**100071011 Computer Networks 2022-2023-2**

**Project-1**

**Reliable File Transfer using Go-Back-N protocol**

**Specification**

|  |  |
| --- | --- |
| **学号 (Student ID)** | **1120200822** |
| **姓名 (Name)** | **郑子帆** |
| **班号 (Class No.)** | **07112002** |
| **授课教师 (Instructor)** | **郑宏** |

**School of Computer**

**Beijing Institute of Technology**

**April 12, 2023**

1. **Requirement Analysis**

**What is GBN?**

GBN is a reliable data transfer protocol that uses sliding window, sequence number, cumulative acknowledgment, checksum and timeout/retransmission mechanisms to deal with frame loss, corruption and disorder. The basic idea of GBN is that the sender can send multiple data frames consecutively without waiting for acknowledgment, within the window size range. The receiver only accepts data frames that arrive in order and sends cumulative ACKs or NAKs if it don’t receive the designated frame in time or receive the wrong frame. If the sender receives duplicate ACKs or NAKs or times out, it resends all the data frames in the window.

**Why GBN?**

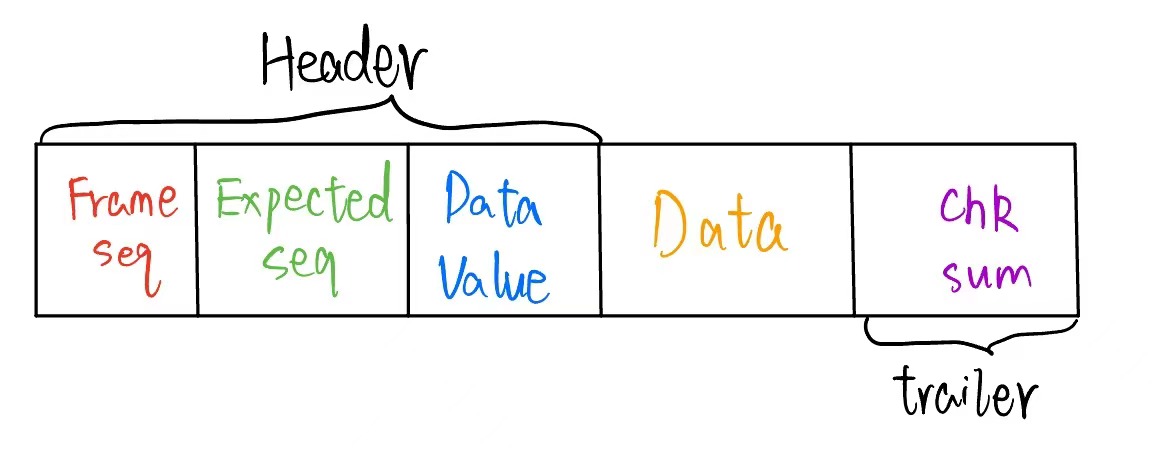
GBN can be used to reliably transfer files, because it can ensure that every frame in the file arrives at the receiver in the correct order and without corruption or loss. It can also reduce network congestion, because it only retransmits frames when necessary.

**Main Issue of this project**

1. Define the frame (PDU) structure. Add a checksum field at the end of the PDU. The checksum uses the CRC-CCITT standard.
2. Use the UDP Socket API to simulate and implement the sending and receiving of PDUs, with each UDP datagram encapsulating one PDU.
3. Simulate the transmission errors and frame loss during data transmission.
4. Configure the protocol related parameters through a configuration file.
5. Record the communication status and rite it to a log file.
6. Implement full-duplex communication.
7. Write a program to analyze the log file.
8. **Design**

**Frame structure**

I designed a PDU structure as follows:



It can be divided into 5 sections:

1. Frame Seq: It represents the sequence number of the present data frames. It takes up for 1 byte. In other words, the maximum sequence number in my program is
2. Expected Seq: It represents the sequence number that is expected to be received by the receivers. Specifically, it is always equal to Frame Seq when transmitting data frames. It takes up for 1 byte.
3. Data Value: The present frame is a data frame if Data Value = 1 and is a ACK/NAK signal frame if Data Value = 0. It takes up for 1byte.
4. Data: If the present frame is a data frame, so this segment means the data bitstream, default to 1024 bytes. But it can be changed through the configuration files.
5. Chk Sum: This is a check field generated by all the contents of a data frame, using the CRC-CCITT standard, occupying 2 bytes.

**CRC-CCITT**

This is a standard for cyclic redundancy code calculation, specifically, it uses the expression to perform XOR calculation.

**Sliding windows**

For the sender, every time a frame is sent to the receiver, the right side of the window slides to the right by one grid, and always keeps the current window size not exceeding the set maximum size. When an ack signal is received, the left side of the window slides to the right by one grid.

For the receiver, the window size is always 1. Whenever a frame that is expected to be received is received, the window moves to the right by one grid.

**Sequence spaces and numbers**

The sequence number of each frame is in the first byte, and the range is 0 to SWSize. After a file is split into multiple frames, the sequence number of each frame is set in order, that is, 0 to SWSize are used cyclically.

**Acknowledgment rules**

For two devices, each device has a sender and a receiver. The receiver of the device can receive data frames as well as ACK/NAK frames. The receiver of both devices remains open and a flag called "recv\_state" is set to describe the current receiving status. When recv\_state=0, it means the device is in a waiting state to receive a file. When recv\_state=1, it means the device is in the process of receiving a file. In both states, ACK/NAK frames can be received (as the states are independent of whether the sending end is transmitting data). When the receiver of a device receives an ACK, it means that the data frame sent by the sending end of the device has been correctly received by the other device, and the sending window can be shifted to the right. When the receiver of a device receives a NAK, it means that the data frame sent by the sending end of the device was not received correctly (due to errors in data or loss of data), and the sending end of the device immediately retransmits the data frames in the sending window.

