Socket Programming Project #2

TCP/UDP 소켓 프로그래밍을 활용한 채팅 Application 개발
2단계 : 파일 전송 기능 구현

2019134006 김준섭
2019161011 박수연
tekcos

학정번호-분반 : CSI4106.02-00

목차

Ⅰ - 개요

Python을 이용, TCP, UDP 소켓프로그래밍을 활용한 채팅 application 개발을 목표로 프로젝트를 수행했다. 프로젝트 목표 2단계에 해당하는 파일 전송 기능을 구현하였다.

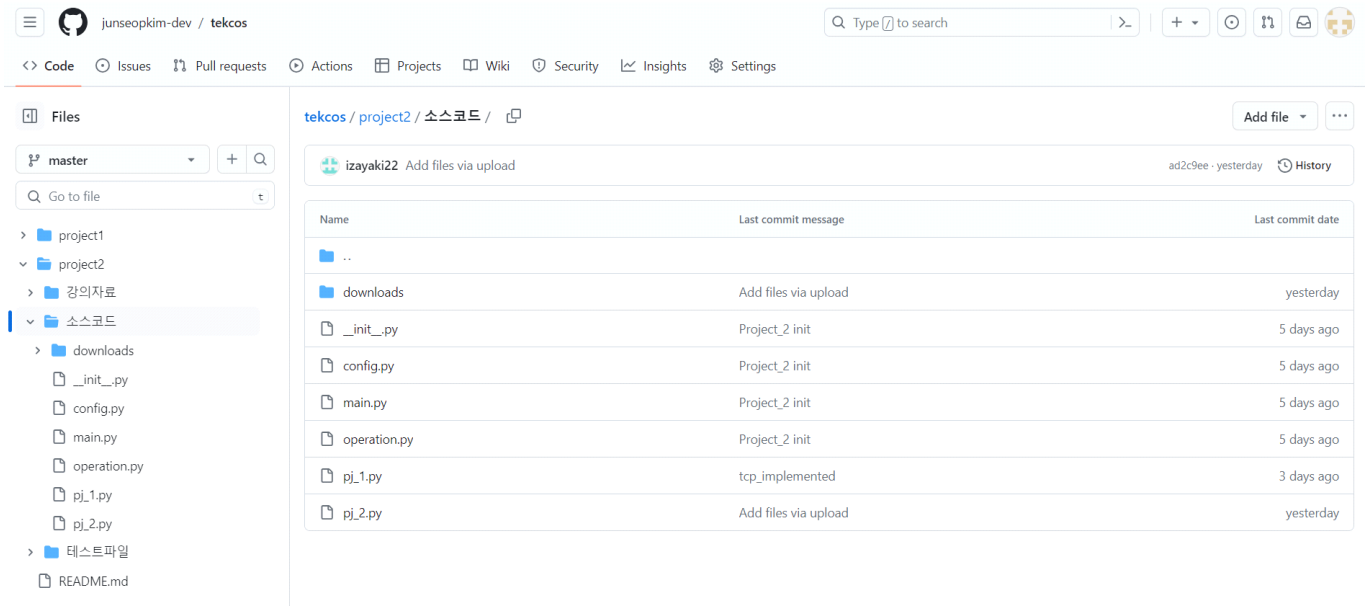
Ⅱ 프로젝트 진행 상황

1. 현황 요약

단계	핵심 구현 기능 목표	완료 여부 (마감 일자)
1	문자열 전송 기능	완료 (2023.10.30)
2	파일 전송 기능	완료 (2023.11.21)
3	주변 device 인식 가능	-

2. 역할 분담
   a. 김준섭 : Git 관리, 코드 테스트, Reliable TCP 파일 전송 구현, 보고서 작성
   b. 박수연 : Git 관리, 코드 테스트, Reliable UDP 파일 전송 구현, 보고서 검토

3. 협업을 위한 Github repository 관리
   a. Github 주소 : https://github.com/junseopkim-dev/tekcos



Ⅲ 사전지식

1. 네트워크 계층구조

유지관리 및 시스템 업데이트를 쉽게 하고자 네트워크 시스템은 계층구조를 이룬다.

크게 2가지의 모델이 존재한다 : Internet Protocol Stack, ISO/OSI Reference Model

a. Internet Protocol Stack : 5개의 계층으로 구성

(1) Application : Supporting network applications.
ex) FTP, SMTP, HTTP

(2) Transport : Process-Process data transfer.
ex) **TCP, UDP**

(3) Network : Routing of datagrams from source to destination.
ex) IP, Routing Protocols

(4) Link : Data transfer between neighboring entwork elements.
ex) Ethernet, 802.111(Wifi), PPP

(5) Physical : Move the individual bits within the frame from one node to the next.

b. ISO/OSI Reference Model : 7개의 계층으로 구성

(1) Application : Supporting network applications.
ex) FTP, SMTP, HTTP

(2) Presentation : Allow applications to interpret meaning of data

(3) Session : Synchronization, checkpointing, recovery of data exchange

(4) Transport : Process-Process data transfer.
ex) **TCP, UDP**

(5) Network : Routing of datagrams from source to destination.
ex) IP, Routing Protocols

(6) Link : Data transfer between neighboring entwork elements.
ex) Ethernet, 802.111(Wifi), PPP

(7) Physical : Move the individual bits within the frame from one node to the next.

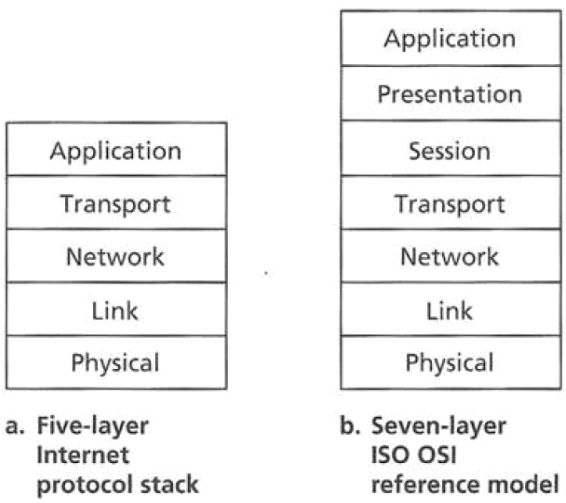


Figure 1.23 ♦ The Internet protocol stack (a) and OSI reference model (b)

2. Transport Layer Protocol : TCP & UDP

a. TCP (Transmission Control Protocol)

(1) 특징 :

(가) reliable (rdt 1.0, 2.0, 2.1, 2.2, 3.0 등)

(나) in-order delivery

(다) 연결 지향적 프로토콜 (3-way handshake)

(2) 장단점

(가) 장점 : rdt 3.0을 통해 높은 신뢰성을 보장한다.

(나) 단점 : UDP보다 속도가 느리다.

b. UDP (User Datagram Protocol)

(1) 특징 :

(가) unreliable

(나) unordered delivery

(다) 비연결형 프로토콜

(2) 장단점

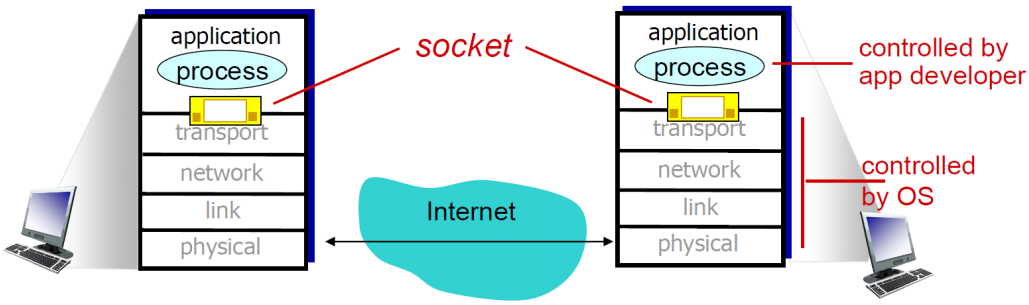
(가) 장점 : 빠른 속도로 전송 가능하다.

(나) 단점 : 신뢰성을 보장하지 못한다.

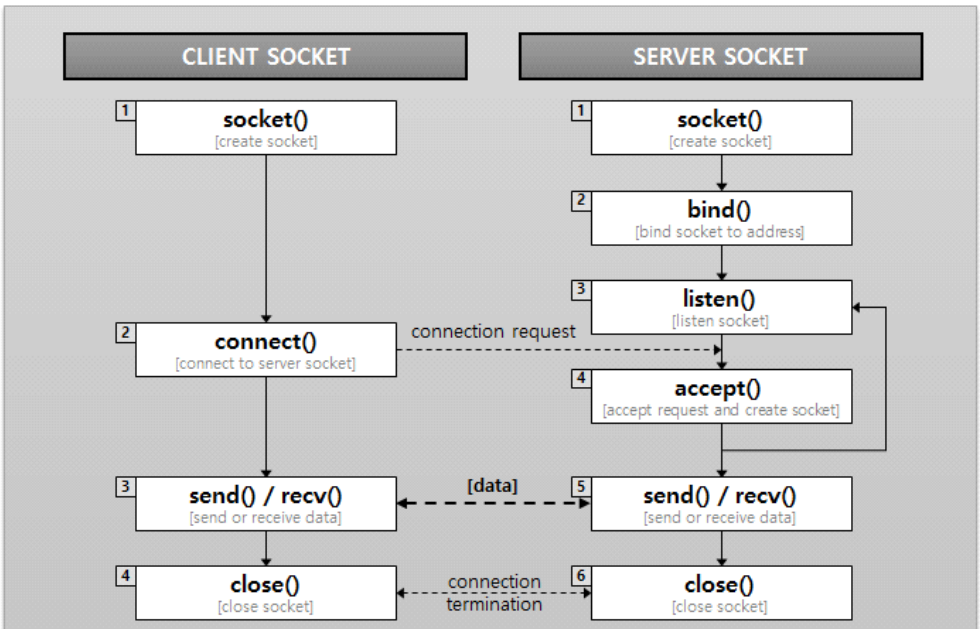
3. Socket

a. A process sends messages into, and receives messages from, the network through a software interface called a **socket**.

b. 프로세스는 집, 소켓은 문에 비유할 수 있다.

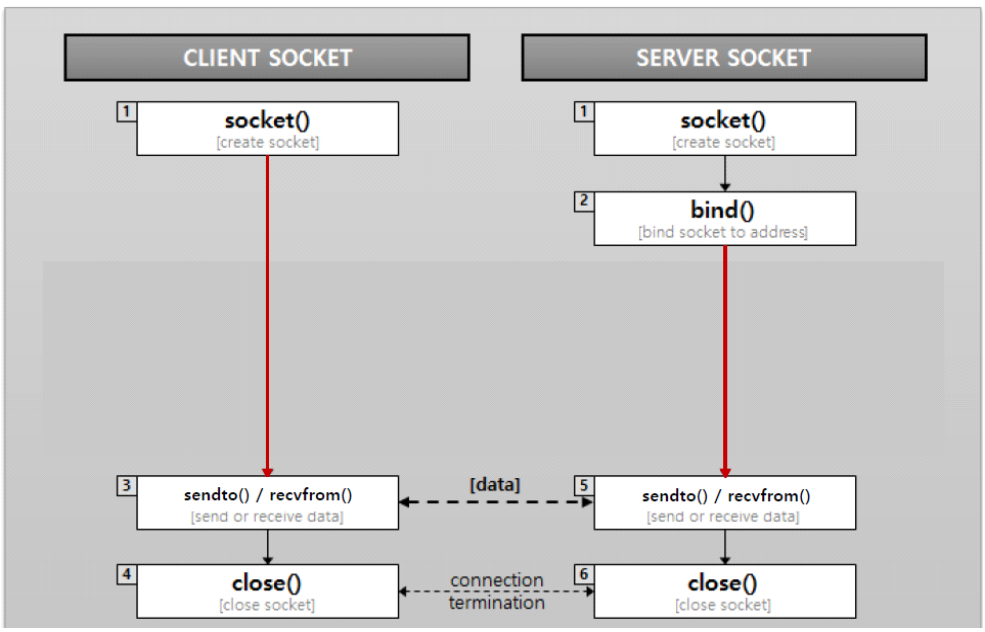


c. Socket Programming with TCP



(1) TCP는 연결 지향성 프로토콜이므로, 데이터 패킷을 주고받기 전에 서로를 연결하는 절차가 필요하다. (3-way handshake)
(2) 서로 데이터를 주고받기 전 서버는 listen()을 통해 클라이언트로부터의 연결요청이 오기를 대기한다.
(3) 서버가 listening하는 사이, 클라이언트는 connect()를 통해 서버에게 연결요청을 보낸다.
(4) 클라이언트로부터 연결요청이 온 경우, 서버는 accept()를 통해 연결을 허가하는 절차를 거친다.
(5) 서버-클라이언트 간 연결이 수립된 이후, send()/recv()를 통해 서로 데이터를 주고받을 수 있게 된다.

d. Socket Programming with UDP



(1) UDP는 TCP와는 달리 비연결형 프로토콜로, 이 때문에 TCP와는 달리 클라이언트의 connect(). 서버의 listen(), accept()가 존재하지 않는다.
(2) 서버-클라이언트 간 sendto(), recvfrom()을 통해 data를 주고받는다.

