪽 끝으로 이동시킵니다. 세로 위치는 기존 말풍선의 아래쪽으로 둡니다.

Container 높이를 새 말풍선까지 포함하도록 resize 로 확장합니다. scrollArea 설정을 통해 스크롤바는 맨 아래로 자동 스크롤되어 최신 메시지가 보이도록 합니다.