**Time-out timer maintain**

Frames sent by the sending end of the device have received the corresponding ACK within a specified time, and whether the receiving end of the device has received the expected data frame within a specified time.

**Retransmissions**

Combining the acknowledgment rules and the time-out timer maintenance mentioned above, it can be concluded that when the timer times out or a NAK frame is received, the data frames in the sending window must be retransmitted immediately.

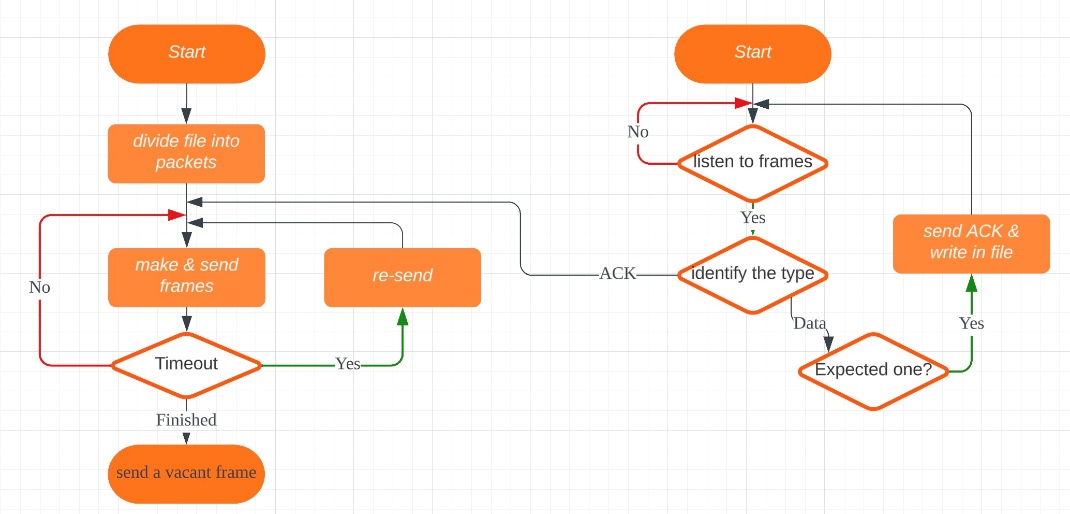
**UDP Socket**

This project is implemented in Python and uses the socket package in Python to achieve UDP communication. The IP address is "localhost" and the port numbers are 40822 and 40823 respectively.

**System model**

图示

描述已自动生成

 **Flow chart(About the operating principles on singe host)**

**Various possible errors and simulations**

According to the requirements in the project document, we have set the Error Rate and Lost Rate. In the specific implementation of the send function, we generate a random value and determine its landing area to decide whether the current frame needs to produce erroneous data or be dropped.

**Multi-thread**

According to the requirements in the project document, we have set the Error Rate and Lost Rate. In the specific implementation of the send function, we generate a random value and determine its landing area to decide whether the current frame needs to produce erroneous data or be dropped.

**Logging**

We have recorded a log file which includes the sender's sequence number (reflecting the number of times sent, starting from 1) or timestamp, the sequence number and status (New, TO, RT) of the PDU sent in this transmission, the sequence number of the PDU already confirmed, the receiver's sequence number (reflecting the number of times received, starting from 1) or timestamp, the expected sequence number of the PDU, the current sequence number and status (DataErr, NoErr, OK) of the received PDU.

**Configuration file and parameters**

We have set up a configuration file which contains the ports of the two terminals, Error Rate, Lost Rate, max sequence number, SWSize, timeout time, etc. We used Python's configparser package to analyze the configuration file and set the parameter values accordingly.

**Queue**

To support continuous transmission of multiple files, I did not implement a simulated message queue. I simplified this problem by not allowing the terminal to input the instruction of the next file to be transmitted while it is sending a file. After completing the current transmission task, the terminal can input and transmit the next file for the next task.

1. **Development and Implementation**

**Development environment & tools**

OS: Windows 10

Language: Python 3.9

development environment: Visual Studio Code

python's packet: socket, time, threading, configparser, random

**Critical functions & Codes**

The functions about PDU: We have *CRCCCITT* to calculate the checksum, *make* to encapsulate a data frame, *extract* to extract the frame’s information.

*def* CRCCCITT(*bitstr*):

    loc = [16, 12, 5, 0]

    p = [0 for i in range(17)]

    for i in loc:

        p[16 - i] = 1   # 反转，p[0]表示16次方，以此类推

    # print(p)

    info\_list = []

    for xstr in *bitstr*:

        bin\_s = bin(xstr).replace('0b', '') # 得到每个字节的二进制串

        bin\_s\_len = len(bin\_s)

        for i in range(8 - bin\_s\_len):  # 不够8位的补0

            info\_list.append(0)

        for j in bin\_s:

            info\_list.append(int(j))

*bitstr* = info\_list.copy()

    strlen = len(*bitstr*)

    for i in range(16):

*bitstr*.append(0)    # 在最后补16个0

    for i in range(strlen):

        if *bitstr*[i] == 1:

            for j in range(17):

*bitstr*[j + i] = *bitstr*[j + i] ^ p[j]

    check\_code = *bitstr*[-16::]  # 取最后16位

    check\_sum = 0

    for i in range(16): # 将16位转换为十进制

        check\_sum = check\_sum \* 2 + check\_code[i]

    return check\_sum

'''