4. Reliable Data Transfer

줄여서 RDT라고 부르며, 데이터를 전송하는 과정에서 손실, 중복, 오류 등이 발생할 수 있는 불안정한 네트워크 환경에서도 안정적으로 신뢰성있는 데이터 전송을 가능케 하는 기술

a. rdt1.0

(1) 특징 : 통신이 완전히 신뢰할 수 있는 경우, Bit error, loss of packet 없음

(2) FSM

a. rdt1.0: sending side

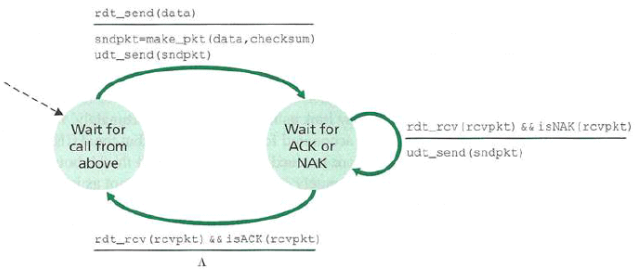
b. rdt1.0: receiving side

Figure 3.9 ♦ rdt1.0 – A protocol for a completely reliable channel

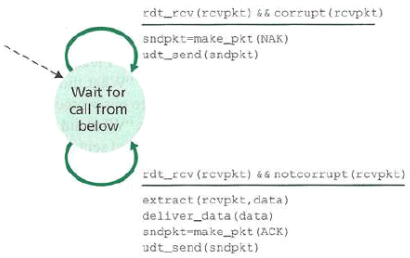
b. rdt2.0

(1) 특징 : 패킷이 전송되는 과정에서 bit error가 발생할 수 있는 경우 보완

(2) FSM



a. rdt2.0: sending side



b. rdt2.0: receiving side

Figure 3.10 ♦ rdt2.0 – A protocol for a channel with bit errors

c. rdt2.1

(1) 특징 : rdt 2.0에서 ACK/NAK에 손상이 발생할 경우 보완

(2) FSM

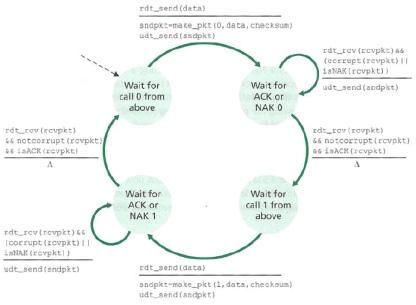


Figure 3.11 ♦ rdt2.1 sender

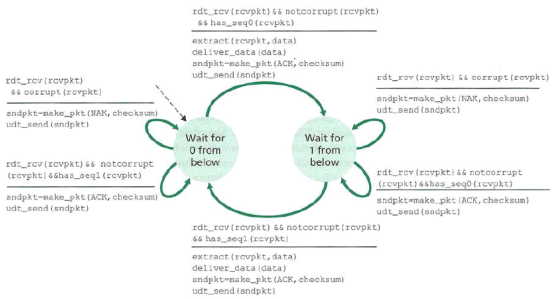


Figure 3.12 ♦ rdt2.1 receiver

d. rdt2.2

(1) 특징 : rdt 2.1과 달리 ACK만을 사용

(2) FSM

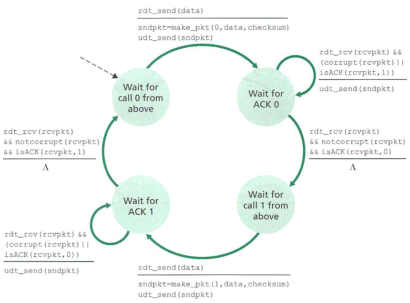


Figure 3.13 ♦ rdt2.2 sender

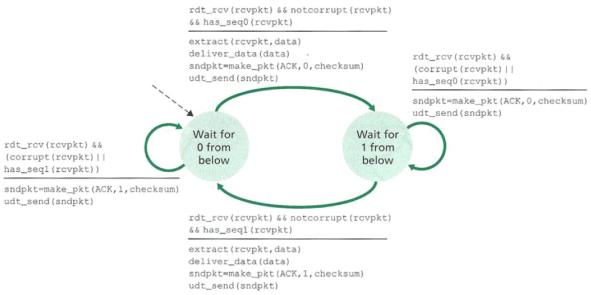


Figure 3.14 ♦ rdt2.2 receiver

e. rdt3.0

(1) 특징 : 패킷 에러 뿐만 아니라 소실(loss)이 발생할 경우 보완

(2) FSM

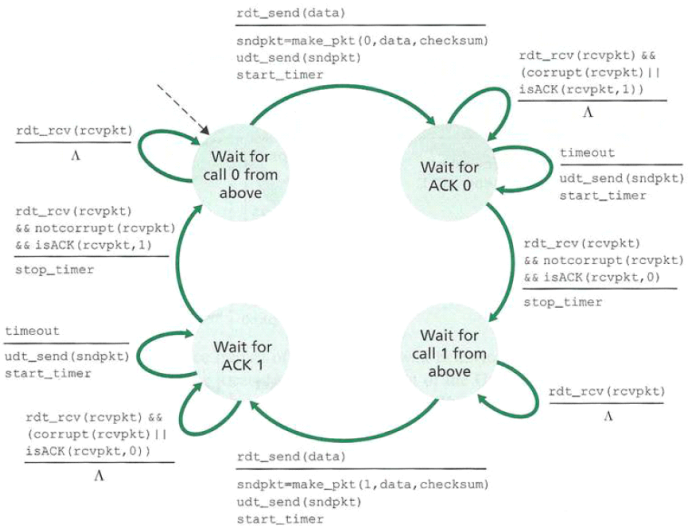


Figure 3.15 ♦ rdt3.0 sender

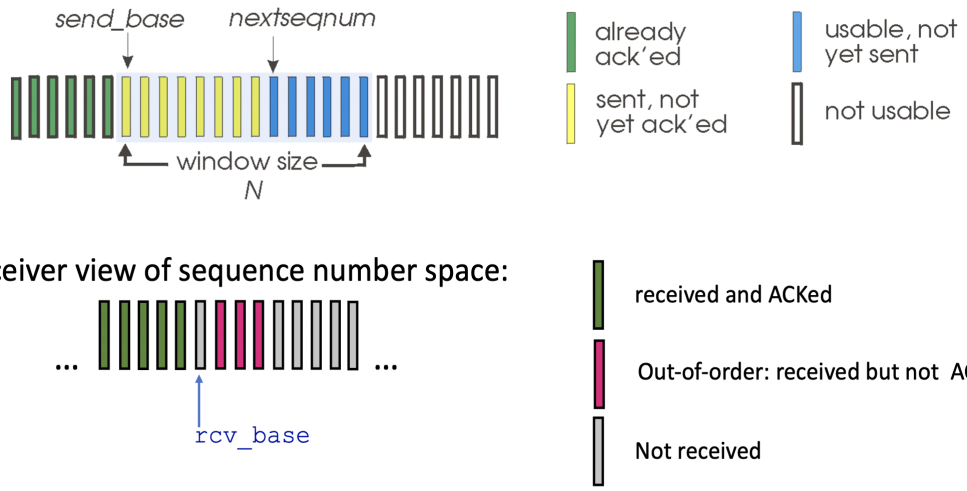
5. Pipelining

패킷을 보낸 후 응답이 올 때까지 기다리는 stop and wait 방식 특징상 느린 속도의 단점을 극복하고자 패킷을 보낸 뒤 기다리지 않고 바로 다음 패킷을 보내는 방식. 크게 2가지 방법이 존재한다.

* Window size는 (사용하는 sequence number /2) 이하여야 한다.

a. Go-Back N (GBN)

(1) 수신자가 어떤 패킷을 받지 못하면 그 패킷부터 모든 패킷을 다시 보내는 방법이다.



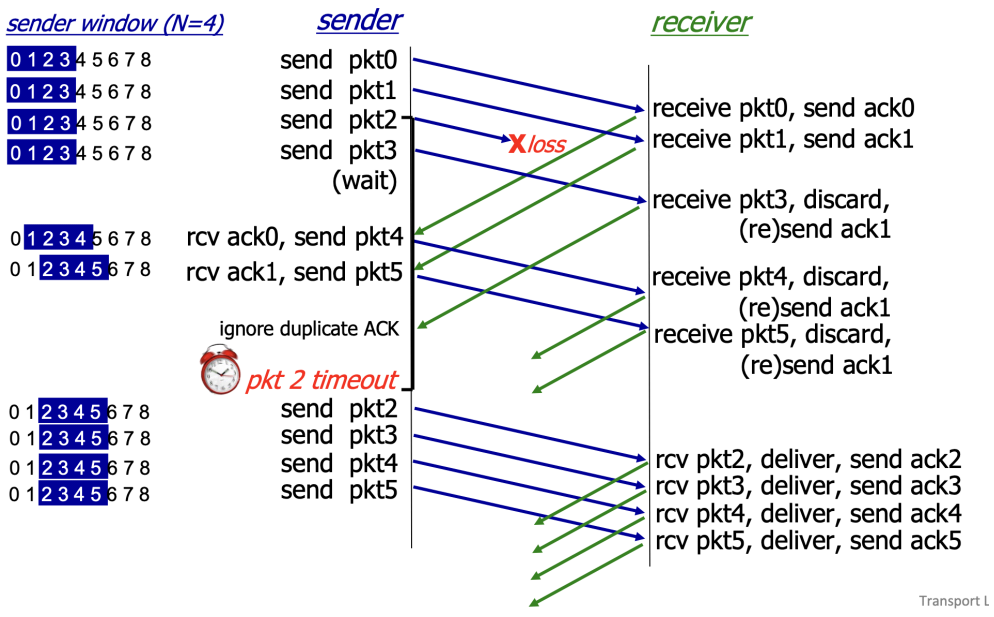
(2) 송신자는 ACK를 받지 않은 상태로 동시에 최대로 보낼 수 있는 패킷의 수를 정하는데, 이를 window size라고 하며, 현재 window 위치를 설정해 window size만큼의 범위를 설정한다.

(3) 송신자는 보낼 패킷들이 들어있는 버퍼의 시작부터 window의 범위를 설정하고, sequence number가 그 window의 범위 안에 들어 있는 패킷을 모조리 보낸다.

(4) 수신자는 패킷들을 연속해서 받고, 각각의 패킷을 받을 때마다 sequence number를 확인하고, ACK를 보낸다.

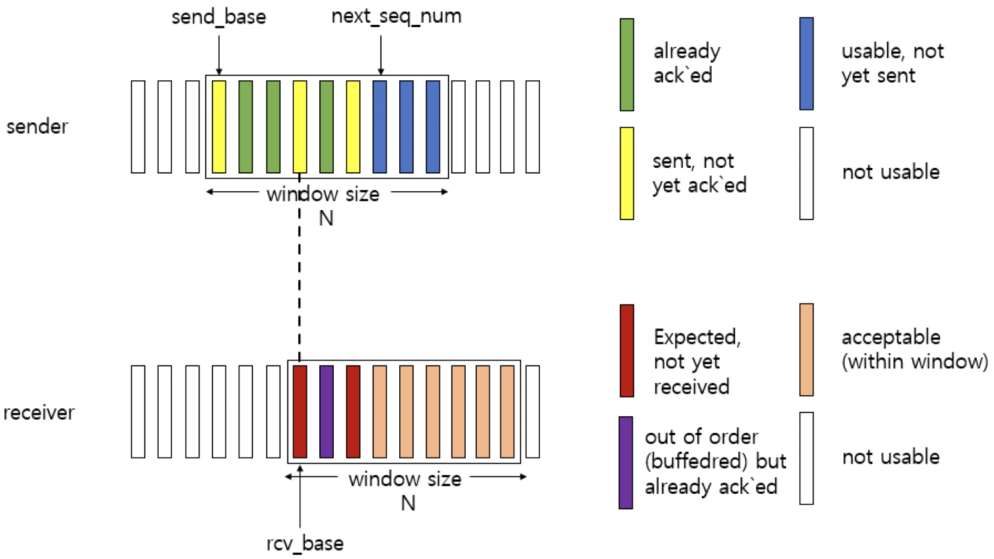
(5) 이 때, 송신자가 보낸 패킷들 중에서 일부가 유실될 수 있다. 그러면 수신자가 받은 패킷들의 sequence number에 빈틈이 생기게 된다. 이 때, 빈틈이 생기기 전에 정상적으로(연속적으로) 받은 패킷의 마지막 번호를 지속적으로 ACK에 실어서 보낸다.

(6) 송신자는 내가 보낸 패킷의 ACK가 정상적으로 왔다면, 그 패킷을 처리됐다고 기록한 후, window를 오른쪽으로 한 칸 밀어서 하나의 패킷을 더 보낼 수 있는 상태로 만든다.



b. Selective Repeat (SR)

(1) 수신자가 받은 각각의 패킷에 대해 ACK를 보내는 방식이다.