制作PDU 包括header、data、trailer

header包括 发送方frame编号(seq\_byte) 接收方期望接受到的frame编号(ack\_byte) 是否有数据(data\_value\_byte)

trailer包括checksum(2 bytes)

'''

*def* make(*frame\_nr*, *frame\_expected*, *data\_value*, *MAX\_SEQ*, *data*=*b*''):

    seq = *frame\_nr* % *MAX\_SEQ*    # frame编号

    seq\_byte = seq.to\_bytes(1, *byteorder*='big', *signed*=False)  # frame编号转换为byte

    ack = *frame\_expected* % *MAX\_SEQ*  # ack编号

    ack\_byte = ack.to\_bytes(1, *byteorder*='big', *signed*=False)  # ack编号转换为byte

    data\_value\_byte = *data\_value*.to\_bytes(1, *byteorder*='big', *signed*=False)    # data\_value转换为byte

    header = seq\_byte + ack\_byte + data\_value\_byte# frame头部

    checksum = CRCCCITT(header + *data*)      # 计算checksum（按照CRCCCITT标准）

    checksum = checksum.to\_bytes(2, *byteorder*='big', *signed*=False)  # checksum转换为byte

    trailer = checksum  # frame尾部

    return header + *data* + trailer

*def* extract(*packet*):    # 解析一个数据包

    seq = *packet*[0] # 第一个字节为seq

    ack = *packet*[1] # 第二个字节为ack

    data\_value = *packet*[2]  # 第三个字节为data\_value

    data = *packet*[3:-2]

    checksum = int.from\_bytes(*packet*[-2:], *byteorder*='big', *signed*=False)

    return seq, ack, data\_value, data, checksum

We use Timer class to count the time of every frame. It has four functions, *start*, *stop,* *running* and *timeout*.

*class* Timer(object):

    defaultT = -1 # 默认的时间戳

    # 初始化计时器

*def* \_\_init\_\_(*self*, *duration*):

*self*.stTime = *self*.defaultT

*self*.edTime = *duration*

    # 开始计时

*def* start(*self*):

        if *self*.stTime == *self*.defaultT:

*self*.stTime = time.time()

    # 停止计时

*def* stop(*self*):

        if *self*.stTime != *self*.defaultT:

*self*.stTime = *self*.defaultT

    # 判读计时器是否在counting

*def* running(*self*):

        return *self*.stTime != *self*.defaultT

    # 判断是否超时

*def* timeout(*self*):

        if not *self*.running():

            return False

        else:

            return time.time() - *self*.stTime >= *self*.edTime

The core of this project is the abstract of client, CLIENT class. It involves four types of private variables, which are the configuration parameters, sender marks, receiver marks and udp socket. The detailed definitions can be found in the annotations of the code below.

*class* CLIENT:

*def* \_\_init\_\_(*self*, *My\_Port*, *Dest\_Port*, *timeout*=0.03, *MAX\_SEQ*=8, *SWS*=5, *error\_rate*=0, *lost\_rate*=0):

        # 设置地址和端口

*self*.MY\_ADDR = ('localhost', *My\_Port*)

*self*.YOUR\_ADDR = ('localhost', *Dest\_Port*)

        # 设置序列长度和窗口大小

*self*.MAX\_SEQ = *MAX\_SEQ*

*self*.SWS = *SWS*

        # 设置错误率、丢包率

*self*.error\_rate = *error\_rate*

*self*.lost\_rate = *lost\_rate*

        # sender用

*self*.send\_timer = Timer(*timeout*)

*self*.nowack = 0  # 目前需要确认的帧号，取值0 ~ num\_packets

*self*.is\_sending = 0 # 是否正在发送

*self*.next\_frame\_to\_send = 0  # 要发的下一帧的帧号，取值0 ~ num\_packets

*self*.needRT = 0     # 是否需要重传

*self*.needTO = 0     # 是否需要超时重传

*self*.max\_frame\_ID = -1   # 当前传过的最大的帧号

        # receiver用

*self*.frame\_expected = 0  # 接收方当前要接受的序号，取值0 ~ MAX\_SEQ-1

*self*.recv\_state = 0   # 当前状态，0表示等待文件传输 1表示正在进行文件传输

        # udp socket

*self*.sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

*self*.sock.bind(*self*.MY\_ADDR)

*self*.sock.settimeout(0.03)  # 设置获取Ack信号的超时时间

        log\_filepath = "log for " + str(*My\_Port*) + ".txt"

*self*.logger = open(log\_filepath, 'w')   # 打开日志文件

*self*.logCount = 1   # 日志编号

For the sender of the terminal/device, it needs to cut the file into multiple packets and send them in order. In addition, it needs to calculate the time to determine whether it is timeout and retransmit the packets in the window if it's timeout. The code is as follows.

*def* sender(*self*, *filename*):

    # 将要发送的文件拆分，存储到packets中

    packets = []

    try:

        file = open(*filename*, 'rb')

    except IOError:

        print('Cannot open file: ', *filename*)

        exit(0)

*self*.is\_sending = 1

    while True:

        data = file.read(1024) # 读取得到字节流，1024B为一个packet

        if not data:

            break

        packets.append(data)

    num\_packets = len(packets)

    print('This file can be divided into ' + str(num\_packets) + ' packets.')

    window\_size = *self*.set\_window\_size(num\_packets)

    # 循环发送数据包，直到数据包全部发出

    while *self*.nowack < num\_packets:

        # 为数据包添加头部和校验位构成frame

        while *self*.next\_frame\_to\_send < *self*.nowack + window\_size and *self*.next\_frame\_to\_send < num\_packets:

            s = PDU.make(*self*.next\_frame\_to\_send % *self*.MAX\_SEQ, *self*.nowack % *self*.MAX\_SEQ, 1, *self*.MAX\_SEQ, packets[*self*.next\_frame\_to\_send])

            utils.send(s, *self*.sock, *self*.YOUR\_ADDR, *self*.error\_rate, *self*.lost\_rate)  # 发送数据包

            if *self*.next\_frame\_to\_send > *self*.max\_frame\_ID:

*self*.max\_frame\_ID = *self*.next\_frame\_to\_send

*self*.needTO = *self*.needRT = 0

            SendMsg = *self*.getSenderMsg()

*self*.logger.write(SendMsg)

*self*.next\_frame\_to\_send += 1

            window\_size = *self*.set\_window\_size(num\_packets)  # 防止packets越界

        # 窗口达到右边沿，打开计时器

        if not *self*.send\_timer.running():

            # print('start timer')

*self*.send\_timer.start()

        flag = 0    # 判断是否有收到Ack/Dek

        # 等待计时器超时或收到Ack

        while *self*.send\_timer.running() and not *self*.send\_timer.timeout():

            if(*self*.next\_frame\_to\_send < *self*.nowack + window\_size or *self*.needRT == 1):

                flag = 1

                break

            time.sleep(0.01)

        if flag == 1:

*self*.send\_timer.stop()

            if *self*.needRT == 1:    # 如果需要重传，则需要重置next\_frame\_to\_send

*self*.next\_frame\_to\_send = *self*.nowack

            continue

        # 计时器超时，需要重发

        if *self*.send\_timer.timeout():

*self*.send\_timer.stop()

*self*.needTO = 1

*self*.next\_frame\_to\_send = *self*.nowack

    utils.send(*b*'', *self*.sock, *self*.YOUR\_ADDR, *self*.error\_rate, *self*.lost\_rate)  # 用于表示发送结束

    # print('already sent')

*self*.clrset()   # 重置相关标识

    return

For the receiver of the terminal/device, it needs to listen in the ACK/NAK as well as data frames. When receive a frame, it firstly determines what type of the frame is (ACK/NAK or data frame). If it's ACK then the sender of this terminal should move the left side of the window one grid to the right. If it's a data frame then extract is and judge whether the frame is the expected one through comparing seq with expected\_frame and write down in the file if all are correct. Moreover, receiver has a tag *recv\_state* to determine whether the received frame is from a new file.

*def* receiver(*self*):

    file\_count = 0

    file = None     # 文件对象

    while True:

        try:

            pkt, addr = utils.recv(*self*.sock, 1029)

        except:

            continue

        if not pkt:  # 所有数据包全部被接收

            file.close()  # 关闭文件

*self*.recv\_state = 0     # 更改当前的状态为接收完毕

*self*.frame\_expected = 0 # 重置期望接受帧号

            continue

        # if self.recv\_state == 0:

        #     print(pkt[0], pkt[1], pkt[2])

        seq, ack, data\_value, data, check\_sum = PDU.extract(pkt) # 解析

        if data\_value == 0:     # 如果不是数据包，则为ACK，NAK信号

            if seq == 0 and data\_value == 0:  # 说明收到了ack信号

                # print('Receive ACK of packet No.', self.nowack)

*self*.nowack += 1

            elif seq == 1 and ack == *self*.nowack and data\_value == 0: # 说明DataError

*self*.needRT = 1

            continue

        elif data\_value == 1 and *self*.recv\_state == 0:   # 如果是接收的该文件的第一个数据包

*self*.recv\_state = 1     # 更改当前的状态为正在接收

            file = open(*f*'copy\_from\_{*self*.YOUR\_ADDR[1]}\_{file\_count}.txt', 'wb') # 创建接收文件

            file\_count += 1

        # print(seq, ack ,data\_value)

        # 否则为数据包

        # 对收到包进行冗余校验，如果检查包有问题直接丢包

        rev\_check\_sum = PDU.CRCCCITT(pkt[:-2])

        # print(seq, self.frame\_expected)

        if seq != *self*.frame\_expected:  # 说明收到的包序号不对，有丢包

            # print(seq, self.frame\_expected)

            NoErrorMsg = *f*'{*self*.logCount},pdu\_exp={*self*.frame\_expected},pdu\_recv={seq},status=NoErr\n'

*self*.logger.write(NoErrorMsg)

*self*.logCount += 1

        elif rev\_check\_sum == check\_sum:    # 说明收到的包的数据没有问题

*self*.frame\_expected = *self*.inc(*self*.frame\_expected)

            pkt\_ack = PDU.make(0, *self*.frame\_expected, 0, *self*.MAX\_SEQ)  # data段为空,data\_value为0

            utils.send(pkt\_ack, *self*.sock, *self*.YOUR\_ADDR, *type*='ack')

            file.write(data)

            OKMsg = *f*'{*self*.logCount},pdu\_exp={seq},pdu\_recv={seq},status=OK\n'

*self*.logger.write(OKMsg)

*self*.logCount += 1

        else:   # 说明收到的包的数据有问题

            # print(rev\_check\_sum, check\_sum)

            pkt\_dek = PDU.make(1, *self*.frame\_expected, 0, *self*.MAX\_SEQ)

            utils.send(pkt\_dek, *self*.sock, *self*.YOUR\_ADDR, *type*='dek')

            DataErrorMsg = *f*'{*self*.logCount},pdu\_exp={*self*.frame\_expected},pdu\_recv={seq},status=DataErr\n'

*self*.logger.write(DataErrorMsg)

*self*.logCount += 1

    return

We wrote a wrapper for the sendto and recvfrom methods of socket and we add the ErrorRate and LostRate in the functions.