(2) 송신자는 GBN 방식과 마찬가지로 window를 가지고 있으며, window 범위 안에 들어오는 패킷들을 다 보내고 나면, 각각의 패킷에 대해 ACK을 기다린다.

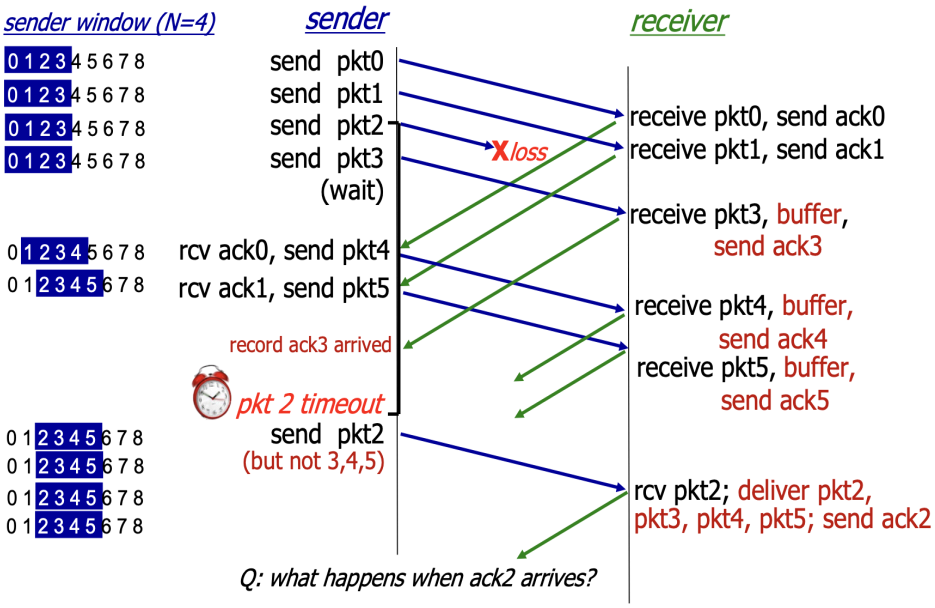
(3) 각각의 패킷에 대해 타이머를 가지고 있고, 일정 시간이 지나도 ACK 가 도착하지 않으면 timeout을 발생시켜 해당 패킷만을 다시 보낸다.

(4) 특정 패킷에 대해 ACK가 들어오면 그 패킷을 완료되었다고 기록한다. 송신자가 보낸 패킷들 중 아직 ACK되지 않은 패킷들 중 가장 번호가 작은 패킷이 ACK되면 window를 한 칸 이동시킨다.

(5) 수신자는 window와 버퍼를 가지고 있다. 어떤 패킷을 받았을 때 이 패킷이 순서(rcv_base)에 어긋난다면 일단 버퍼에 저장해 둔다.

(6) 순서에 맞는 패킷이 들어오면 버퍼에 있는 패킷들까지 모조리 ACK을 한꺼번에 보낸다. 그리고 ACK를 보낸 패킷 수만큼 window를 이동한다.

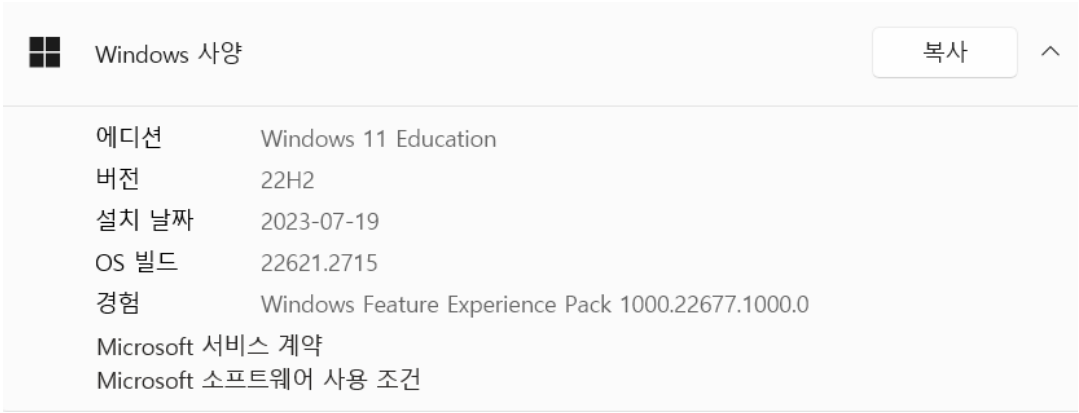
(7) window의 범위보다 이전에 속하는 패킷이 도달한다면, ACK에 오류가 생겨 재전송된 패킷이라 간주하고 ACK만 보내고 별다른 처리는 하지 않는다.



Ⅳ 구현

1. 구현 목표 : 파이썬을 이용한 소켓 프로그래밍을 이용해 TCP, UDP의 신뢰할 수 있는 파일 전송 기능 구현

2. 구현 환경
   a. OS 정보
      (1) 에디션 : Windows 11 Education
      (2) 버전 : 22H2
      (3) 설치 날짜 : 2023-07-19
      (4) OS 빌드 : 22621.2715
      (5) 경험 : Windows Feature Experience Pack 1000.22677.1000.0



   b. Python 버전 : 3.11.5

3. 구현 코드 : pj_1.py, pj_2.py에서 변경이 이루어졌다.

a. pj_1.py 변경 사항:

지난 제출본과 동일한 pj_1.py로부터, 지시에 따라 tcp와 udp의 recv, recvfrom 함수의 parameter를 config.py를 참조한 APP_HEADER_LEN+PACKET_SIZE로 변경하였다.

변경 전

```
(기존과 동일)

    @staticmethod
    def udp_server_connect(udp_server_socket:socket.socket):
        # udp_client_socket 함수가 전송한 packet으로부터
            client의 udp address(udp_client_addr) 반환

        data,addr =udp_server_socket.recvfrom(1024)
        return addr

(중략, 기존과 동일)

    def tcp_recv(self)->bytes:
        # TCP socket(tcp_socket)으로 들어오는 packet의 data 반환

        data =self.tcp_socket.recv(1024)

        return data

    def udp_recv(self)->bytes:
        # UDP socket(udp_socket)으로 들어오는 packet의 data 반환

        data ,addr =self.udp_socket.recvfrom(1024)

        return data
(이하 동일)
```

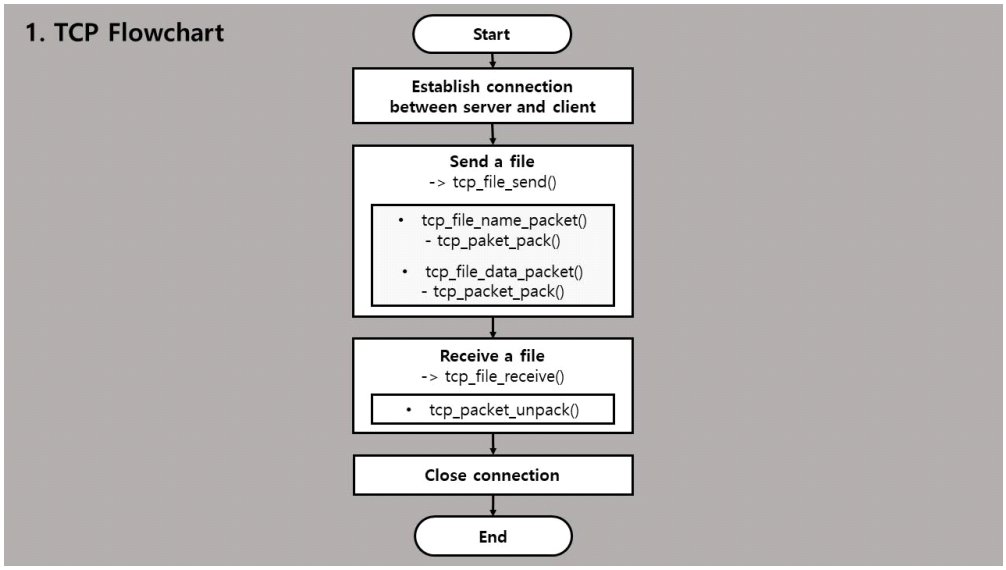
변경 후

```
(기존과 동일)

    @staticmethod
    def udp_server_connect(udp_server_socket:socket.socket):
        # udp_client_socket 함수가 전송한 packet으로부터
            client의 udp address(udp_client_addr) 반환

        data,addr =udp_server_socket.recvfrom(APP_HEADER_LEN+PACKET_SIZE)
        return addr

(중략, 기존과 동일)

    def tcp_recv(self)->bytes:
        # TCP socket(tcp_socket)으로 들어오는 packet의 data 반환

        data =self.tcp_socket.recv(APP_HEADER_LEN+PACKET_SIZE)

        return data

    def udp_recv(self)->bytes:
        # UDP socket(udp_socket)으로 들어오는 packet의 data 반환

        data ,addr =self.udp_socket.recvfrom(APP_HEADER_LEN+PACKET_SIZE)

        return data
(이하 동일)
```

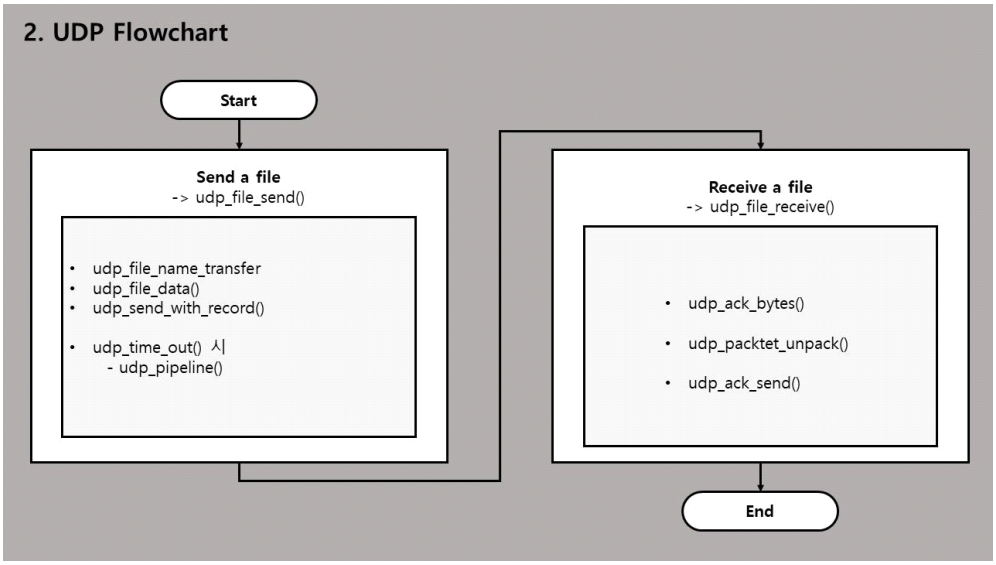
b. pj_2.py : UDP의 pipeline은 GBN으로 구현되었다.