*def* send(*packets*, *sock*, *addr*, *ErrorRate*=0, *LostRate*=0, *type*='data'):

    if *type* != 'data':  # 发送如ack信号时，不会丢包

*sock*.sendto(*packets*, *addr*)

        return

    if *ErrorRate* != 0 and random.randint(0, int(100/*ErrorRate*)) == 0:  # 出错帧

        errorArr = [0]

*packets* = *packets*[0:-2] + bytes(0)  # 将check\_sum的最后一个字节置为0

    if *LostRate* != 0 and random.randint(0, int(100/*LostRate*)) == 0: # 丢失帧

        return

*sock*.sendto(*packets*, *addr*)

    return

*def* recv(*sock*, *max\_size*=1029):

    packets, addr = *sock*.recvfrom(*max\_size*)

    return packets, addr

At last, the terminal. We need to use threading methods to create a thread for the receiver of the terminal/device and run the sender of the terminal/device at the main process.

if \_\_name\_\_ == '\_\_main\_\_':

    '''

    读取配置文件Config.ini, 包含:

    host:

        error\_rate

        loss\_rate

        max\_seq

        SWS

        timeout

    host1:

        port

    host2:

        port

    '''

    config\_path = input('Enter the config file path: ')

    config = configparser.ConfigParser()

    config.read(config\_path)    # 读取配置文件

    host\_port = config.getint('host1', 'port')

    host\_error\_rate = config.getint('host', 'error\_rate')

    host\_loss\_rate = config.getint('host', 'loss\_rate')

    host\_max\_seq = config.getint('host', 'max\_seq')

    host\_SWS = config.getint('host', 'SWS')

    host\_timeout = config.getfloat('host', 'timeout')

    host\_port1 = config.getint('host2', 'port')

    # 创建发送对象和接收对象

    client1 = client.CLIENT(host\_port, host\_port1, host\_timeout, host\_max\_seq, host\_SWS, host\_error\_rate, host\_loss\_rate)

    recv\_thread = threading.Thread(*target*=client1.receiver)

    recv\_thread.start()

    # 输入文件路径并发送文件

    while True:

        filename = input('Enter the file path(Auto receiving, Or enter \"q\" to quit): ')

        if filename == 'q':

            stop\_threads()

            break

        client1.sender(filename)

1. **System Deployment, Startup, and Use**

The system can be deployed on any machine supporting Python and the required libraries. The startup process involves reading the configuration file, initializing the clients, and creating threads for sending and receiving files.

Before running the project, please check whether you have installed all the packages that I used in the code.

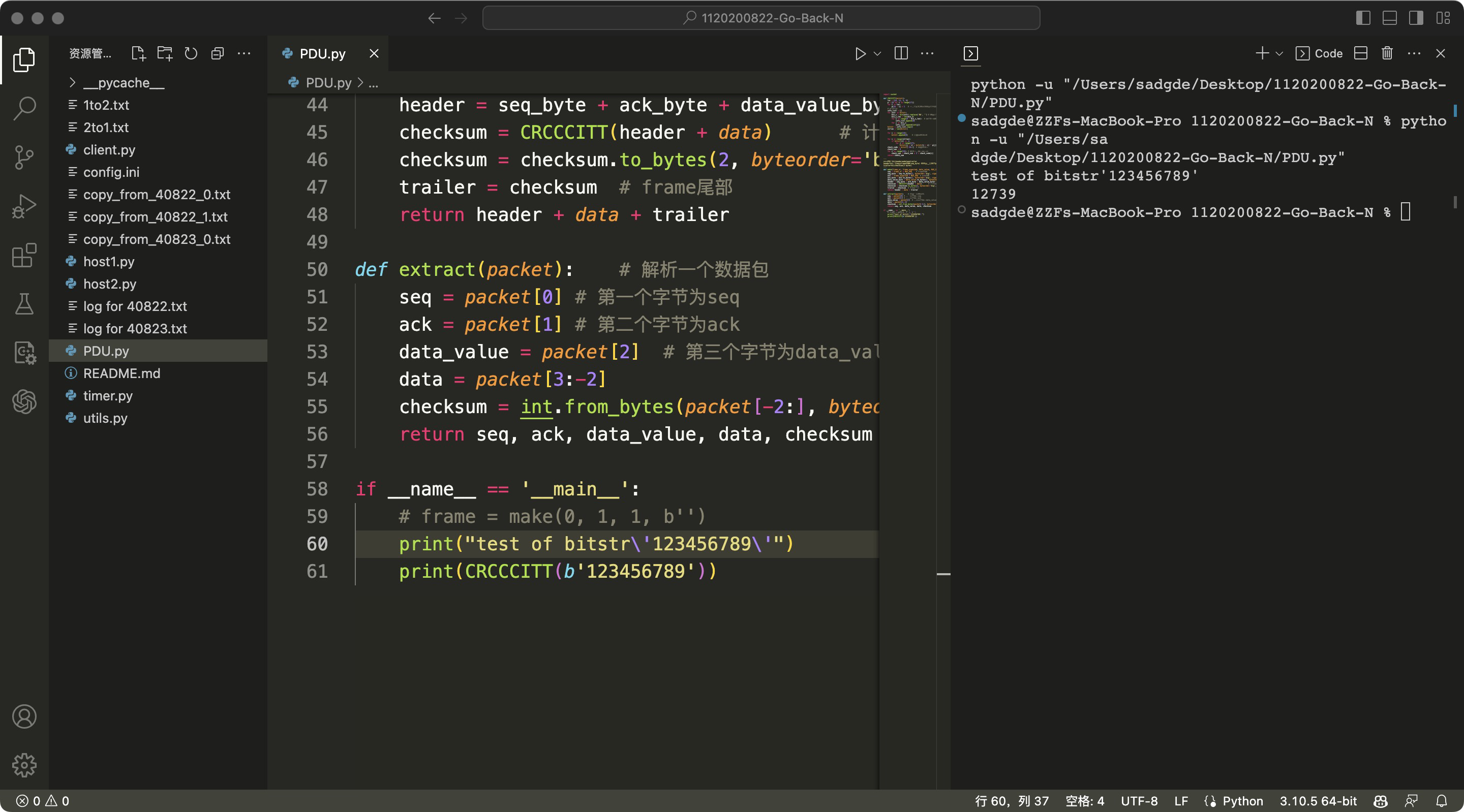
When running the project, please run the host1.py and host2.py in two terminals respectively. Firstly, input the configuration file’s path in the two terminals. Then, input the path of the file that you want to transmit in the terminal. If you want to quit, enter ‘q’ and quit.

1. **System Test**

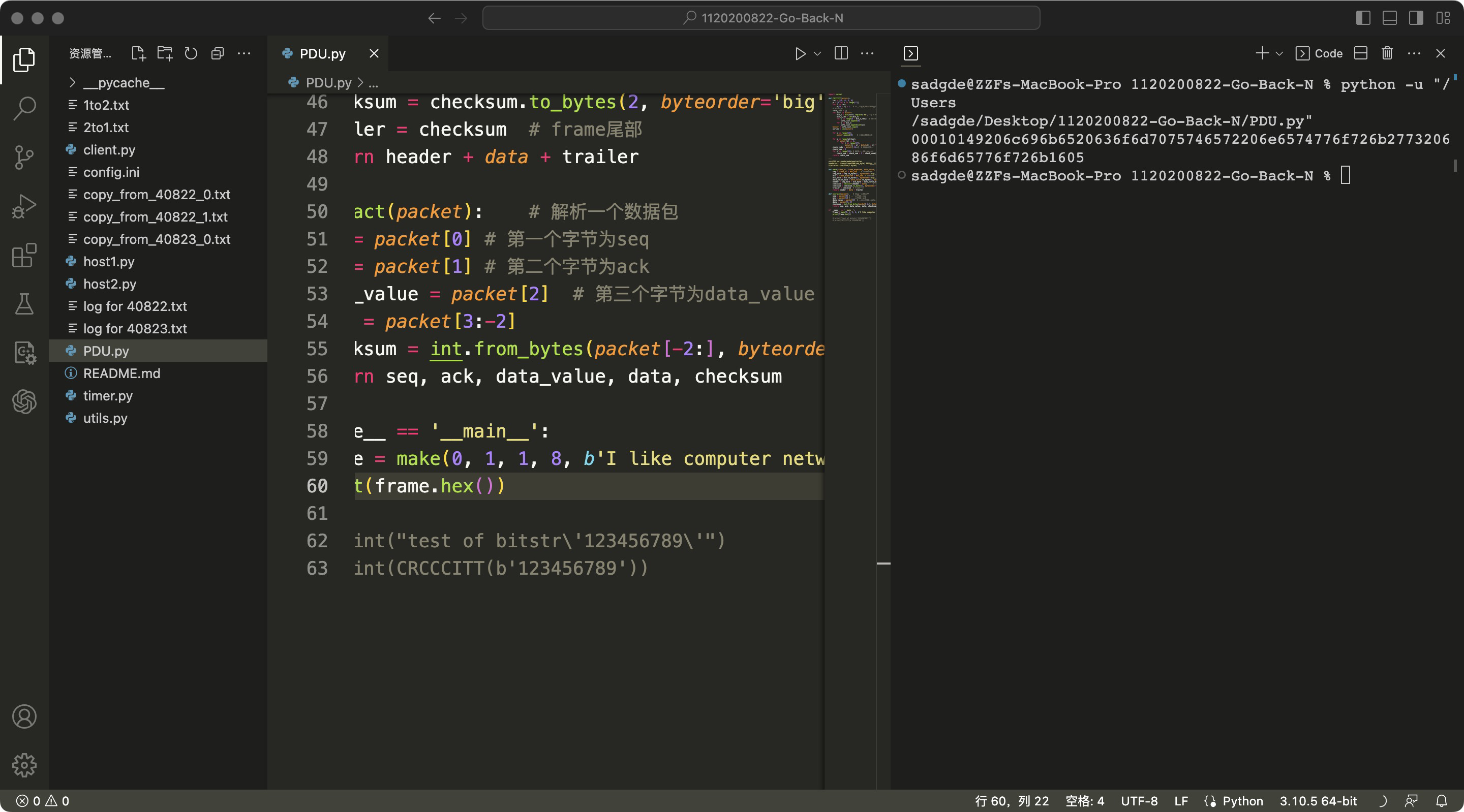
Unit tests and integrated tests are conducted to ensure the proper functioning of each component and the overall system. Test cases include various scenarios such as normal file transfer, packet loss, data corruption, and retransmission due to timeouts. The results of each test case are documented with screenshots to verify the system's reliability and performance.