(1) Flow chart

(가) TCP Flowchart



(나) UDP Flowchart



(2) 전체 코드

```
import os
from threading import TIMEOUT_MAX
from tkinter.messagebox import NO
# from turtle import st

from config import *
from collections.abc import Callable

import struct
from typing import Tuple,Any

from time import time
from time import sleep

from tkinter import END

UDP_WINDOW_SIZE =100
UDP_MAX_ACK_NUM =int(2**16)
UDP_TIMEOUT =5
UDP_WAIT =0.05

PACKET_TYPE_FILE_START =b'₩x00'
PACKET_TYPE_FILE_DATA =b'₩x01'
PACKET_TYPE_FILE_END =b'₩x02'
PACKET_TYPE_FILE_ACK =b'₩x03'

TCP_FILE_TRANSFER_END =PACKET_TYPE_FILE_END +bytes(PACKET_SIZE-1)# TCP에서의 파일 전송 종료를 알리기 위한 패킷


class FileTransfer:
    def __init__(self)->None:
        self.file_pointer =None
        self.udp_recv_packet =[bytes(PACKET_SIZE)for _in range(UDP_MAX_ACK_NUM)]
        self.udp_recv_flag =[False for _in range(UDP_MAX_ACK_NUM)]
        self.udp_send_packet =dict()
        self.udp_ack_windows =[False for _in range(UDP_MAX_ACK_NUM)]
        self.udp_ack_num =0
        self.udp_last_ack_num =0
        self.file_packet_start =0
        self.file_name =None

    @staticmethod
    def tcp_packet_pack(packet_type:bytes,data:bytes)->bytes:
        data_len =len(data)
        packet =packet_type +struct.pack(">H",data_len)+data
        packet =packet +bytes(PACKET_SIZE -len(packet))# packet 크기 맞추기
        return packet

    @staticmethod
    def tcp_packet_unpack(packet:bytes)->Tuple[bytes,bytes]:
        packet_type =packet[:1]
        data_len =struct.unpack(">H",packet[1:3])[0]
        data =packet[3:3+data_len]
        return packet_type,data
```

```
@staticmethod
def udp_packet_pack(packet_type:bytes,ack_num:Any,data:bytes)->bytes:
    data_len =len(data)
    if type(ack_num)==int:
        packet =packet_type +struct.pack(">HH",ack_num,data_len)+data
    elif type(ack_num)==bytes:
        packet =packet_type +ack_num +struct.pack(">H",data_len)+data
    packet =packet +bytes(PACKET_SIZE -len(packet))# packet 크기 맞추기
    return packet

@staticmethod
def udp_packet_unpack(packet:bytes)->Tuple[bytes,int,bytes]:
    packet_type =packet[:1]
    ack_num,data_len =struct.unpack(">HH",packet[1:5])
    data =packet[5:5+data_len]
    return packet_type,ack_num,data

@staticmethod
def udp_ack_bytes(packet:bytes)->bytes:
    return packet[1:3]

def tcp_file_name_packet(self,file_name:str)->bytes:
    # TCP 통신에서의 file 이름 전송용 패킷 생성
    # 패킷 구조: ₩x00 + (이름 data 크기) + (파일 이름 data)
    data =file_name.encode(ENCODING)
    return self.tcp_packet_pack(PACKET_TYPE_FILE_START,data)


def tcp_file_data_packet(self)->Tuple[bool,bytes]:
    # tcp sener가 가진 self.file_pointer에서
    # 전송을 위한 packet을 생성한다,
    # 결과값: 패킷이 존재 여부, 생성된 패킷
    # 패킷 구조: ₩x01 + (data 크기) + (file data)
    data =self.file_pointer.read(PACKET_SIZE -1 -2)
    if data:
        return True,self.tcp_packet_pack(PACKET_TYPE_FILE_DATA,data)
    else:
        return False,None

def udp_file_data(self)->Tuple[bool,bytes]:
    # udp sener가 전송할 file data를 얻는다
    # 결과값: file data
    data =self.file_pointer.read(PACKET_SIZE -1 -2 -2)
    if data:
        return True,data
    else:
        return False,None

def tcp_file_name_transfer(self,filename:str,tcp_send_func:Callable)->None:
    # TCP 통신에서 sender에게 파일 전송이 시작을 알리면서 파일 이름을 전송한다.
    packet =self.tcp_file_name_packet(filename)
    tcp_send_func(packet)
```

```
    def tcp_file_send(self,filename:str,tcp_send_func:Callable)->None:
        basename =os.path.basename(filename)
        self.file_pointer =open(filename,"rb")

        # packet의 파일 이름(basename)을 전송한다.
        self.tcp_file_name_transfer(basename,tcp_send_func)
        # 이름 전송 종료


        # 파일을 구성하는 data를 전송한다.
        # tcp_file_data_packet이 생성하는 packet을 tcp를 이용해 전부 전송한다.
        data_ready,packet =self.tcp_file_data_packet()

        while data_ready:

            tcp_send_func(packet)
            data_ready,packet =self.tcp_file_data_packet()
        # 파일 data 전송 종료


        # TCP_FILE_TRANSFER_END을 전송하여
        # 파일의 전송이 끝냈음을 알린다.
        tcp_send_func(TCP_FILE_TRANSFER_END)
        # TCP_FILE_TRANSFER_END을 전송 종료


        # 파일 닫기
        self.file_pointer.close()
        self.file_pointer =None


    def tcp_file_receive(self,packet)->int:
        packet_type,data =self.tcp_packet_unpack(packet)

        if packet_type ==PACKET_TYPE_FILE_START:
            basename =data.decode(ENCODING)
            self.file_name =basename
            file_path ='./downloads/(tcp) '+basename
            # 파일의 이름을 받아 file_path 위치에 self.file_pointer를 생성한다.
            self.file_pointer=open(file_path,"wb")

            return 0

        elif packet_type ==PACKET_TYPE_FILE_DATA:
            # self.file_pointer에 전송 받은 data를 저장한다.
            self.file_pointer.write(data)

            return 1

        elif packet_type ==PACKET_TYPE_FILE_END:
            # 파일 전송이 끝난 것을 확인하고 file_pointer를 종료한다.
            self.file_pointer.close()
            self.file_pointer =None
            return 2

    def udp_file_name_transfer(self,file_name:str,udp_send_func:Callable)->None:
        data =file_name.encode(ENCODING)
        self.udp_send_with_record(PACKET_TYPE_FILE_START,data,udp_send_func)

    def udp_send_with_record(self,packet_type:bytes,data:bytes,udp_send_func:Callable)->None:
        packet =self.udp_packet_pack(packet_type,self.udp_last_ack_num,data)
        udp_send_func(packet)
        # GBN, SR을 통한 재전송을 위해 packet과 전송 시간을 self.udp_send_packet에 저장한다.
        # 또한 self.udp_lask_ack_num을 update하여 새로 전송할 packet의 ack_num을 update한다.
        self.udp_send_packet[self.udp_last_ack_num]=(time(),packet)
        self.udp_last_ack_num =(self.udp_last_ack_num +1)%UDP_MAX_ACK_NUM
```

```
def udp_file_send(self,filename:str,udp_send_func:Callable)->None:
        basename =os.path.basename(filename)
        self.file_pointer =open(filename,"rb")

        # udp를 통해 파일의 basename을 전송하고 ack를 기다린다.
        # hint : self.udp_file_name_transfer 함수를 활용할 것
        self.udp_file_name_transfer(basename,udp_send_func)

        while self.udp_ack_num !=self.udp_last_ack_num:
            if self.udp_ack_num ==0:
                if self.udp_time_out():
                    self.udp_file_name_transfer(basename,udp_send_func)
                    break
                else:
                    sleep(UDP_WAIT)

        data_ready,data =self.udp_file_data()

        while data_ready:

            if len(self.udp_send_packet)<UDP_WINDOW_SIZE:
                  #window의 크기보다 전송한 패킷의 양의 적은 경우
                self.udp_send_with_record(PACKET_TYPE_FILE_DATA,data,udp_send_func)
                data_ready,data =self.udp_file_data()# 다음 전송할 data를 준비한다.
            else:

                # PIPELINE을 위한 window를 전체를 사용하여 ack를 기다리며 timeout에 대처한다.
                # Timeout이 아닌 경우에는 Sleep(UDP_WAIT)를 사용한다.
                while not self.udp_ack_windows[self.udp_ack_num]:

                    if self.udp_time_out():
                        self.udp_pipeline(udp_send_func)
                        break
                    else:
                        sleep(UDP_WAIT)

                pass

        # 모든 파일 data의 ack를 기다리고 timeout에 대처한다.
        while not self.udp_ack_windows[self.udp_ack_num]:

            if self.udp_ack_num ==self.udp_last_ack_num:
                break

            if self.udp_time_out():

                self.udp_pipeline(udp_send_func)
            else:
                sleep(UDP_WAIT)

        # 파일 전송이 완료되었음을 알리고 ack에 대비한다.
        self.udp_send_with_record(PACKET_TYPE_FILE_END,b'',udp_send_func)

        while self.udp_ack_num !=self.udp_last_ack_num:

            if self.udp_ack_num !=self.udp_last_ack_num:
                if self.udp_time_out():
                    self.udp_send_with_record(PACKET_TYPE_FILE_END,b'',udp_send_func)
                    break
                else:
                    sleep(UDP_WAIT)

        # 파일 포인터를 제거한다.
        self.file_pointer.close()
        self.file_pointer =None
```

```
    def udp_file_receive(self,packet:bytes,udp_send_func:Callable)->int:
        ack_bytes =self.udp_ack_bytes(packet)
        packet_type,ack_num,data =self.udp_packet_unpack(packet)

        if packet_type !=PACKET_TYPE_FILE_ACK:
            # 받은 packet에 대한 ack를 전송한다.
            self.udp_ack_send(ack_bytes,udp_send_func)
            pass

        if packet_type ==PACKET_TYPE_FILE_START:  # file transfer start
            if self.file_pointer is not None:
                self.file_pointer.close()
            basename =data.decode(ENCODING)
            self.file_name =basename
            file_path ='./downloads/(udp) '+basename
            # 파일의 이름을 받아 file_path 위치에 self.file_pointer를 생성하고.
            # 그다음 받을 파일의 data의 시작 packet의 ack_num를 self.file_packet_start에 저장하여
            # 연속된 packet을 받을 수 있게 준비한다.
            self.file_pointer =open(file_path,'wb')
            self.file_packet_start =ack_num +1
            return 0

        elif packet_type ==PACKET_TYPE_FILE_DATA:  # file transfer
            if not self.udp_recv_flag[ack_num]:
                # 처음 받은 packet인지 확인하고
                # 처음 받은 packet이라면 self.udp_recv_packet[ack_num]에 저장하고
                # self.udp_recv_flag[ack_num]에서 확인할 수 있게 표시한다.
                self.udp_recv_packet[ack_num]=data
                self.udp_recv_flag[ack_num]=True

            # self.udp_recv_packet에 self.file_packet_start에서 부터 연속된
            # 패킷이 저장되어 있다면 이를 self.file_pointer를 이용해 파일로 저장하고
            # self.udp_recv_flag를 update한다.
            # 또한 self.file_packet_start 역시 update한다.

            while self.udp_recv_flag[self.file_packet_start]:
                self.file_pointer.write(self.udp_recv_packet[self.file_packet_start])
                self.udp_recv_flag[self.file_packet_start]=False
                self.file_packet_start =(self.file_packet_start +1)%UDP_MAX_ACK_NUM
            return 1

        elif packet_type ==PACKET_TYPE_FILE_END:  # file transfer end
            # 파일 전송이 끝난 것을 확인하고 파일을 종료한다.
            if self.file_pointer is not None:
                self.file_pointer.close()
                self.file_pointer =None
            return 2

        elif packet_type ==PACKET_TYPE_FILE_ACK:  # ack
            # GBN, SR을 위해 self.udp_ack_windows를 update한다.
            # hint: self.udp_ack_num으로 부터 연속되게 ack를 받은 경우
            # window를 옮겨준다 (self.udp_send_packet에 저장된 packet도 처리해줄 것)

            if self.udp_ack_num not in self.udp_send_packet.keys():
                return 1

            if self.udp_ack_num ==ack_num:
                self.udp_ack_windows[self.udp_ack_num]=True
                del self.udp_send_packet[self.udp_ack_num]
                self.udp_ack_num =(self.udp_ack_num +1)%UDP_MAX_ACK_NUM
            return 1
        return 1
```

```
def udp_time_out(self)->bool:
    if time()-self.udp_send_packet[self.udp_ack_num][0]>UDP_TIMEOUT:# timeout
        return True
    else:
        return False

def udp_pipeline(self,udp_send_func:Callable)->None:
    # GBN, SR 중 하나의 알고리즘을 선택하여 ACK를 관리한다.
    # def udp_gbn () or def udp_sr()로 구현
     # hint: self.udp_send_packet[ack_num]에 저장시
    # (send time, packet)형태로 저장할 것
    # udp_file_send()에서 사용


    for ack_num in range(self.udp_ack_num,self.udp_last_ack_num):

        send_time,packet =self.udp_send_packet[ack_num]
        udp_send_func(packet)
        self.udp_send_packet[ack_num]=(time(),packet)



def udp_ack_send(self,ack_bytes:bytes,udp_send_func:Callable):
    packet =PACKET_TYPE_FILE_ACK +ack_bytes
    packet =self.udp_packet_pack(PACKET_TYPE_FILE_ACK,ack_bytes,b'')
    udp_send_func(packet)
```

c. FileTransfer.__init__() 내 변수 정의

Subject	변수	설명
	file_name	전송하고자 하는 file의 이름
	file_pointer	File을 read/write 하기 위한 객체
Receiver	file_packet_start	최근 수신한 연속된 패킷 중 가장 마지막 패킷의 sequence number ex) 1,2,4,5를 수신한 경우 file_packet_start는 2
	udp_recv_packet	수신한 패킷을 저장하는 list (file을 순서대로 재조립하기 위해 사용됨)
	udp_recv_flag	패킷 수신 여부를 기록하기 위한 list
Sender	udp_send_packet	현재 windows 만큼의 패킷을 저장하고 있는 dictionary (Key: packet number / Value: (전송한 시각 time, packet)
	udp_ack_windows	현재 windows 만큼의 패킷에 대한 ACK 수신 여부를 기록하기 위한 list
	udp_ack_num	현재 windows에서 가장 첫 번째 위치한 패킷의 sequence number (= send_base)
	udp_last_ack_num	다음 전송해야 할 패킷의 sequence number (= nextseqnum)

d. FileTransfer 내 함수

(1) 함수 tcp_packet_pack

함수 tcp_packet_pack

```
    @staticmethod
    def tcp_packet_pack(packet_type:bytes,data:bytes)->bytes:
        data_len =len(data)
        packet =packet_type +struct.pack(">H",data_len)+data
        packet =packet +bytes(PACKET_SIZE -len(packet))
        return packet
```

설명

Return value : packet (bytes)
요약 : 전송할 데이터를 패킷으로 변환

data_len =len(data)
전송할 데이터의 길이(data_len)를 계산
packet =packet_type +struct.pack(">H",data_len)+data
패킷을 구성, 다음과 같이 구성된다:
1. 패킷 타입(packet_type)
2. 데이터 길이(data_len)를 빅 엔디안 방식으로 포맷팅한 후(struct.pack(">H", data_len))
3. 실제 데이터(data)
packet =packet +bytes(PACKET_SIZE -len(packet))
패킷의 크기를 고정된 크기(PACKET_SIZE)에 맞추기 위해,
패킷의 길이가 PACKET_SIZE보다 작은 경우, 남은 부분을 0으로 채워서 패킷의 크기를 조정
return packet
조합된 패킷을 반환

(2) 함수 tcp_packet_unpack

함수 tcp_packet_unpack

```
    @staticmethod
    def tcp_packet_unpack(packet:bytes)->Tuple[bytes,bytes]:
        packet_type =packet[:1]
        data_len =struct.unpack(">H",packet[1:3])[0]
        data =packet[3:3+data_len]
        return packet_type,data
```

설명

Return value : packet_type, data 튜플로 반환
요약 : 수신한 패킷을 실제 데이터로 변환

```
packet_type =packet[:1]
        패킷의 첫 번째 바이트를 추출하여 패킷 타입(packet_type)으로 지정
data_len =struct.unpack(">H",packet[1:3])[0]
        패킷의 2번째와 3번째 바이트(packet[1:3])를 사용하여 데이터의 길이(data_len)를 추출
        빅 엔디안 방식으로 인코딩된 2바이트 정수를 해석하여 그 값을 반환
        [0]은 unpack 함수가 반환한 튜플의 첫 번째 요소를 의미
data =packet[3:3+data_len]
        패킷에서 데이터 길이(data_len)에 해당하는 부분을 추출하여 실제 데이터(data)로 지정
        패킷의 4번째 바이트부터 시작하여 data_len만큼의 길이를 가진 데이터 부분
return packet_type,data
        추출된 패킷 타입(packet_type)과 데이터(data)를 튜플 형태로 반환
```

(3) 함수 udp_packet_pack

함수 udp_packet_pack

```
    @staticmethod
    def udp_packet_pack(packet_type:bytes,ack_num:Any,data:bytes)->bytes:
        data_len =len(data)
        if type(ack_num)==int:
            packet =packet_type +struct.pack(">HH",ack_num,data_len)+data
        elif type(ack_num)==bytes:
            packet =packet_type +ack_num +struct.pack(">H",data_len)+data
        return packet
```

설명

Return value : packet
요약 : 전송할 데이터를 패킷으로 변환

```
data_len =len(data)
        전송할 데이터의 길이(data_len)를 계산
if type(ack_num)==int:
    packet =packet_type +struct.pack(">HH",ack_num,data_len)+data
        ack_num이 정수형인 경우
        ack_num과 data_len을 빅 엔디안 방식으로 포맷팅하여 패킷을 조합
elif type(ack_num)==bytes:
    packet =packet_type +ack_num +struct.pack(">H",data_len)+data
        ack_num이 바이트형인 경우
        ack_num을 직접 패킷에 추가하고, data_len을 빅 엔디안 방식으로 포맷팅하여 패킷을 조합
return packet
        조합된 패킷 반환
```

(4) 함수 udp_packet_unpack

함수 udp_packet_unpack

```
    @staticmethod
    def udp_packet_unpack(packet:bytes)->Tuple[bytes,int,bytes]:
        packet_type =packet[:1]
        ack_num,data_len =struct.unpack(">HH",packet[1:5])
        data =packet[5:5+data_len]
        return packet_type,ack_num,data
```

설명

Return value : packet type, data
요약 : 수신한 패킷을 실제 데이터로 변환

packet_type =packet[:1]
패킷의 첫 번째 바이트를 추출하여 패킷 타입(packet_type)으로 지정
ack_num,data_len =struct.unpack(">HH",packet[1:5])
패킷의 2번째부터 5번째 바이트(packet[1:5])를 사용하여
확인 번호(ack_num)와 데이터 길이(data_len)를 추출
빅 엔디안 방식으로 인코딩된 두 개의 2바이트 정수를 해석하여 그 값을 반환
data =packet[5:5+data_len]
패킷에서 데이터 길이(data_len)에 해당하는 부분을 추출하여 실제 데이터(data)로 지정
패킷의 6번째 바이트부터 시작하여 data_len만큼의 길이를 가진 데이터 부분에 해당
return packet_type,ack_num,data
추출된 패킷 타입(packet_type), 확인 번호(ack_num), 데이터(data)를 튜플 형태로 반환

(5) 함수 udp_ack_bytes

함수 udp_ack_bytes

```
    @staticmethod
    def udp_ack_bytes(packet:bytes)->bytes:
        return packet[1:3]
```

설명

Return value : ACK number (bytes)
요약 : 수신한 패킷에서 ACK number 반환

return packet[1:3]
패킷의 2번째와 3번째 바이트(packet[1:3])를 추출하여 반환

(6) 함수 tcp_file_name_packet

함수 tcp_file_name_packet

```
    def tcp_file_name_packet(self,file_name:str)->bytes:
        # TCP 통신에서의 file 이름 전송용 패킷 생성
        # 패킷 구조: ₩x00 + (이름 data 크기) + (파일 이름 data)
        data =file_name.encode(ENCODING)
        return self.tcp_packet_pack(PACKET_TYPE_FILE_START,data)
```

설명

Return value : File 이름이 담긴 packet
요약 : File 이름을 전송하기 위한 패킷 생성

data =file_name.encode(ENCODING)
제공된 파일 이름(file_name)을 지정된 인코딩(ENCODING)을 사용하여 바이트로 인코딩
return self.tcp_packet_pack(PACKET_TYPE_FILE_START,data)
tcp_packet_pack 메서드를 사용하여 실제 패킷을 생성
패킷 타입(PACKET_TYPE_FILE_START)과 인코딩된 파일 이름 데이터(data)를 전달합니다.
생성된 패킷은 반환됨.

(7) 함수 tcp_file_data_packet

함수 tcp_file_data_packet

```
def tcp_file_data_packet(self)->Tuple[bool,bytes]:
    data =self.file_pointer.read(PACKET_SIZE -1 -2)
    if data:
        return True,self.tcp_packet_pack(PACKET_TYPE_FILE_DATA,data)
    else:
        return False,None
```

설명

Return value : 데이터 존재 여부(Boolean), packet
요약 : File을 구성하는 data를 읽고 패킷 단위로 생성하여 전송

```
data =self.file_pointer.read(PACKET_SIZE -1 -2)
        현재 열려 있는 파일(self.file_pointer)에서
        지정된 크기(PACKET_SIZE - 1 - 2)만큼의 데이터를 읽어 data에 저장.
        PACKET_SIZE - 1 - 2는 패킷 타입과 데이터 길이를 위한 공간을 제외한 크기를 의미
if data:
    return True,self.tcp_packet_pack(PACKET_TYPE_FILE_DATA,data)
        읽어온 데이터가 존재하는 경우(if data)
        tcp_packet_pack 메서드를 사용하여 실제 패킷을 생성하고, True와 함께 생성된 패킷을 반환
else:
    return False,None
        데이터가 없는 경우(else), False와 None을 반환
        더 이상 전송할 데이터가 없음을 의미
```

(8) 함수 udp_file_data

함수 udp_file_data

```
def udp_file_data(self)->Tuple[bool,bytes]:
    data =self.file_pointer.read(PACKET_SIZE -1 -2 -2)
    if data:
        return True,data
    else:
        return False,None
```

설명

Return value : 데이터 존재 여부(Boolean), packet
요약 : 현재 전송할 data 찾기

```
data =self.file_pointer.read(PACKET_SIZE -1 -2 -2)
        현재 열려 있는 파일(self.file_pointer)에서
        지정된 크기(PACKET_SIZE - 1 - 2 - 2)만큼의 데이터를 읽어 data에 저장
if data:
    return True,data
        읽어온 데이터가 존재하는 경우(if data), True와 함께 읽어온 데이터를 반환
else:
    return False,None
        데이터가 없는 경우(else), False와 None을 반환
        더 이상 전송할 데이터가 없음을 의미
```

(9) 함수 tcp_file_name_transfer

함수 tcp_file_name_transfer

```
def tcp_file_name_transfer(self,filename:str,tcp_send_func:Callable)->None:
    packet =self.tcp_file_name_packet(filename)
    tcp_send_func(packet)
```

설명

Return value : None
요약 : TCP 통신을 사용하여 파일 이름을 전송

packet =self.tcp_file_name_packet(filename)
파일 이름을 전송하기 위한 TCP 패킷을 생성
tcp_send_func(packet)
생성된 패킷을 전송하는 데 사용할 함수(tcp_send_func)에 패킷을 전달하여 실제로 전송

(10) 함수 tcp_file_send

함수 tcp_file_send

```
def tcp_file_send(self,filename:str,tcp_send_func:Callable)->None:
    basename =os.path.basename(filename)
    self.file_pointer =open(filename,"rb")
    self.tcp_file_name_transfer(basename,tcp_send_func)

    data_ready,packet =self.tcp_file_data_packet()
    while data_ready:
        tcp_send_func(packet)
        data_ready,packet =self.tcp_file_data_packet()

    tcp_send_func(TCP_FILE_TRANSFER_END)

    self.file_pointer.close()
    self.file_pointer =None
```

설명

Return value : None
요약 : TCP 통신을 사용하여 파일 전송
(같은 디렉토리에 존재하는 operation.py에서 pj_1.py와 연계되어 활용됨)

basename =os.path.basename(filename)
전송할 파일의 경로에서 파일 이름만 추출

self.file_pointer =open(filename,"rb")
지정된 파일을 바이너리 읽기 모드("rb", read binary)로 열기

self.tcp_file_name_transfer(basename,tcp_send_func)
tcp_file_name_transfer 함수 통해 파일 이름을 TCP의 tcp_send_func 함수로 전송

data_ready,packet =self.tcp_file_data_packet()
패킷으로부터 데이터를 읽어오고, 데이터의 준비 상태(data_ready)와 패킷(packet)을 가져오기

while data_ready:
    tcp_send_func(packet)
    data_ready,packet =self.tcp_file_data_packet()
data_ready가 참인 동안, 즉 데이터가 남아있는 동안, 패킷을 TCP를 통해 전송
이후 다음 패킷 준비

tcp_send_func(TCP_FILE_TRANSFER_END)
파일 전송이 완료되었음을 나타내는 신호(TCP_FILE_TRANSFER_END)를 전송

self.file_pointer.close()
self.file_pointer =None
열려 있는 파일 포인터를 닫고, None으로 설정하여 더 이상의 파일 작업이 없음을 알림

(11) 함수 tcp_file_receive

함수 tcp_file_receive

```
def tcp_file_receive(self,packet)->int:
    packet_type,data =self.tcp_packet_unpack(packet)

    if packet_type ==PACKET_TYPE_FILE_START:
        basename =data.decode(ENCODING)
        self.file_name =basename
        file_path ='./downloads/(tcp) '+basename
        self.file_pointer=open(file_path,"wb")
        return 0

    elif packet_type ==PACKET_TYPE_FILE_DATA:
        self.file_pointer.write(data)
        return 1

    elif packet_type ==PACKET_TYPE_FILE_END:
        self.file_pointer.close()
        self.file_pointer =None
        return 2
```

설명

Return value : 정수 0 또는 1 또는 2
요약 : TCP를 통한 File 수신 및 저장

```
packet_type,data =self.tcp_packet_unpack(packet)
        수신된 패킷을 타입과 데이터로 분해

if packet_type ==PACKET_TYPE_FILE_START:
        받은 패킷의 타입이 파일 시작(PACKET_TYPE_FILE_START)을 나타내는 경우

basename =data.decode(ENCODING)
self.file_name =basename
file_path ='./downloads/(tcp) '+basename
self.file_pointer=open(file_path,"wb")
return 0
        패킷에서 파일 이름을 추출하고, 해당 이름으로 파일 경로를 생성
        파일은 바이너리 쓰기 모드("wb", write binary)로 열림
        파일 시작 처리 완료되었음을 나타내는 0 반환

elif packet_type ==PACKET_TYPE_FILE_DATA:
        받은 패킷의 타입이 파일 데이터(PACKET_TYPE_FILE_DATA)를 나타내는 경우

self.file_pointer.write(data)
return 1
        수신된 데이터를 열린 파일에 쓰기
        파일 데이터 처리 완료되었음을 나타내는 1 반환

elif packet_type ==PACKET_TYPE_FILE_END:
        받은 패킷의 타입이 파일 종료(PACKET_TYPE_FILE_END)를 나타내는 경우

self.file_pointer.close()
self.file_pointer =None
return 2
        파일 전송이 끝났으므로, 파일을 닫고 파일 포인터를 초기화
        파일 종료 처리가 완료되었음을 나타내는 2 반환
```