**Unit Test**

Test the CRC-CCITT function.



Test the make function.



**System Test**

Firstly, we tested One-way file transmission. The configuration parameters are as follows.

A\_Port: 40822

B\_Port: 40823

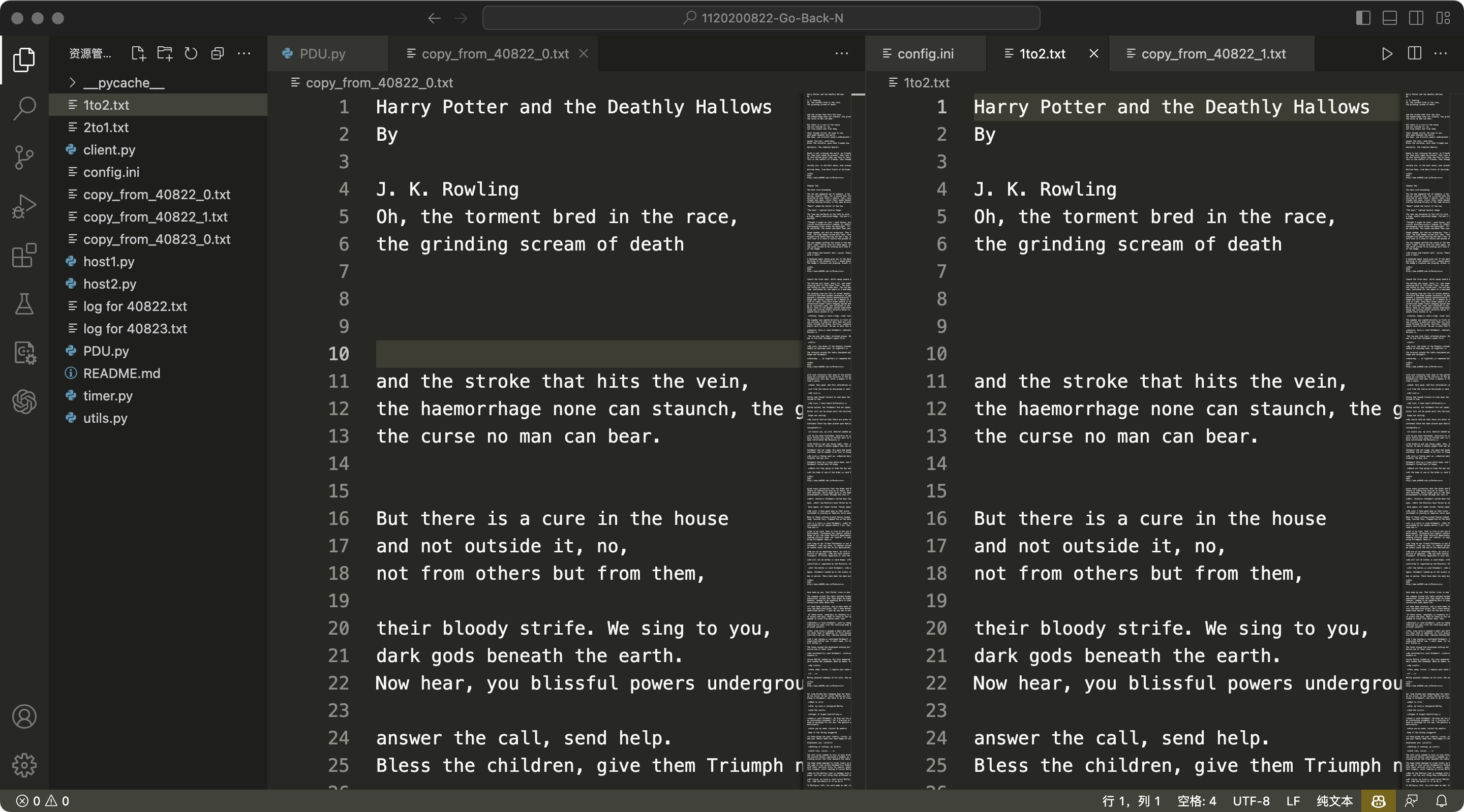
ErrorRate: 3

LostRate: 3

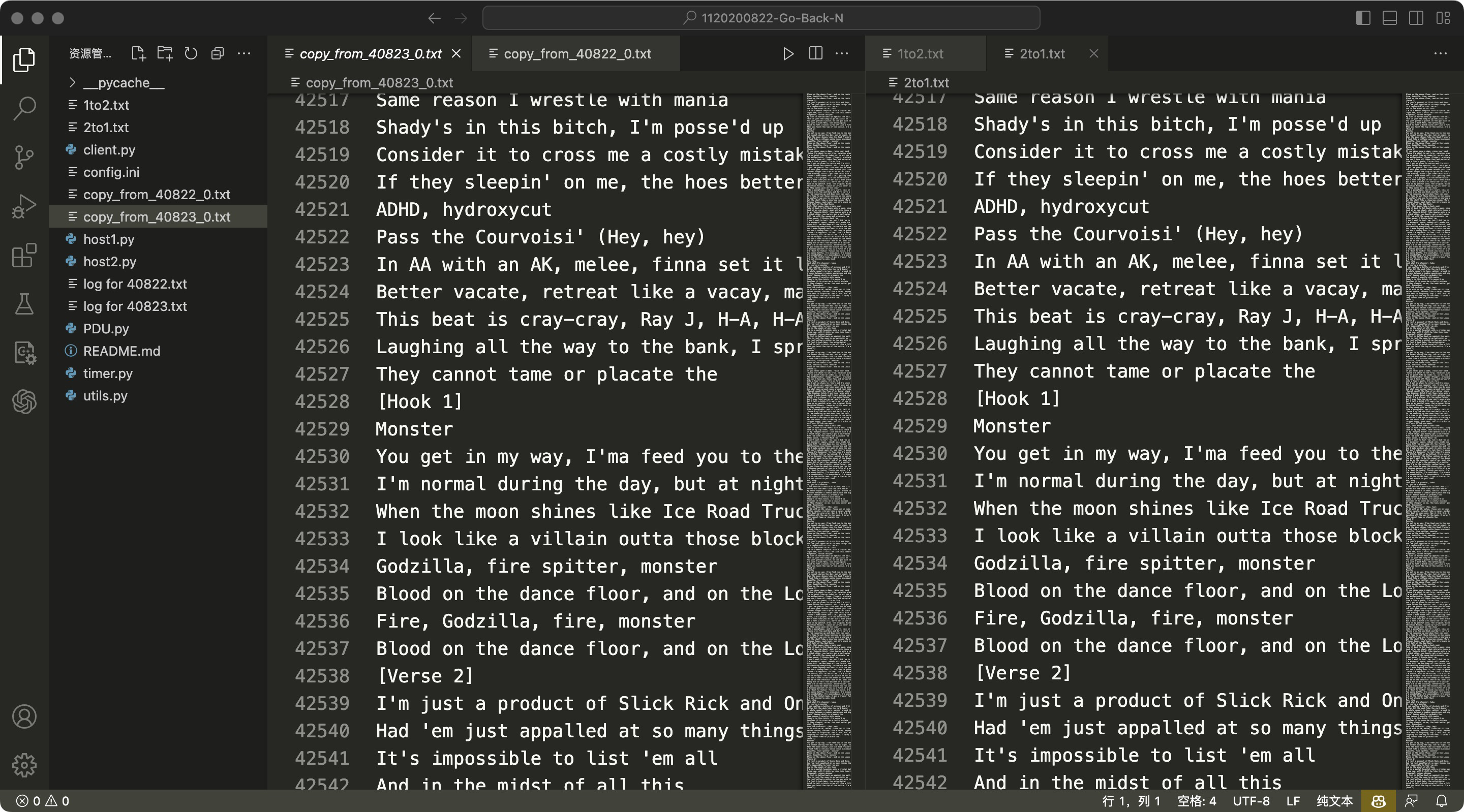
Max\_seq: 8

SWS: 5

Timeout: 0.05(s)



Then, we tested two-ways transmission.



1. **Performance and Analysis**

The performance of the system is analyzed in terms of file transfer speed, error detection and correction, and retransmission handling. The results are presented in data sheets and figures to provide a clear understanding of the system's capabilities and limitations.

We defined “retransmission rate” as the ratio of log messages with status “TO” or “RT” to the whole log messages from sender. The python file *LogAnalyzer.py* calculates the retransmission rate of a single Log file.

Change the Error Rate and Lost Rate, and observe the retransmission rate given by the program. We set Lost Rate = 3% and change Error Rate, the statistics are as follows.

|  |  |  |
| --- | --- | --- |
| ErrorRate(%) | Time(s) | Retransmission Rate(%) |
| 0 | 44.8 | 13.9 |
| 0.1 | 46.7 | 15.2 |
| 0.2 | 47.6 | 19.3 |
| 0.3 | 50.4 | 21.5 |
| 0.5 | 54.0 | 24.2 |

Then set Error Rate = 3% and change Lost Rate, the statistics are as follows.

|  |  |  |
| --- | --- | --- |
| LostRate(%) | Time(s) | Retransmission Rate(%) |
| 0 | 42.7 | 10.5 |
| 0.1 | 44.2 | 13.8 |
| 0.2 | 47.6 | 16.1 |
| 0.3 | 49.8 | 21.5 |
| 0.5 | 52.2 | 24.0 |

The transmission rate and time increase as the error rate and lost rate increase, which is in line with the working principle of the Go-Back-N protocol.

1. **Summary or Conclusions**