(12) 함수 udp_file_name_transfer

함수 udp_file_name_transfer

```
def udp_file_name_transfer(self,file_name:str,udp_send_func:Callable)->None:
    data =file_name.encode(ENCODING)
    self.udp_send_with_record(PACKET_TYPE_FILE_START,data,udp_send_func)
```

설명

Return value : None
요약 : UDP 통신을 사용하여 파일 이름을 전송

data =file_name.encode(ENCODING)
제공된 파일 이름(file_name)을 지정된 인코딩(ENCODING)을 사용하여 바이트로 인코딩
self.udp_send_with_record(PACKET_TYPE_FILE_START,data,udp_send_func)
파일 이름을 나타내는 데이터(data)와 함께
파일 시작을 나타내는 패킷 타입(PACKET_TYPE_FILE_START)을
udp_send_with_record 함수에 전달하여 UDP를 통해 전송

(13) 함수 udp_send_with_record

함수 udp_send_with_record

```
def udp_send_with_record(self,packet_type:bytes,data:bytes,udp_send_func:Callable)->None:
    packet =self.udp_packet_pack(packet_type,self.udp_last_ack_num,data)
    udp_send_func(packet)
    # GBN, SR을 통한 재전송을 위해 packet과 전송 시간을 self.udp_send_packet에 저장한다.
    # 또한 self.udp_lask_ack_num을 update하여 새로 전송할 packet의 ack_num을 update한다.
    self.udp_send_packet[self.udp_last_ack_num]=(time(),packet)
    self.udp_last_ack_num =(self.udp_last_ack_num +1)%UDP_MAX_ACK_NUM
```

설명

Return value : None
요약 : 전송할 data를 패킷으로 변환 후,
패킷 정보(시각,패킷)저장 및 새로 전송할 패킷 number(self.udp_last_ack_num)를 업데이트

packet =self.udp_packet_pack(packet_type,self.udp_last_ack_num,data)
udp_packet_pack 메서드를 사용하여 UDP 패킷을 생성
udp_send_func(packet)
이를 udp_send_func 함수를 통해 실제로 전송.
패킷은 패킷 타입, self.udp_last_ack_num, 그리고 데이터를 포함
self.udp_send_packet[self.udp_last_ack_num]=(time(),packet)
현재 패킷과 그 전송 시간을 self.udp_send_packet 딕셔너리에 기록
이때 키로는 self.udp_last_ack_num를 사용
self.udp_last_ack_num =(self.udp_last_ack_num +1)%UDP_MAX_ACK_NUM
self.udp_last_ack_num를 하나 증가
이를 UDP_MAX_ACK_NUM으로 모듈로 연산하여 확인 번호의 범위를 관리
번호 최대 도달시 다시 0으로 순환

(14) 함수 udp_file_send

함수 udp_file_send

```
def udp_file_send(self,filename:str,udp_send_func:Callable)->None:
    basename =os.path.basename(filename)
    self.file_pointer =open(filename,"rb")
    # udp를 통해 파일의 basename을 전송하고 ack를 기다린다.
    # hint : self.udp_file_name_transfer 함수를 활용할 것
    self.udp_file_name_transfer(basename,udp_send_func)

    while self.udp_ack_num !=self.udp_last_ack_num:
        if self.udp_ack_num ==0:
            if self.udp_time_out():
                self.udp_file_name_transfer(basename,udp_send_func)
                break
            else:
                sleep(UDP_WAIT)

    data_ready,data =self.udp_file_data()

    while data_ready:

        if len(self.udp_send_packet)<UDP_WINDOW_SIZE:
            #window의 크기보다 전송한 패킷의 양의 적은 경우
            self.udp_send_with_record(PACKET_TYPE_FILE_DATA,data,udp_send_func)
            data_ready,data =self.udp_file_data()# 다음 전송할 data를 준비한다.
        else:
            # PIPELINE을 위한 window를 전체를 사용하여 ack를 기다리며 timeout에 대처한다.
            # Timeout이 아닌 경우에는 Sleep(UDP_WAIT)를 사용한다.
            while not self.udp_ack_windows[self.udp_ack_num]:

                if self.udp_time_out():
                    self.udp_pipeline(udp_send_func)
                    break
                else:
                    sleep(UDP_WAIT)

            pass

    # 모든 파일 data의 ack를 기다리고 timeout에 대처한다.
    while not self.udp_ack_windows[self.udp_ack_num]:

        if self.udp_ack_num ==self.udp_last_ack_num:
            break

        if self.udp_time_out():

            self.udp_pipeline(udp_send_func)
        else:
            sleep(UDP_WAIT)
    # 파일 전송이 완료되었음을 알리고 ack에 대비한다.
    self.udp_send_with_record(PACKET_TYPE_FILE_END,b'',udp_send_func)

    while self.udp_ack_num !=self.udp_last_ack_num:

        if self.udp_ack_num !=self.udp_last_ack_num:
            if self.udp_time_out():
                self.udp_send_with_record(PACKET_TYPE_FILE_END,b'',udp_send_func)
                break
            else:
                sleep(UDP_WAIT)

    # 파일 포인터를 제거한다.
    self.file_pointer.close()
    self.file_pointer =None
```

설명

Return value : None
요약 : UDP 통신을 사용하여 파일 전송
(같은 디렉토리에 존재하는 operation.py에서 pj_1.py와 연계되어 활용됨)

```
basename =os.path.basename(filename)
        전송할 파일의 경로에서 파일 이름만 추출

self.file_pointer =open(filename,"rb")
        파일을 바이너리 읽기 모드("rb", read binary)로 열기

self.udp_file_name_transfer(basename,udp_send_func)
        udp_file_name_transfer 메서드를 사용하여 파일 이름(basename)을 UDP를 통해 전송

while self.udp_ack_num !=self.udp_last_ack_num:
    if self.udp_ack_num ==0:
        if self.udp_time_out():
            self.udp_file_name_transfer(basename,udp_send_func)
            break
        else:
            sleep(UDP_WAIT)
         file_start에 대한 ack 대기. timeout 발생하면 재전송 수행

data_ready,data =self.udp_file_data()
        파일로부터 데이터를 읽어오고, 데이터의 준비 상태(data_ready)와 데이터(data)를 가져오기

while data_ready:
         보낼 데이터가 존재하는 동안 수행
    if len(self.udp_send_packet)<UDP_WINDOW_SIZE:
        self.udp_send_with_record(PACKET_TYPE_FILE_DATA,data,udp_send_func)
        data_ready,data =self.udp_file_data()
         데이터 패킷 전송 및 다음 데이터 준비
    else:
         아직 ack를 받지 않은 패킷들이 udp_window_size보다 많은 경우 ack를 기다림
        while not self.udp_ack_windows[self.udp_ack_num]:
            if self.udp_time_out():
                self.udp_pipeline(udp_send_func)
                break
            else:
                sleep(UDP_WAIT)
        pass
         timeout 발생 시 udp_pipeline을 통해 재전송 수행

while not self.udp_ack_windows[self.udp_ack_num]:
    if self.udp_ack_num ==self.udp_last_ack_num:
        break
    if self.udp_time_out():
        self.udp_pipeline(udp_send_func)
    else:
        sleep(UDP_WAIT)
         모든 파일 데이터의 전송이 완료된 후, 마지막 ack를 받을 때까지 대기
         timeout 발생 시 udp_pipeline을 통해 재전송 수행

self.udp_send_with_record(PACKET_TYPE_FILE_END,b'',udp_send_func)
        파일 전송 완료를 나타내는 신호(PACKET_TYPE_FILE_END)를 전송

while self.udp_ack_num !=self.udp_last_ack_num:
    if self.udp_ack_num !=self.udp_last_ack_num:
        if self.udp_time_out():
            self.udp_send_with_record(PACKET_TYPE_FILE_END,b'',udp_send_func)
            break
        else:
            sleep(UDP_WAIT)
         file_end에 대한 ack 대기. timeout 발생하면 재전송 수행

self.file_pointer.close()
self.file_pointer =None
        열려 있는 파일 포인터를 닫고, None으로 설정하여 더 이상의 파일 작업이 없음을 알림
```

(15) 함수 udp_file_receive

함수 udp_file_receive

```
def udp_file_receive(self,packet:bytes,udp_send_func:Callable)->int:
    ack_bytes =self.udp_ack_bytes(packet)
    packet_type,ack_num,data =self.udp_packet_unpack(packet)

    if packet_type !=PACKET_TYPE_FILE_ACK:
        self.udp_ack_send(ack_bytes,udp_send_func)
        pass

    if packet_type ==PACKET_TYPE_FILE_START:
        if self.file_pointer is not None:
            self.file_pointer.close()
        basename =data.decode(ENCODING)
        self.file_name =basename
        file_path ='./downloads/(udp) '+basename
        self.file_pointer =open(file_path,'wb')
        self.file_packet_start =ack_num +1
        return 0

    elif packet_type ==PACKET_TYPE_FILE_DATA:
        if not self.udp_recv_flag[ack_num]:
            self.udp_recv_packet[ack_num]=data
            self.udp_recv_flag[ack_num]=True

        while self.udp_recv_flag[self.file_packet_start]:
            self.file_pointer.write(self.udp_recv_packet[self.file_packet_start])
            self.udp_recv_flag[self.file_packet_start]=False
            self.file_packet_start =(self.file_packet_start +1)%UDP_MAX_ACK_NUM
        return 1


    elif packet_type ==PACKET_TYPE_FILE_END:
        if self.file_pointer is not None:
            self.file_pointer.close()
            self.file_pointer =None
        return 2


    elif packet_type ==PACKET_TYPE_FILE_ACK:

        if self.udp_ack_num not in self.udp_send_packet.keys():
            return 1

        if self.udp_ack_num ==ack_num:
            self.udp_ack_windows[self.udp_ack_num]=True
            del self.udp_send_packet[self.udp_ack_num]
            self.udp_ack_num =(self.udp_ack_num +1)%UDP_MAX_ACK_NUM

        return 1

    return 1
```

설명

Return value : 정수 0 또는 1 또는 2
요약 : UDP를 통한 File 수신 및 저장
(같은 디렉토리에 존재하는 operation.py에서 pj_1.py와 연계되어 활용됨)

```
ack_bytes =self.udp_ack_bytes(packet)
packet_type,ack_num,data =self.udp_packet_unpack(packet)
        수신된 패킷을 해석해
        ack 바이트(ack_bytes), 패킷 타입(packet_type), 확인 번호(ack_num), 데이터(data)를 추출

if packet_type !=PACKET_TYPE_FILE_ACK:
    self.udp_ack_send(ack_bytes,udp_send_func)
    pass
        수신된 패킷이 ack 패킷이 아닐 경우, 해당 패킷에 대한 ack를 전송

if packet_type ==PACKET_TYPE_FILE_START:
    if self.file_pointer is not None:
        self.file_pointer.close()
    basename =data.decode(ENCODING)
    self.file_name =basename
    file_path ='./downloads/(udp) '+basename
    self.file_pointer =open(file_path,'wb')
    self.file_packet_start =ack_num +1
    return 0
        파일 전송 시작을 나타내는 패킷을 처리
        파일 이름을 추출하고, 새 파일을 생성
        file_packet_start도 ack_num+1로 update

elif packet_type ==PACKET_TYPE_FILE_DATA:
    if not self.udp_recv_flag[ack_num]:
        self.udp_recv_packet[ack_num]=data
        self.udp_recv_flag[ack_num]=True
         첫번째 도착한 데이터인지 확인하고
         맞으면 udp_recv_packet에 저장 후 udp_recv_flag을 True로 업데이트

    while self.udp_recv_flag[self.file_packet_start]:
        self.file_pointer.write(self.udp_recv_packet[self.file_packet_start])
        self.udp_recv_flag[self.file_packet_start]=False
        self.file_packet_start =(self.file_packet_start +1)%UDP_MAX_ACK_NUM
    return 1
         파일 데이터 패킷을 처리
         연속된 패킷이 수신되면 파일에 데이터를 기록하고
         udp_recv_flag를 다시 False로 업데이트

elif packet_type ==PACKET_TYPE_FILE_END:
    if self.file_pointer is not None:
        self.file_pointer.close()
        self.file_pointer =None
    return 2
         파일 전송 종료를 나타내는 패킷을 처리
         파일 전송이 끝났음을 확인하고 파일을 닫음

elif packet_type ==PACKET_TYPE_FILE_ACK:
    if self.udp_ack_num not in self.udp_send_packet.keys():
        return 1
         보낸 데이터 패킷에 대한 ACK가 아닐 경우 무시

    if self.udp_ack_num ==ack_num:
        self.udp_ack_windows[self.udp_ack_num]=True
        del self.udp_send_packet[self.udp_ack_num]
        self.udp_ack_num =(self.udp_ack_num +1)%UDP_MAX_ACK_NUM

    return 1
return 1
         udp_ack_windows에 수신된 ack 패킷 기록
         udp_send_packet에 ack를 받은 패킷을 삭제
         udp_ack_num을 증가시켜 다음 패킷에 대한 ack를 기다림
```