In general, the project demonstrates the successful implementation of a reliable file transfer system using the Go-Back-N protocol. The system effectively handles various scenarios such as packet loss, data corruption, and retransmission, ensuring the reliable and ordered delivery of data packets. Through thorough testing and performance analysis, the system proves to be a suitable solution for reliable file transfer in computer networks.

1. **References**

[1] Wetherall D J, Tanenbaum A S. Computer networks[M]. Pearson Education, 2013.

[2] UDP Communication. https://wiki.python.org/moin/UdpCommunication

[3] An Intro to Threading in Python . https://realpython.com/intro-to-python-threading/

[4] recv and recvfrom, socket programming using python. https://stackoverflow.com/questions/

36115971/recv-and-recvfrom-socket-programming-using-python

[5] Go-Back-N Protocol. https://www.baeldung.com/cs/networking-go-back-n-protocol

Note: Because this project is not a research assignment, but a programming assignment, thus I did not read any related papers during the process of completing this project, but learned about technical issues from various blogs, python official website and other websites.

1. **Comments**

I spent a lot of time on this experiment, including two consecutive all-nighters. At first, I didn't know how to start, but after consulting some materials, I gradually completed the program and spent one night debugging it. Although it was painful, I learned a lot of knowledge through this project, deepened my understanding of some protocol content in Computer Networks, as well as improved my coding ability. Of course, it would be even better if this project had a higher weight in the score(lol).