(16) 함수 udp_time_out

함수 udp_time_out

```
    def udp_time_out(self)->bool:
        if time()-self.udp_send_packet[self.udp_ack_num][0]>UDP_TIMEOUT:# timeout
            return True
        else:
            return False
```

설명

Return value : True 또는 False
요약 : 전송한 packet의 timeout 여부 반환

```
if time()-self.udp_send_packet[self.udp_ack_num][0]>UDP_TIMEOUT:
        현재 시각과
        마지막으로 ack를 기다리고 있는 패킷(self.udp_send_packet[self.udp_ack_num][0])이
        전송된 시각의 차이를 계산
    return True
        이 차이가 설정된 타임아웃 값(UDP_TIMEOUT)보다 크다면
        타임아웃이 발생했음을 나타내는 True를 반환
else:
    return False
        그렇지 않으면, False를 반환
```

(17) 함수 udp_pipeline (GBN으로 구현)

함수 udp_pipeline

```
    def udp_pipeline(self,udp_send_func:Callable)->None:
        for ack_num in range(self.udp_ack_num,self.udp_last_ack_num):
            send_time,packet =self.udp_send_packet[ack_num]
            udp_send_func(packet)
            self.udp_send_packet[ack_num]=(time(),packet)
```

설명

Return value : None
요약 : Timeout 후 패킷 재전송 (GBN)

```
for ack_num in range(self.udp_ack_num,self.udp_last_ack_num):
        GBN 방식이기 때문에
        self.udp_ack_num부터 self.udp_last_ack_num까지 전부 재전송

    send_time,packet =self.udp_send_packet[ack_num]
        재전송하기 위해 udp_send_packet에서 send_time과 packet을 다시 꺼냄

    udp_send_func(packet)
        패킷을 재전송
    self.udp_send_packet[ack_num]=(time(),packet)
        패킷을 보낸 시간을 다시 업데이트 (timer restart)
```

(18) 함수 udp_ack_send

함수 udp_ack_send

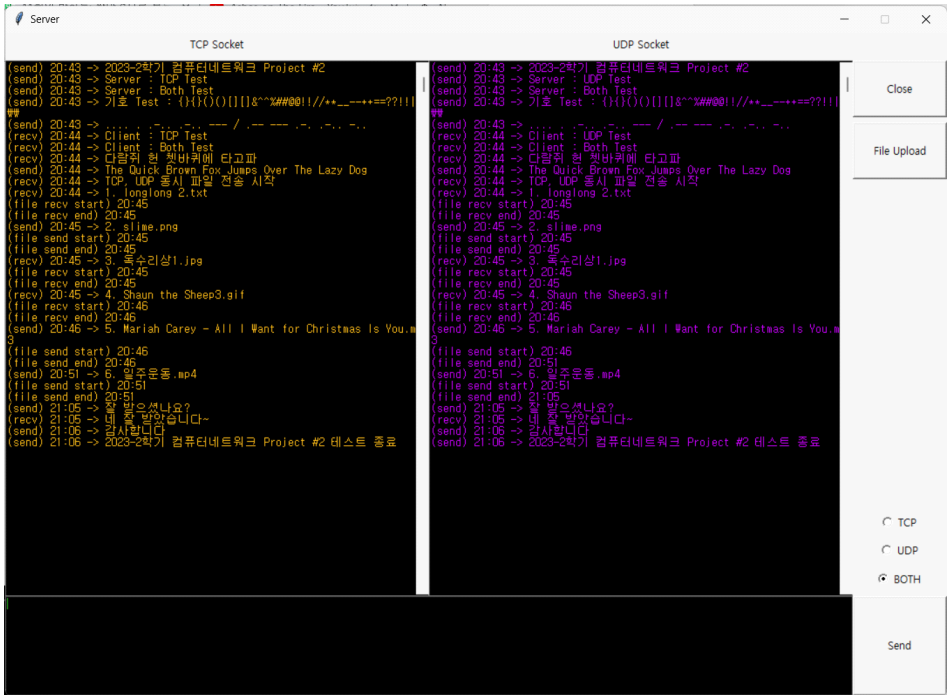
```
    def udp_ack_send(self,ack_bytes:bytes,udp_send_func:Callable):
        packet =PACKET_TYPE_FILE_ACK +ack_bytes
        packet =self.udp_packet_pack(PACKET_TYPE_FILE_ACK,ack_bytes,b'')
        udp_send_func(packet)
```

설명

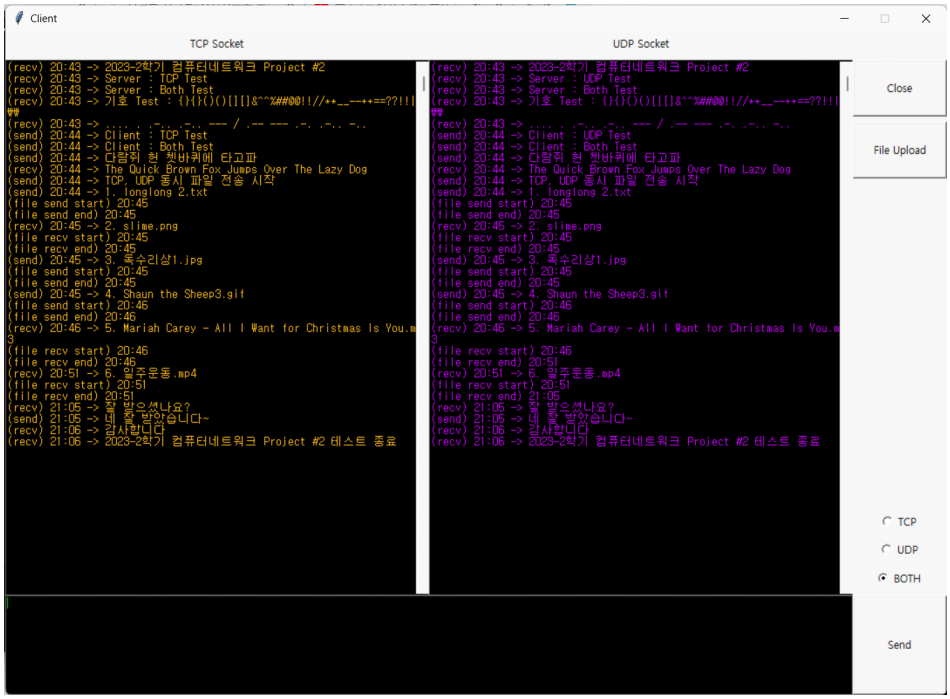
Return value : None
요약 : 수신한 패킷에 대한 ACK 전송

```
packet =PACKET_TYPE_FILE_ACK +ack_bytes
packet =self.udp_packet_pack(PACKET_TYPE_FILE_ACK,ack_bytes,b'')
        udp_packet_pack 메서드를 사용하여 ack 패킷 생성
        패킷은 ack 패킷 타입(PACKET_TYPE_FILE_ACK),
        ack 바이트(ack_bytes), 그리고 빈 데이터(b'')를 포함

udp_send_func(packet)
        생성된 ack 패킷을 udp_send_func 함수를 사용하여 실제로 네트워크를 통해 전송
```

4. 동작 화면 : Server, Client 양쪽에서 서로 10회 이상 다른 메시지를 주고받을 수 있는지, 여기에 더해 양쪽에서 UDP, TCP를 통한 파일 전송이 이루어지는지 확인하였다.



Server

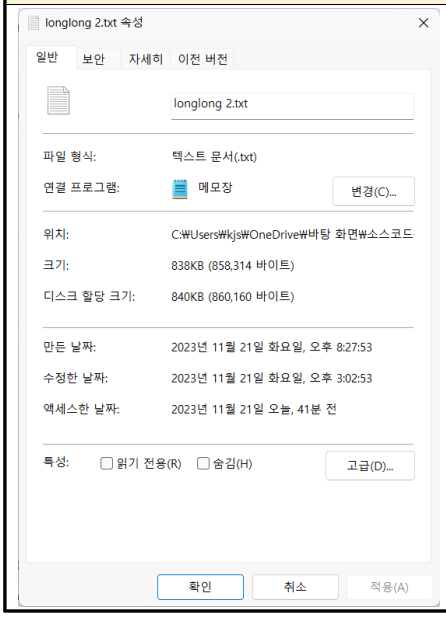
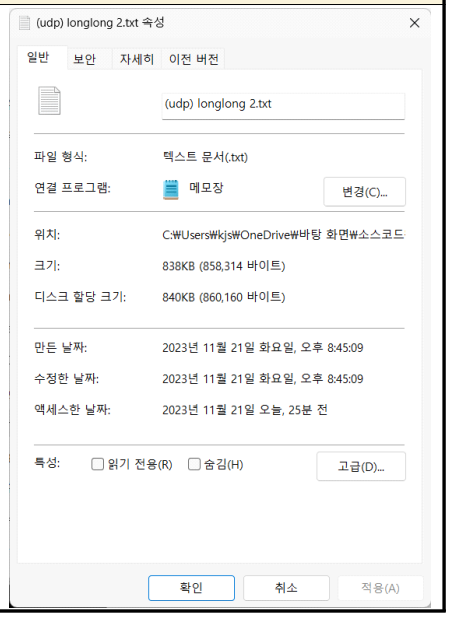


Client

5. 전송된 파일의 Data loss 여부 확인 :
   텍스트, 이미지, 오디오, 비디오 모두 Data loss 없이 잘 전송됨을 확인할 수 있었다.

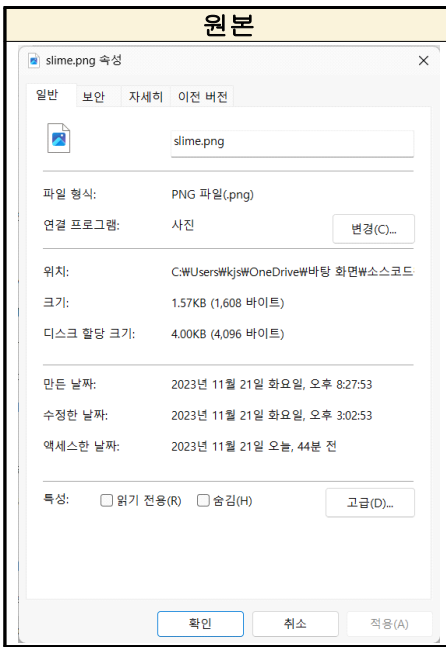
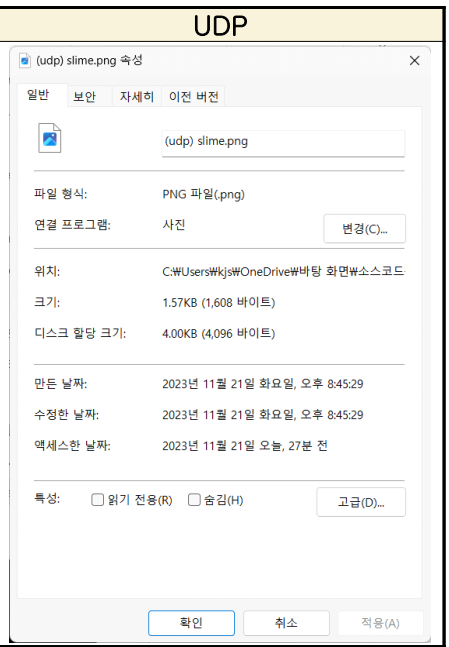
a. 텍스트

(1) txt : longlong 2.txt

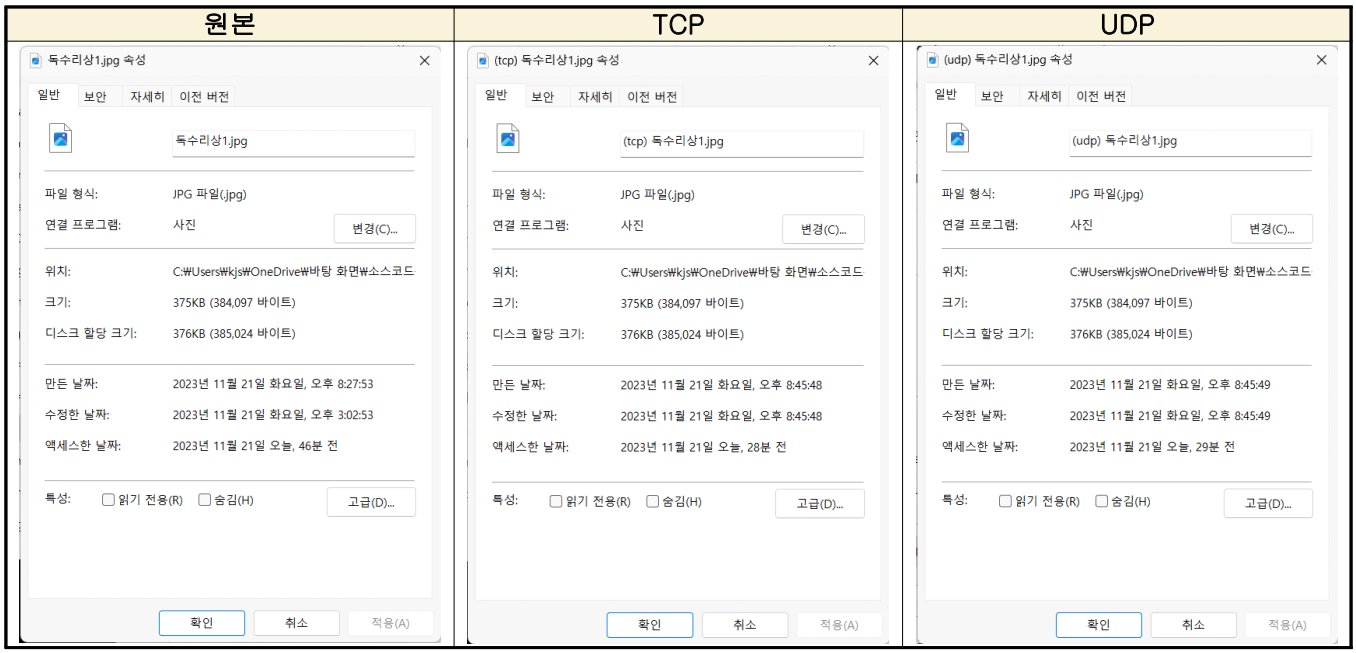
원본	TCP	UDP
		

b. 이미지

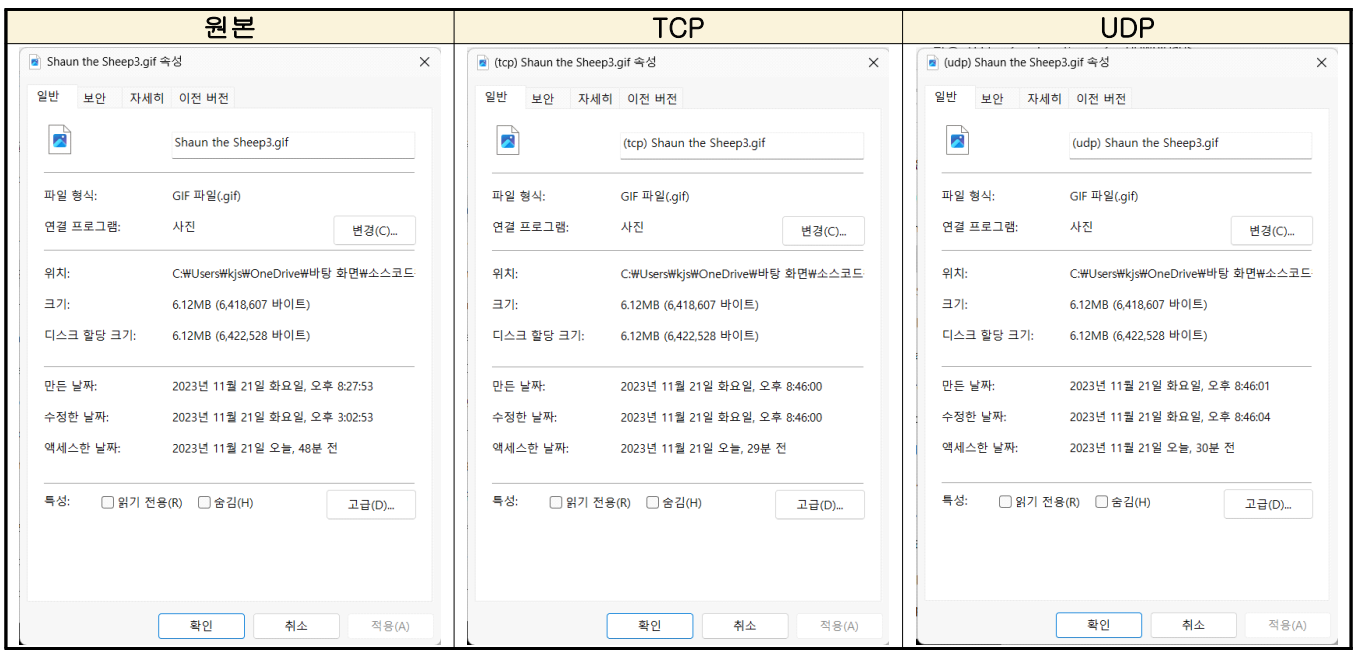
(1) png : slime.png

원본	TCP	UDP
		

(2) jpg : 독수리상1.jpg

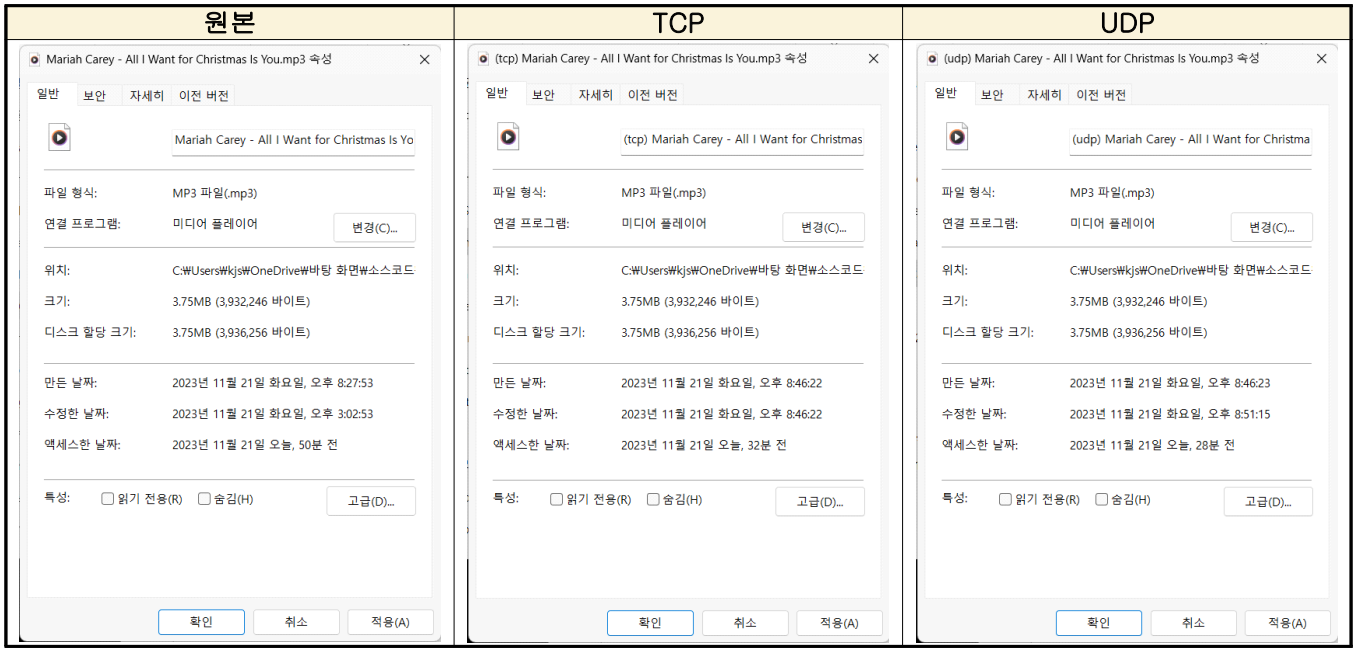


(3) gif : Shaun the Sheep3.gif



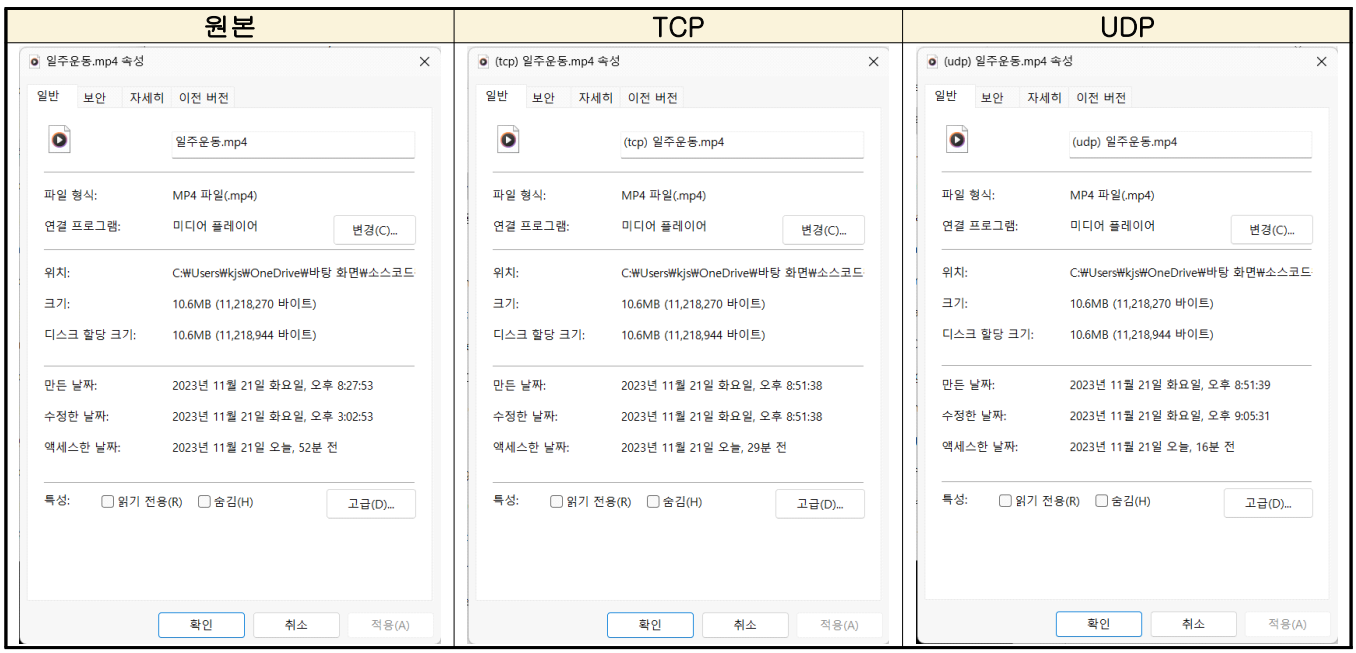
c. 오디오

(1) mp3 : Mariah Carey - All I Want for Christmas Is You.mp3



d. 비디오

(1) mp4 : 일주운동.mp4



V 고찰

1. TCP는 느린 전송속도에 비해 높은 신뢰성의 패킷 전송을 보장하는 반면, UDP는 패킷 전송의 신뢰성은 낮지만 빠른 전송속도를 갖는 것으로 인식되었지만, UDP 역시 신뢰할 수 있는 파일 전송을 구현가능함을 확인할 수 있었다.
2. 하지만 파이썬 소켓프로그래밍에서 UDP에 pipelined protocol을 이용한 신뢰할 수 있는 파일 전송 기능을 구현한 결과 데이터 전송의 신뢰성은 확보되었지만, TCP로 전송할 때보다 더욱 전송속도가 느려지는 상황이 발생하며 빠른 전송속도라는 UDP의 장점이 상쇄되어버리는 역효과를 가져왔다.
3. 즉, 개발하고자 하는 프로그램의 목적에 따라 사용하는 프로토콜도 적재적소에 맞게 선택하는 것이 매우 중요할 것이다. 예를 들어, 메일과 같이 시간에 대한 제약이 적고 신뢰도가 높은 파일의 전송이 요구되는 프로그램은 TCP로, 스트리밍 및 센서 피드백 제어 등 실시간으로 빠른 데이터 교환이 요구되는 프로그램에는 UDP를 사용하는 것이 적절할 것이다.
4. UDP를 통한 신뢰성있는 파일 전송을 수행할 때, 같은 파일이라도 클라이언트에서 서버로의 파일 전송 시 걸리는 시간에 비해, 서버에서 클라이언트로의 파일 전송 시 걸리는 시간이 대체로 더 긴 경향성이 관찰되었다. 정확한 원인 파악을 위해 향후 추가연구가 필요할 것으로 생각된다.

VI 참고문헌

1. Kurose, J., & Ross, K. (2018, October 23). Computer Networking: A Top-Down Approach, Global Edition. Pearson Higher Ed.
2. 컴퓨터네트워크 강의자료 Chapter 1. Introduction
3. 컴퓨터네트워크 강의자료 Chapter 2. Application Layer (2.1-2.3)
4. 컴퓨터네트워크 강의자료 Chapter 3. Transport Layer (3.1-3.4)
5. 컴퓨터네트워크 강의자료 Chapter 3. Transport Layer (3.5-3.7)
6. 컴퓨터네트워크 강의자료 231011_Socket_Programming_Project1
7. 컴퓨터네트워크 강의자료 231101_Socket_Programming_Project2
8. 컴퓨터네트워크 강의자료 Socket_Programming_Projec2 추가설명